Web Development for Liberty Mutual

Version 1.0.0 June 2017



Copyright © Descriptor Systems, 2017
All Rights Reserved.

This document may not be copied or reproduced in any form without prior written consent of Joel Barnum of Descriptor Systems.

All trademarks belong to their respective companies.

You can contact Descriptor Systems at:

http://www.descriptor.com
sales@descriptor.com
319-362-3906
P.O. Box 461
Marion, IA 52302

Web Development for Liberty Mutual

Chapter 1: Introduction to the Course

Web Development for Liberty Mutual....1-1
Legal Information....1-2
Web Development for Liberty Mutual....1-3
Introductions....1-4
Course Description....1-5
Course Objectives....1-6
Sample Agenda....1-7
Sample Agenda, cont'd....1-8
Sample Agenda, cont'd....1-9
Sample Agenda, cont'd....1-10
Sample Agenda, cont'd....1-11
Course Logistics....1-12

Chapter 2: Introduction to JavaScript

Introduction to JavaScript....2-1
What is JavaScript?....2-2
Where Does JavaScript Run?....2-3
JavaScript vs Java....2-4
History of JavaScript....2-5
ECMAScript Versions....2-6
Browser Support for ECMAScript Versions....2-7
JavaScript is Typically Interpreted....2-8
Four Categories of JavaScript Functionality....2-9
JavaScript Syntax Overview....2-10
Try it Now!....2-11
What is an Object?....2-12
Object Terminology....2-13
Objects Exercise....2-14
Objects in JavaScript....2-15
Client-Side JavaScript Objects....2-16

Embedding JavaScript in HTML 2-17
Using the script Tag 2-18
Using an External File 2-19
Defining Functions 2-20
Modifying Page Elements 2-21
The Form Submission Event 2-22
Chapter Summary 2-23

Chapter 3: Data Types and Assignment

Data Types and Assignment 3-1
JavaScript Comments 3-2
JavaScript Basics 3-3
Variables 3-4
Rules for Identifiers 3-5
Reserved Words 3-6
Numbers 3-7
Booleans 3-8
Truthy and Falsy 3-9
Truthy and Falsy, cont'd 3-10
Strings 3-11
String Literal Escape Codes 3-12
Dates and Times 3-13
Scope of Variables 3-14
Quick Practice 3-15
Chapter Summary 3-16

Chapter 4: JavaScript Operators and Flow Control

JavaScript Operators and Flow Control 4-1
JavaScript Operators 4-2
Compound Assignment Operators 4-3
Increment and Decrement Operators 4-4
Equality Operators 4-5
Comparison Operators 4-6
Boolean Logical Operators 4-7
Conditional Operator 4-8

String Concatenation....4-9
Type Conversion....4-10
Quiz....4-11
Other Operators....4-12
The if Statement....4-13
The if else Statement....4-14
The if else if Statement....4-15
The if else if Statement, cont'd....4-16
The while Loop....4-17
The while Loop, cont'd....4-18
The for Loop....4-19
The for Loop, cont'd....4-20
Chapter Summary....4-21

Chapter 5: JavaScript Objects and Arrays

JavaScript Objects, Arrays and Functions....5-1
What is an Object?....5-2
Categories of Objects....5-3
What Are Properties?....5-4
Creating Objects....5-5
Using Object Literal Syntax....5-6
Using the new Operator....5-7
Objects Are Reference Types....5-8
Prototype Inheritance....5-9
Working with Properties....5-10
The for-in Lop....5-11
What is an Array?....5-12
Creating an Array....5-13
Array Indexes....5-14
Iterating Arrays....5-15
Appending Elements....5-16
Inserting Elements at the Beginning....5-17
Deleting Array Elements....5-18
The splice() Method....5-19

Using splice() to Insert....5-20
The map() Method....5-21
Other Useful Array Methods....5-22
What is a Function?....5-23
Functions are Objects....5-24
Defining Functions....5-25
Function Declarations....5-26
Function Definition Expression....5-27
Function Definition Expression, cont'd....5-28
Optional Arguments....5-29
Global Functions vs Methods....5-30
Scope of Variables Review....5-31
Defining Functions within Functions....5-32
Closures....5-33
Closures, cont'd....5-34
Chapter Summary....5-35

Chapter 6: Introduction to JavaScript AJAX

Introduction to JavaScript AJAX....6-1
What is AJAX?....6-2
Traditional Web Processing....6-3
AJAX Processing....6-4
AJAX Applications....6-5
The XMLHttpRequest Object....6-6
XMLHttpRequest: Raw JavaScript....6-7
Sending a Request: Raw JavaScript....6-8
Sending a Request: Raw JavaScript, cont'd....6-9
Updating the Document: Raw JavaScript....6-10
What is JSON?....6-11
Basic JSON Syntax....6-12
Using JSON in Raw JavaScript....6-13
JavaScript AJAX Libraries....6-14
What is a Promise?....6-15
Introduction to axios....6-16

Using axios to GET JSON....6-17
Chapter Summary....6-18

Chapter 7: Introduction to React

Introduction to React....7-1
Single Page Web Applications....7-2
The React Framework....7-3
What React Isn't....7-4
History of React....7-5
React Architecture....7-6
React Components....7-7
What is Flux?....7-8
React Hello, World: Plain JavaScript....7-9
The createElement API....7-10
Rendering the DOM Tree....7-11
Introduction to JSX....7-12
React Hello, World in JSX....7-13
Defining Components in JSX....7-14
Passing Data to Components and Iterating....7-15
React and AJAX....7-16
React AJAX Example....7-17
Developing Production React Apps....7-18
Chapter Summary....7-19

Chapter 8: Introduction to the Spring Framework

Introduction to the Spring Framework....8-1
What is the Spring Framework?....8-2
History of Spring....8-3
Spring Fundamentals....8-4
Spring Modules....8-5
Writing Spring Applications....8-6
What is a Spring Bean?....8-7
The Spring Container....8-8
Configuring the Container....8-9
Programming To Interfaces....8-10

Dependent Objects Without Spring....8-11
Introduction to Dependency Injection....8-12
Configuring with Annotations....8-13
Configuring with Java Config....8-14
Configuring with Java Config, cont'd....8-15
Configuring with XML....8-16
Injecting Simple Values....8-17
Configuring Annotation-Based Apps....8-18
Creating the Container (Annotation or XML)....8-19
Creating the Container (Java Configuration)....8-20
Steps for a Simple Spring Application....8-21
Hello World From Spring....8-22
Chapter Summary....8-23

Chapter 9: Introduction to Spring Boot

Introduction to Spring Boot....9-1
What is Spring Boot?....9-2
Building Spring Boot Projects....9-3
Generating a Starter Project....9-4
Gradle Project Structure....9-5
The Generated Build Script....9-6
Generating an Eclipse Project....9-7
The 'main' Program....9-8
Spring Boot Autoconfiguration....9-9
Running the Project....9-10
Chapter Summary....9-11

Chapter 10: Spring Annotations

Spring Annotations....10-1
Configuring Spring Beans....10-2
Introduction to Annotations....10-3
Annotations and Spring....10-4
Enabling Annotations....10-5
The @Value Annotation....10-6
Using Annotations for Dependency Injection....10-7

The @Component Annotation 10-8
The @Resource Annotation: By Name 10-9
The @Resource Annotation: By Type 10-10
Autowiring with Annotations 10-11
@Resource vs @Autowired 10-12
The @Qualifier Annotation 10-13
Other Annotations 10-14
Spring and JSR-330 10-15
Chapter Summary 10-16

Chapter 11: Spring Java Configuration

Spring Java Configuration 11-1
Configuring Spring Beans 11-2
Introduction to Java-Based Configuration 11-3
Java-Based Configuration Example 11-4
Why Use Java Configuration? 11-5
The @Bean Annotation 11-6
Performing Setter Injection 11-7
Referencing Other Beans 11-8
Performing Constructor Injection 11-9
Using Autowiring in a Configuration Class 11-10
Autowiring Bean-Creation Methods 11-11
Multiple @Configuration Classes 11-12
Referencing Multiple Configuration Classes 11-13
Importing a Configuration 11-14
Importing Other Configuration Types 11-15
Complete Example 11-16
Using the Groovy DSL for Configuration 11-17
What is Groovy? 11-18
Groovy DSL Example 11-19
Chapter Summary 11-20

Chapter 12: Introduction to JEE

Introduction to JEE 12-1
What is JEE? 12-2

JEE Version History....12-3
Benefits of JEE....12-4
JEE Vendors....12-5
JEE Development Tools....12-6
JEE Technologies....12-7
Deployment Descriptors....12-8
JEE Development and Packaging....12-9
Introduction to Servlets....12-10
Introduction to JSPs....12-11
Introduction to JSF....12-12
Introduction to EJBs....12-13
Introduction to CDI....12-14
Introducing the Java Persistence API....12-15
Introduction to JNDI....12-16
Introduction to JDBC....12-17
Introduction to JMS....12-18
Introduction to JTA....12-19
Introduction to SOAP Web Services....12-20
Introduction to REST....12-21
Introduction to JavaMail....12-22
Chapter Summary....12-23

Chapter 13: Introduction to Servlets

Introduction to Servlets....13-1
What is a Servlet?....13-2
Why Use Servlets?....13-3
Servlet Containers....13-4
The Servlet API....13-5
Writing a Servlet....13-6
A Sample Servlet....13-7
Chapter Summary....13-8

Chapter 14: Introduction to JavaServer Pages

Introduction to JavaServer Pages....14-1
What are JavaServer Pages?....14-2

JSPs Are Actually Servlets....14-3
Why Use JSPs?....14-4
JSP Version History....14-5
Model 1 vs Model 2....14-6
Intro to the JSP Expression Language....14-7
A JSP Consists Of....14-8
Writing Scriptlets....14-9
Deploying JSPs....14-10
Separating Content From Presentation....14-11
What is a JavaBean?....14-12
What is a JavaBean Property?....14-13
Using a JavaBean from a JSP....14-14
Referencing a JavaBean with Actions....14-15
JavaBean Scope....14-16
Retrieving Properties....14-17
Using Expressions to Read Properties....14-18
Intro to the JSP Standard Tag Library....14-19
JSTL Core Tags....14-20
c:forEach Example....14-21
Chapter Summary....14-22

Chapter 15: Introduction to Spring MVC

Introduction to Spring MVC....15-1
Spring and the Web....15-2
The Spring WebApplicationContext....15-3
The Spring WebApplicationContext, cont'd....15-4
Introduction to Spring MVC....15-5
What is the MVC Design Pattern?....15-6
Spring MVC Architecture....15-7
Spring MVC Controllers....15-8
Form Processing Flow: Phase 1....15-9
Form Processing Flow: Phase 2....15-10
Spring MVC Development Steps....15-11
Configuring the Web Application....15-12

A Service Bean....15-13
A Spring MVC Input Form....15-14
A Spring MVC Result Page....15-15
Configuring Controller Annotation Processing....15-16
What is a ViewResolver?....15-17
A Spring MVC Command Controller....15-18
The Spring Configuration File....15-19
MVC With Java Configuration....15-20
Using Thymeleaf Instead of JSP....15-21
Thymeleaf/Spring Recipe....15-22
Chapter Summary....15-23

Chapter 16: Spring and JDBC

Spring and JDBC....16-1
The DAO Design Pattern....16-2
What's Wrong with JDBC?....16-3
Introducing the Spring JDBC Module....16-4
The JdbcTemplate Class....16-5
Spring JDBC Exceptions....16-6
JDBC Connection Review....16-7
Working with Data Sources in Spring....16-8
Configuring a DriverManager Source (XML)....16-9
Configuring a DriverManager (Java Config)....16-10
Configuring a JNDI-Based DataSource (XML)....16-11
Configuring JNDI (Java Config)....16-12
Writing a DAO....16-13
JdbcTemplate Query Methods....16-14
Example: Querying Row Count....16-15
Mapping Rows to Objects....16-16
Example: Query All....16-17
Example: Find Student By ID....16-18
JdbcTemplate Insert, Update and Delete....16-19
Example: Insert....16-20
Complete DAO Example....16-21

Chapter Summary....16-22

Chapter 17: Introduction to Web Services

Introduction to Web Services....17-1
What are Web Services?....17-2
Benefits of Web Services....17-3
Web Services Challenges....17-4
Service Oriented Architectures....17-5
What is SOAP?....17-6
Example SOAP Message....17-7
Example RPC Response....17-8
What is WSDL?....17-9
What is UDDI?....17-10
Using UDDI....17-11
Java APIs for Web Services....17-12
Toolkits for Web Services....17-13
Chapter Summary....17-14

Chapter 18: Introduction to REST

Introduction to REST....18-1
What is REST?....18-2
REST vs SOAP....18-3
Principles of REST....18-4
Resource IDs....18-5
Resources With Multiple Representations....18-6
Use Links to Connect Resources....18-7
Use Standard HTTP Methods....18-8
Stateless Communication....18-9
REST in Java....18-10
What is Restlet?....18-11
A Simple Restlet Resource....18-12
JAX-WS and REST....18-13
JAX-RS....18-14
A JAX-RS Service....18-15
Spring MVC REST....18-16

Chapter Summary....18-17

Chapter 19: Introduction to JAX-RS

Introduction to JAX-RS....19-1
Introduction to JAX-RS....19-2
JAX-RS Annotations....19-3
Writing a Resource Class....19-4
Writing an Application Class....19-5
Content Negotiation....19-6
Negotiation via HTTP Accept Header....19-7
Content Negotiation Example #1....19-8
Content Negotiation Example #2....19-9
Assigning a URI Path....19-10
Accessing Request Parameters....19-11
Accessing URI Parameters....19-12
The @Context Annotation....19-13
@Context Example....19-14
Mapping Java Objects to Content Types....19-15
Writing a JAX-RS Application....19-16
Complete Example....19-17
Chapter Summary....19-18

Chapter 20: RESTful CRUD Services

RESTful CRUD Services....20-1
The Richardson Maturity Model....20-2
RESTful CRUD Service Overview....20-3
HTTP Review....20-4
Media Types....20-5
Implementing Create....20-6
Implementing Retrieve....20-7
Implementing Update....20-8
Implementing Delete....20-9
Complete JAX-RS CRUD Example....20-10
Chapter Summary....20-11

Chapter 21: Spring and REST

Spring and REST....21-1
Spring Support for REST....21-2
Message Converters....21-3
Content Negotiation....21-4
Negotiation via HTTP Accept Header....21-5
Spring REST Annotations....21-6
A Simple RESTful Controller....21-7
Responding to POST Requests....21-8
Responding to PUT Requests....21-9
Configuring a Spring REST Application....21-10
Chapter Summary....21-11

Chapter 22: Additional Spring Boot Topics

Additional Spring Boot Topics....22-1
Dependent Libraries....22-2
Traditional Approaches for Dependencies....22-3
Spring Boot Starter Dependencies....22-4
Viewing Dependencies....22-5
Eliminating Starter Dependencies....22-6
Using a Specific Library....22-7
Auto Configuration....22-8
How Auto Configuration Works....22-9
Overriding Auto Config Externally....22-10
Executable JAR Deployment....22-11
Building a WAR....22-12
Deploying the WAR....22-13
Chapter Summary....22-14

Chapter 23: Transactions in Spring

Transactions in Spring....23-1
Introduction to Transactions....23-2
A Program with No Transactions....23-3
A Program with Transactions....23-4
ACID Transactions....23-5
Global and Local Transactions....23-6

Distributed Transactions....23-7
Spring and Transactions....23-8
Spring Transaction Managers....23-9
Configuring a DataSource Transaction Manager (XML)....23-10
Configuring a DataSource Transaction Manager (Java Config)....23-11
Configuring a JTA Transaction Manager (XML)....23-12
Configuring a JTA Transaction Manager (Java Config)....23-13
Using a Vendor-Specific Transaction Manager....23-14
Transaction Demarcation and Management....23-15
Using Declarative Transaction Management....23-16
Spring Transaction Exception Handling....23-17
Transaction Propagation....23-18
Transaction Propagation Attributes....23-19
Transaction Propagation Example....23-20
A Typical Architecture....23-21
Exceptions and Rolling Back....23-22
Transaction Timeouts....23-23
Transaction Isolation....23-24
Transaction Isolation Levels....23-25
Read Only Transactions....23-26
Specifying Transaction Attributes....23-27
Transaction Annotations....23-28
The @Transactional Annotation....23-29
Complete Declarative Transaction Example....23-30
Chapter Summary....23-31

Chapter 24: WSDL Fundamentals

WSDL Fundamentals....24-1
What is an IDL?....24-2
What is WSDL?....24-3
WSDL and SOA....24-4
History of WSDL....24-5
Creating WSDL....24-6
Using WSDL....24-7

Structure of a WSDL Document 24-8
Structure of a WSDL Document, cont'd 24-9
Abstract versus Concrete WSDL 24-10
The Definitions Element 24-11
The Types Section 24-12
The Messages Section 24-13
The PortType Section 24-14
The Bindings Section 24-15
The Bindings Section, cont'd 24-16
The Bindings Section, cont'd 24-17
The Service Section 24-18
Factoring WSDL 24-19
WSDL 2.0 Overview 24-20
Using WSDL with UDDI 24-21
WS-Policy Overview 24-22
Chapter Summary 24-23

Web Development for Liberty Mutual

Version 1.0.0 June 2017



1 - 1

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Legal Information

Copyright (C) Descriptor Systems 2001, 2017

All Rights Reserved

All trademarks are owned by their respective companies

You can contact Descriptor Systems at:
http://www.descriptor.com
jbarnum@descriptor.com

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Web Development for Liberty Mutual

- Introductions
- Objectives

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introductions

- Please introduce yourself:
  - Name:
  - Java background:
  - HTML, CSS and JavaScript background:
  - Your goal for the course:

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Description

Description: This course covers both client (JavaScript) and server side (Java/Spring) programming for web applications. The course emphasizes RESTful applications using Spring Boot on the server and React on the client.

Prerequisites: Basic Java and HTML/CSS background is assumed.

Audience: Java developers that want to learn JavaScript with React, the Spring Framework and RESTful services.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Objectives

After taking this course, you will be able to:

- Be able to code client-side code using JavaScript and the React user-interface library
- Understand the basics of the Spring Framework including Spring Boot
- Code server-side web applications that use servlets/JSPs and SpringMVC
- Write RESTful clients and services

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda

Day 1

Introduction to the Course

Introduction to JavaScript

JavaScript Data Types

JavaScript Operators and Flow Control

JavaScript Objects, Arrays and Functions

Introduction to AJAX

Introduction to React

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 2

Introduction to the Spring Framework

Introduction to Spring Boot

Spring Annotations and Configuration

Spring Java Configuration

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 3

Introduction to JEE

Introduction to Servlets

Introduction to JSP

Introduction to Spring MVC

Spring and JDBC

Transactions in Spring OPTIONAL

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 4

Introduction to SOAP-based Web Services

Introduction to WSDL OPTIONAL

Introduction to REST

Introduction to JAX-RS

RESTful CRUD (Create, Retrieve, Update, Delete) Applications

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

Day 5

Spring MVC and REST

Additional Spring Boot Topics

Case Study/Wrap Up

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Logistics

- Schedule
  - Start time
  - Lunch
  - End time
  - Breaks

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JavaScript

- What is JavaScript?
- JavaScript Objects

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is JavaScript?

- JavaScript is a modern, object-oriented programming language
- JavaScript is considered to be a **weakly typed** language and supports **inheritance by prototype** and **functional programming**

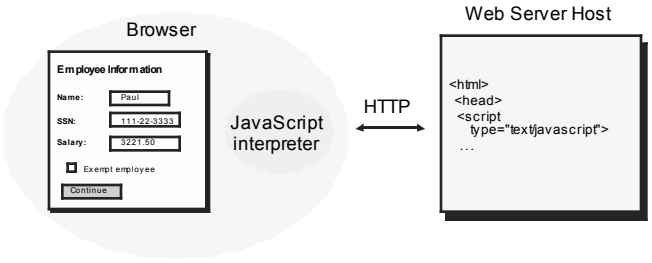


From Wikipedia: In computer science, functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data. It emphasizes the application of functions, in contrast to the imperative programming style, which emphasizes changes in state.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Where Does JavaScript Run?

- A very common environment for JavaScript is referred to as **client side** in which JavaScript runs within a Web browser
- You can use JavaScript in a browser to:
  - Validate forms
  - Expand/collapse page elements
  - Communicate with the server via Ajax
- JavaScript also can run on the server or in **embedded mode** within other applications



The good news here is that almost all modern browsers, even those on mobile platforms such as iOS and Android, provide JavaScript. The bad news is that there are significant differences between the implementations, making cross-browser coding more difficult.

Ajax ostensibly stands for "Asynchronous JavaScript and XML", but the definition is a bit misleading since there's no requirement to use XML.

JavaScript also runs on the server: perhaps the best known example is Node.js, which acts as a programmable Web server. One advantage of using Node.js is that developers can use the same language to write code that runs on both the client and server.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript vs Java

- JavaScript is not really related to the Java programming language, except for a roughly similar syntax



Java is a standalone, general purpose, object-oriented programming language invented by Sun Microsystems and now owned by Oracle.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

History of JavaScript

- JavaScript was originally developed by Netscape in 1995 under the name **LiveScript**
- Microsoft added JavaScript support to Internet Explorer in version 3.0
- Netscape submitted the language to the European Computer Manufacturers Association (ECMA) - the standardized version is known as **ECMAScript**

ECMA adopted the language in 1997.

For trademark reasons, Microsoft's language is known as JScript.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

ECMAScript Versions

- **Version 1.0 1997** - First edition
- **Version 2.0 1998** - Minor changes
- **Version 3.0 1999** - Added regular expressions, exception handling etc.
- **Version 4.0** - Abandoned
- **Version 5.0 2009** - Strict mode, JSON support, etc.
- **Version 6.0 2015** - Focused on writing applications; includes *modules* and additional collection types

---

As of this writing, ECMA is working on Version 7.

Strict mode imposes restrictions on code to help developers avoid common pitfalls (e.g. misuse of global variables). Once strict mode support becomes universal, it would be a good idea for developers to adopt it.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Browser Support for ECMAScript Versions

- Most modern browsers support ECMAScript 3, but version 5 support is becoming universal, even on mobile browsers
- See http://kangax.github.com/es5-compat-table for a chart
- Also see http://caniuse.com/use-strict for ECMAScript 5 **strict mode** support

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript is Typically Interpreted

- Most JavaScript implementations (e.g. in a Web browser) **interpret** the JavaScript source code as it executes
- Thus, users must have access to the actual source code

See http://en.wikipedia.org/wiki/Interpreter_(computing) for a good discussion of the pros and cons of interpreted vs. compiled languages.

Note however, that many modern JavaScript implementations actually perform a just-in-time compile to yield better runtime performance.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Four Categories of JavaScript Functionality

- **Core**: The common syntax and function
- **Client-side**: Implemented by Web browsers
- **Server-side**: Implemented by engines, for example, Node.js
- **Embedded**: Implemented by engines such as Rhino

The most common way to use JavaScript is to use the core functionality along with the client-side functionality provided by Web browsers.

Also note that JavaScript is embedded into the Java virtual machine. Java version 8 provides an embedded JavaScript engine known as Nashorn.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Syntax Overview

- JavaScript is cASe SENsitive
- JavaScript statements can span more than one line and should be terminated with a semicolon
- You define variables using the **var** keyword
- JavaScript supports various ways to add comments, including //
- You can create character strings delimiting with double-quotes or single quotes

```
1    var x = 12;
2
3    alert(x);  // display in a message box
4
5    // display in browser's developer tools console
6    console.log("x: " + x);
```

JavaScript enforces variable type safety less stringently than langauges such as Java or C#.

Note that in the last line, we use the + operator to concatenate a literal string with the numeric value of a variable.

JavaScript's rules on whether semicolons are required or not are somewhat arcane. But you should be OK if put them at the end of statements in the same fashion you might in Java or C#.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Try it Now!

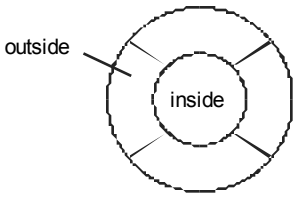
- Open a Web browser and in the URL entry field, type:

```
javascript:alert("Hello, world");
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is an Object?

- In object-oriented programming, an **object** is a self-contained chunk of **state** and **behavior**
- Objects respond to a stimulus (function call) often by working on the object's state
- For example, an "account" object might support a **debit()** behavior that reduces the object's **balance** state



Object Name	Sample State	Sample Behavior
lassie	weight	bark
account12	balance	debit
policy765	amount	payOut

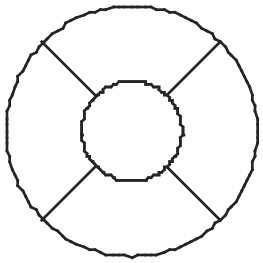
Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

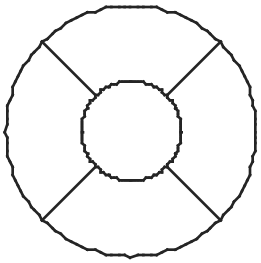
Object Terminology

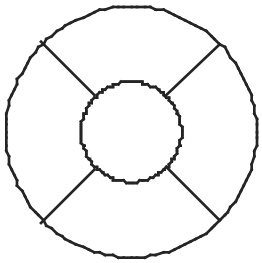
- An object's state is data - synonyms include **properties** and **attributes**
- An object's behavior is program logic provided by **methods** also known as **functions** or **subroutines**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Objects Exercise







Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Objects in JavaScript

- JavaScript itself provides several predefined objects that provide **properties** (state), **methods** (behavior) and **events**
- To access an object, you use the **dot** operator
- It's also possible to create your own custom JavaScript objects using **prototypes**

An "event" is when an object notifies other code that something significant happened to the object. For example, a JavaScript button object fires an event when the user clicks the button.

Unlike languages like Java, C++ or C#, you cannot use "inheritance" to create new classes. Prototyping lets you accomplish much the same effect by adding properties, methods and events to a prototype object.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Client-Side JavaScript Objects

- Client-side JavaScript is runs in a browser - the browser vendor provides the JavaScript interpreter
- The two most important client-side JavaScript **objects** are:
    - **window** - represents the browser's window (or frame)
    - **document** - represents the page

```
1    // store last modified date in a variable 's'
2    var s = document.lastModified;
3
4    // print the contents of the browser window
5    window.print();
```

The first example shows access a property on the document object, while the second shows calling a method on the window object.

The document object exposes the API known as the Document Object Model (DOM).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Embedding JavaScript in HTML

- There are four ways to embed JavaScript in HTML
    - **Inline** using the **script** tag
    - **External file** using the **src** attribute of a script tag
    - **Within an event handler** such as a button's **onclick** event
    - **Within a javascript: URL**

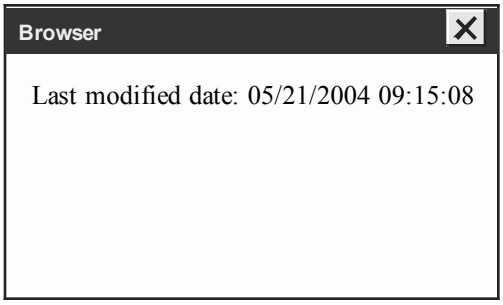
The "javascript:" URL approach is likely the least commonly used.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using the script Tag

- You can embed one or more **script** elements in a Web page's *head* or *body*

```
1    <html>
2      <body>
3        <script>
4          document.write ("Last modified date: ");
5          document.write (document.lastModified);
6        </script>
7      </body>
8    </html>
```



document.write() is a simple way to write to the document and thus the browser, but it's not appropriate for most real-world applications, since the first document.write() clears any existing document. It is useful for debugging and for creating "secondary" windows.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using an External File

- For reuse and for large complex scripts, you can place the script in its own file and reference from an HTML page
- The **src** attributes specifies the URL of the external script
- The script file can be located local to the HTML page or on an external Web server

```
1    file1.html
2    ----------
3
4    <html>
5     <body>
6      <script src="first.js">
7      </script>
8     </body>
9    </html>
```

```
1    first.js
2    ----------
3
4    document.write
5      ("Last modified date: ");
6    document.write
7      (document.lastModified);
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Functions

- A **function** is a block of re-usable code that can accept parameters and return values
- It's common to define functions in the **head** and then call them from other scripts

```
1    <head>
2      <script>
3        function toCelsius(f)
4        {
5          return (5/9) * (f-32);
6        }
7      </script>
8      <script>
9         alert(toCelsius(-40));
10     </script>
11   </head>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Modifying Page Elements

- It's common in client-side JavaScript to respond to **events** and then modify something on the page
- The **document** object provides the **Document Object Model** (DOM) API

```
1    <div id="mydiv">Initial text</div>
2    <button onclick="changeIt()">Press Me</button>
3
4    <script>
5      function changeIt()
6      {
7        var theDiv = document.getElementById("mydiv");
8        theDiv.innerHTML = "New text";
9      }
10   </script>
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Form Submission Event

- If a page defines a form with a submit button, you can write JavaScript that fires when the user submits the form - the JavaScript runs before the browser sends the request to the server

```
1    <script>
2      function confirm() {
3        var aNumber = Math.random();
4        console.log("aNumber: " + aNumber);
5        if(aNumber < 0.5) return true;
6        else return false;
7      }
8    </script>
9
10   <form onsubmit="return confirm()"
11     action="http://localhost/myserver">
12     <input type="submit">
13   </form>
```

If onsubmit evaluates to false, then the browser will not send a request to the server.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- What JavaScript is, and what it is not
- The categories of JavaScript functionality
- How to write and embed simple scripts into HTML

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Data Types and Assignment

- Defining Variables
- Data Types

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Comments

- JavaScript supports two kinds of comments:
    - Single line comments
    - Block comments

```
1    // single-line comment
2
3    /* multi-line comment
4
5      all of this is not processed
6
7      neither is this
8
9    */
```

Single-line comments comment-out until the end of the line.

Block comments comment-out until the end of the comment block.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Basics

- In general, JavaScript programs consist of **statements**, each of which is **optionally** terminated with a semicolon
- JavaScript is a **case-sensitive** language - most predifined keywords are in lower case

```
1    var x = 1;   // define a variable
2
3    alert(x);    // call a method (function)
```

The alert() function is actually not part of Core JavaScript - it's furnished by the "window" object provided by Web browsers.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Variables

- **Variables** are locations in memory that hold a value
- You declare a variable using the **var** keyword
- Even though JavaScript supports data types, variables are **untyped**
- Datatypes include:
    - Primitives: **numbers**, **strings** and **booleans**
    - Objects: **Date**, **Math**, etc.

```
1    var count = 12;                // number
2    var interestRate = 4.5;        // number
3    var name ="Paul Westerberg";   // string
4    var flag = false;              // boolean
5    var now = new Date();          // Date object
6
7    count = "abcd";   // change variable's type!
3 - 4
```

Any programming language needs variables to hold data for processing. In JavaScript, the basic syntax to define a variable requires you to use the keyword "var", followed by the variable's name, followed by an optional initialization value.

Using the "var" keyword for global variables is not technically required, but is for variables defined within functions (local variables).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Rules for Identifiers

- Any length combination of letters, digits and special characters: ('_ ', '$') but must begin with an letter or special character
- JavaScript is case sensitive!

Valid	Invalid
count	1account
interestRate	order-number
purchase_Number1	delivery date

The rules for identifiers are straightforward. You can (and should) use names that are descriptive and you can use underscores or uppercase characters to separate words within identifiers.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Reserved Words

break	this	private
case	throw	protected
catch	try	public
continue	typeof	static
debugger	var	yield
default	void	const
delete	while	
do	with	
else	class	
finally	enum	
for	export	
function	extends	
if	import	
in	super	
instanceof	implements	
new	interface	
return	let	
switch	package	

This page lists the reserved identifiers that you must not use when you assign names to your variables and other identifiers. Some of these keywords are reserved, but not implemented.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Numbers

- Internally, JavaScript stores numbers as IEEE 64-bit floating-point values
- You can specify both integer and floating-point **literals** in expressions

```
1    var a = 12;     // integer literal
2    var b = 0xff;   // integer literal (hex)
3
4    var c = 2.3;    // floating-point literal
5
6    var d = 6.032e23; // floating-point literal
```

The range of numbers is approximately 5e-324 (negative) to 1.7976931348623157e+308 (positive).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Booleans

- The boolean data type allows only the literal values **true** and **false**
- The most common use of booleans is in flow control statements (e.g. the "if" statement)

```
1    var finished = false;
2    var ready = true;
3
4    var ok = confirm("Continue?");
5    if(ok)
6    {
7      // do something
8    }
```

Programmers often use boolean variables in loops or conditional statements (e.g. "if" statements). The keywords "true" and "false" are considered literal boolean constants.

The "confirm()" function is provided by the "window" object in a Web browser and presents the user with a simple dialog box.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Truthy and Falsy

- JavaScript converts the following to **false** and are considered to be **falsy**:
    - undefined
    - null
    - 0
    - -0
    - NaN
    - ""
- All other values are considered to be **truthy**

You can use the "null" keyword to indicate a lack of a value - you can assign it to any variable.

JavaScript also has the notion of "undefined", which is not a keyword and works like null, but slightly differently. JavaScript assigns undefined to variables that have not been initialized.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Truthy and Falsy, cont'd

```
1    var a = "hello";
2    if (a)
3    {
4      // a is truthy, we will get here
5    }
6
7    var b = "";
8    if (b)
9    {
10     // b is falsy, won't get here
11   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Strings

- Strings hold sequences of characters
- To delimit a literal string, use single or double quotes
- Strings define a **length** property and several methods
- You can concatenate strings using the + operator

```
1    var s1 = "If Only You Were Lonely";
2    var index = s1.indexOf ('Only');
3
4    var s2 = s1 + " Not";
5    var count = s1.length;
```

Internally, JavaScript uses the Unicode standard instead of older systems like ASCII and EBCIDIC. Unicode lets you represent all of the world's characters, including those from non-Western languages such as Chinese, Japanese, Arabic and Hebrew. Each character occupies 16 bits of storage. You can read more about Unicode at www.unicode.org.

You can think of strings as arrays of characters, where the first character is at index zero.

Note that JavaScript also defines a RegExp (regular expression) object to perform pattern matching on strings.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

String Literal Escape Codes

- JavaScript defines special **escape codes** for common non-printable characters
- You can initialize a character to any Unicode value using an integer constant

```
\b      Backspace
\t      Horizontal tab
\n      Linefeed
\f      Form feed
\r      Carriage return
\"      Double quote
\'      Single quote
\\      Backslash
\uxxxx  Hexadecimal Unicode code point
\xxx    Octal Unicode code point
```


You can specify any Unicode character using the character's code point and the '\uxxxx' notation, but common characters have shortcuts (e.g. \n). Examples:

var ch1 = '\n'; var ch2 = '\u00DC'; // German U umlaut

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dates and Times

- JavaScript programs use **constructors** to create objects
- Core JavaScript provides a **Date** constructor to let you work with dates and times

```
1    var d1 = new Date();  // current date/time
2
3    var d2 = new Date(1993, 0, 18); // January 18, 1993
4
5    // January 18, 1993 6:10:20PM local
6    var d3 = new Date(1993, 0, 18, 18, 10, 20);
7
8    var year = d2.getFullYear();
9    var month = d2.getMonth();
10   var day = d2.getDate();
11   var dayOfWeek = d2.getDay(); // 0 - Sunday
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Scope of Variables

- JavaScript supports **global** and **local** variables
- Variables defined within a function (method) are **local** and can only be used by that method

```
1    var a = "My global variable";
2
3    function function1()
4    {
5      var b = "My local variable";
6
7      alert(a);  // access global
8    }
9
10   function function2()
11   {
12     alert(a);  // access global
13     alert(b);  // error - out of scope
14   }
```

Global variables are actually implemented as properties on the so-called "global object". In client-side JavaScript in a Web browser, the "window" object acts as the global object.

A function is a block of code with a name and optional arguments. To declare a function, use the "function" keyword, provide a proper identifier, open-and-close parenthesis and a pair of "curly braces".

Unlike languages such as Java and C, JavaScript doesn't have "block scope", in which variables defined within a block are local to that block.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Quick Practice

- Define a variable and initialize it to your favorite whole number:
- Define a variable and initialize it to your favorite floating-point number:
- Define a variable and initialize it to your name:

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The rules for defining and initializing variables
- The JavaScript data types

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Operators and Flow Control

- JavaScript Operators and Type Conversion
- If Statements and Looping

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Operators

- Operators let you perform math, comparisons and so forth
- JavaScript provides 47 operators, which we can divide into categories:
    - Math
    - Equality and Relational
    - Assignment
    - Boolean Logical
    - Integer Bitwise
    - Shift

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Compound Assignment Operators

- JavaScript provides several compound assignment operators

```
*= /= %= += -= <<= >>= >>>= &= ^= |=
```

```
1    var i1 = 12;
2
3    i1 += 45; // same as i1 = i1 + 45;
4
5    alert(i1);  // 57
```


Compound assignment operators are shorthand syntax for performing an operation and an assignment. The example demonstrates using a compound assignment to perform an addition and assignment.

We will cover the other operators later in this chapter.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Increment and Decrement Operators

- These shortcut operators come in two forms: **prefix** and **postfix**
- Postfix operators work after the expression is evaluated, prefix before

```
                ++ --

1    var i1 = 12;
2    i1++; // same as i1 = i1 + 1;
3
4    alert(i1);  // 13
5
6    var i2 = --i1 * 3;
7
8    alert(i2); // 36
9
10   alert(i1); // 12
```

These operators work only on primitives and change the value of a variable. The interesting thing about them is that you can use them in expressions -- they are especially useful in looping constructs.

The example shows using the postfix ++ and prefix --.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Equality Operators

- Equality operators return a boolean result and are most often used in "if" statements or loops
- The **strict equality operator** === compares without performing a type conversion

```
                          == != ===
1    var i1 = 12;
2    var i2 = 14;
3
4    var b = i1 == i2;
5    alert(b);   // false
6
7    var x = "23";
8    var y = 23;
9
10   alert(x == y);  // true
11   alert(x === y); // false
```

The operators always return a boolean type.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Comparison Operators

- The numeric comparison operators work on numbers and strings and return a boolean result

```
            < <= > >=
```

```
1    var i1 = 12;
2    var i2 = 14;
3
4    var b = i1 >= i2;
5    alert(b);  // false
6
7    var q = "abc";
8    var r = "abd";
9    alert(q < r); // true
10
11   var x = "Sam";
12   var y = "sam";
13   alert (x < y); // true
4 - 6
```

The relational operators let you compare numeric values or strings and always return a boolean result. They are especially handy for "if" statements and looping constructs.

Note that string comparison is case-sensitive and that upper-case characters collate before upper-case characters, at least for Western characters.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Boolean Logical Operators

- You can use logical operators to build more complex testing expressions
- These operators work with booleans and "truthy" and "falsy" values

&& || !

```
1    var atHome = true;
2    var eatingDinner = false;
3
4    var result = atHome && eatingDinner;
5    alert(result);  // false
6
7    result = atHome || eatingDinner;
8    alert(result); // true
9
10   result = atHome && !eatingDinner;
11   alert(result); // true
```

These operators are most often used in "if" statements or looping constructs. For an "and" expression to be true, both operands must be true. For "or" to be true, either operand can be true.

We refer to these operators short-circuit operators because JavaScript will not bother evaluating the second part of the expression if it doesn't need to. For example, given:

```
var i1 = 12;
var i2 = 13;

var result = (i1 < i2) || (++i1 == i2 );
```

The ++ will not execute.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Conditional Operator

- The conditional operator is a shortcut for an if-else statement

```
? :
```

```
1    var i1 = 12;
2    var i2 = 14;
3
4    var min = ( i1 < i2 ) ? i1 : i2;
5    alert(min); // 12
```

This ternary (three operand) operator is a syntax inherited from the C language. JavaScript evaluates the first operand -- if it evaluates to true, the entire expression evaluates to the second operand. If false, the entire expression evaluates to the third operand.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

String Concatenation

- You can use the + and += operators to concatenate strings

```
1    var s1 = "Hello";
2    var s2 = s1 + " world";
3
4    alert(s2); // Hello world
5
6    s1 += " Dolly";
7
8    alert(s1); // Hello Dolly
9
10   var s3 = s1 + 23;
11
12   alert(s3); // Hello Dolly23
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Type Conversion

- JavaScript is a weakly typed language and will often convert variables between types "on the fly" during operations
- Depending on the operator, data will be converted from string to number or vice versa

```
1    var a = 9 + "hello";   // "9hello" (string)
2    var b = "3" * "4";     // 12 (number)
3    var c = 3 * "4";       // 12 (number)
4    var d = 2 - "abc";     // NaN
```


JavaScript also provides functions that let you perfom explicit type conversion:

```
var a = Number("3");  // 3 (number)
var b = String(false);  // "false" (string)
var c = Boolean("");    // false
```

There is an equivalent function for Object() not shown here.

Note that you can use the typeof operator to determine any variable's type. It returns a string:

```
var a = 123;
alert(typeof a);   // number
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Quiz

- What is the value of z in each script?

```
A:
  var x = 2;
  var y = "4";
  var z = x * y;
B:
  x = 2;
  y = "4";
  z = x + y;
C:
  x = "2";
  y = "4";
  z = x + y;
D:
  x = true;
  y = "4";
  z = x + y;
E:
  x = true;
  y = 4;
  z = x + y;
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Other Operators

in	Tests if an object contains a property
delete	Removes an object property or array element
instanceof	Tests if an object is in a class
typeof	Returns type name of a value
eval()	A function that dynamically executes code

For completeness, here we show the operators not covered explicitly in this chapter. We will see most of them later in the course.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The if Statement

- The *if* statement examines an arbitrarily complex boolean expression for "truthiness"

```
if ( <expr> )
{
  // statements
}


if ( <expr> )
  // one statement
```


The "if" statement evaluates the provided boolean expression and executes the block or single statement that follows. The parenthesis are required.

If you only need a single statement, you can use the form that does not define a block. However, it never hurts to define the block and can save time later.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The if else Statement

- The *if else* statement examines an arbitrarily complex boolean expression

```
if ( <expr> )
{
  // statements
}
else
{
  // statements
}
```


The "if - else" statement lets you conditionally execute one of two blocks. And as shown on the previous page, if you have only one statement in either block, you can omit the curly braces.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The if else if Statement

- The *if else if* statement examines an arbitrarily complex boolean expression

```
if ( <expr> )
{
  // statements
}
else if ( <expr> )
{
  // statements
}
else
{
  // statements
}
```


The "if - else if - else" lets you create complex decisions. Note that the final "else" is optional.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The if else if Statement, cont'd

```
1    var a = Math.random();
2    var b = Math.random();
3    var c = Math.random();
4
5    if ( (a + b) > c )
6    {
7      alert( "Sum is greater" );
8    }
9    else if ( (a+b) == c )
10   {
11     alert ( "The same" );
12   }
13   else
14   {
15      alert( "Sum NOT greater" );
16   }
```

This simple program retrieves three random integers, then determines if the sum of the first two is greater than, less than, or equal to the third.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The while Loop

- The *while* loop evaluates a boolean expression and stays in the loop as long as the express is "truthy"

```
while ( <expr> )
{
  // statements
}

do
{
  // statements
}
while ( <expr> );
```

The difference between "while" and "do while" is that "do while" always does at least one pass through the loop.

Also note that if you have only one statement, you can omit the curly braces.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The while Loop, cont'd

```
1    var i = 0;
2    while ( i < 10 )
3    {
4      console.log( "in while: " + i );
5      i++;
6    }
7
8    do
9    {
10     console.log( "in do-while: " + i );
11     i++;
12   } while ( i < 10 );
```


This program has two loops. The first runs 10 times with "i" ranging from 0 to 9.

The second loop only runs once.

Note that console.log() is a function supported by many Web browsers.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The for Loop

- The *for* loop is a convenient and concise form of the *while* loop

```
// form #1
for ( <init statement(s)>;<expr>;<exec statement(s)> )
{
  // statements
}


// form #2
for ( variable in object )
{
  // statements
}
```

The "for" loop is "syntactic sugar" that lets you concisely write common looping constructs. The syntax requires a parenthesis with three fields, separated with semicolons. The first field contains statements that are executed before the loop begins -- if you have multiple statements, separate them with commas. The second field (between the semicolons), must be an expression that evaluates to a boolean -- the loop continues as long as the expression is true. The third field is any statements that are executed at the end of the loop -- again, if you have more than one statement, use commas to separate them.

The second form of for, using the "in" keyword, is useful with objects (covered later).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The for Loop, cont'd

```
1    for ( var i = 0; i < 10; i++ )
2    {
3      console.log( "for loop: " + i );
4    }
5
6    for ( var i = 10,j = 0; i > 0; i--, j++ )
7    {
8      console.log( "(" + i + "," +
9                                  j + ")" );
10   }
```


Here we show two examples of the "for" loop. The first, is a simple loop that counts from 0 to 9. The second illustrates using multiple "initial statements" and "executing statements". Its output looks like: (10,0)
 (9,1)
 (8,2)
 (7,3)
 (6,4)
 (5,5)
 (4,6)
 (3,7)
 (2,8)
 (1,9)

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to work with variables using JavaScript operators
- How to use conditional and flow-control statements to organize logic

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Objects, Arrays and Functions

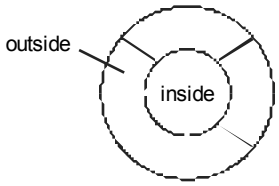
- Objects and Properties
- Arrays and Array Methods
- Defining and Calling Functions

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is an Object?

- In classic object orientation, an **object** is a set of state (data) and behavior (code)
- JavaScript provides an interesting and powerful way for programs to work with objects
- JavaScript does not have **classes** in the same sense as other languages like Java, but does support inheriting state and behavior via a **prototype** mechanism



Object Name	Sample State	Sample Behavior
lassie	weight	bark
account12	balance	debit
policy765	amount	payOut

Compared to Java, JavaScript is loosely typed and dynamic – you can add state and behavior to objects on the fly!

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Categories of Objects

- **Built-in** (native) objects such as arrays and functions
- **Host** objects such as those provided by a browser (*document*, *window* and so forth)
- **User-defined** objects created by programs

Any value in a JavaScript program that's not a number, string or boolean is an object.

It might seem odd that functions are actually objects, but that's how it works in JavaScript!

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What Are Properties?

- Properties consist of a name and value where the name is unique in the object
- Methods are really just a special case of a property - they are properties that associate a name with a set of statements
- Unlike in many languages, JavaScript lets you add/delete properties to objects dynamically

JavaScript doesn't support encapsulation using public/private keywords as do other OO languages, but you can define private properties/methods in the "constructor function" that we will cover later. For more details, see: http://javascript.crockford.com/private.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating Objects

- JavaScript provides three ways to create new objects:
  - Using **object literal** syntax
  - Using the **new** operator
  - Using the ECMAScript v5 Object.create() function

ECMAScript v5 implementations provide the Object.create() function that let you create objects by specifying a prototype. We will not cover it here.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Object Literal Syntax

- JavaScript provides a special syntax so that you can create user-defined objects

```
1    var nothing = {};
2    var person =  {
3      name: "Sue",
4      age:  32
5    };
6    var book = {
7       title: "101 JavaScript Tricks",
8       author: {
9          name: "Paul Westerberg",
10         email: "paul@westerberg.com"
11      }
12   };
13   alert("Person name: " + person.name);
14   alert("Book author email: " + book.author.email);
```

This syntax is the basis for the JSON (JavaScript Object Notation) format often used in RESTful applications.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using the new Operator

- You can use the **new** operator to create empty objects

```
1    var nothing = new Object();
2    var person = new Object();
3    person.name = "Sue";
4    person.age = 32;
5
6    var book = new Object();
7    book.title = "101 JavaScript Tricks";
8    book.author = new Object();
9    book.author.name = "Paul Westerberg";
10   book.author.email = "paul@westerberg.com";
11
12   alert("Person name: " + person.name);
13   alert("Book author email: " + book.author.email);
```

This syntax demonstrates that you can add properties to objects dynamically. In other words, if you do an assignment to a nonexistent property on an object, JavaScript creates and initializes it.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Objects Are Reference Types

- The value returned from creating an object is a **reference**, or pointer to the actual object
- You can define aliases to the object

```
1    var sue =
2    {
3      name: "Sue",
4      age:  32
5    };
6
7    var sam = sue;
8    sam.age = 44;
9
10   alert("Sue age: " + sue.age); // Sue age: 44
11   alert("Sam age: " + sam.age); // Sam age: 44
```

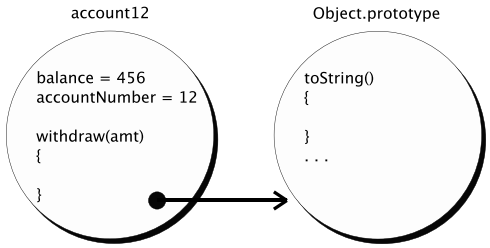
This is similar to languages like Java. In fact, JavaScript's "garbage collection" works similarly to Java - the JavaScript runtime gets rid of objects that no longer have any references.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Prototype Inheritance

- Most JavaScript objects have another object chained to them - the **prototype**
- So far, the prototype has been **Object.prototype**
- Objects inherit properties/methods from their prototype - Object.prototype defines several methods like **hasOwnProperty()**, **toString()** and so forth

```
1    var nothing = new Object();
2    alert("nothing.toString(): " + nothing.toString());
```



Object.prototype is predefined by JavaScript.

What this means is that when you access a property or call a method on an object, JavaScript looks to see if it's defined on the object itself. If not, then it looks through the prototype chain. If the property isn't defined anywhere in the chain, then it evaluates to the JavaScript keyword "undefined". Note that reading an undefined property is not an error, but calling an undefined method is (the error message will appear in the JavaScript console if you run the program in a browser).

The return from Object.prototype's toString() method isn't very useful, but this code demonstrates that we can call the method even though it's not explicitly define in the "sue" object. But if you care, the output is:

```
[object Object]
```

There are actually a few exceptional objects that don't have a prototype.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Working with Properties

- You can read or write properties using either the dot (.) or square bracket operators
- Using square brackets is more typing, but more powerful:
    - You can have property names that are JavaScript keywords (e.g. *for*) or contain spaces
    - You can use a string variable

```
1    var person = new Object();
2    person.name = "Sue";
3    var myvar = "age";
4    person[myvar] = 32;
5    person['new'] = true;
6    person["acct num"] = 1234;
7    alert(person.age);  // 32
```

We will see the square brackets again when we cover arrays. JavaScript arrays and objects are closely related.

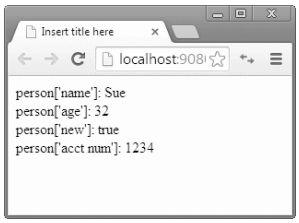
Note that ECMAScript v5 relaxes the restriction on using keywords for property names using the dot syntax.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The for-in Lop

- JavaScript provides a special looping syntax that lets you enumerate an object's properties

```
1    for (prop in person)
2    {
3      document.write("person['" + prop + "']: " +
4          person[prop] + "<br>");
5    }
```



This loop enumerates all properties defined by the object, and any properties from its prototype. Note that it's possible to exclude a property from enumeration by setting the property's "enumerable" attribute to false - properties inherited from Object.prototype are not enumerable, and thus are not reported by this loop.

But if the object contained methods, they WOULD be reported, since methods are a type of property.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is an Array?

- In JavaScript, an array is an ordered collection of values
- Arrays and objects are closely related - arrays are actually objects!
- If you use arrays with integer indexes, the indexes start at zero

0	1	2	3	4	5	6	7	8	9
0	2	4	6	8	10	12	14	16	18

Since JavaScript provides indexed arrays and since objects can act like hash tables, JavaScript doesn't need "collections" as found in other languages like Java.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating an Array

- You can create an array either using the square-bracket [] syntax or using the Array() constructor function

```
1    // empty array
2    var array1 = [];
3    // array with 10 integers
4    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
5    // array with 3 strings
6    var array3 = ["tom", "dick", "harry"];
7    // array with a mixture of types
8    var array4 = ["sue", 1, false];
9    // empty array
10   var array5 = new Array();
11   // array with space for 10 elements
12   var array6 = new Array(10);
```

It's usually easier to use the square-bracket syntax instead of the Array() constructor function

In any case, when you create an array, it's actually an object that chains to the Array.prototype object. This prototype has several useful properties and methods.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Array Indexes

- There are basically two ways to treat an array:
  - Using consecutive non-negative integer indexes
  - Using arbitrary keys (arrays are objects, and thus can have properties)
- Using integer indexes is the "normal" way, and provides the best performance

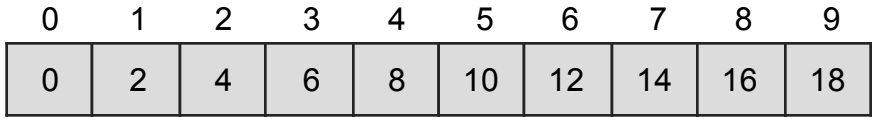
```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2    array2[5] = 42;
3
4    var array7 = [];
5    array7['phone'] = "555-1212";
6    array7[-42] = 654;
```

If you use only non-negative integer indexes, the JavaScript engine can use a sequential data structure and provide constant-time access to elements. If you use anything else, JavaScript treats the array as a normal object using a hash table, which might not be as fast.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Iterating Arrays

- Arrays provide a **length** property that you can use to iterate the array



0	1	2	3	4	5	6	7	8	9
0	2	4	6	8	10	12	14	16	18

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2
3    for(var i = 0; i < array2.length; i++)
4    {
5      document.write(array2[i] + " ");
6    }
7    // 0 2 4 6 8 10 12 14 16 18
```

The "length" property is actually provided by Array.prototype.

You can also use the "for-in" loop covered in the Objects chapter, but it's generally safer to use the loop shown here, since "for-in" iterates inherited properties - if some other code (say a library) added properties to Array.prototype, you'd iterate those, too.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Appending Elements

- You can append elements by assigning using the **length** as an index or by calling the **push()** method

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2
3    array2[array2.length] = 20;
4    array2.push(22);
5
6    for(var i = 0; i < array2.length; i++)
7    {
8      document.write(array2[i] + " ");
9    }
10   // 0 2 4 6 8 10 12 14 16 18 20 22
```

The push() method is inherited from Array.prototype. Note that there's also a pop() method that removes the element at the end of the array and returns it - you can use push() and pop() to treat an array as a LIFO (last-in, first-out) stack data structure.

Note that the push() method accepts multiple arguments:

```
array2.push(22, 24);
// 0 2 4 6 8 10 12 14 16 18 20 22 24
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Inserting Elements at the Beginning

- You can use the **unshift()** method to insert one or more elements at the beginning of an array, moving existing elements back

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2
3    array2.unshift(42);
4
5    for(var i = 0; i < array2.length; i++)
6    {
7      document.write(array2[i] + " ");
8    }
9    // 42 0 2 4 6 8 10 12 14 16 18
```

There's also a shift() method that removes the element at the beginning of the array and returns it - you can use shift() and unshift() to treat an array like a FIFO (first-in, first-out) data structure.

The unshift() method accepts multiple arguments:

```
array2.unshift(42, 44);
// 42 44 0 2 4 6 8 10 12 14 16 18
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Deleting Array Elements

- Since arrays are objects, and array elements are properties, you can delete them in the same fashion as you can delete "normal" object properties
- If you delete in the middle, the array becomes **sparse**

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2
3    delete array2[3];
4
5    for(var i = 0; i < array2.length; i++)
6    {
7      document.write(array2[i] + " ");
8    }
9    // 0 2 4 undefined 8 10 12 14 16 18
```

We will cover sparse arrays in a moment.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The splice() Method

- The **splice()** method is a powerful way to insert or remove elements from an array
- The method takes one to four arguments, and operates differently depending on how many you provide

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2    var array3 = array2.splice(3);
3
4    document.write("array2: ");
5    for(var i = 0; i < array2.length; i++) {
6      document.write(array2[i] + " ");
7    }
8    document.write("<br>array3: ");
9    for(var i = 0; i < array3.length; i++) {
10     document.write(array3[i] + " ");
11   }
12   // array2: 0 2 4
13   // array3: 6 8 10 12 14 16 18
```

If you supply a single index argument to splice(), it deletes the remaining elements from the original array and returns a new array with the deleted elements.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using splice() to Insert

- By providing splice with a "delete count" of zero and additional elements, you can insert them into the array, moving existing elements back

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2    var array3 = array2.splice(3, 0, 999, 888);
3
4    document.write("array2: ");
5    for(var i = 0; i < array2.length; i++)
6    {
7      document.write(array2[i] + " ");
8    }
9    document.write("<br>array3: ");
10   for(var i = 0; i < array3.length; i++)
11   {
12     document.write(array3[i] + " ");
13   }
14   // array2: 0 2 4 999 888 6 8 10 12 14 16 18
15   // array3:
```


---

splice() interprets the first argument as the index where to work, the second as a count to delete, and any remaining arguments as values to insert.

On the last page, we provided just one argument (no count), so that example deleted until the end of the array.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The map() Method

- The **map()** method creates a new array by invoking a **callback** method that you provide

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2
3    function squareIt(val)
4    {
5        return val*val;
6    }
7
8    var array3 = array2.map(squareIt);
9
10   for(var i = 0; i < array3.length; i++)
11   {
12       document.write(array3[i] + " ");
13   }
14   // 0 4 16 36 64 100 144 196 256 324
```

The map() method invokes the callback for each element in the original array and assembles the output array from whatever the callback returns.

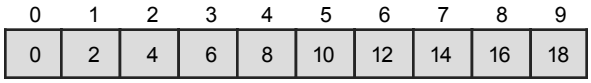
The callback method accepts up to three arguments: a value from the original array, the current index and a reference to the input array. The last two arguments are optional.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Other Useful Array Methods

- **join()** - converts all elements to a string and returns it
- **reverse()** - used with indexed arrays, it reverses the element order
- **sort()** - sorts the array alphabetically or via a user-defined criteria
- **concat()** - creates a new array with additional elements appended to the original
- **slice()** - returns a subset of an array:

```
1    var array2 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18];
2    var array3 = array2.slice(4, 6);
3    // array3:  8 10
```



The string type provides a split() method that works as the inverse of join() - it splits a string into an array, using a delimiter.

Note that ECMAScript v5 provides a few additional array methods.

The slice() method accepts two arguments: the starting and ending indexes of the slice to return. The last index is non-inclusive. slice() does not modify the source array, but does return the slice. If you give slice() a single argument, it returns an array from that index to the end of the source array. Note that you can use negative indexes to slice at the end of the array.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Function?

- A **function** is a chunk of code that you define once but can call as many times as needed
- We refer to a function that is part of an object as a **method**
- Functions can accept **parameters** and optionally return a value

```
1    function sum(a, b)
2    {
3       return a+b;
4    }
5
6    var result = sum(5, 8);
```

In other languages, functions are referred to as "subroutines", "procedures" or subprograms.

Functions can use a predefined variable named "arguments", that represents the parameters actually passed to the function. This object contains a "length" property containing the number of arguments.

```
function function7(a, b, c)
{
  // displays 1
   alert(arguments.length);
}
function7(12);
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Functions are Objects

- In JavaScript, functions are a type of **object**, and thus can be passed as parameters to another function or returned
- We can thus say that JavaScript functions are **first-class functions** as opposed to languages like Java (pre Java v8) in which functions (methods) can only be defined and invoked

Java version 8 does support first-class methods.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Functions

- JavaScript provides two syntaxes for defining functions:
    - Function declaration
    - Function definition expression
- The difference is that the **definition expression** syntax assigns the function object to a variable

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Function Declarations

- **Function declarations** let you define and name functions

```
1    // function declaration syntax
2    function function1(p1, p1)
3    {
4      alert(p1+p2);
5    }
6
7    function function2()
8    {
9      return "hi";
10   }
11
12   function1(2, 3);
13   var x = function2();
```

Behind the scenes, this syntax creates a variable with the function name and assigns the function object to the variable. Thus calling code that invokes the function is really using the variable.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Function Definition Expression

- **Function definition expressions** let you define a function, optionally name it and optionally invoke it immediately

```
1    // function definition expression
2    var function3 = function(p1) {return p1;}
3
4    // naming is useful for recursion
5    var f4 = function function4() {return function4};
6
7    function3(12);
8    var x = f4();    // references function4
9    // function4();  // not visible
```

The syntax where you use both a variable and a function name is useful for writing recursive functions - in the function's body, you can invoke the function itself. Note that the function name is only defined within the body - outside code must use the variable to invoke the method.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Function Definition Expression, cont'd

- You can use the function definition expression syntax to define and invoke **anonymous** functions on the fly

```
1    // define and invoke a function in one fell swoop
2    var f5 = function(){alert("hi");}();  // "hi"
3
4    // define and invoke an anonymous function
5    // note the extra set of parens
6    (function(p1){alert(p1);}(42));  // 42
7
8    // pass an anonymous function to another function
9    function function6(obj){alert(obj(72));}
10   function6(function(p1) {return p1;});  // 72
```


In the line that defines and invokes the anonymous function, the extra set of parenthesis is required to avoid a syntax error. This syntax is sometimes referred to as an immediately invoked function expression.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Optional Arguments

- If a caller fails to provide an argument, JavaScript passes **undefined**
- You can use that implement optional function arguments

```
1    function myFunction(a, b, c)
2    {
3      b = b || 'bDefault';
4      c = c || 'cDefault';
5
6      console.log(a);
7      console.log(b);
8      console.log(c);
9    }
10
11   myFunction('a');
12   myFunction('a', 'b');
13   myFunction('a', 'b', 'c');
```

In this function,we treat the first argument, a, as "required" and the other two as optional. Note that we define reasonable defaults for the optional arguments.

The || operator returns its first argument if that arg is truthy and the second arg otherwise. We can use this trick to conditionally initialize the optional arguments.

If the caller fails to provide the required argument, this function will print "undefined" for its value.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Global Functions vs Methods

- Functions that are part of an object are known as **methods**
- JavaScript provides methods with an implicit **this** reference that refers to the object on which the method is running

```
1    var sue = {
2      name: "Sue",
3      age:  32
4    };
5
6    sue.ageInDogYears = function() {
7       return this.age * 7;
8    }
9
10   alert("Age in dog years: " + sue.ageInDogYears());
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Scope of Variables Review

- JavaScript supports **global** and **local** variables
- Variables defined within a function (method) are **local** and can only be used by that method

```
1    var a = "My global variable";
2
3    function function1()
4    {
5      var b = "My local variable";
6
7      alert(a);  // access global
8    }
9
10   function function2()
11   {
12     alert(a);  // access global
13     alert(b);  // error - out of scope
14   }
```

Global variables are actually implemented as properties on the so-called "global object". In client-side JavaScript in a Web browser, the "window" object acts as the global object.

A function is a block of code with a name and optional arguments. To declare a function, use the "function" keyword, provide a proper identifier, open-and-close parenthesis and a pair of "curly braces".

Unlike languages such as Java and C, JavaScript doesn't have "block scope", in which variables defined within a block are local to that block.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining Functions within Functions

- JavaScript programs can nest functions within other functions - the nested functions can access variables defined in the enclosing function

```
1    //  a nested function can access
2    //  variables defined by enclosing function
3      function f1() {
4         var x = 12;
5
6         function f2() {
7            alert(x);
8         }
9
10        f2();
11     }
12
13     f1();  // 12
14     f2();  // undefined
```


Note that the nested function is not directly visible outside of its enclosing function.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Closures

- An enclosing function can return a nested function back to a caller, which can then invoke the nested function
- The nested function can still access local variables defined by the enclosing function, even if there's no call to the enclosing function - this is known as a **closure**

Closures are a fundamental part of functional programming, in which you write programs that consist of functions that don't modify any external state, and thus have no side effects.

The benefit of functional programming is that programs are easier to understand, maintain and debug, since there's no mutable "global" data. In addition to JavaScript, other languages that support functional programming include Lisp, Scheme, Haskell, Clojure and Scala.

There is a useful library that helps you implement functional programming in JavaScript at:

`http://osteele.com/sources/javascript/functional/`

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Closures, cont'd

```
1       function f1()
2       {
3          var x = 12;
4
5          function f2()
6          {
7             alert(x);
8          }
9
10         return f2;
11      }
12
13      // demonstrate a closure in action
14      var f2 = f1();
15      f2();             // 12
```


JavaScript internally implements closures by maintaining a "scope chain" – when a nested function references a variable, JavaScript looks first in the nested function itself, and then in the enclosing function.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About objects and their properties
- How to work with arrays and array methods
- Various ways to define and invoke functions

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JavaScript AJAX

- What is AJAX?
- Introduction to JSON
- What are Promises?
- JavaScript AJAX Libraries, including axios

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is AJAX?

- AJAX (Asychronous JavaScript and XML) uses a combination of existing technologies:
  - JavaScript
  - The XMLHttpRequest Object
  - HTML Document Object Model
- AJAX lets you create Web applications with a higher degree of interactivity



The notion of asynchronous request processing using the XMLHttpRequest object has been around for several years, but the term "AJAX" was coined by Jesse James Garrett of Adaptive Path.

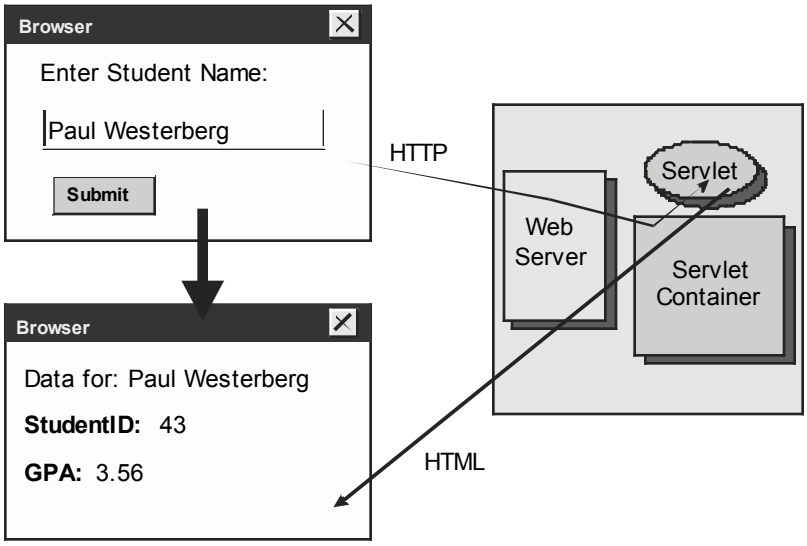
You can read his essay at www.adaptivepath.com/publications/essays/archives/000385.php.

Note that to use AJAX, you're not required to use XML.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Traditional Web Processing

- In the traditional model, users submit pages and Web applications generate complete pages
- This can be inefficient, especially if most of the page is unchanged from one request to the next



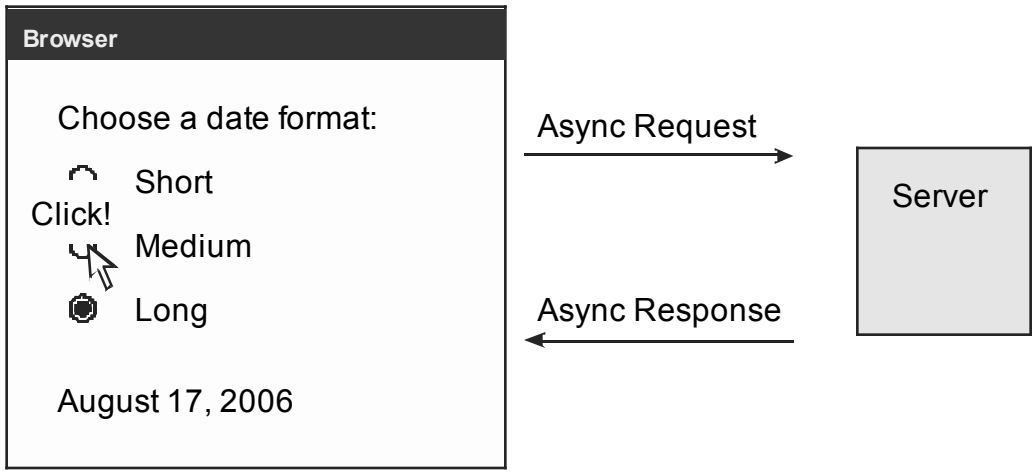
The Web started as a classic two-tier, client-server system in which the Web browser client is nothing more than a display device. All of the processing power and page-generation logic resides on the server.

While this model has been fantastically successful, it's not necessarily the correct fit for all types of applications.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AJAX Processing

- With AJAX, **any** user action can trigger a request to the server for more or different data
- Since the request is asynchronous, the user can keep working while the request is processed



In an AJAX Web application, JavaScript on the client side can send requests without requiring the user to press a Submit button. In this figure, we show an application that updates a date display whenever the user clicks a radio button. In response, the server sends back a differently formatted string containing the current date.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

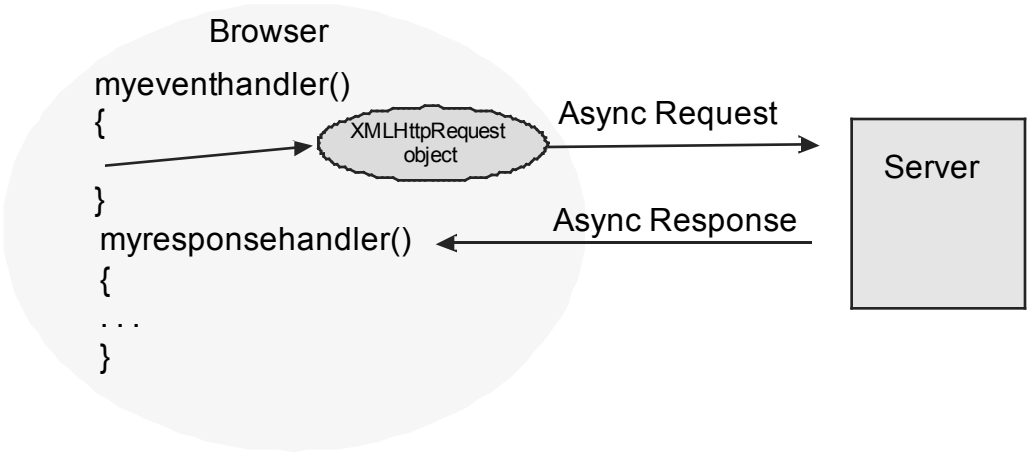
AJAX Applications

- Perhaps the first AJAX application was Microsoft's Outlook Web Access
- Here is a partial listing of others:
    - Flickr
    - Google Maps
    - Google Mail
    - Yahoo Mail

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The XMLHttpRequest Object

- The **XMLHttpRequest** object was first implemented by Microsoft Internet Explorer, but is now a part of other browsers such as Firefox and Opera
- This object defines an API for making remote requests, either synchronously or asynchronously



The XMLHttpRequest object is the foundation of AJAX.

This object lets a JavaScript event handler (e.g. from a radio-button click), send a request to the server and establish another handler to process the response. The response handler can use the DOM to modify the page content based on the returned data. This technique is what lets us write AJAX applications that perform requests without requiring a full page reload.

The XMLHttpRequest object has an interesting history. First implemented by Microsoft Internet Explorer, it has since been "ported" to other browsers and has become a standard part of modern browsers.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

XMLHttpRequest: Raw JavaScript

- Retrieving the XMLHttpRequest object reference is somewhat browser dependent

```
1    function getXMLHttpRequest()
2    {
3      var xRequest=null;
4      // handle non-Microsoft browsers
5      if (window.XMLHttpRequest)
6      {
7        xRequest=new XMLHttpRequest();
8      }
9      // handle Microsoft browsers
10     else if (typeof ActiveXObject != "undefined")
11     {
12       xRequest=new ActiveXObject("Microsoft.XMLHTTP");
13     }
14     return xRequest;
15   }
```


While the XMLHttpRequest object itself is mostly standard across browsers, the act of creating the object is different.

Here we show a simple JavaScript method that returns a reference to the object on different browsers.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sending a Request: Raw JavaScript

- The XMLHttpRequest object supports both HTTP **GET** and **POST** requests
- Here we show using a **GET** with parameters and an asychronous *callback handler*

```
1    var req;
2    function sendAsyncGetRequest(url)
3    {
4      req = getXMLHttpRequest();
5      if (req)
6      {
7        req.open("GET", url, true);
8        req.onreadystatechange = callback;
9        req.send(null);
10     }
11   }
```

```
var url =
  "MyServlet?myparm='abc'";
sendAsyncGetRequest(url);
```

This code listing shows defining "req" as a global JavaScript variable and then a method that sends an asynchronous "get" request to the server.

In line 4, we call the method shown on the last page that creates the XMLHttpRequest object. Assuming that works, we then open a request using "get", the passed URL and specify "true" to configure an asynchronous request. Then in line 8, we register our asynchronous response handler (code shown on the next page). In line 9, we send the request with no additional parameters.

The sample call shown in the figure invokes a server-side program with URL "MyServlet", passing a request parameter named "myparm" with value set to 'abc'.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sending a Request: Raw JavaScript, cont'd

```
1    function callback()
2    {
3      if (req.readyState == 4)
4      {
5        if (req.status == 200)
6        {
7          var text = req.responseText;
8          . . . // update the page
9        }
10       else
11       {
12         alert("Problem: " + req.statusText);
13       }
14     }
15   }
```


Here we show the JavaScript callback function that we registered to handle the response. It turns out that there are several possible ways that the browser invokes this function for a single request. Here we ignore all of the notifications except for the one that indicates that the request is complete (status = 200). At that point, we could then use the DOM to modify the page to contain the data received from the server, which is referenced by the "text" variable in line 7..

Note that this function uses the global "req" variable we showed on the last page.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Updating the Document: Raw JavaScript

- To update the HTML page, the **callback handler** can use the DOM to replace or update a **text node** containing the response text

```
1    function callback()
2    {
3      . . .
4      var text = req.responseText;
5      setParagraph(text);
6      . . .
7    }
8
9    function setParagraph(text)
10   {
11     var myPara=document.getElementById('c');
12     myPara.firstChild.nodeValue=text;
13   }
```

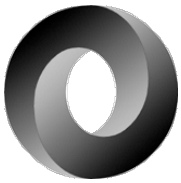
After the callback handler retrieves the response, it can use the DOM to change the page to display the returned data. Here we show the callback handler calling a JavaScript function that locates a paragraph by ID and then changes its text-node child to contain the text returned from the server.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is JSON?

- JSON is the **JavaScript Object Notation** and was originally syntax in JavaScript for initializing objects, but has now become a standard data format rivaling XML

```
{
  "id"  : 17,
  "name": "Sue Smith",
  "gpa":3.45
}
```



You can read more about JSON at http://www.json.org.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Basic JSON Syntax

- JSON has a simple, easy to read and understand syntax, both for humans and computers

```
1    -- sample object
2    {
3      "gpa"      : 3.33
4      "name"     : "Sue Smith",
5      "address"  :
6      {
7         "street":      "123 Elm",
8         "city"  :      "Marion"
9      }
10   }
11
12   -- sample array
13   [ "red", "green", "blue" ]
```

Objects can contain sub-objects and arrays. Arrays can contain objects and sub-arrays. And so forth.

Note that you must use double quotes rather than single quotes for values.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using JSON in Raw JavaScript

- JavaScript applications can use JSON to create and initialize objects and arrays

```
1    var rect1 =
2    {
3       upperLeft: { x: 2, y: 2 },
4       lowerRight: { x: 4, y: 5 }
5    };
6
7    var str = JSON.stringify(rect1);
8
9    var rect2 = JSON.parse(str);
```


JavaScript also provides the eval() method that parses strings containing JSON, but this is potentially unsafe since it actually executes as JavaScript code. Also note that the JSON.stringify and JSON.parse methods require an ECMAScript version 5 implementation.

JSON is actually a subset of the JavaScript "object-literal notation" - in other words, every legal JSON string is also legal JavaScript OLN.

Popular JavaScript libraries such as jQuery also often include JSON support.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript AJAX Libraries

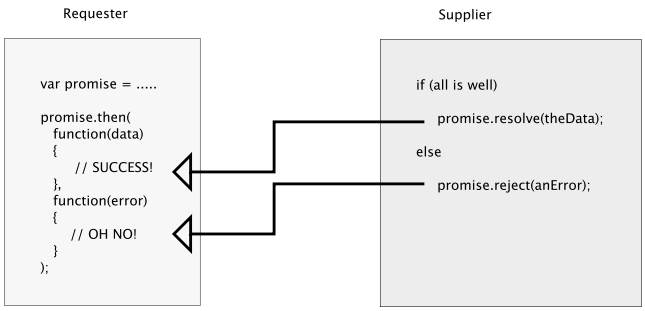
- Issuing AJAX requests using plain old JavaScript is tedious, repetitive and error-prone
- Using a JavaScript library such as jQuery, fetch or axios makes AJAX much easier
- Many modern libraries use **promises** to abstract the asynchronous nature of AJAX

jQuery is a very large library that does much more than just AJAX, while fetch and axios are single-purpose AJAX libraries.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Promise?

- **Promises** is an emerging API for writing asynchronous code
- Many popular JavaScript libraries implement this API
- A **promise** is an object that represents an asynchronous action
- The "supplier" calls **resolve()** or **reject()** to fulfill or reject the promise
- The "requester" calls **then()** to register one or two callbacks



You can read more about the concept of promises at https://www.promisejs.org

Note that you can pass two functions to then(): the first acts as the "resolve" function, the second as "reject".

One interesting thing is that you can "chain" promises to handle an event in sequence. The then() method itself returns a promise so that you can call then() on it, registering another callback. When the promise is fulfilled, the first callback will run, followed by the second.

```
var promise = ...;
var another = promise.then(function(){/* callback #1 */});
var yetanother = another.then(function(){/* callback #2 */});
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to axios

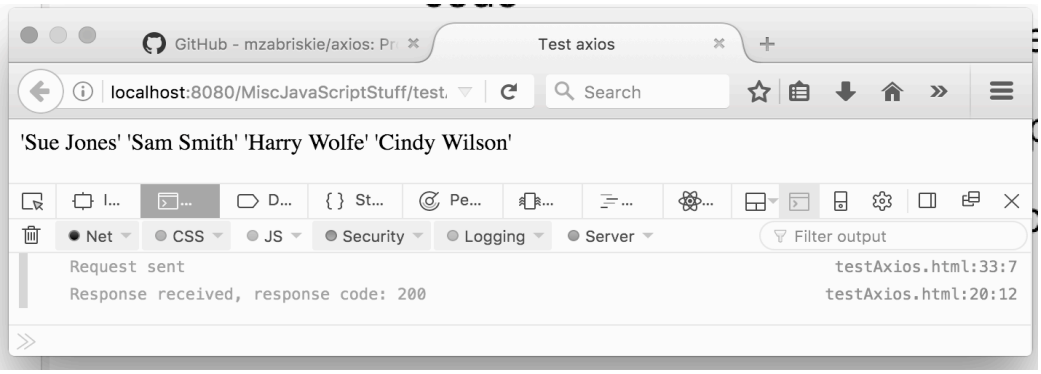
- axios is a JavaScript, promised-based HTTP client that works from within a browser or from within a server-side node.js application
- You can read more at https://github.com/mzabriskie/axios

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using axios to GET JSON

students.json

testAxios.html



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    [
2        {"id":12, "name":"Sue Jones", "gpa":3.45},
3        {"id":45, "name":"Sam Smith", "gpa":2.76},
4        {"id":73, "name":"Harry Wolfe", "gpa":3.98},
5        {"id":9,  "name":"Cindy Wilson", "gpa":2.87}
6    ]
```

6 - 17A : students.json

```
1    <!DOCTYPE html>
2    <html>
3      <head>
4        <meta charset="UTF-8">
5        <title>Test axios</title>
6          <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
7      </head>
8      <body>
9
10       <div id="target"></div>
11
12       <script>
13
14         var target = document.getElementById("target");
15
16         var promise = axios.get('students.json');
17
18         promise.then(function (response)
19         {
20              console.log("Response received, response code: "
21                  + response.status);
22
23              var students = "";
24              for(var i=0; i < response.data.length; i++)
25              {
26                      students += "'" + response.data[i].name + "' ";
27              }
28              target.innerHTML = students;
29         })
30         .catch(function (error)
31         {
32             target.innerHTML = error;
33         });
34
35         console.log("Request sent");
36
37       </script>
38     </body>
39   </html>
```

Chapter Summary

In this chapter, you learned:

- What AJAX is and how to issue requests from plain old JavaScript
- About JSON as a general-purpose data format
- How to issue AJAX requests using the axios library

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to React

- React Architecture
- Building React Components in Plain JavaScript
- Building React Components in JSX

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Single Page Web Applications

- When the web was first invented, browsers completed loaded a new page when you navigated to a new page
- With the advent of JavaScript, the Document Object Model (DOM) and AJAX, developers can now create **single-page applications** that only update a part of the page
- There are several JavaScript frameworks for SPA including React and Angular

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The React Framework

- React is a JavaScript library for developing user interfaces
- You can build UIs using either "pure" JavaScript or React's XML-based JSX syntax

We will cover using React for web applications, but you can instead use React Native to build mobile applications for iOS and Android.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What React Isn't

- Unlike other frameworks such as Angular, React focuses solely on building the user interface
- React can integrate with other libraries for functionality such as AJAX

Other frameworks, in particular Angular, try to be a complete SPA solution, providing APIs for managing state, executing AJAX requests, routing and so forth. React is a smaller, lighter-weight library that concentrates just on the user interface (generating HTML in web applications). You might find it convenient to think of React as implementing the V (view) in Model-View-Controller.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

History of React

- React was created at Facebook, and was first deployed at Facebook in 2011
- Facebook converted the project to open source in 2013 and it's now available on GitHub
- React Native was introduced and open-sourced in 2015

You can read more about the history of React on Wikipedia:

`https://en.wikipedia.org/wiki/React_(JavaScript_library)`

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

React Architecture

- React emphasizes developing reusable, components that compose your application's user interface
- React provides a virtual DOM so it can optimize changes to the actual page DOM
- React apps are typically written using ES6 (EcmaScript 2015) and use a **transpiler** such as Babel to convert to ES5, which has widespread browser support

Babel is a separate open-source project and can be used outside of React. That being said, Babel's aware of React - not only can it convert ES6 to E65, but also knows how to transpile React's JSX syntax. You can read more about Babel at https://babeljs.io.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

React Components

- Though optional, React developers favor creating reusable components to compose user interfaces
- React provides serveral techniques for creating components, including:
    - React.createClass()
    - ES6 class extending React.Component
    - Stateless function objects using the React plain JavaScript API or JSX
- In this chapter, we will focus on using stateless functions

If your component needs state, then you can use either createClass() or write an ES6 class. The React documentation hints that createClass() may be deprecated at some point, so it's a good idea to use ES6 classes as much as possible.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Flux?

- **Flux** is a pattern for managing state and flow through an application
- Using Flux is optional for React apps, but the Flux architecture provides a scalable, reliable, one-way flow data flow that works well for applications that use dynamic data
- We will not cover Flux in detail in this chapter

The official "Getting Started" with Flux is at: https://github.com/facebook/flux/tree/master/examples/flux-concepts. Here's another good introduction: http://blog.andrewray.me/flux-for-stupid-people.

The popular Redux library is "inspired by the Flux pattern". Read more at http://redux.js.org.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

React Hello, World: Plain JavaScript

- Here we create a simple React element and attach it to a **div** in the host page's DOM

plainJS-try1-hello.html

The two React scripts define a couple of global JavaScript objects: React and ReactDOM. The React global object lets you define components, while ReactDOM lets you bind components to a web-page DOM.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html>
2    <html lang="en">
3      <head>
4        <title>React Hello, World</title>
5        <meta charset="utf-8">
6      </head>
7      <body>
8
9        <!-- Target container -->
10       <div id="react-container"></div>
11
12       <script src="https://unpkg.com/react@15.4.2/dist/react.js">
13       </script>
14       <script src="https://unpkg.com/react-dom@15.4.2/dist/react-dom.js">
15       </script>
16
17       <script>
18         var heading =
19           React.createElement("h1", null, "Hello, World");
20
21         ReactDOM.render(heading,
22             document.getElementById('react-container'));
23       </script>
24
25     </body>
26   </html>
```

The createElement API

- The **createElement()** API creates an element in React's virtual DOM
- The new element can reference a standard HTML element such as <h1> or reference a React component

```
var heading = React.createElement(

  "h1",             // type: here an HTML element-name string
  null,             // props: passed to element
  "Hello, World");  // children: here text for the &lt;h1&gt;
```


The API docs for createElement() are at:

https://facebook.github.io/react/docs/react-api.html#createelement

Once created, the element is immutable, but you can clone and modify using cloneElement().

If the first argument is a string, createElement() interprets it as an HTML element name.

The optional "props" argument lets you pass data that is stored with the element in the virtual DOM.

The optional "children" argument(s) let you specify child elements - for example, you could create a <ul> element and its <li> children all at once. This argument accepts multiple values.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Rendering the DOM Tree

- You can call **ReactDOM.render()** whenever you want to "flush" React's virtual DOM to the actual page DOM
- React optimizes the updates so that only changes to the virtual DOM are applied to the "real" DOM

```
ReactDOM.render(
  heading,                  // element: to render into the DOM
  document.getElementById(  // container: where to render
       'react-container'))
```


The API docs for render() are at:

https://facebook.github.io/react/docs/react-dom.html#render

After an initial render, on subsequent renders(), React updates the page DOM instead of completely rendering it. So if you called ReactDOM.render() twice, back-to-back, without changing anything in the virtual DOM, the second call to render() would be a NO-OP.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JSX

- Creating components with plain JavaScript is not difficult, but is verbose, so React provides an alternative syntax: JSX
- JSX's syntax is roughly based on XML
- You can embed snippets of JavaScript within JSX using curly braces as delimiters
- Since JSX is not native JavaScript, you need to use a **transpiler** such as Babel to convert to native JavaScript

---

The official documentation for JSX is at:

https://facebook.github.io/react/docs/jsx-in-depth.html

Since JSX is XML-based, you need to be sure to close all tags - if the tag is empty, you can use the syntax like <someTag />.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

React Hello, World in JSX

- Here we create a simple React element using JSX and pass it to ReactDOM.render()
- Note how the page includes a script reference to Babel and specifies the JSX script's type as **text/babel** - this is simple for learning and development

JSX-try1-hello.html

JSX-try1-compiled.js

Note that the pages that use JSX cannot be directly displayed by browsers since browsers don't recognize JSX. In other words, if you use JSX, you MUST use a transpiler such as Babel to convert the JSX into plain JavaScript.

Note also how the first argument to render() is NOT a string (no quotes) - Babel will transpile that content into JavaScript that creates a React element. So essentially, the transpiler calls createElement() for you.

In production, you probably would use a build script to run Babel instead of including the script and type as shown here.

The second listing on this page shows how Babel transpiled the StudentList component into raw JavaScript. You can see the transpiled output by using the tool at:

https://babeljs.io/repl

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html>
2    <html lang="en">
3      <head>
4        <meta charset="utf-8">
5        <title>React Hello, World (JSX)</title>
6        <script src="https://unpkg.com/react@15.4.2/dist/react.js">
7        </script>
8        <script src="https://unpkg.com/react-dom@15.4.2/dist/react-dom.js">
9        </script>
10       <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.29/browser.
11       </script>
12     </head>
13     <body>
14
15       <div id="react-container"></div>
16
17       <script type="text/babel">
18
19         ReactDOM.render(
20               <h1>Hello, world!</h1>,
21               document.getElementById('react-container')
22         );
23
24       </script>
25
26     </body>
27   </html>
```

```
1    'use strict';
2
3    ReactDOM.render(React.createElement(
4                    'h1',
5                    null,
6                    'Hello, world!'
7    ), document.getElementById('react-container'));
```

7 - 13B : JSX-try1-compiled.js

Defining Components in JSX

- Here we define a function using ES6 **arrow syntax** - the function defines the React component
- We then call ReactDOM.render(), specifying the component using JSX syntax

JSX-try2-component2.html

JSX-try2-compiled.js

You should name your components with an upper-case character so that React can differentiate between JSX components and standard HTML elements.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>React: Iterating</title>
  </head>
  <body>

    <div id="react-container"></div>

    <script src="https://unpkg.com/react@15.4.2/dist/react.js">
    </script>
    <script src="https://unpkg.com/react-dom@15.4.2/dist/react-dom.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.29/browser.
    </script>

    <script type="text/babel">

      /* define a React component */
       const StudentList = () =>
         <ul>
           <li>Sue Jones</li>
           <li>Sam Smith</li>
           <li>Harry Wolfe</li>
           <li>Cindy Wilson</li>
        </ul>;

       ReactDOM.render(
          <StudentList />,
             document.getElementById('react-container'));

  </script>

  </body>
</html>
```

```
1    "use strict";
2
3    var StudentList = function StudentList() {
4       return React.createElement(
5          "ul",
6          null,
7          React.createElement(
8             "li",
9             null,
10            "Sue Jones"
11         ),
12         React.createElement(
13            "li",
14            null,
15            "Sam Smith"
16         ),
17         React.createElement(
18            "li",
19            null,
20            "Harry Wolfe"
21         ),
22         React.createElement(
23            "li",
24            null,
25            "Cindy Wilson"
26         )
27      );
28   };
```

Passing Data to Components and Iterating

- You can pass immutable data to React components using an object (typically named **props**)
- Here we then use Array.map() and an ES6 arrow function to create multiple list-item child elements to build the unordered list

JSX-try3-iterateComponent.html

JSX-try3-compiled.js

To avoid warnings in the browser console, we specify a unique key for each list item - that makes React happier!

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html>
2    <html lang="en">
3      <head>
4        <meta charset="utf-8">
5        <title>React: Iterating</title>
6      </head>
7      <body>
8
9        <div id="react-container"></div>
10
11       <script src="https://unpkg.com/react@15.4.2/dist/react.js">
12       </script>
13       <script src="https://unpkg.com/react-dom@15.4.2/dist/react-dom.js">
14       </script>
15       <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.29/browser.
16       </script>
17
18       <script type="text/babel">
19
20          var data =
21             [
22               {id:12, name:"Sue Jones", gpa:3.45},
23               {id:45, name:"Sam Smith", gpa:2.76},
24               {id:73, name:"Harry Wolfe", gpa:3.98},
25               {id:9,  name:"Cindy Wilson", gpa:2.87}
26             ];
27
28          /* define a React component */
29          const StudentList = (props) =>
30           <ul>
31             {
32                props.students.map((student) =>
33                  <li key={student.id}>{student.name}</li>
34             )}
35           </ul>;
36
37          ReactDOM.render(
38             <StudentList students={data} />,
39                document.getElementById('react-container'));
40
41       </script>
42
43   </body>
44   </html>
```

```
1    "use strict";
2
3    var data =
4      [
5        {id:12, name:"Sue Jones", gpa:3.45},
6        {id:45, name:"Sam Smith", gpa:2.76},
7        {id:73, name:"Harry Wolfe", gpa:3.98},
8        {id:9,  name:"Cindy Wilson", gpa:2.87}
9      ];
10
11   /* define a React component */
12   var StudentList = function StudentList(props) {
13       return React.createElement(
14           "ul",
15           null,
16           props.students.map(function (student) {
17               return React.createElement(
18                   "li",
19                   { key: student.id },
20                   student.name
21               );
22           })
23       );
24   };
25
26   ReactDOM.render(
27     React.createElement(StudentList,
28       { students: data }),
29       document.getElementById('react-container'));
```

React and AJAX

- While React only provides support for the user-interface piece of a web application, there are several popular JavaScript AJAX libraries including **jQuery**, **fetch** and **axios**
- You can use one of those libraries to issue the AJAX call, then update and render the virtual DOM when the response arrives
- Remember that React optimizes virtual DOM updates to the real page DOM!

The last bullet point means that when the AJAX response returns, you can just call ReactDOM.render() again - React will only update the parts of the page DOM that changed due to the response.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

React AJAX Example

- Here, instead of hard-coding the student list in JavaScript, we use axios to fetch it using AJAX

JSX-try4-iterateAjaxComponent.html

students.json

Admittedly, we are still hard-coding the data in a JSON file, but the programming would be the same if it was a real, live RESTful source.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>React: Iterating Component using axios AJAX</title>
  </head>
  <body>

    <h1>Student List</h1>

    <div id="react-container"></div>

    <script src="https://unpkg.com/react@15.4.2/dist/react.js">
    </script>
    <script src="https://unpkg.com/react-dom@15.4.2/dist/react-dom.js">
    </script>
    <script src="https://unpkg.com/axios/dist/axios.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.29/browser.
    </script>

    <script type="text/babel">

      /* define initial content until AJAX response */
      ReactDOM.render(
        React.createElement("div", null, "Placeholder for AJAX data"),
            document.getElementById('react-container'));

     /* define a React component */
     const StudentList = (props) =>
       <table>
         <tbody>
         {
             props.students.map((student) =>
               <tr key={student.id}>
                 <td>{student.id}</td>
                 <td>{student.name}</td>
                 <td>{student.gpa}</td>
               </tr>
         )}
         </tbody>
       </table>;

      /* issue AJAX request */
      axios.get('students.json')
```

```
46          .then(function (response)
47          {
48             var data = response.data;
49
50             /* re-render when AJAX response arrives. React will
51                only update changes to the DOM */
52             ReactDOM.render(
53                <StudentList students={data} />,
54                document.getElementById('react-container'));
55          })
56          .catch(function (error)
57          {
58             console.log(error);
59          });
60
61       </script>
62
63     </body>
64   </html>
```

```
1    [
2        {"id":12, "name":"Sue Jones", "gpa":3.45},
3        {"id":45, "name":"Sam Smith", "gpa":2.76},
4        {"id":73, "name":"Harry Wolfe", "gpa":3.98},
5        {"id":9,  "name":"Cindy Wilson", "gpa":2.87}
6    ]
```

Developing Production React Apps

- Most React developers use a build script of some sort to transpile, modularize and minify their applications
- React provides a Node Package Manager (npm) script named **create-react-app** that gives you a good start

For more info, see https://facebook.github.io/react/docs/installation.html.

The create-react-app script also configures a live web server - you can start it and then modify your pages - the server will automatically refresh once you save the sourc e files.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How React implements the view of a web application by providing an optimized virtual DOM
- How to create simple React components using plain JavaScript and JSX
- How to combine React and axios for AJAX to access RESTful data

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to the Spring Framework

- What is the Spring Framework?
- Spring Architecture
- Spring Hello, World

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is the Spring Framework?

- The **Spring Framework** is an open-source, Java and JEE framework that aims to make developing applications easier
- Spring applications are based on **plain old Java objects** (POJO) also known as JavaBeans
- Objects in a Spring application should not depend on Spring and should be easy to test



The homepage for the Spring Framework is at:

http://www.springsource.org/

Note that Spring is also available for Microsoft's .NET.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

History of Spring

- The Spring Framework was created in 2003 by Rod Johnson based on material from his book *Expert One-on-One J2EE Design and Development*
- Spring 2 was released in 2006 and added basic support for Java 5 and configuration extensions
- Spring 2.5 adds support for Java 5 annotations
- Spring 3 contains support for RESTful Web services and other new features including **Java-based configuration**
- Spring 4 adds support for Java 8 and removes many old deprecated classes and methods

Rod Johnson publicized his new framework with a post to theserverside.com:
http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Fundamentals

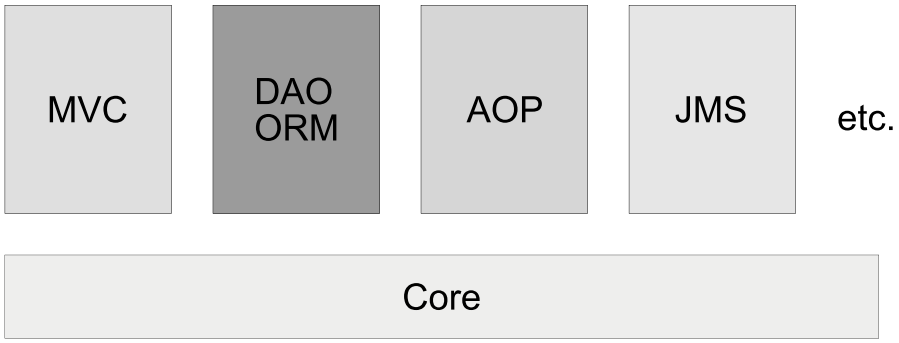
- Use of Spring beans (POJOs) instead of requiring classes to implement framework interfaces
- Loose coupling between objects
- **Dependency injection** (DI) to give objects resources instead of requiring lookups
- **Aspect-oriented programming** (AOP) to free objects from performing system functions such as logging and transactions
- Spring doesn't re-invent the wheel, but can integrate with other frameworks (e.g. Java Persistence)

Dependency injection is also known as "Inversion of Control" (IoC).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Modules

- The Spring Framework itself is modular so you can use only the pieces needed by your application
- This makes the Spring runtime lightweight



This notion of modular design allows Spring developers to use only the pieces they need for their application.

Note that there are additional modules not shown here.

Many developers use a build tool such as Maven or Gradle to help manage the Spring JARs and their dependencies.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing Spring Applications

- To write Spring programs, you:
  - Write Spring **bean** classes
  - Configure the **container** so it can manage and initialize the beans
  - Bootstrap the container, retrieve bean references and write application code

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Spring Bean?

- Unlike in earlier frameworks, Spring **beans** don't need to implement any non-domain interfaces
- These POJOs are like JavaBeans, though they don't need to implement Serializable and such

```
1    @Component
2    public class Student
3    {
4       private int id;
5       . . .
6
7       public int getId() {return id;}
8       public void setId(int i) {id = i;}
9       . . .
10   }
```

One of the "knocks" against frameworks like EJB2 and Struts is that they require you to write classes that implement framework interfaces and then require the framework's runtime behavior to exist. This coupling between the objects and the framework makes it difficult to re-use business objects and also makes testing outside of the runtime difficult or impossible.

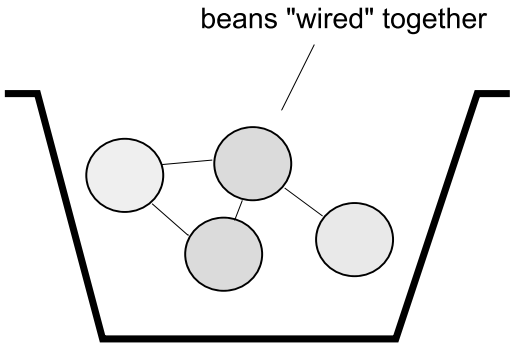
Spring avoids that by allowing the use of POJOs that have no dependency on Spring itself, except perhaps for the @Component annotation that names the bean.

JavaBeans is a native Java specification that defines the notion of Java components. Spring POJOs typically follow a relaxed version of the full JavaBean specification.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Spring Container

- To manage the lifecycle of the beans, Spring provides a lightweight **container**
- Spring apps create the container, then retrieve bean references from the container
- The container can **wire**, or hook up beans to each other via *dependency injection*



A Spring container

A container is a set of runtime that performs useful tasks. In Spring, one such useful task is ensure that beans have their dependencies satisfied. This is quite different from traditional JEE, where objects use JNDI to find their own dependencies. Here, the container injects the dependencies so the object doesn't need to do any look ups.

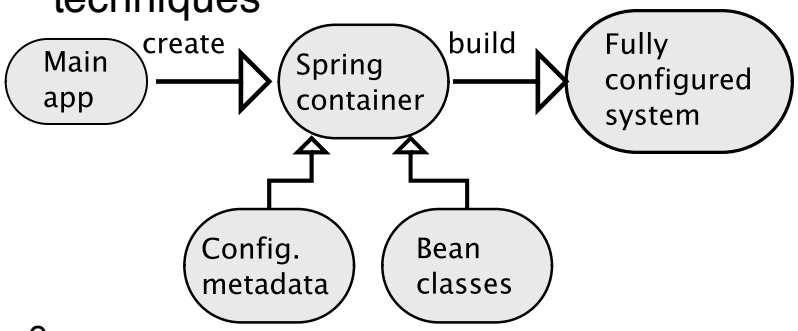
Beans can have references to other beans or to external resources. The container can connect a bean to its referenced objects as part of creating the set of managed objects.

This process, known as "wiring" is typically configured using annotations or in an XML file that the container reads when it starts.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring the Container

- Spring provides three* ways to configure Spring beans:
  - **Annotations** - Markup bean classes themselves
  - **Java configuration** - Write one or more classes that perform configuration
  - **XML** - Write one or more XML configuration files
- Spring applications can use any mixture of these techniques



* - Spring 4 added the ability to use the Groovy language both for writing and configuring Spring beans.

XML-based configuration was the first approach and was the only way to configure in early versions of Spring. It has the advantage that you can keep your beans configuration-free and consolidate all of the configuration, but does require you to learn and use XML.

Annotation-based configuration reduces the amount of XML required.

Java-based configuration is relatively new. It has the advantage that you can keep your beans configuration-free and perform configuration in the same language that you are using to write the beans (Java).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Programming To Interfaces

- To decrease coupling between objects, Spring developers often start by writing an **interface** for some of their beans
- This eases maintenance, and makes beans easier to test

```
1    public interface MyService
2    {
3        public void myMethod();
4    }
5
6    public class MyClass implements MyService
7    {
8       public void myMethod() {...};
9    }
10
11   public class OtherClass
12   {
13      private MyService theService;
14   . . .
15   }
```


Writing both an interface and an implementation is not required by Spring, but it's considered a best practice.

Note that dependent classes maintain a reference that's typed to the interface, not the implementation.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dependent Objects Without Spring

- Most real-world objects use other objects; we refer to this as a **dependency**
- In non-Spring environments, objects typically satisfy their own dependencies

```
1    public class OtherClass
2    {
3       // MyService is an interface
4       private MyService theService;
5
6       public OtherClass()
7       {
8          // MyClass implements MyService
9          theService = new MyClass();
10         . . .
11      }
12   }
```

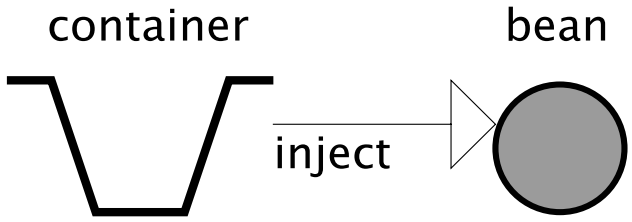
Here we show the OtherClass constructor explicitly and manually satisfying a dependency to a MyService object.

While this is straightforward and simple, it presents a few problems, most notably that OtherClass must know the name of the MyService implementation class. That makes it difficult to subsitute some other implementation, perhaps for testing.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Dependency Injection

- **Dependency injection** is a technique in which the container instantiates and provides references for beans
- To inject a bean reference, the container directly accesses a field or calls a bean's "set" method or constructor
- DI lessens coupling between beans and decouples beans from services such as JNDI



Dependency injection is a form of Inversion of Control</i> where the container supplies a bean with its dependencies.

The term "Dependency injection" was coined by Martin Fowler.

Spring developers typically write annotations and/or an XML file to configure their beans and how the Spring container should inject dependencies. Another option is to use the "Java configuration" technique added in Spring 3.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring with Annotations

- Spring lets you configure injection using **annotations**

```
1    package sample;
2
3    @Component("myService")
4    public class MyClass implements MyService
5    {
6       public void myMethod() {...};
7    }
```

```
1    package sample;
2
3    @Component("other")
4    public class OtherClass
5    {
6       @Resource(name="myService")
7       private MyService theService;
8       public void setTheService(MyService svc) {...}
9    . . .
10   }
```

Here we show injecting the MyClass reference into the OtherClass bean.

Note how in the Java code, OtherClass doesn't know about the implementation class, only the interface.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring with Java Config

- Java configuration is the newest approach and lets you configure using the same language you use to write the beans themselves
- To use Java config, create one or more Java classes annotated with **@Configuration** containing bean-creation methods annotated with **@Bean**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring with Java Config, cont'd

```
1    @Configuration
2    public class MyConfig
3    {
4      @Bean
5      public MyService myService()
6      {
7         return new MyClass();
8      }
9
10     @Bean
11     public OtherClass otherClass()
12     {
13       OtherClass other = new OtherClass();
14       other.setTheService(myService());
15       return other;
16     }
17   }
```


Here we provide two bean-creation methods to create the same beans we configured with annotations and XML-based config.

Note in the otherClass() method, we perform dependency injection by calling the myService() method and passing its return to the OtherClass bean's setTheService() method.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring with XML

- XML configuration was the original configuration approach in Spring

```
1    <!-- sample.xml -->
2
3    <?xml version="1.0" encoding="UTF-8"?>
4    <beans ...>
5
6      <bean id="myService" class="sample.MyClass"/>
7
8      <bean id="other" class="sample.OtherClass">
9        <property name="theService" ref="myService" />
10     </bean>
11
12   </beans>
```


In older versions of Spring, there was no support for annotations, and most developers used XML. Even in modern Spring, some developers still prefer the XML approach since it lets you perform all of the configuration in a central location.

Note how the property name, "theService" maps to the "setTheService()" method in the bean. When using XML injection, Spring can call such "set" methods to perform the injection. With annotations, Spring doesn't need the "set" methods.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Injecting Simple Values

- You can use annotations, XML or Java config to intialize properties to values

```
1    // using annotations
2    public class MyClass
3    {
4       @Value("1234")
5       private int myInteger;
6       public void setMyInteger(int myInt) {...}
7    }
```

```
1    <!-- using XML -->
2
3    <bean id="myBean" class="sample.MyClass">
4      <property name="myInteger" value="1234" />
5    </bean>
```

Whether you use annotations or XML, Spring must convert the supplied textual value to the appropriate Java type. If it can't convert, the container throws org.springframework.beans.factory.BeanCreationException.

If you plan to use the XML approach, the "set" method is required, but not if you use annotations.

The @Value annotation is from the org.springframework.beans.factory.annotation package.

Injecting values with Java config is trivial - just call the "setter" method.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Annotation-Based Apps

- To enable Spring to scan your beans for annotations, you need to provide at least a minimal XML file

```
1    <!-- sample.xml -->
2
3    <?xml version="1.0" encoding="UTF-8"?>
4    <beans ...>
5
6      <context:component-scan base-package="sample"/>
7
8    </beans>
```

This simple XML file configures Spring so it can process the @Component and @Resource annotations shown previously. Note how Spring needs to know which packages to scan for the annotations; here the MyClass and OtherClass classes are in the "sample" package.

We don't need the component-scan element if we use XML or Java configuration to configure the beans.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating the Container (Annotation or XML)

- Spring applications with a *main* method can create a Spring container and then retrieve beans from it
- Spring provides several techniques for creating a container - here we retrieve the configuration file from the file system and use it to create the container

```
1    AbstractApplicationContext ctx =
2     new FileSystemXmlApplicationContext("sample.xml");
3
4    OtherClass oc =
5       (OtherClass)ctx.getBean("other");
6
7    . . .
8
9    ctx.close();
```

The AbstractApplicationContext interface specifies the basic functionality of a Spring container. Note that once we've created the container, we can use it to retrieve managed bean references.

The imports for this program might look like:

```
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating the Container (Java Configuration)

- If you are using Java configuration, you do not need any XML configuration - just reference the configuration class

```
1    AnnotationConfigApplicationContext ctx =
2      new AnnotationConfigApplicationContext(
3         MyConfig.class);
4
5    OtherClass oc =
6       (OtherClass)ctx.getBean("other");
7
8    . . .
9
10   ctx.close();
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Steps for a Simple Spring Application

- As desired, create interfaces for bean classes
- Implement the beans, using dependency injection as required
- Write an XML configuration file (if necessary)
- Write a class with a *main* method that creates a container and retrieves bean instances
- Invoke the beans' methods to perform the application function

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Hello World From Spring

HelloService.java

HelloImpl.java

Greeter.java

HelloApp.java

hello.xml

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package hello;
2
3    public interface HelloService
4    {
5        public void sayHello();
6    }
```

8 - 22A : HelloService.java

```java
package hello;

import org.springframework.stereotype.Component;

@Component("helloService")
public class HelloImpl implements HelloService
{
    public void sayHello()
    {
        System.out.println("Hello, from Spring");
    }
}
```

8 - 22B : HelloImpl.java

```java
package hello;

import javax.annotation.Resource;

import org.springframework.stereotype.Component;

@Component("greeter")
public class Greeter
{
    @Resource(name="helloService")
    private HelloService helloService;

    public void sendGreetings()
    {
        helloService.sayHello();
    }
}
```

```
1    package hello;
2
3    import org.springframework.context.support.AbstractApplicationContext;
4    import org.springframework.context.support.FileSystemXmlApplicationContext;
5
6    public class HelloApp
7    {
8        public static void main(String[] args) throws Exception
9        {
10           AbstractApplicationContext factory =
11              new FileSystemXmlApplicationContext("hello.xml");
12
13           Greeter greeter = (Greeter) factory
14                   .getBean("greeter");
15           greeter.sendGreetings();
16
17           ctx.close();
18       }
19   }
```

8 - 22D : HelloApp.java

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <beans xmlns="http://www.springframework.org/schema/beans"
3        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4        xmlns:context="http://www.springframework.org/schema/context"
5        xsi:schemaLocation="
6     http://www.springframework.org/schema/beans
7     http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
8     http://www.springframework.org/schema/context
9     http://www.springframework.org/schema/context/spring-context-3.1.xsd">
10
11       <context:component-scan base-package="hello"/>
12
13
14   </beans>
```

8 - 22E : hello.xml

Chapter Summary

In this chapter, you learned

- About the basic architecture of the Spring Framework
- How to use Dependency Injection to inject resources into a bean

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Spring Boot

- What is Spring Boot?
- Starter Projects
- Autoconfiguration
- Building and Running

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Spring Boot?

- Spring Boot is an add-on project to Spring that provides:
    - A generator for starter projects that defines common configurations (e.g SpringMVC with JPA)
    - Easier configuration with no XML
    - Ability to run Web applications without deploying to an application server
    - An optional **command-line** interface (CLI)

The home page of Spring Boot is at:

`http://projects.spring.io/spring-boot`

Spring Boot provides several other features, including runtime instrumentation to check the health of an application (the Actuator).

The CLI lets you generate generate starter projects and build them. The Actuator provides a Web-based console or SSH access to the application so you examine many aspects of the app, including getting a list of beans, HTTP trace and so forth.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Building Spring Boot Projects

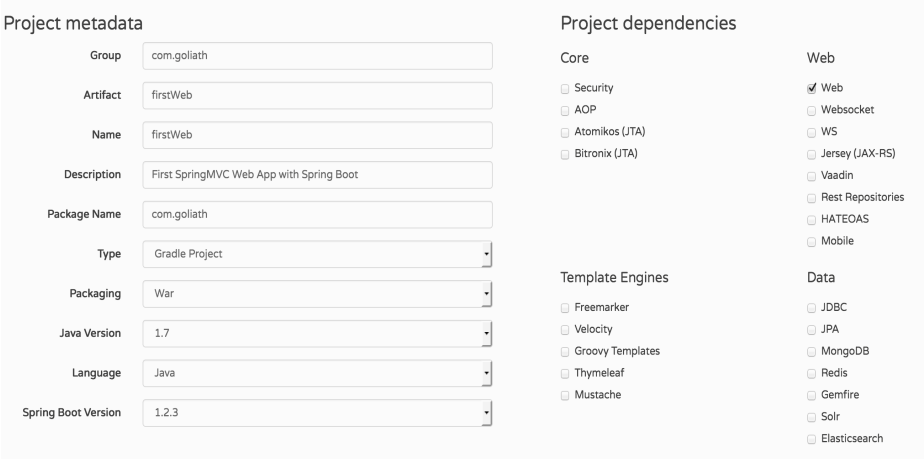
- Spring Boot supports both Maven and Gradle and can generate starter build scripts for both tools
- You can read more about Maven at: https://maven.apache.org/
- You can read more about Gradle at: https://gradle.org/

In this chapter, we'll use Gradle.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Generating a Starter Project

- Spring Boot provides the **Initializr** to generate projects with commonly used features
- You can access the Initializr on the Web at: http://start.spring.io or via the Spring Boot Command-Line Interface



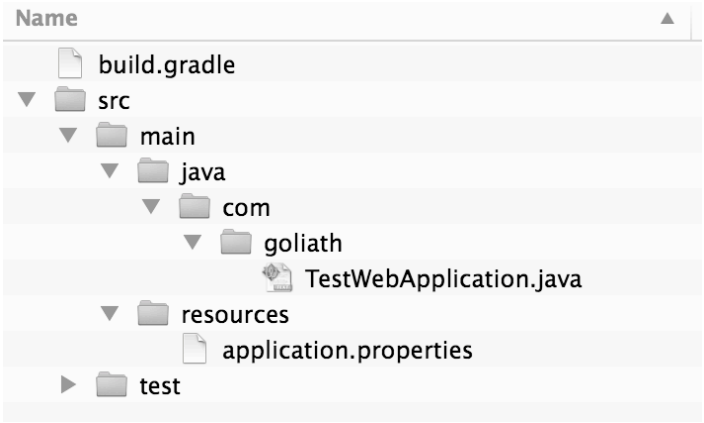
The Intitializr generates a ZIP file that you can download and unzip.

You can also run the Initializr from within the Spring Tool Suite Eclipse environment.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Gradle Project Structure

- Once you unzip the project generated by the Initializr, you will have a directory structure:



The Initializr generates a basic harness for unit testing. We will not cover it in this overview.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Generated Build Script

- The Initializr generated a Gradle build script that specifies project metadata, **plugins**, **repositories** and **dependencies**

```
1    ...
2    apply plugin: 'java'
3    apply plugin: 'eclipse'
4    apply plugin: 'idea'
5    apply plugin: 'spring-boot'
6    apply plugin: 'io.spring.dependency-management'
7
8    repositories {
9        mavenCentral()
10   }
11
12   dependencies {
13       compile(
14         "org.springframework.boot:spring-boot-starter-web")
15   }
```

Plugins add "tasks" that you can execute. For example, the java plugin knows how to compile the Java code in your project.

Repositories are sources of dependencies, i.e. libraries. You can use Maven, Ivy or locally defined repositories.

Dependencies are libraries that your code needs. Note that you can specify when the dependency should be satisfied, i.e. at compile time. Gradle fetches the dependencies from the repositories. Note that Gradle supports "transitive dependencies" - if a specified dependency itself requires other code, Gradle will that that too.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Generating an Eclipse Project

- The project generated by Initialzr does not contain support for an IDE out of the box, but the build script contains **eclipse** and **idea** plugins
- For Eclipse, run **gradle eclipse**, then import the project into your workspace
- You can then create the Java classes, Web pages and so forth that comprise your application



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The 'main' Program

- The Initializr generated a **main()** method class so you can run the application, even if it's a Web application
- The **@SpringBootApplication** annotation configures the class to use Spring Boot's autoconfiguration facility so we don't need any XML configuration

```
1    @SpringBootApplication
2    public class FirstWebApplication {
3
4        public static void main(String[] args) {
5            SpringApplication.run(
6              FirstWebApplication.class, args);
7        }
8    }
```

You shouldn't need to modify the class under normal circumstances.

According to the Spring Boot docs, the @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration and @ComponentScan with their default attributes.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Boot Autoconfiguration

- Without Spring Boot, applications need to explicitly configure beans such as data sources, Spring MVC view mappings and so forth
- Spring Boot automatically configures default beans based on what it finds on the application's CLASSPATH
- You can override the defaults if needed, but if the defaults are OK, you can run an application with little or no Spring configuration

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Running the Project

- The Spring Boot plugin provides the **bootRun** task to run Web applications on an embedded Tomcat instance
- This is an easy and fast way to run your application without needing to load it onto an application server



It's also possible to build a standard WAR file for later deployment. See http://docs.spring.io/spring-boot/docs/current/reference/html/howto-traditional-deployment.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- What Spring Boot is all about
- How to configure, write and run a Spring Boot Web application

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Annotations

- Enabling Annotations
- Annotations for Dependency Injection

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Spring Beans

- Spring provides several techniques so that developers can provide metadata for beans, including:
    - XML
    - Annotations
    - Java Configuration
- XML configuration was the original approach, but this chapter focuses on annotations

Spring version 4 added Groovy configuration as well.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Annotations

- **Annotations** were added to Java in JDK5 and let you provide **metadata** in your source code
- You can use annotations to decorate classes, fields or methods
- Java defines several built-in annotations and frameworks like Spring can define their own "custom" annotations
- The Java compiler ignores custom annotations, so they need to be processed by the framework

```
1    @MyCustomAnnotation
2    public class Test
3    {
4      @SuppressWarnings("unused")
5      private int x;
6
7       @Override
8       public String toString() {...}
9    }
```


JDK 5 introduced a few standard, or built-in annotations:

@Override, @Deprecated, @SuppressWarnings,@Retention,@Target

The "parameters" to annotations are referred to as "elements". Annotations can accept no elements, a single element or multiple elements.

Technically, an annotation is similar to an interface, in that it defines a new type and must be fully qualified or imported when used.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Annotations and Spring

- Annotations were first added to Spring in version 2.0 and expanded in version 2.5
- Spring defines its own annotations and also uses annotations from other Java specifications (e.g. EJB3 and AOP)

```
1    import org.springframework.stereotype.Component;
2    import javax.annotation.Resource;
3
4    @Component("theStudent")
5    public class Student
6    {
7      @Resource(name="theAdvisor")
8      private Advisor theAdvisor;
9      public void setTheAdvisor(Advisor theAdvisor)
10       {...}
11   }
```

This code snippet shows using the @Resource annotation, which Spring borrows from JEE, and the @Component annotation, which is Spring-specific.

We will cover what these particular annotations do in a moment.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Enabling Annotations

- In Spring, annotations are processed by BeanPostProcessors, so you must configure the post processor(s) in the Spring configuration
- Spring lets you enable ALL annotations or selectively enable only the ones you are using

```
<!-- option 1: enable individual BeanPostProcessors -->
<bean class=
  "org.springframework.context.annotation.CommonAnnotationBeanPostProcessor"/>
<bean class=
  "org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"

<!-- option 2: enable ALL BeanPostProcessors -->
<context:annotation-config/>

<!-- option 3: enable ALL BeanPostProcessors AND @Component scanning -->
<context:component-scan base-package="mypackage" />
```


A BeanPostProcessor is a Spring component that runs as part of the container's startup sequence. The BeanPostProcessors mentioned here scan your beans looking for annotations and then taking appropriate action.

Configuring all of the annotation processors adds overhead if your Spring program is only using a few types of annotations, but is certainly is easier.

Programs that use @Component have no options here - they must use the third approach. But programs that primarily use XML for configuration and only use annotations besides @Component can decrease container startup time by selectively enabling on the BeanPostProcessors they need.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Value Annotation

- This is a Spring-specific annotation that lets you inject a simple value

```
1    public class Student
2    {
3    . . .
4
5      @Value("3.33")
6      public void setGpa(double gpa)
7      {. . .}
8    }
```


Spring also provides the Spring Expression Language (SpEL) that lets you perform sophisticated injection using @Value. For example, this expression injects a random number:

```
#{ T(java.lang.Math).random() * 100.0 }
```

The "T" in the expression means Type and lets you assign a Java type to a part of the expression, in this case the java.lang.Math class.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Annotations for Dependency Injection

- As we've already seen, a common use of annotations in Spring is to perform **dependency injection**
- You use @Component to configure beans into the container and then @Resource, @Autowire or @Inject to inject references

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Component Annotation

- This is a Spring-specific annotation that obviates the need to define beans in the XML configuration file
- You must write the **context:component-scan** element in the configuration file

```
1    package university.faculty;
2    import org.springframework.stereotype.Component;
3
4    @Component("advisor1")
5    public class Advisor {. . . .}
```

```
1    <beans ...>
2
3       <!-- <context:annotation-config/> -->
4       <context:component-scan base-package="university"/>
5
6    </beans>
```


The context:component-scan XML element is a superset of context:annotation-config that additionally scans for @Component, @Resource and other annotations. If you have multiple "base packages", you can separate them with commas. Note that you can also fine-tune which packages it scans using context:include-filter and context:exclude-filter child elements.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Resource Annotation: By Name

- This is a standard JEE annotation that Spring uses to perform dependency injection
- @Resource injects either by **name** or by **type**, depending on how you use it

```
1    package university;
2
3    import javax.annotation.Resource;
4    import org.springframework.stereotype.Component;
5
6    @Component(name="student1")
7    public class Student
8    {
9      // inject bean reference by name
10     @Resource(name="advisor1")
11     public void setTheAdvisor(Advisor theAdvisor)
12     {. . .}
13   }
```

Here are assuming that there is an Advisor bean defined named "advisor1". That bean could either be defined in the XML, or using the @Component annotation (covered on a later page).

The placement for this annotation is either on the field to be injected or on its "set" method.

The @Resource annotation is from JSR-250 and is also used by JEE application servers that implement JEE Version 5 or later.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Resource Annotation: By Type

- You can use @Resource to inject by **type**
- You can use it to inject bean references or references to standard Spring types such as ApplicationContext

```
1    package university;
2
3    import javax.annotation.Resource;
4    import org.springframework.stereotype.Component;
5
6    @Component(name="student1")
7    public class Student
8    {
9      // inject reference by type
10     @Resource
11     private ApplicationContext ctx;
12     . . .
13   }
```

This usage of @Resource accepts no elements (parameters). The container determines what to look up by examining the type of the property.

The "standard" Spring objects include the BeanFactory interface, the ApplicationContext interface, the ResourceLoader interface, the ApplicationEventPublisher interface and the MessageSource interface.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Autowiring with Annotations

- @Autowired is a Spring-specific annotation that injects by **type**

```
1    public class MyBean
2    {
3      @Autowired
4      private ServiceBean1 sob1;
5
6      @Autowired
7      public void doSomething(ServiceBean sob2)
8      {
9          . . .
10     }
11
12     @Autowired
13     public MyBean(ServiceBean sob3) { . . . }
14   }
```

You can use the @Autowired annotation on fields, "set" methods, constructors, and other methods that accept bean references as parameters. If the wiring fails, the container will throw NoSuchBeanDefinitionException.

Note that @Autowired wires by type, at least by default. You can use the @Qualifier annotation in conjunction with @Autowired to have finer control over how it works.

@Autowired will fail if there's no candidate bean to wire, but you can specify @Autowired(required=false) to make it optional.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

@Resource vs @Autowired

- There is some overlap between the functionality of @Resource and @Autowired - both can inject by type
- A good rule of thumb is to use @Resource only for injecting by name and to use @Autowired to inject by type

Here's a note from the Spring documentation:

Tip: If you intend to express annotation-driven injection by name, do not primarily use @Autowired, even if is technically capable of referring to a bean name through @Qualifier values. Instead, use the JSR-250 @Resource annotation, which is semantically defined to identify a specific target component by its unique name, with the declared type being irrelevant for the matching process.

You can read the docs at:

http://docs.spring.io/spring/docs/3.0.0.M4/spring-framework-reference/html/ch03s09.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Qualifier Annotation

- You can use **@Qualifier** with autowiring when there are multiple candidate beans

```
1    @Component("firstImpl")
2    public class FirstImpl implements MyService
3    {
4    . . .
5    }
6    @Component("secondImpl")
7    public class SecondImpl implements MyService
8    {
9    . . .
10   }
11   public class Client
12   {
13     @Autowired @Qualifier("secondImpl")
14     private MyService theService;
15    . . .
16   }
```

This essentially is the same as autowiring by name.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Other Annotations

- Spring defines many more annotations, including:

Annotation	Notes
@Scope	Configures bean scope
@PostConstruct	For bean lifecycle
@PreDestroy	For bean lifecycle
@Controller	For Spring MVC @Component stereotype
@RequestMapping	For Spring MVC
@RequestParameter	For Spring MVC
@Service	@Component stereotype for REST
@Respository	@Compoment stereotype for DAOs
@Transactional	Configure transactions
@Pointcut	For Spring AOP
@Before	For Spring AOP
@RunWith	For testing

The "stereotype" annotations are special cases of @Compoment for components that reside in the tiers of a typical JEE application. @Controller is for the Web tier, @Service for the business-logic tier and @Repository is for the data tier. You use the stereotypes instead of @Component mainly for documentation, though @Repository does provide additional function (it translates database exceptions into Spring database exceptions).

Note that modern versions of JEE support the Common Dependency Injection (CDI) specification, which defines alternatives to many of these Spring-specific annotations.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and JSR-330

- The **Dependency Injection for Java** (JSR-330) specification defines a standard set of annotations for DI in Java
- JSR-330 defines @Named and @Inject annotatations similar to Spring's @Component and @Autowired
- Spring applications that need to be JEE standards compliant can use the JSR-330 annotations if they include the **javax-inject.jar** in the CLASSPATH

JSR-330's design was heavily influenced by Spring and the Google Guice framework.

The Contexts and Dependency Injection (JSR-299) specification extends JSR-330 and is a standard part of JEE 6 (and later) designed to help you integrate tiers, e.g. Web and domain logic (EJB)

CDI provides additional annoations such as @RequestScoped and @ConversationScoped that are useful in Web applications.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to inject collection properties
- About autowiring
- How to use annotations in Spring 2.x or later as an alternative way to provide configuration metadata

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Java Configuration

- Why Use Java Configuration?
- Writing a Configuration Class
- Using Dependency Injection

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Spring Beans

- Spring provides several techniques so that developers can provide metadata for beans, including:
    - XML
    - Annotations
    - Java Configuration
- XML configuration is the traditional approach

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Java-Based Configuration

- String in Spring 3, you can provide configuration metadata by writing Java code instead of XML, giving you fine-grained control over how the container creates objects
- Spring provides the **@Configuration** and **@Bean** annotations as well as the **AnnotationConfigApplicationContext** application context type

This approach lets you avoid writing ANY XML at all.

Other Java-based configuration annotations include @Import (to import configuration from another class) and @Scope to specify the scope.

Note that in versions of Spring prior to 3.2, Java-based configuration requires the CGLIB JAR to be in the program's CLASSPATH.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Java-Based Configuration Example

```
1    @Configuration
2    public class HelloWorldConfig
3    {
4       @Bean
5       public HelloWorld helloWorld()
6       {
7          return new HelloWorld();
8       }
9    }
```

```
1    AnnotationConfigApplicationContext ctx =
2      new AnnotationConfigApplicationContext(
3         HelloWorldConfig.class);
4
5    HelloWorld helloWorld =
6       ctx.getBean(HelloWorld.class);
7
8    . . .
```


This example shows configuration and using a HelloBean object (HelloBean class is not shown).

Note how the main program uses the AnnotationConfigApplicationContext and retrieves the bean by specifying its class.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Use Java Configuration?

- Java configuration has some of the same advantages as XML configuration:
    - Centralized configuration
    - Cleaner, pure-POJO beans (no annotations)
- Main advantage of Java configuration over XML is compiler-time checking for typos and type-safety

One big issue with XML configuration is that you don't discover problems until runtime.

A potential downside to Java configuration is that Java itself is somewhat verbose. A potential solution is to use the Groovy language for configuration instead of Java. Groovy is a simplified syntax language based on Java. For more info on using Groovy for Spring configuration, see:

```
https://spring.io/blog/2014/03/03/groovy-bean-configuration-in-spring-framework-4
http://hantsy.blogspot.com/2013/12/spring-4-groovy-dsl-bean-definition.html
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Bean Annotation

- You use the **@Bean** annotation on methods that produce Spring beans
- The container calls such methods, then adds the returned reference to its internal bean list
- By default, the new bean's name is taken from the bean-creation method's name
- This annotation has optional elements to let you specify the bean's name, scope and initialize and destroy methods

```
1    @Bean
2    public Student myStudent()
3    {
4      Student s = new Student();
5
6      return s;
7    }
```

For the example here to work, the Student class would need to have a public, zero-argument constructor.

We will cover scope and init and destroy methods later in the class.

Note that is possible to assign beans names and then retrieve them by name:

```
@Bean(name="student01")
public Student createStudent() {....}

. . .

Student s = (Student)ctx.getBean("student01");
```

If you don't assign a name, Spring uses the bean-creation method's name.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Performing Setter Injection

- In the bean-creation method, you can easily call **set()** methods to inject dependencies

```
1    @Bean
2    public Advisor myAdvisor()
3    {
4       Advisor a = new Advisor();
5       a.setAdvisorID(12);
6       a.setName("Sue Smith");
7       a.setSalary(40000);
8
9       return a;
10   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Referencing Other Beans

- If the bean-creation methods are in the same configuration class, you can call them directly to obtain references to other beans

```
1    @Configuration
2    public class MyConfig
3    {
4       @Bean
5       public Student myStudent()
6       {
7           Student s = new Student();
8           s.setAdvisor(myAdvisor());
9           return s;
10      }
11
12      @Bean
13      public Advisor myAdvisor() {....}
14   }
```


We will see how to use multiple configuration classes later in this chapter.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Performing Constructor Injection

- In the bean-creation method, you can provide dependencies via **constructor injection** if the bean class supports it

```
1    @Bean
2    public Student myStudent()
3    {
4       Student s = new Student(myAdvisor());
5       s.setName("Harry Wolfe");
6       s.setGpa(3.33);
7       s.setStudentID(45);
8
9       return s;
10   }
```

For the above code to work, the Student class would need a one-argument constructor that accepts an Advisor reference.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Autowiring in a Configuration Class

- Instead of directly calling other bean-creation methods, configuration classes can autowire related beans by type or name

```
1    @Configuration
2    public class StudentConfiguration
3    {
4      @Autowired private Advisor a;  // autowire by type
5
6      @Bean
7      public Student myStudent()
8      {
9        Student s = new Student(a);
10
11       return s;
12     }
13   }
```

This is especially useful if you write multiple configuration classes and the related bean is defined in a different configuration class.

Note that you could instead autowire by name:

```
@Resource(name="myAdvisor")
private Advisor a;
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Autowiring Bean-Creation Methods

- The bean-creation methods themselves can take advantage of dependency injection by defining arguments that Spring will inject

```
1    @Configuration
2    public class StudentConfig
3    {
4       @Bean
5       public Student myStudent(Advisor a)
6       {
7           Student s = new Student(a);
8           return s;
9       }
10   }
```


In this case, Spring will find a reference to an Advisor and pass it into the myStudent() method. Using this technique, you don't need to define any fields for autowiring.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Multiple @Configuration Classes

- The **@Configuration** annotation identifies a class as a providing beans
- You can write multiple Configuration classes to modularize the bean definitions

```
1    @Configuration
2    public class StudentConfig
3    {
4       @Bean public Student myStudent() {.....}
5    }
```

```
1    @Configuration
2    public class AdvisorConfig
3    {
4       @Bean public Advisor myAdvisor() {.....}
5    }
```

Be sure to never put any application code in Configuration beans!

@Configuration is actually an @Component stereotype.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Referencing Multiple Configuration Classes

- Applications can use **AnnotationConfigApplicationContext** or **AnnotationConfigWebApplicationContext** to reference configuration class(es)

```
1    AnnotationConfigApplicationContext ctx =
2     new AnnotationConfigApplicationContext(
3        StudentConfig.class,
4        AdvisorConfig.class);
5
6    Student s = (Student)ctx.getBean(Student.class);
7    Advisor a = (Advisor)ctx.getBean(Advisor.class);
```


Note that instead of providing the configuration classes when you create the context, you can call its "register()" method to provide the classes. You should then call refresh() on the context to ensure it's in a stable state.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Importing a Configuration

- You can use the **@Import** annotation to reference other Configurations
- Then, the "main" program only needs to specify the "root" Configuration

```
1    @Configuration
2    @Import(AdvisorConfig.class)
3    public class StudentConfig
4    {
5    . . .
6    }
```

```
1    AnnotationConfigApplicationContext ctx =
2      new AnnotationConfigApplicationContext(
3        StudentConfig.class);
4
5    Student s = (Student)ctx.getBean(Student.class);
6    Advisor a = (Advisor)ctx.getBean(Advisor.class);
```

In the bottom listing, we can use Advisors even though we didn't specify the AdvisorConfig class when we created the context.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Importing Other Configuration Types

- You can use the **@ImportResource** annotation to reference configurations defined in XML or Groovy
- You can use the **@ComponentScan** annotation to enable the same functionality as in XML's **context: component-scan** element
- These techniques lets you mix configuration approaches and migrate from XML and/or annotation-based configuration to Java configuration

```
1    @Configuration
2    @ImportResource({"classpath:spring.xml"})
3    @ComponentScan(basePackages="mypackage")
4    public class MyConfig
5    {
6    . . .
7    }
```

@ImportResource examines the file extension - if it ends in .groovy, then Spring uses a GroovyBeanDefinitionReader. If it's XML, then Spring uses XmlBeanDefinitionReader.

The string(s) you provide to this annotation can contain prefixes such as "file:" to read from the file system, "classpath:" to read from the CLASSPATH and so forth. See the Spring docs for Resources for details:

http://docs.spring.io/autorepo/docs/spring/3.2.x/spring-framework-reference/html/resources.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example

Advisor.java

Professor.java

Student.java

AdvisorConfig.java

StudentConfig.java

TestUniversity.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package university;
2
3    import java.util.ArrayList;
4    import java.util.Collection;
5
6    public class Advisor
7    {
8        private int advisorID;
9        private String name;
10       private double salary;
11       private Professor theProf;
12
13       private Collection<Student> students = new ArrayList<Student>();
14
15       public Professor getTheProf()
16       {
17           return theProf;
18       }
19
20       public void setTheProf(Professor theProf)
21       {
22           this.theProf = theProf;
23       }
24
25       public int getAdvisorID()
26       {
27           return advisorID;
28       }
29
30       public void setAdvisorID(int advisorID)
31       {
32           this.advisorID = advisorID;
33       }
34
35       public String getName()
36       {
37           return name;
38       }
39
40       public void setName(String name)
41       {
42           this.name = name;
43       }
44
45       public double getSalary()
```

```
46      {
47          return salary;
48      }
49
50      public void setSalary(double salary)
51      {
52          this.salary = salary;
53      }
54
55      public Collection<Student> getStudents()
56      {
57          return students;
58      }
59
60      public void setStudents(Collection<Student> students)
61      {
62          this.students = students;
63      }
64
65  }
```

```
1    package university;
2
3    public class Student
4    {
5        private int studentID;
6        private String name;
7        private double gpa;
8
9        private Advisor advisor;
10
11       public Student() {}
12
13       public Student(Advisor advisor)
14       {
15           this.advisor = advisor;
16       }
17
18       public int getStudentID()
19       {
20           return studentID;
21       }
22
23       public void setStudentID(int studentID)
24       {
25           this.studentID = studentID;
26       }
27
28       public String getName()
29       {
30           return name;
31       }
32
33       public void setName(String name)
34       {
35           this.name = name;
36       }
37
38       public double getGpa()
39       {
40           return gpa;
41       }
42
43       public void setGpa(double gpa)
44       {
45           this.gpa = gpa;
```

```
46        }
47
48        public Advisor getAdvisor()
49        {
50            return advisor;
51        }
52
53        public void init()
54        {
55            System.out.println("In init method");
56        }
57
58    }
```

```java
package university.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import university.Advisor;
import university.Professor;

@Configuration
public class AdvisorConfig
{
    @Bean
    public Advisor myAdvisor()
    {
        Advisor a = new Advisor();
        a.setAdvisorID(12);
        a.setName("Sue Smith");
        a.setSalary(40000);

        a.setTheProf(theProfessor());

        return a;
    }

    @Bean
    public Professor theProfessor()
    {
        Professor p = new Professor();
        p.setName("I.M. Mean");
        p.setProfessorID(42);

        return p;
    }
}
```

```
1    package university.config;
2
3    import javax.annotation.Resource;
4
5    import org.springframework.context.annotation.Bean;
6    import org.springframework.context.annotation.Configuration;
7    import org.springframework.context.annotation.Import;
8
9    import university.Advisor;
10   import university.Student;
11
12   @Configuration
13   @Import(AdvisorConfig.class)
14   public class StudentConfig
15   {
16     @Autowired
17     private Advisor a;
18
19   //    @Resource(name="myAdvisor")
20   //    private Advisor a;
21
22       @Bean(name="student01")
23   //  public Student myStudent(Advisor a)
24       public Student myStudent()
25       {
26           Student s = new Student(a);
27           s.setName("Harry Wolfe");
28           s.setGpa(3.33);
29           s.setStudentID(45);
30
31           return s;
32       }
33   }
```

```
1    package university.test;
2
3    import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4    import org.springframework.context.support.AbstractApplicationContext;
5
6    import university.Student;
7    import university.config.AdvisorConfig;
8    import university.config.StudentConfig;
9
10   public class TestUniversity
11   {
12
13       public static void main(String[] args)
14       {
15           AbstractApplicationContext ctx = new AnnotationConfigApplicationContext(
16                   StudentConfig.class, AdvisorConfig.class);
17
18           // Student s = (Student)ctx.getBean(Student.class);
19           Student s = (Student) ctx.getBean("student01");
20           System.out.println("Student name: " + s.getName());
21           System.out.println("Advisor name: " + s.getAdvisor().getName());
22           System.out.println("Professor name: " +
23              s.getAdvisor().getTheProf().getName());
24
25           ctx.close();
26       }
27
28   }
```

```java
package university;

import java.util.ArrayList;

public class Professor
{
    private int professorID;
    private String name;
    private ArrayList<Advisor> advisors;

    public int getProfessorID()
    {
        return professorID;
    }

    public void setProfessorID(int professorID)
    {
        this.professorID = professorID;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public ArrayList<Advisor> getAdvisors()
    {
        return advisors;
    }

    public void setAdvisors(ArrayList<Advisor> advisors)
    {
        this.advisors = advisors;
    }
}
```

Using the Groovy DSL for Configuration

- Starting in Spring 4, you can use the Groovy language to define Spring beans and/or to provide configuration metadata
- Steps for using Groovy:
    - Add Groovy dependency to Gradle or Maven build script
    - Optionally use Groovy to create Spring bean classes
    - Optionally use Spring's Groovy DSL to configure beans written in Java or Groovy

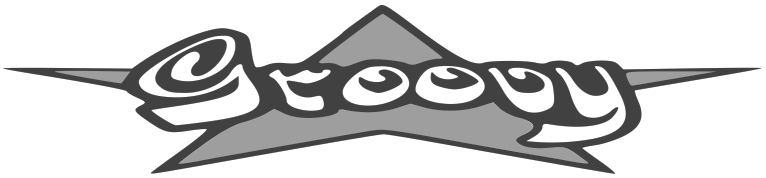
Spring 4 adapted the Domain Specific Language (DSL) from the Grails project so you can configure beans using Groovy. Using the DSL is nice because its syntax is simpler than Java, but still gives the expressive power of standard Java configuration.

We will not cover how to create Spring bean classes in Groovy in this chapter, since this chapter is all about configuration.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Groovy?

- Groovy is a dynamic, typing optional language with syntax similar to Java that runs on the JVM
- Some differences from Java:
  - Semicolons optional
  - Delimit strings with single or double quotes (double quoted support **interpolation**)
  - Can omit parenthesis for method calls
  - **print** and **println** shortcuts for System.out.println



The home page for Groovy is at: http://www.groovy-lang.org/index.html

Like Java 8, Groovy supports functional programming and closures.

Interpolated strings can contain variable references, for example:

```
def name = "John Adams"
println "Hello, $name"
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Groovy DSL Example

```
1    // build.gradle
2    dependencies {
3     compile 'org.springframework:spring-context:4.2.6.RELEASE'
4     compile "org.codehaus.groovy:groovy-all:2.1.0"
5    }
```

```
1    // spring.groovy
2    import university.TeachingAssistant
3
4    beans {
5      ta23 (TeachingAssistant) {  // bean-name (bean-type)
6        taID = 465                // set property
7        name = "Sue Mi"           // set property
8      }
9    }
```

```
1    // StudentConfig.java
2    @ImportResource({"classpath:spring.groovy"})
3    public class StudentConfig {. . .}
```

"beans" is part of the DSL. Behind the scenes, Spring creates a GroovyBeanDefinitionReader that parses the DSL and configures Spring beans into the container.

For more information on the Groovy DSL, see:

https://spring.io/blog/2014/03/03/groovy-bean-configuration-in-spring-framework-4

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About the advantanges of Spring Java Configuration
- How to use Java Configuration to perform dependency injection

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JEE

- What is JEE?
- JEE Development Process
- Overview of JEE Technologies

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is JEE?

- Java Enterprise Edition is a set of specifications for server-side Java programming
- Vendors implement JEE by creating **containers** within an application server
- From the Oracle Web site:

> Java Platform, Enterprise Edition (Java EE) builds on the solid foundation of Java Platform, Standard Edition (Java SE) and is the industry standard for implementing enterprise-class service-oriented architecture (SOA) and next-generation web applications.

Java consists of three programming platforms:

- Java Standard Edition, which is the core of Java

- Java Micro Edition, which allows Java applications to run mobile devices

- Java Enterprise Edition, covered here.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Version History

- J2EE 1.2 (December 12, 1999)
- J2EE 1.3 (September 24, 2001)
- J2EE 1.4 (November 11, 2003)
- Java EE 5 (May 11, 2006)
- Java EE 6 (December 10, 2009)
- Java EE 7 (May 28, 2013)

Note the name change that occured with JEE Version 5.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Benefits of JEE

- Portability across platforms
- Portability across vendors
- Containers provide high-level services
- Many implementations from which to choose
- Wide array of tools for development, testing and deployment

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Vendors

A partial list of JEE vendors/products:

- IBM WebSphere
- Oracle/BEA WebLogic
- Oracle GlassFish
- JBoss/Wildfly
- Apache Tomcat (Web container only)
- Apache Geronimo

The key is that it should be fairly easy to port applications from one vendor to another since JEE defines standard specifications.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Development Tools

A partial list of JEE development tools:

- IBM Rational Application Developer (RAD)
- Eclipse
- Oracle Netbeans
- Oracle JDeveloper
- JetBrains IntelliJ

The goal of these tools is to make the JEE development and deployment process easier.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Technologies

A partial list of the Application Programming Interfaces (APIs) defined by JEE

- Servlets
- JavaServer Pages
- JavaServer Faces
- Enterprise JavaBeans
- Java Persistence API
- Java Naming and Directory Interface
- JDBC
- Java Messaging Service
- Java SOAP Web Services
- Java RESTful Services

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Deployment Descriptors

- Many JEE technologies use XML-based **deployment descriptors** for configuration
- Many container vendors provide high-level tools to make it easier to work with these deployment descriptors
- One focus of JEE Version 5 is to reduce the requirement to write deployment descriptors

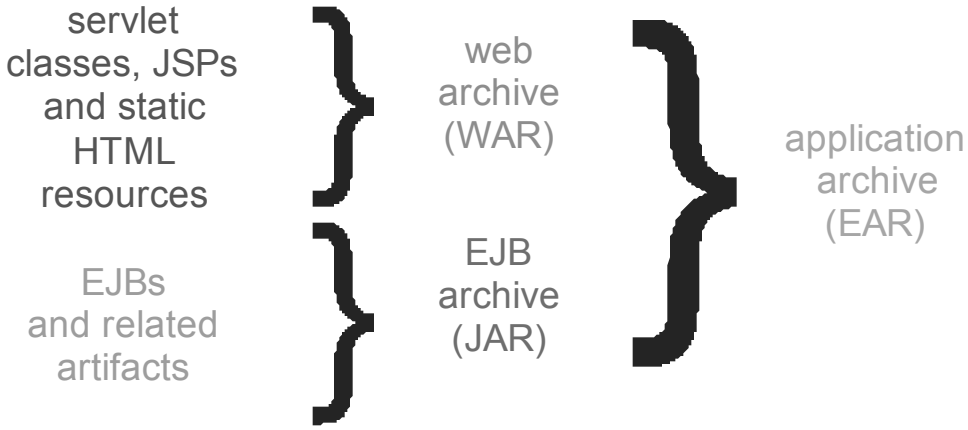
```
<?xml version="1.0"?>
<web-app>
  <servlet>
    <servlet-name>HelloCount</servlet-name>
    <servlet-class>HelloCount</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloCount</servlet-name>
    <url-pattern>HelloCount</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>InputCount.html</welcome-file>
  </welcome-file-list>
</web-app>
```


In JEE 5 and later, programmers can use code "annotations" instead of writing verbose XML configuration files.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Development and Packaging

- Since JEE is a Java technology, programmers write JEE Java code and compile it in the normal fashion
- JEE provides various ways to package complete applications to make them easier to deploy



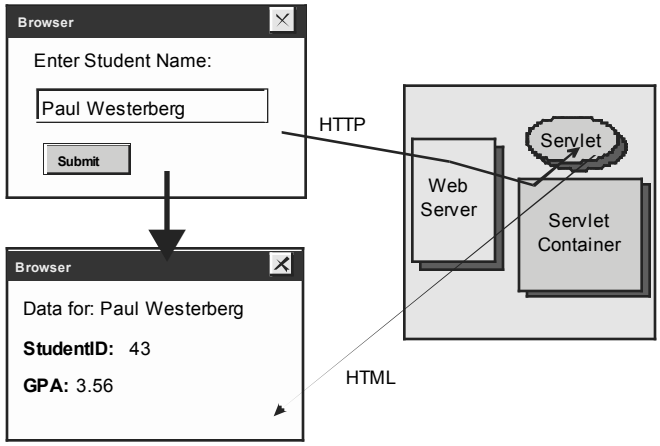
These archives are internally similar to ZIP files.

Note that each type of archive has its own deployment descriptor. The deployment descriptor shown on the last page was for a WAR file.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Servlets

- Servlets are a Java replacement for traditional CGI programs and let you add programmability to your Web sites
- Servlets perform well and are portable across container vendors and platforms
- A common use of servlets is to respond to HTML forms



When servlets were first introduced, their primary role was to generate HTML. This has changed somewhat with the introduction of JSPs -- now, servlets are generally more specialized -- they respond to requests, execute presentation-tier logic and then transfer control to the JSP that will generate the HTML.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JSPs

- Like servlets, JSPs let you add programmability to Web sites, but JSPs are more focused on presentation rather than Java coding
- Upon first access, the container converts JSPs into servlets, so they have the same performance and portability characteristics

```
1    <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
2       prefix="c"%>
3    <html>
4      <body>
5        <c:forEach begin="0" end="5">
6          <p>Hello, World</p>
7        </c:forEach>
8      </body>
9    </html>
```

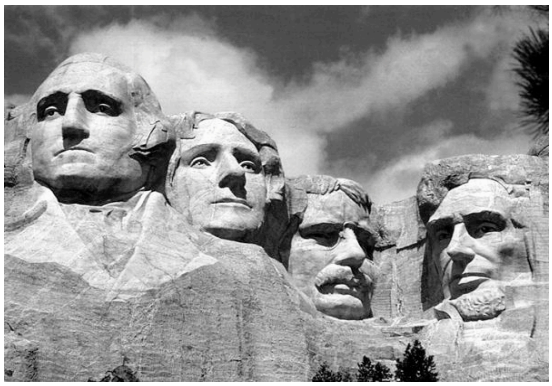

Though you could create an entire Web site using only JSPs, it turns out that putting a lot of code in a JSP makes them difficult to maintain. So as mentioned on the last page, a good strategy is to combine servlets and JSPs, putting the bulk of the presentation-tier code in the servlet and letting the JSP handle HTML generation.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JSF

- JavaServer Faces is a **component-based** framework for developing Java-based Web applications
- JSF supports a "rapid application development" programming style, especially coupled with a visual development tool
- JSF is a standard part of Java Enterprise Edition (JEE)





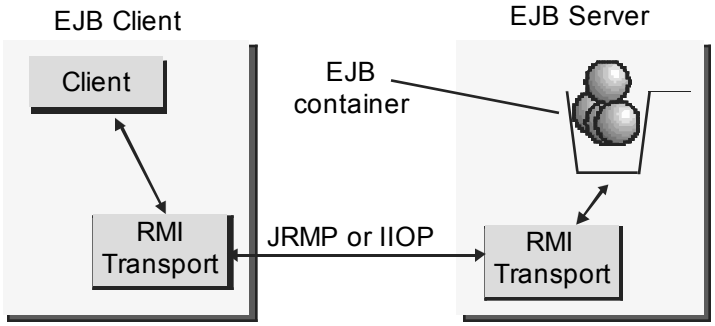
JSF borrows inspiration from client-side user-interface systems like Java's Swing, where you also build "screens" from components.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to EJBs

- Enterprise Java Beans are distributed components that run within the bounds of an EJB container
- The container provides services such as pooling, transactions, security and persistence
- From the EJB specification:

> Enterprise JavaBeans is an architecture for component-based computing. Enterprise beans are components of transaction-oriented enterprise applications.



One key to understanding EJBs is that they can optionally live remotely from the program that calls them. It's quite common in JEE applications to architect a servlet as the EJB "client".

JRMP and IIOP are protocols for distributed objects -- EJBs can use either one.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to CDI

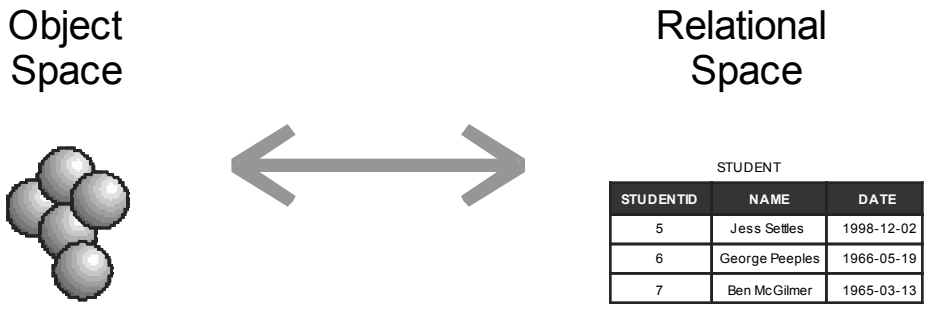
- The **Contexts and Dependency Injection** specification provides a programming model for **domain** objects in Java Enterprise Edition
- CDI is a standard part of JEE 6 and later and provides code **annotations** so that programs can request references to objects that provide a useful service
- CDI was inspired by the Spring Framework

```
public class Test
{
   @Inject
   private MyService service;
    . . .
}
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introducing the Java Persistence API

- **Java Persistence** is the official Oracle API for object/relational mapping
- JPA is a required part of Java Enterprise Edition, Version 5

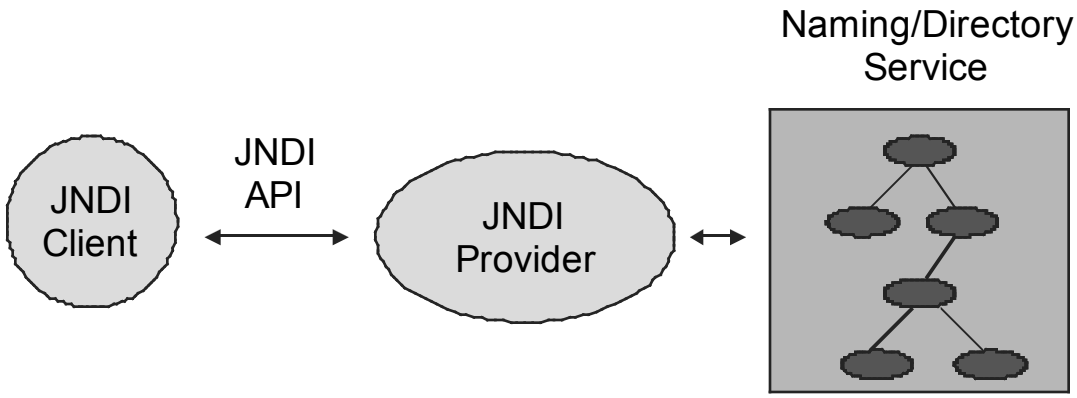


You can also use JPA in standalone JSE applications.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JNDI

- The Java Naming and Directory Interface provides a standard way to access naming services such as LDAP
- All JEE compliant containers must provide a JNDI implementation since other JEE technologies such as EJBs use JNDI

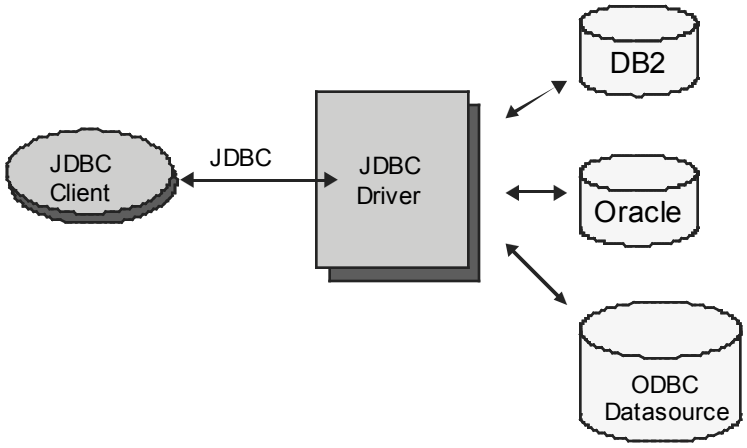


Most enterprise-class applications need the ability to look up resources in some sort of directory. Prior to JNDI, Java programmers needed to code to a particular naming/directory service and then if they changed services, all of the code would need to be rewritten. JNDI avoids that by providing a standard programming interface to any naming/directory service that has a JNDI provider written for it.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JDBC

- The JDBC API provides a standard way to access relational data so that your Java programs can remain independent of the actual database product
- To use JDBC, you must obtain a JDBC driver for your particular database



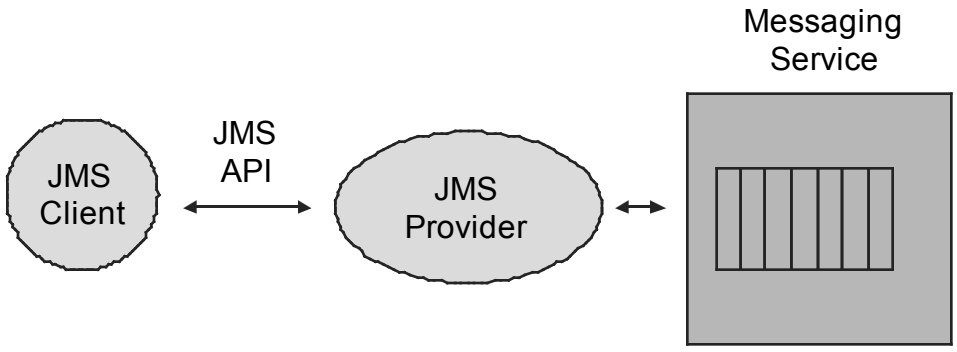
JDBC is one of the most well known and widely used JEE APIs (in fact, JDBC actually predates JEE!).

JDBC provides a standard programming interface for relational databases so that your Java code is independent of the actual database product. That way, if you need to switch database vendors, your code should still work.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JMS

- The Java Messaging Service provides a standard way to access message-oriented middleware (MOM) products such as IBM's WebSphere MQ in a product-independent fashion
- To use JMS, you must obtain a JMS provider for your particular MOM product



Messaging systems provide a way to create loosely coupled applications that are resistant to failure. There are several messaging services available, each with their own proprietary API. The JMS API provides a standard programming interface so that Java programs are independent of the actual messaging service. That way, if you need to change services, your code should still work.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JTA

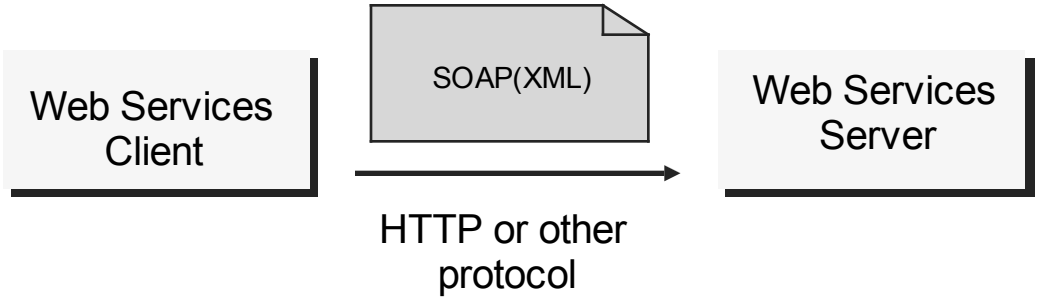
- The Java Transaction API provides a standard interface to transaction managers such as those included with database products like IBM's DB2 or Oracle
- Most JEE programmers do not use JTA directly, but instead use **declarative** transactions and let the container handle transactions

Declarative security means that you configure security constraints in a deployment descriptor rather than coding to the JTA directly.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to SOAP Web Services

- Web services use the SOAP protocol to allow for remote procedure calls and document exchange
- JEE defines the JAX-WS API so that JEE developers can use Web services in a vendor-independent fashion

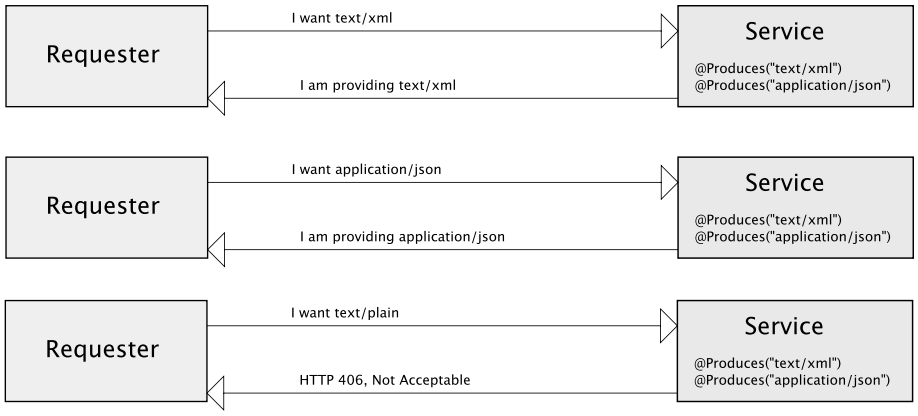


Web services provide a platform and language-independent way to invoke remote services or pass arbitrary business documents. Large companies such as Microsoft and IBM have adopted Web services as an integration technology.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to REST

- **Representational State Transfer (REST)** is an architectural style that emphasizes stateless transfer of resource representations using standard HTTP methods
- JEE defines the JAX-RS API so that JEE developers can develop RESTful applications



REST was introduced by Roy Fielding, who was one of the authors of HTTP itself.

The figure shows how a requester (client) and provider might negotiate which type of data to transfer.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JavaMail

- The JavaMail API provides a standard way for Java applications to send and receive Internet email without having to program the SMTP protocol directly

```
1    Properties p = System.getProperties();
2    p.put("mail.host", "localhost" );
3
4    MimeMessage message =
5          new MimeMessage ( Session.getInstance(p, null ) );
6    message.setFrom
7           ( new InternetAddress ( "me@mycompany.com" ) );
8    message.setRecipients(Message.RecipientType.TO,
9          InternetAddress.parse ( "you@yourcompany.com" ) );
10
11   message.setSubject ( "Great Subject" );
12   message.setText( "The best message ever!" )
13   Transport.send(message);
```

JavaMail is an API for sending and receiving email in Java programs. The code fragment on this page shows sending a message to "you@yourcompany.com" from "me@mycompany.com" with the subject "Great Subject" and message body containing the text "The best message ever".

While it's not important at this point that you understand this code completely, it should give you a feel for how the JavaMail API works.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- What JEE is all about
- Fundamentals of several JEE technologies

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Servlets

- Why Use Servlets?
- The Servlet Container
- The Servlet API
- The Servlet Lifecycle

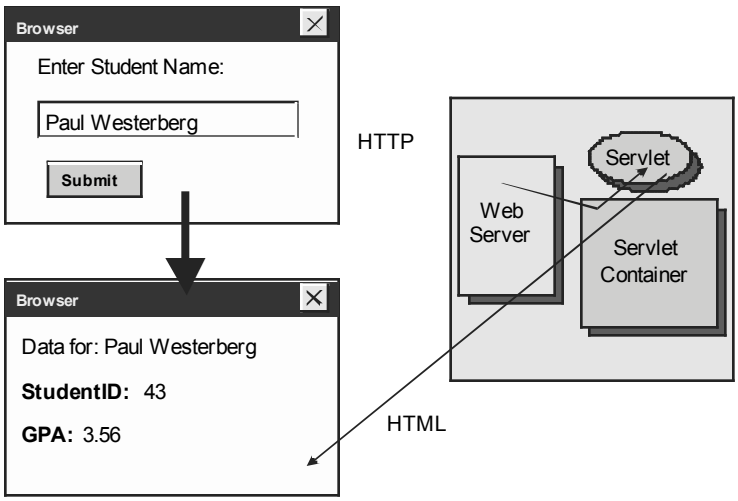
In this chapter, you will learn the basics of the servlet architecture and will examine a simple but complete servlet.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Servlet?

- A servlet is a Java program that's invoked by a Web server -- servlets typically generate HTML
- Servlets can take the place of traditional CGI programs



13 - 2

This figure shows a typical use of a servlet: responding to an HTML form. When the user submits the form, the Web server passes control to the servlet, which generates an HTML response back to the client.

The picture is much the same as in the last chapter on CGI -- the difference is that we now use a servlet instead of a Perl program. Note also that we refer to the servlet- support runtime as the servlet container</i>.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Use Servlets?

- Servlets scale better than CGI programs where the Web server spawns a separate process for each request. In contrast, servlets can stay in memory once they are loaded
- Since servlets are written in Java, they are portable across platforms
- Unlike other proprietary schemes, servlets are widely supported by application servers and thus are portable across vendors

Recall from the last chapter, that some CGI implementations execute a separate process for each request, causing problems when a large number of users access the site at the same time. Servlet applications don't have this problem since the servlet container can keep a servlet in memory for subsequent requests. Thus servlet applications scale well as more users connect to the application.

The key advantage of all Java server-side technologies is that they are portable, both across platforms, and across vendors. For example, say you write a servlet that works with a particular vendor's servlet runtime. Since Java is portable, you should be able run that servlet on any platform that the vendor supports, for example Windows or Linux.

But even more important, you should also be able to change vendors with minimum impact on your servlet code. For example, suppose you initially deploy your servlet under IBM WebSphere, but then decide to use BEA WebLogic instead. Since the servlet specification is vendor independent, you should be able to do that without changing your code.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Servlet Containers

- Servlet support is not included with web servers like Apache. To support servlets, you must install a a container for the servlet objects
- There are several popular servlet containers including:
  - Apache Tomcat
  - IBM WebSphere
  - Oracle WebLogic

This page lists some of the popular servlet containers available today. Note that just installing a Web server is not sufficient: if you want to deploy servlets, you also need a servlet container.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Servlet API

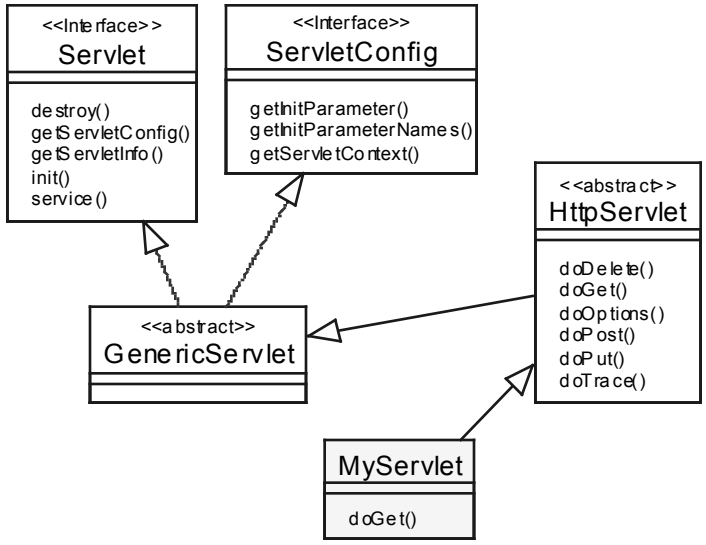
- Sun has defined a set of standard interfaces that you must implement when you write a servlet. The container calls the methods in the interfaces at well defined times
- The container also implements a standard set of interfaces that the servlet can call if required

In the rest of this course, you will learn much of the servlet API. Note that the API falls into two categories: callbacks from the container into the servlet and calls from the servlet to the container. We will examine both categories in this course.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Servlet

- You can extend the HttpServlet abstract class to implement your servlet
- We will examine these interfaces in more detail later in the course



This figure is a class diagram that uses Unified Modeling Language (UML) notation to describe the inheritance hierarchy for a typical servlet. This servlet extends HttpServlet and overrides the doGet method so it can respond to a "get" request from a browser.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Sample Servlet

```
1    import java.io.*;
2    import javax.servlet.*;
3    import javax.servlet.http.*;
4    import javax.servlet.annotation.*;
5
6    @WebServlet("/MyServlet")
7    public class MyServlet extends HttpServlet
8    {
9        public void doGet ( HttpServletRequest request
10                           , HttpServletResponse response)
11           throws ServletException, IOException
12       {
13           response.setContentType ( "text/html" );
14           PrintWriter out = response.getWriter();
15           out.println ("<html><body>" );
16           for ( int i=0; i < 5; i++ )
17               out.println ("<p>Hello, <b>world</b></p>" );
18           out.println ("</body></html>" );
19       }
20   }
```

Here is the code for the sample servlet modeled on the last page. In lines 1-4, it imports the java.io package and the standard servlet packages.

In line 6, we use an annotation introduced in Java Enterprise Edition (JEE) Version 6 to assign a URL to the servlet.

In line 7, the servlet extends HttpServlet as shown in the UML diagram on the last page. In line 9, we override doGet, accepting request and response arguments. In line 13, we use the response argument to set the content type. Then, in line 13, we retrieve an output writer to which we can send the HTML response, which we generate in lines 15 to 18.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The advantages of using servlets
- About servlet containers
- How to write a simple servlet

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JavaServer Pages

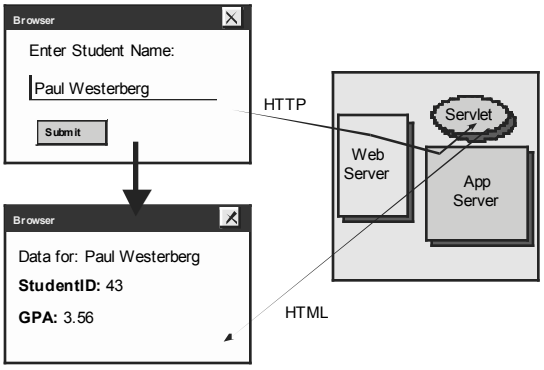
- What are JavaServer Pages?
- Separating Content From Presentation

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What are JavaServer Pages?

- JSPs are a J2EE technology that combine static HTML with embedded Java code
- In order to use JSPs, you need a JSP container as part of your application server
- Popular JSP containers include IBM WebSphere, JBoss, Apache Tomcat and BEA WebLogic



JSPs are similar to other server-side scripting technologies such as Microsoft's Active Server Pages (ASP). Like ASPs, well-written JSPs are mostly HTML with small snippets of Java code (not JavaScript).

And just as with servlets, to use JSPs, you need a container to host the JSPs. In fact, many vendors supply both JSP and servlet containers in the same application server product.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSPs Are Actually Servlets

- You write a JSP by coding HTML with snippets of Java code referred to as *scriptlets*
- When the JSP container loads the JSP, it translates it into a servlet and then compiles it on the fly

```
1    <jsp:useBean id="now" class="java.util.Date"/>
2    <html>
3      <body>
4        <h3>My First Page</h3>
5        <p>
6          The <b>current</b> time is: ${now}
7        </p>
8      </body>
9    </html>
```


This page shows a simple JSP that outputs HTML for "Hello, world" five times. Note that the JSP consists of standard HTML boilerplate in lines 1 and 8, and a chunk of Java code bracketed by <% and %>. The code within this bracket is regular, old Java code. Note that it references an implicit variable</i> named "out" -- this variable is defined for the JSP and is similar to the PrintWriter object used by servlets to write content back to the client.

To run a JSP, a client enters the JSP's URL into a browser.

It's important to realize the JSPs end up being servlets -- the first time a reference is made to the JSP's URL, the container translates the JSP into a servlet and then compiles it and deploys it into a servlet container. The servlet container then calls the normal methods (for example "init") on the generated servlet.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Use JSPs?

- JSPs are easier than servlets for non-programmers to create - you can use standard HTML authoring tools
- Since JSPs are compiled into servlets, they perform well
- JSPs can act as the View portion of a Model-View-Controller architected web application

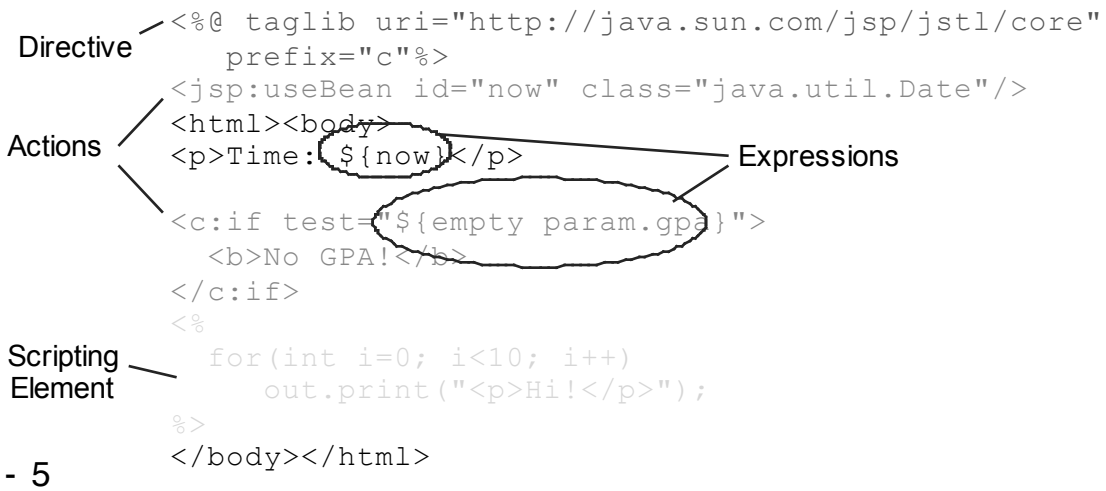
Since JSPs end up being servlets, you may wonder why you would JSPs instead of servlets. The main reason is that JSPs are mainly HTML and thus can be written by web-page designers using standard HTML authoring tools. And since they are really servlets, they have all of the advantages of servlets including good performance, platform independence and so forth.

Finally, JSPs work well together with servlets using an architecture known as Model-View-Controller. We will cover MVC in the next chapter.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSP Version History

- Version 1.0 introduced JSPs and defined basic syntax of **directives** and **Java scripting elements**
- Version 1.1 added **action elements** and **tag libraries**
- Version 2.0 added an **expression language** and a simpler way to write tag libraries



A "directive" is a command to the page translator, while a "scripting element" lets you embed executable Java statements within a JSP.

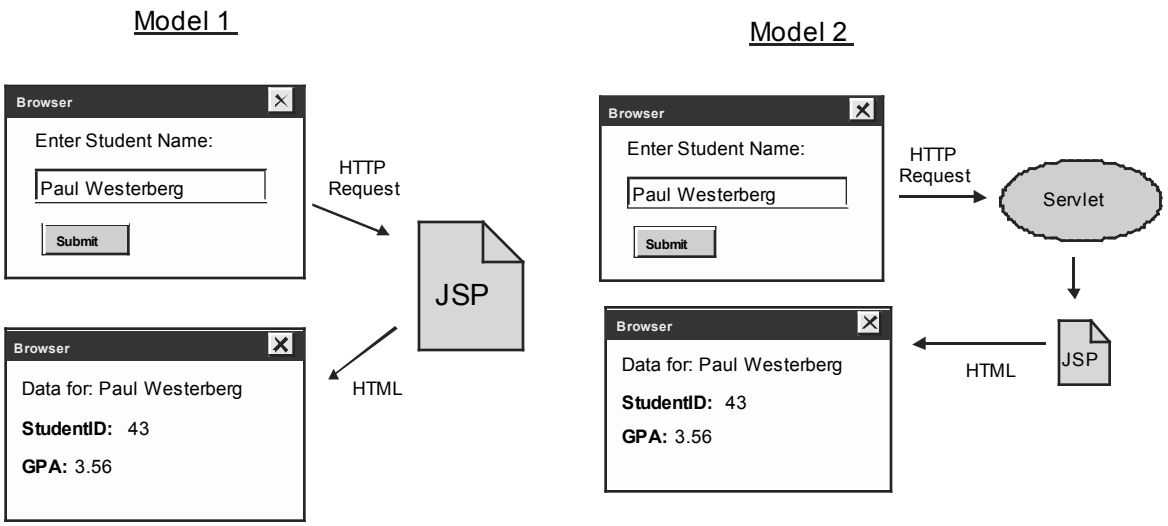
"Action elements" are like directives -- they extend the set of commands that the JSP translator recognizes. A "tag library" is a collection of user-written custom action elements.

An "expression language" is a set of syntax that lets you perform common operations like adding or subtracting numbers, accessing well-known variables and so forth.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Model 1 vs Model 2

- In a **Model 1** architected application, JSPs respond directly to HTML forms
- In a **Model 2** (aka MVC) application, servlets process forms and then transfer control to JSPs



There are two common ways to architect J2EE Web applications -- Model 1-based applications use JSPs exclusively, while Model 2 apps use a mixture of servlets and JSPs. Model 2-based designs are considered superior for large applications, since Model 2 provides easier maintenance. On the other hand, Model 1 is simpler for smaller applications.

Note there is a third possibility -- using only servlets. There's no catchy "Model x" moniker for that choice however.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Intro to the JSP Expression Language

- The JSP **expression language** is a language, based on Java, that makes it easy for JSP authors to perform calculations and access data defined in *JavaBeans*
- Prior to JSP 2.0, you could only use the expression language in conjunction with the *JSP Standard Tag Library* (JSTL)

```
<p>
  The total is: ${item.quantity * item.price}
</p>
```


Expressions are delimited by '${' at the start and '}' at the end. The expression shown on this page multiples the value of a JavaBean property named "quantity" times another property named "price".

Before the advent of the expression language, the only way to add programmability to JSPs was to write scripting elements, which consist of Java code. The downside to scripting elements is that the page author needs to be familiar with Java syntax and notions such as garbage collection, primitives versus references and so forth. The expression language is much easier for non-programmers to learn than full-blown Java.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A JSP Consists Of...

- Template Text
  - Typically HTML or XML
  - Written unmodified back to the client via the implicit *out* variable
- Elements
  - Use syntax based upon XML
  - Elements include:
    - Directives
    - Scripting elements (scriptlets)
    - Action elements

This page shows the parts of a JSP. Template text, which is usually HTML, comprises the bulk of a well-written JSP. The servlet that's generated from the JSP sends template text directly back to the client.

Elements add programmability to the JSP. Directives act as commands to the container, scriptlets hold the embedded Java code and action elements perform various other tasks such as giving a JSP access to JavaBeans.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing Scriptlets

- Scriptlets contain Java code -- you can write as many as you wish, but the JSP container combines them all sequentially into the generated servlet's _jspService method

```
1    <%
2        for ( int i = 0; i < 5; i++ )
3        {
4    %>
5    <p> Hello world</p>
6    <%
7        }
8    %>
```

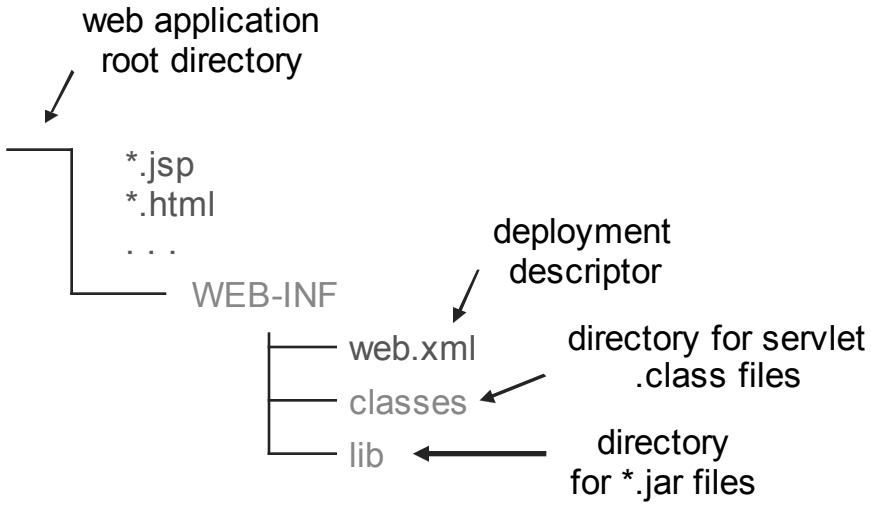

One kind of element is a scripting element or scriptlet. Each scriptlet must reside within a bracket of <% and %> characters. This page fragment shows two scriptlets that when combined, make up a complete "for" loop. Between the two scriptlets in line 5 is some template text. We'll leave it as an exercise for the student to show that this fragment has similar output to the JSP shown previously in this chapter.

With the advent of the JSP Expression Language, scriptlets are less common -- some people consider them to be "bad form".

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Deploying JSPs

- You normally deploy JSP files in the same directory hierarchy as static HTML files



Even though JSPs end up being servlets, you deploy them into the root directory of the web application's directory hierarchy. The container will then translate the JSP into a servlet, compile it and deploy it in a container-specific fashion.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Separating Content From Presentation

- If you are not careful, you can end up with JSPs that are "cluttered" with too much code in scriptlets, obscuring the static HTML
- That makes pages hard to understand and maintain -- in addition, if the business logic is in the JSP itself, you need to update the JSP if the logic changes
- A better design is to separate the bulk of the logic into external JavaBeans or EJBs and concentrate on the presentation in the JSP

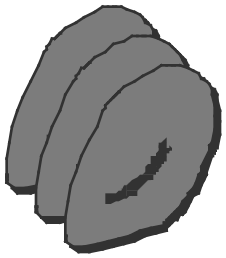
JSPs with too much code obviate the advantages of JSPs: they cannot easily be written or maintained by non-programmers. And since the JSP is a single "source" file, too much code quickly becomes unwieldy and hard to edit. In fact, the best JSPs have no code -- all the programmability is provided by action elements.

So if you find yourself writing a lot of code in your JSPs, you should consider re-architecting your system so that the JSPs call out to external Java code. We will cover that technique next.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a JavaBean?

- A JavaBean is a Java class that obeys a few simple conventions:
  - Use get/set methods to expose properties
  - Use public methods to expose behavior
  - Support firing events
- Typically, we don't use JavaBean events with JSPs



One recommended technique of minimizing JSP "code bloat" is to have the JSP reference JavaBeans that reside in the same Java Virtual Machine as the generated servlet. Sun wrote the JavaBean specification to allow just this sort of thing.

This page shows some of the conventions that you should follow so that a Java class can be considered a JavaBean. The most useful characteristic of JavaBean for JSPs is the notion of properties that we will cover on the next

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a JavaBean Property?

- A property is defined by the existence of "getter" and "setter" methods
- A readonly property defines only a "getter" method

```
1    public String getAddress()
2    {
3       ...
4    }
5    public void setAddress ( String s )
6    {
7      ...
8    }
9    public int getSerialNumber ()
10   {
11     ...
12   }
```

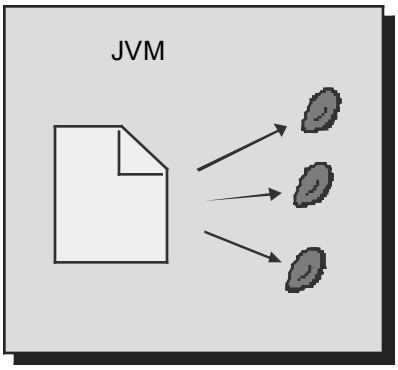

JavaBeans can define properties by providing methods that follow a naming convention. A normal property is made up of a "get" method and a "set" method -- the name of the property is what follows after the "get" or "set". On this page, lines 1 to 8 define a property named "address". Note that the property name starts with a lower-case character. Every property also has a type -- the "address" property's type is String.

This page also shows a read-only property of type "int" named "serialNumber".

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a JavaBean from a JSP

- There are two ways to access a JavaBean from a JSP:
  - Using built-in **JSP actions**
  - Using the **JSP expression language**



Now that you know how to write a Bean, let's turn our attention on how to write a JSP that uses the Bean.

Those of you taking the class that are Java programmers know that to access any Java object, you must first call the "new" operator to instantiate the object and then call the object's methods. To understand that process, you first need to learn quite a few fundamental Java concepts such as references, garbage collection and the like. Fortunately, non-programmers writing JSPs can avoid that learning curve by using the special syntax that we will cover next.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Referencing a JavaBean with Actions

- The **jsp:useBean** action element assigns a Bean references to the variable specified by the *id* attribute
- If the Bean does not exist, jsp:useBean will instantiate it. If the Bean does exist, the action will return a reference to the existing Bean

```
<jsp:useBean id="theStudent"
       class="studentinfo.StudentBean" />
```


This action element is roughly equivalent to the Java "new" operator -- it returns a reference to a Java object, in this case, a JavaBean. The jsp:useBean element accepts at least two attributes: the "id" attribute, which is the JSP's name for the Bean reference and the "class" attribute, which specifies the Java class of the Bean.

This element differs from the Java "new" operator in that it first checks to see if a Bean with the specified id already exists. If so, it sets the id attribute to the existing Bean.

Those students who are familiar with XML will note that jsp:useBean is what is known as an "empty element" -- note the slash immediately before the closing angle bracket. That's simply short-hand for writing a complete end-tag like </jsp:useBean>.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaBean Scope

- The **jsp:useBean** action element has a *scope* attribute that specifies where the action element should look for an existing bean
- Valid *scope* values include:
  - page (default value)
  - request
  - session
  - application

```
<jsp:useBean id="theStudent"
        class="studentinfo.StudentBean"
        scope="session" />
```


The JSP specification defines several "scopes" that indicate how global the JavaBean is. The default scope, "page" indicates that the Bean's lifespan is the same as the page -- in other words, when the page finishes execution, the Bean reference goes out of scope and is subject to garbage collection.

Request scope indicates that the bean reference is valid until the current request finishes. That means that if the JSP forwards the request to another JSP or servlet, the Bean reference will still be accessible.

Session scope is demonstrated in the code snippet. As you learned in an earlier chapter, a session is a series of requests from the same user. Session scope allows JSPs and servlets to maintain longer-lived data, for example a shopping cart.

Finally, application scope is specific to J2EE applications, which have the notion of well-defined Web applications. Application scope lets servlets and JSPs running within the same web application to maintain state, even if several users are accessing the application.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Retrieving Properties

- The **jsp:getProperty** action element retrieves the specified property and converts it to a String (calls the "getter" method)

```
<jsp:useBean id="theStudent"
         class="studentinfo.StudentBean" />

The student's gpa:
<jsp:getProperty name="theStudent"
                 property="gpa" />
```


Of course if you can set properties, you can also query them. This code snippet shows how to query the "gpa" property and send it to the HTML output stream. Again note that even the "gpa" property is a double, the getProperty element converts it to a String.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Expressions to Read Properties

- You can use JSP expressions to access bean properties using the dot operator
- The expression evaluator searches for the bean at each of the four scopes: **page**, **request**, **session** and **application** (jsp:useBean not required)

```
1    <!- assume a bean named "student"
2         is stored at session scope -->
3
4    <p>The student's GPA: ${student.gpa}</p>
5    <p>
6      The student's name:
7         ${sessionScope.student.name}
8    </p>
```


Since the expression evaluator automatically searches the four scopes, you can reference a bean with only its name. However, as the second example shows, you can explicitly specify the scope if desired.

Behind the scenes, the expression evaluator calls the "get" methods on a bean to retrieve its properties.

Note that to use the expressions, you don't need to use the jsp:useBean action (assuming that another JSP or servlet stores the bean at some scope).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Intro to the JSP Standard Tag Library

- A **tag library** is a set of **custom actions** that extend the JSP syntax
- The JSTL is a collection of custom actions that aims to make it easier for non-programmers to write useful JSPs without writing scripting elements
- By providing a standard set of actions, avoids "re-inventing the wheel"
- The JSTL, along with the EL, supplies a Perl or PHP-like environment where the page author can treat everything as strings

---

The JSTL library was developed under the Java Community Process and draws inspiration from tag libraries from the Apache Jakarta project and other authors.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSTL Core Tags

- The core tags provide basic functions that are useful in many kinds of JSPs

```
<%@ taglib prefix="c"
   uri="http://java.sun.com/jsp/jstl/core" %>
```

Tag	Description
c:out	Write to output stream
c:if	Simple conditional
c:choose,c:when,c:otherwise	Like Java switch-case
c:forEach	Iterating (looping)
c:set	Define or modify an EL variable
c:get	Retrieve an EL variable
c:exception	Handle exception locally
c:import	Similar to jsp:include
c:url	Encodes, rewrites URLs
c:redirect	Redirects to a URL

JSTL provides several categories of tags -- here we examine the "core" tags that have wide applicability.

Before you can use any tag library like the JSTL, the JSP must write the approriate "taglib" directive as shown here.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

c:forEach Example

```
1    <c:forEach var="i"
2               begin="0"
3               end="5">
4        <p>${i}. My List </p>
5    </c:forEach>
```

0. My List
1. My List
2. My List
3. My List
4. My List
5. My List

This example shows iterating with a fixed count. This is similar to a "for" loop in Java, but is not quite as flexible: the "step" must be > zero and the "begin" must be >= zero.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The basics of writing JSPs
- How to access JavaBeans from a JSP

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Spring MVC

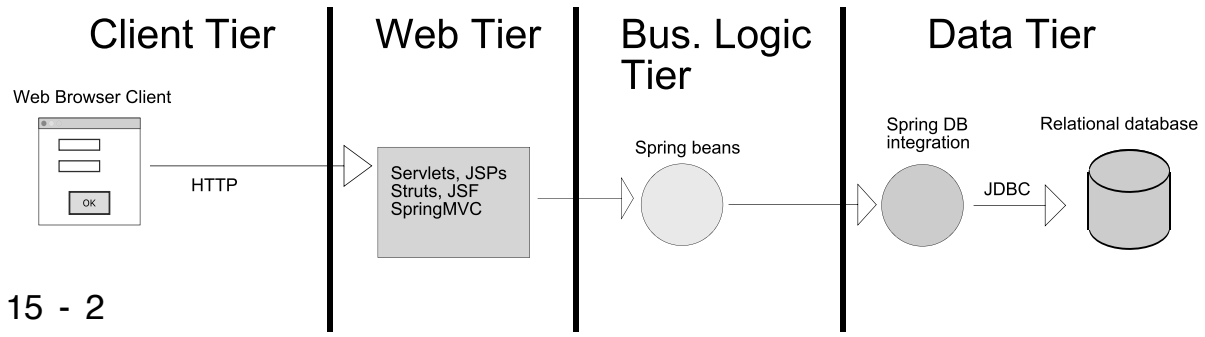
- MVC Fundamentals
- Spring MVC Architecture
- Spring MVC Sample Application

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and the Web

- There are three basic strategies for fronting Spring applications with a Web front end:
    - Use standard JEE servlets and JSPs as the Web tier
    - Use a JEE Web framework such as Struts or JSF as the Web tier
    - Use the SpringMVC framework as the Web tier



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Spring WebApplicationContext

- If you simply want to use a Spring business layer from a servlet/JSP, Struts or JSF Web app, Spring provides a **context listener** and a **WebApplicationContext**

```
1    web.xml fragment
2    ----------------
3
4    <context-param>
5      <param-name>contextConfigLocation</param-name>
6      <param-value>/WEB-INF/spring.xml</param-value>
7    </context-param>
8    <listener>
9      <display-name>ContextLoaderListener</display-name>
10     <listener-class>
11      org.springframework.web.context.ContextLoaderListener
12     </listener-class>
13   </listener>
```

A ServletContextListener is a standard part of JEE Web apps it's a class that implements that interface and is configured in web.xml. The interface has two methods: contextCreated and contextDestroyed the servlet container calls those methods when the Web app starts and is about to go away. Spring provides a ServletContextListener that creates a Spring ApplicationContext.

The bad news here is the unfortunate overloading of the term context. We have a servlet context (aka Web application context) and we have a Spring application context. They are different things. The servlet context is a part of the JEE specification, while, as we've already seen, the Spring application context represents the Spring container and does stuff like letting you retrieve beans by name.

The "context-param" element tells the listener where the Spring configuration files are located, typically within the WEB-INF folder of the Web application.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Spring WebApplicationContext, cont'd

```
1    Servlet class fragment
2    ----------------------
3
4    protected void doGet(HttpServletRequest request,
5     HttpServletResponse response)
6       throws ServletException, IOException
7    {
8      ServletContext servletContext = getServletContext();
9      WebApplicationContext ctx =
10       WebApplicationContextUtils.
11          getRequiredWebApplicationContext(
12           servletContext);
13
14     Student s1 = (Student)ctx.getBean("student1");
15
16     response.setContentType("text/html");
17     PrintWriter out = response.getWriter();
18     out.println("Student name: " + s1.getName() + "<br>");
19   }
```


A WebApplicationContext is an extension of the "normal" application context covered earlier in the course. The primary difference is that it can work with JEE Web application "scopes" such as request and session.

Here we show a fragment of a servlet that first retrieves the WebApplicationContext reference from the location where the Spring context-listener stored it. We use the Spring-provided WebApplicationContextUtils class to retrieve the context.

Once we have the WebApplicationContext, we can then retrieve and access beans in the normal Spring fashion.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

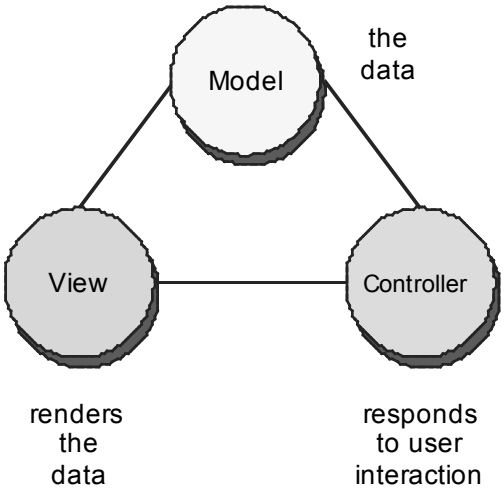
Introduction to Spring MVC

- Spring MVC is a Spring module that makes it fairly easy to create sophisticated Web front ends that use the Model-View-Controller design pattern
- You can using Spring MVC as an alternative to standard servlets/JSPs, Struts, JSF or other JEE Web frameworks
- Version 2.5 and later support using **annotations** to configure how Spring MVC maps *controllers* and maps requests

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is the MVC Design Pattern?

- The **Model-View-Controller** (MVC) architecture originated in the 1980s in the Smalltalk language, where it was used to build graphical user interfaces
- MVC is effective since it decomposes applications into components that you can develop, debug and maintain separately

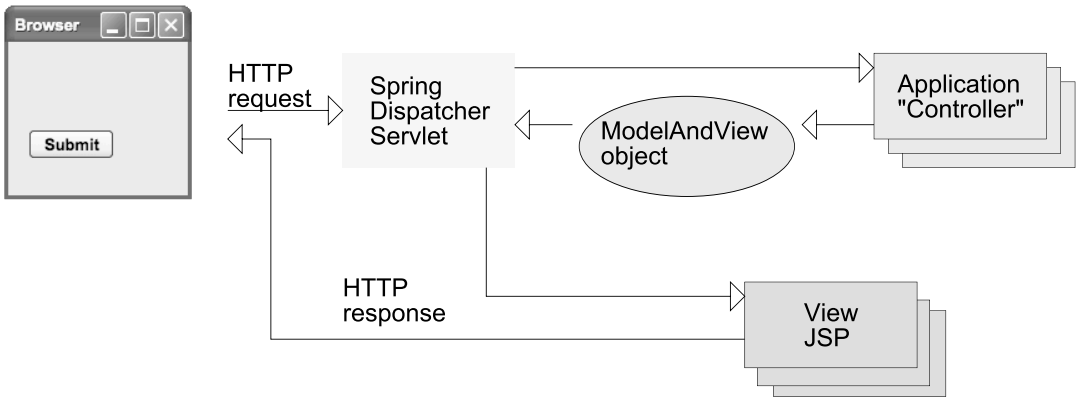


Other Java Web frameworks such as Struts, JSF and JEE 8 MVC use this pattern to varying degrees.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring MVC Architecture

- Spring MVC supplies a **front controller** servlet that acts as a single point of entry to an application
- Developers write "command controllers" that respond to requests from client



Note that if you use annotations, you don't need to explicitly create the ModelAndView object.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring MVC Controllers

- An application **controller** (not to be confused with a *front* controller), is an object that responds to a request, typically by populating a model and specifying a view
- In Spring 2.5 and later, controllers can be POJOs that you configure using annotations

```
1    @Controller
2    @RequestMapping("/formula")
3    public class QuadraticController
4    {
5      . . .
6    }
```


The @Controller annotation is a special case of the @Component annotation and configures the class as a Spring MVC controller - for it to work, you need to write an entry in a Spring configuration XML file:

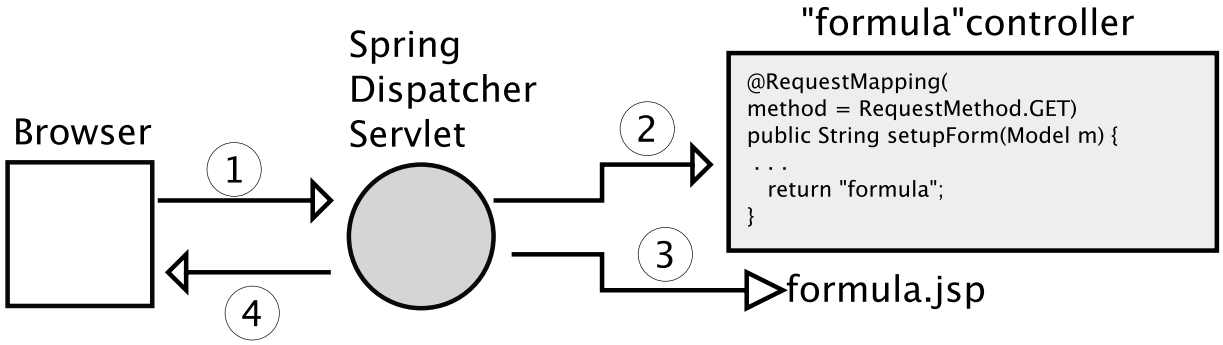
```
<context:component-scan base-package="my-controller-package" />
```

The @RequestMapping annotation assigns a URL to the controller.

In earlier versions of SpringMVC, controllers typically subclass either AbstractController and override handleRequestInternal(), or SimpleFormController and override onSubmit().

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Form Processing Flow: Phase 1



1. Browser sends GET /formula.html
2. DispatcherServlet finds controller with "formula" URL and invokes a "setup" method configured with @RequestMapping(method = RequestMethod.GET). The "setup" method optionally initializes a model and returns a String with a JSP name.
3. The DispatcherServlet executes the JSP, initializing form fields from model
4. DispatcherServlet sends response with form back to browser

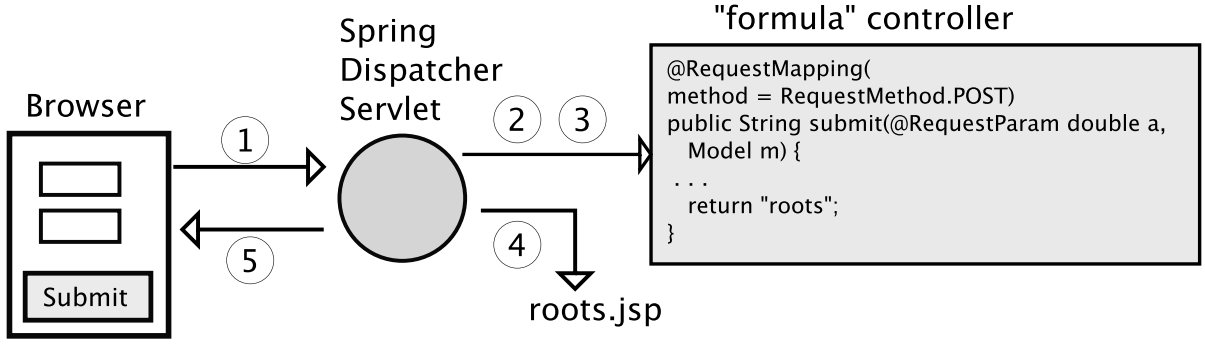
The first phase of the flow happens when a browser issues a GET request. The DispatcherServlet finds the matching controller bean, finds a "setup" method, which returns a String that specifies a JSP name. The servlet generates a HTML response back to the browser using the JSP as a template.

After this phase is complete, the browser displays a user input form, complete with a submit button. The user then completes the form and presses the submit button, which initiates Phase 2.

In Spring 4.3 and later, instead of using @RequestMapping on the setupForm() method, you can use a new annotation, @GetMapping, which is more descriptive.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Form Processing Flow: Phase 2



1. Browser sends POST /formula.html
2. DispatcherServlet maps HTTP request parameters to "submit" method parameters
3. DispatcherServlet invokes "submit" method annotated with @RequestMapping(method = RequestMethod.POST). The "submit" method processes request, populates model and specifies view JSP
4. DispatcherServlet forwards request to view JSP - the JSP accesses the model and generates a response
5. DispatcherServlet returns response to browser

Phase 2 entails actually processing the request and generating a response. The DispatcherServlet can differentiate between the phases since Phase 1 is an HTTP GET, while Phase 2 is a POST.

In Spring 4.3 and later, instead of using @RequestMapping on the submit() method, you can use a new annotation, @PostMapping, which is more descriptive.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring MVC Development Steps

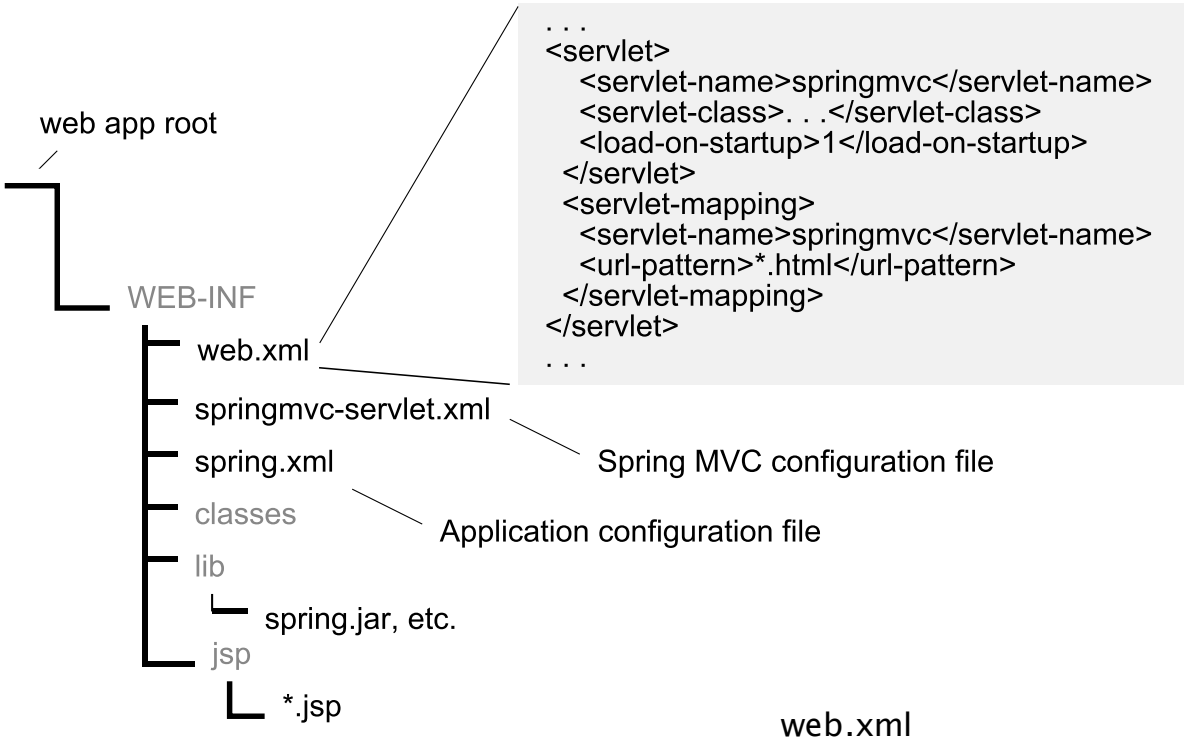
For a Spring SimpleFormController app:

- Configure the Web application:
  - Configure Spring MVC Dispatcher servlet in web.xml
  - Add Spring MVC JARS to WEB-INF/lib
- Create an HTML input form, optionally using Spring MVC custom actions
- Create an application **controller** class to process the form
- Write Spring configuration XML file(s)

Note that if we use annotations or Java configuration, the Spring XML file(s) can be minimal.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring the Web Application



Here we show the Web application configuration for a simple application. It's a good idea to create a separate folder in the WEB-INF directory for the JSPs - that will make it easier to map them as Spring MVC views.

We show a fragment of the web.xml deployment descriptor. Note the servlet configuration - we used the name "springmvc" and set up a URL pattern of *.html. That will route all requests ending in .html to the Spring MVC servlet.

Note also that since we named the servlet "springmvc", the Spring MVC configuration file is named "springmvc-servlet.xml". You can define additional "application" Spring configuration files using the context-listener approach shown earlier in this chapter.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/n
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
    id="WebApp_ID" version="3.0">

    <listener>
        <listener-class>org.springframework.web.util.Log4jConfigListener</listener
    </listener>
    <listener>
        <display-name>ContextLoaderListener</display-name>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring.xml</param-value>
    </context-param>
    <servlet>
        <servlet-name>springmvc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springmvc</servlet-name>
        <url-pattern>*.html</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

A Service Bean

- In the normal fashion, you can write Spring beans that provide application services
- As always, it's a good idea to write an interface first for application service beans

$ax^2 + bx + c = 0$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

SolveQuadratic.java

SolveQuadraticImpl.java

This service bean represents the quadratic formula, which solves equations of the form ax2 + bx +c.

Note how we annotate the implementation class with @Component so we don't need to configure it in an XML file.

We will need to configure annotation processing in the "application" spring.xml file:

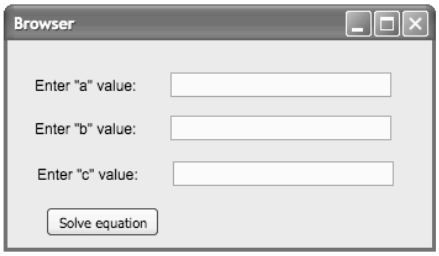
```
<context:annotation-config/>
<context:component-scan base-package="equations"/>
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package equations;
2
3    public interface SolveQuadratic
4    {
5        public double getRoot1(double a, double b, double c);
6        public double getRoot2(double a, double b, double c);
7    }
```

```
1    package equations;
2
3    @Component("solveQuadratic")
4    public class SolveQuadraticImpl implements SolveQuadratic
5    {
6
7        public double getRoot1(double a, double b, double c)
8        {
9            return (-b + Math.sqrt(b * b - 4 * a * c)) / 2 * a;
10       }
11
12       public double getRoot2(double a, double b, double c)
13       {
14           return (-b - Math.sqrt(b * b - 4 * a * c)) / 2 * a;
15       }
16   }
```

A Spring MVC Input Form



formula.jsp

Here we show formula.jsp, which resides in the "jsp" folder and defines a Spring MVC view containing an HTML input form.

Note the form's "action" - formula.html. We map all URLs ending *.html to the Spring MVC servlet, so it will process the request and invoke our controller.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <html>
2      <head>
3        <title>Solve The Quadratic Formula</title>
4      </head>
5      <body>
6        <h3>Solve The Quadratic Formula</h3>
7        <p>
8          The Quadratic Formula solves equations of the form   
9          <img src="equation.png"> using this formula:
10       </p>
11       <p>
12         <img src="QuadraticFormula.jpg">
13       </p>
14       <form action="formula.html" method="post">
15         <table>
16           <tr>
17             <td>Enter <i>a</i> value:</td>
18             <td><input type="text" name="a" value="${a}" /></td>
19           </tr>
20           <tr>
21             <td>Enter <i>b</i> value:</td>
22             <td><input type="text" name="b" value="${b}" /></td>
23           </tr>
24           <tr>
25             <td>Enter <i>c</i> value:</td>
26             <td><input type="text" name="c" value="${c}" /></td>
27           </tr>
28           <tr>
29             <td>
30               <input type="submit" value="Solve equation">
31             </td>
32           </tr>
33         </table>
34       </form>
35     </body>
36   </html>
```

A Spring MVC Result Page

```
1    <html>
2      <head>
3        <title>Equation Solved</title>
4      </head>
5      <body>
6         <h3>Equation Solved</h3>
7         <p>For input values:</p>
8
9          a=${a}<br>
10         b=${b}<br>
11         c=${c}<br>
12
13        <p><b>Root 1:</b> ${root1}</p>
14        <p><b>Root 2:</b> ${root2}</p>
15     </body>
16   </html>
```

Here we show the view JSP that displays the results of solving the equation.

Note that it uses the JSP Expression Language to display values named "a", "b", "c", "root1" and "root2" - our controller will need to store those values into the model. Then the dispatcher servlet will write them into request scope.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Controller Annotation Processing

- Since our controller uses annotations, we must enable annotation processing in the Spring MVC configuration file

```
1    Spring MVC configuration file
2    ----------------------------
3
4    <context:component-scan
5         base-package="controllers" />
```


This configuration is in the springmvc-servlet.xml file, and tells Spring MVC which packages potentially contain controllers annotated with @Controller, @ReqestMapping and @RequestParam.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a ViewResolver?

- So that controllers do not need to hard-code the file names of JSPs, SpringMVC provides **ViewResolver** classes that map logical strings into actual file names

```
1    Spring MVC configuration file
2    -----------------------------
3
4    <bean id="viewResolver"
5      class=
6    "org.springframework.web.servlet.view.InternalResourceViewResolver">
7      <property name="viewClass"
8        value="org.springframework.web.servlet.view.JstlView" />
9      <property name="prefix" value="/WEB-INF/jsp/" />
10     <property name="suffix" value=".jsp" />
11   </bean>
```

For example, if a controller method returns "roots" as a view logical string, this resolver will map to a JSP named "/WEB-INF/jsp/roots.jsp"

This configuration is in the springmvc-servlet.xml file.

The InternalResourceViewResolver does a simple mapping by applying both a prefix and a suffix to a logical view name.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Spring MVC Command Controller

QuadraticEquationController.java

Here we show a listing for a controller for the quadratic equation processing application.

We provide a "setupForm" method to provide a model that will intialize the form fields. Note that we pre-initialize it with sample values for the equation.

We provide a "submitForm" method to process the form. The Spring MVC servlet maps form HTTP parameters to method parameters, then invokes our method when the user posts the form.

After solving the equation, we put the original request values and the two roots into the model, then return a view JSP name. The Spring MVC servlet stores any such model attributes at request scope so that the view JSP can access the model data.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```java
package controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

import equations.SolveQuadratic;

@Controller
@RequestMapping("/formula")
public class QuadraticEquationController
{
    @Autowired private ApplicationContext ctx;

    @RequestMapping(method = RequestMethod.GET)
    public String setupForm(Model model)
    {
        model.addAttribute("a", "1");
        model.addAttribute("b", "3");
        model.addAttribute("c", "-4");

        return "formula";
    }

    @RequestMapping(method = RequestMethod.POST)
    public String submitForm(@RequestParam double a, @RequestParam double b,
            @RequestParam double c, Model model)
    {
        SolveQuadratic solver = (SolveQuadratic)ctx.getBean("solveQuadratic");

        model.addAttribute("a", a);
        model.addAttribute("b", b);
        model.addAttribute("c", c);

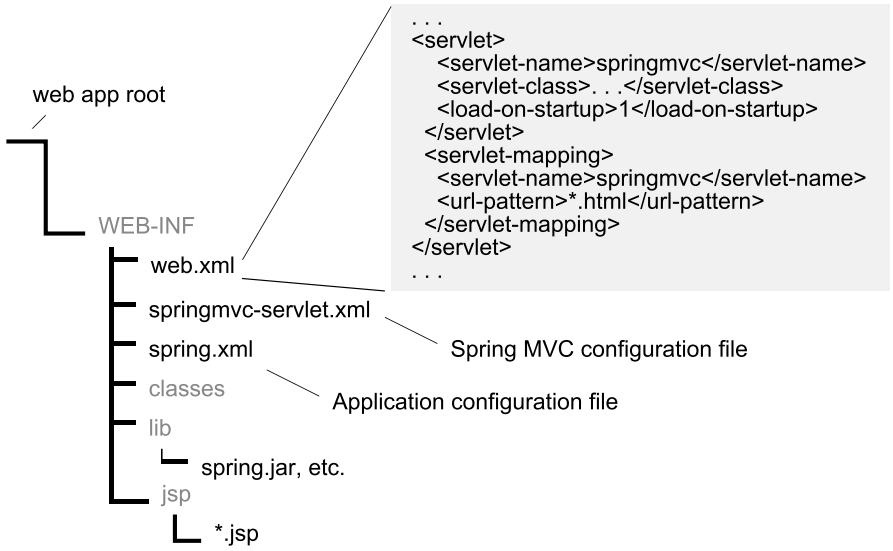
        model.addAttribute("root1", solver.getRoot1(a, b, c));
        model.addAttribute("root2", solver.getRoot2(a, b, c));

        return "roots";
    }
}
```

The Spring Configuration File

- By default, the Spring MVC servlet looks in the WEB-INF folder for a file with name xxxxx-servlet.xml where xxxxx is the servlet name in web.xml

springmvc-servlet.xml



Here we show the complete Spring MVC configuration file. The first bean, the "viewResolver", tells Spring MVC how to map view names to JSPs. We've configured it so that a view named "myview" corresponds to a file named "myview.jsp" in a directory named "jsp" in the Web application's WEB-INF directory.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.sp
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee" xmlns:jms="http://www.sp
    xmlns:lang="http://www.springframework.org/schema/lang" xmlns:tx="http://www.s
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-2.5.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
        http://www.springframework.org/schema/lang
        http://www.springframework.org/schema/lang/spring-lang-2.5.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-2.5.xsd">

    <context:component-scan base-package="controllers" />

    <bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass"
            value="org.springframework.web.servlet.view.JstlView" />
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
```

15 - 19A : springmvc-servlet.xml

MVC With Java Configuration

- If you use Java configuration for Spring MVC, you can avoid writing any Spring XML
- Recipe:
    - Write a configuration class annotated with @Configuration, @ComponentScan and @EnableWebMvc
    - In the configuration class, write bean-creation methods to configure a **view resolver** and any other application beans
    - Configure **contextClass** and **contextConfigLocation** in web.xml

javaconfg-web.xml

QuadraticConfig.java

If you use Java configuration, you don't need any Spring configuration files, but you may need to write elements in the standard web.xml.

The proper contexClass is org.springframework.web.context.support.AnnotationConfigWebApplicationContext.

If you are deploying to a JEE Web container that supports the servlet specification version 3 or later, you can even avoid writing the web.xml file and do all of your configuration in Java. For an example, see:

https://stackoverflow.com/questions/22315672/

how-to-configure-spring-mvc-with-pure-java-based-configuration

NOTE: The name of the Web deployment descriptor must be web.xml. We have changed its name for this presentation to avoid a name conflict, so if you use the one shown here, be sure to rename it web.xml.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3       xmlns="http://java.sun.com/xml/ns/javaee"
4       xmlns:web="http://java.sun.com/xml/ns/javaee"
5       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
6       id="WebApp_ID" version="3.0">
7       <servlet>
8           <servlet-name>springmvc</servlet-name>
9           <servlet-class>
10            org.springframework.web.servlet.DispatcherServlet
11            </servlet-class>
12          <init-param>
13              <param-name>contextClass</param-name>
14              <param-value>
15                org.springframework.web.context.support.AnnotationConfigWebApplicati
16              </param-value>
17          </init-param>
18          <init-param>
19              <param-name>contextConfigLocation</param-name>
20              <param-value>config.QuadraticConfig</param-value>
21          </init-param>
22          <load-on-startup>1</load-on-startup>
23      </servlet>
24      <servlet-mapping>
25          <servlet-name>springmvc</servlet-name>
26          <url-pattern>*.html</url-pattern>
27      </servlet-mapping>
28      <welcome-file-list>
29          <welcome-file>index.jsp</welcome-file>
30      </welcome-file-list>
31  </web-app>
```

```
1    package config;
2
3    import org.springframework.context.annotation.Bean;
4    import org.springframework.context.annotation.ComponentScan;
5    import org.springframework.context.annotation.Configuration;
6    import org.springframework.web.servlet.ViewResolver;
7    import org.springframework.web.servlet.config.annotation.EnableWebMvc;
8    import org.springframework.web.servlet.view.InternalResourceViewResolver;
9
10   import equations.SolveQuadratic;
11   import equations.SolveQuadraticImpl;
12
13   @Configuration
14   @EnableWebMvc
15   @ComponentScan("controllers")
16   public class QuadraticConfig
17   {
18       @Bean
19       public ViewResolver configureViewResolver()
20       {
21           InternalResourceViewResolver viewResolve =
22               new InternalResourceViewResolver();
23           viewResolve.setPrefix("/WEB-INF/jsp/");
24           viewResolve.setSuffix(".jsp");
25
26           return viewResolve;
27       }
28
29       @Bean
30       public SolveQuadratic solveQuadratic()
31       {
32           return new SolveQuadraticImpl();
33       }
34   }
```

15 - 20B : QuadraticConfig.java

Using Thymeleaf Instead of JSP

- **Thymeleaf** is a template engine that many Spring developers prefer instead of JSPs for the view:
  - Supports HTML5 out of the box and lets you better preview pages in a browser (JSPs often need to run tag libraries)
  - There is a Spring/Thymeleaf integration library



Thymeleaf's home page is http://www.thymeleaf.org/index.html.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Thymeleaf/Spring Recipe

- Steps for using Thymeleaf:
    - Load the Thymeleaf libraries into the project
    - Write HTML5, XML-parseable Thymeleaf **templates** instead of JSPs
    - Configure Thymeleaf template-engine and view-resolver beans

formula.html

roots.html

QuadraticThymeleafConfig.java

Note that Thymeleaf templates must be well formed XML – that means that you need to close all tags, even those such as <img> that we don't normally explicitly close in HTML5.

Also note that with Thymeleaf, we generally map the Spring Dispatcher servlet to a URL pattern of '/'. That means that we need to do a bit more configuration so that the dispatcher servlet sends static resources such as images from the Web app's root directory.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html>
2    <html>
3      <head>
4        <title>Solve The Quadratic Formula</title>
5      </head>
6      <body>
7        <h3>Solve The Quadratic Formula</h3>
8        <p>
9          The Quadratic Formula solves equations of the form   
10         <img src="equation.png" /> using this formula:
11       </p>
12       <p>
13         <img src="QuadraticFormula.jpg" />
14       </p>
15       <form action="formula.html" method="post">
16         <table>
17           <tr>
18             <td>Enter <i>a</i> value:</td>
19             <td><input type="text" name="a" th:value="${a}"/></td>
20           </tr>
21           <tr>
22             <td>Enter <i>b</i> value:</td>
23             <td><input type="text" name="b"  th:value="${b}"/></td>
24           </tr>
25           <tr>
26             <td>Enter <i>c</i> value:</td>
27             <td><input type="text" name="c"  th:value="${c}" /></td>
28           </tr>
29           <tr>
30             <td>
31               <input type="submit" value="Solve equation" />
32             </td>
33           </tr>
34         </table>
35       </form>
36     </body>
37   </html>
```

```
1    <!DOCTYPE html>
2    <html>
3      <head>
4        <title>Equation Solved</title>
5      </head>
6      <body>
7         <h3>Equation Solved</h3>
8         <p>For input values:</p>
9
10         <div>
11            a=<span th:text="${a}"></span>
12         </div>
13         <div>
14            b=<span th:text="${b}"></span>
15         </div>
16         <div>
17            c=<span th:text="${c}"></span>
18         </div>
19
20        <p><b>Root 1:</b> <span th:text="${root1}"></span></p>
21        <p><b>Root 2:</b> <span th:text="${root2}"></span></p>
22     </body>
23   </html>
```

```
1    package config;
2
3    import org.springframework.context.annotation.Bean;
4    import org.springframework.context.annotation.ComponentScan;
5    import org.springframework.context.annotation.Configuration;
6    import org.springframework.web.servlet.config.annotation.DefaultServletHandlerConf
7    import org.springframework.web.servlet.config.annotation.EnableWebMvc;
8    import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
9    import org.thymeleaf.spring4.SpringTemplateEngine;
10   import org.thymeleaf.spring4.view.ThymeleafViewResolver;
11   import org.thymeleaf.templateresolver.ServletContextTemplateResolver;
12
13   import equations.SolveQuadratic;
14   import equations.SolveQuadraticImpl;
15
16   /*
17    *   Note that we extend WebMvcConfigurerAdaptor
18    *   so we can override configureDefaultServletHandling().
19    *   That lets us configure the Spring MVC servlet so
20    *   it loads static resources (e.g. images) from the
21    *   WebContent folder (context root).
22    */
23
24   @Configuration
25   @EnableWebMvc
26   @ComponentScan("controllers")
27   public class QuadraticThymeleafConfig extends WebMvcConfigurerAdapter
28   {
29       @Bean
30       public ServletContextTemplateResolver templateResolver()
31       {
32           ServletContextTemplateResolver resolver =
33              new ServletContextTemplateResolver();
34
35           resolver.setPrefix("/WEB-INF/templates/");
36           resolver.setSuffix(".html");
37           resolver.setTemplateMode("HTML5");
38           resolver.setCacheable(false);
39
40           return resolver;
41       }
42
43       @Bean
44       SpringTemplateEngine templateEngine()
45       {
```

```
46          SpringTemplateEngine templateEngine = new SpringTemplateEngine();
47
48          templateEngine.setTemplateResolver(templateResolver());
49
50          return templateEngine;
51      }
52
53      @Bean
54      public ThymeleafViewResolver viewResolver()
55      {
56          ThymeleafViewResolver tvr = new ThymeleafViewResolver();
57          tvr.setTemplateEngine(templateEngine());
58
59          return tvr;
60      }
61
62      @Override
63      public void configureDefaultServletHandling(
64              DefaultServletHandlerConfigurer configurer)
65      {
66          configurer.enable();
67      }
68
69      @Bean
70      public SolveQuadratic solveQuadratic()
71      {
72          return new SolveQuadraticImpl();
73      }
74  }
```

Chapter Summary

In this chapter, you learned:

- Fundamental Model-View-Controller concepts
- The basics of the Spring MVC module

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and JDBC

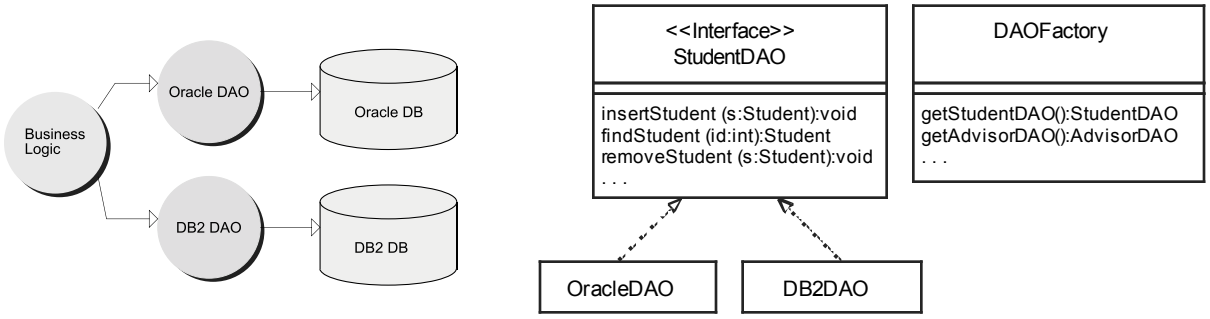
- The DAO Design Pattern
- JdbcTemplate and SimpleJdbcTemplate
- JdbcDaoSupport

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The DAO Design Pattern

- Spring supports and encourages the JEE DAO design pattern and simplifies coding, even if you you use basic JDBC



In a perfect world, or if JDBC was perfect, changing databases would not require any new code. However, in the real world, if you decide to change databases, at least some of your JDBC code probably will be affected. The goal of the DAO pattern is to minimize such changes if we do make a change.

By defining an interface for the DAO, you can minimize the impact of changes on the business logic. Here we show defining a StudentDAO interface that specifies the required behavior for any Student data-access object. The Oracle-specific and DB2-specific classes implement the interface with database-specific behavior.

The DAO Factory interface is an optional part of the pattern that is not really needed in Spring, since Spring programs generally use dependency injection to "hide" the creation of objects, including the DAO objects themselves.

You can read more about the DAO design pattern at:

`http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html`

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What's Wrong with JDBC?

- Even if you use DAOs, JDBC involves a lot of low-level, tedious and repetitive coding
- Managing JDBC connections is prone to errors
- OR/M solutions like Hibernate help with some of these issues

```
1    Connection con = null;
2    try {
3      con = dataSource.getConnection();
4      . . .
5    }
6    catch (SQLException exc) {
7    . . . }
8    finally {
9      try {
10       if (con != null) con.close();
11     catch (Exception e) {}
12   }
```


The snippet of code shown here should look familiar to anyone who's worked much with JDBC. Note the "try-catch" and "finally" blocks and how such programs need to manually manage connections.

While this code is not all that conceptually difficult, it is tedious and verbose.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

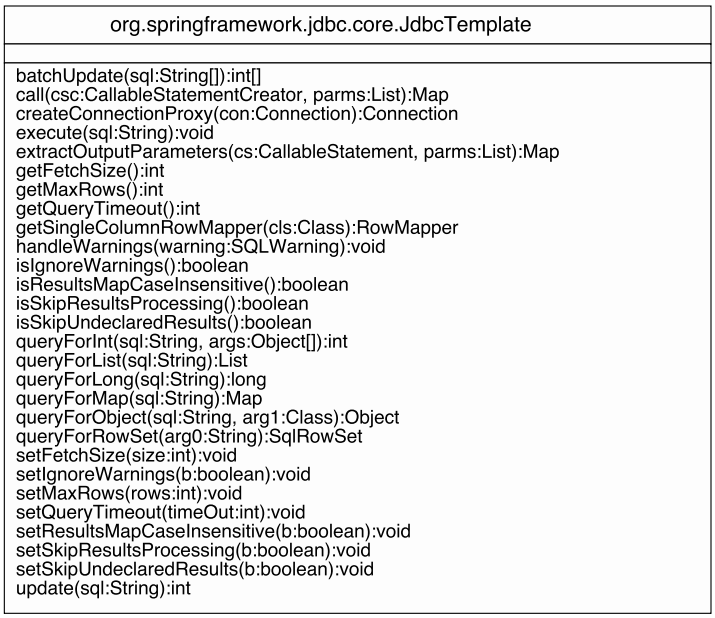
Introducing the Spring JDBC Module

- Spring provides the following support for database access:
    - Classes that retrieve an underlying JDBC **data source**
    - **Template** classes that simplify database coding, including templates for basic JDBC, Hibernate and JPA
    - **Support** classes that you can subclass when writing your DAOs
    - An improved exception hierarchy that's based on unchecked (runtime) exceptions

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The JdbcTemplate Class

- This is the workhorse class for programs that use Spring's basic JDBC support
- It provides methods to let you query, update, insert and delete rows in a table

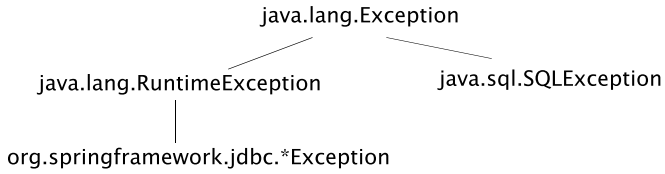


Note that there are many more methods in this class that we didn't show in the figure so that the figure fits on the page.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring JDBC Exceptions

- JDBC provides only the single SQLException and it's a **checked** exception, which can require messy nested **try-catch** blocks
- Spring provides many more precisely named **unchecked** exceptions -- exception handling is simpler and more intuitive



Selected Spring JDBC Exceptions (in org.springframework.jdbc)

BadSqlGrammarException
CannotGetJdbcConnectionException
IncorrectResultSetColumnCountException
InvalidResultSetAccessException
JdbcUpdateAffectedIncorrectNumberOfRowsException
LobRetrievalFailureException
SQLWarningException
UncategorizedSQLException

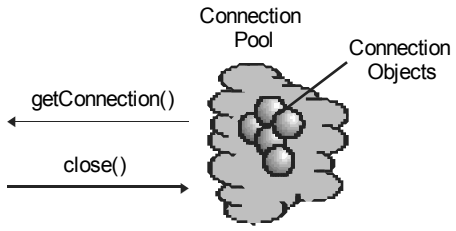
The Spring exceptions are unchecked, so you don't need try-catch blocks. This is a standard Spring philosophy. The idea is that you cannot recover from most JDBC exceptions, so why clutter the code with all of the try-catch blocks? You will need a try-catch block SOMEWHERE in the program to avoid uncaught exceptions, however.

To support this, Spring provides the SQLErrorCodeSQLExceptionTranslator class which consults a Spring-provided table of vendor error codes and their corresponding Spring exceptions.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JDBC Connection Review

- Non-Spring JDBC applications can obtain a connection in one of two ways:
  - Using the DriverManager interface
  - Using a DataSource
- The DriverManager approach is appropriate for standalone Java programs or for testing of enterprise applications
- The DataSource approach requires a JEE runtime that provides a JNDI service, but allows for **pooled** connections



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Working with Data Sources in Spring

- Spring lets you treat both JDBC approaches in a similar fashion
- Spring provides:
    - A DriverManagerDataSource class that uses the JDBC DriverManager approach
    - A JndiObjectFactoryBean class or a <jee: jndi-lookup> configuration element that locate JNDI-based DataSources
- Since you configure these in the Spring configuration file, you can easily switch between the two approaches

In Spring 1, we used the JndiFactoryBean, but in Spring 2 and later, it's easier to use the "jndi-lookup" element in the Spring configuration file to retrieve DataSource objects from JNDI.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a DriverManager Source (XML)

- The Spring DriverManagerDataSource wraps the JDBC DriverManager and is appropriate for standalone programs
- You can inject the data source bean into your DAOs using XML, annotations or Java configuration

```
1    <bean id="studentDataSource"
2      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
3      <property name="driverClassName"
4        value="org.apache.derby.jdbc.EmbeddedDriver"/>
5      <property name="url"
6        value="jdbc:derby:c:\springclass\db\StudentDB"/>
7      <property name="username"
8        value="SA"/>
9      <property name="password"
10       value="SA"/>
11   </bean>
```


The DriverManagerDataSource encapsulates a non-pooled JDBC DataSource. Note that we must provide connection properties.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a DriverManager (Java Config)

- With Java configuration, you can use the **DriverManagerDataSource** to create and initialize a driver-manager DataSource

```
1    @Bean
2    public DataSource dataSource() throws SQLException
3    {
4      DriverManagerDataSource ds =
5         new DriverManagerDataSource();
6      ds.setDriverClassName(
7         "org.apache.derby.jdbc.EmbeddedDriver");
8      ds.setUsername("SA");
9      ds.setPassword("SA");
10     ds.setUrl("jdbc:derby:c:\springclass\db\StudentDB");
11     return ds;
12   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a JNDI-Based DataSource (XML)

- You can use <jee:jndi-lookup> to create a data source via a JNDI lookup
- If your "client" defines a **resource-reference**, then you should specify **resource-ref="true"** - Spring will prepend *java:comp/env* to the provided JNDI name

```
<jee:jndi-lookup id="studentDataSource"
  jndi-name="studentDS" resource-ref="true">
```


With this technique, the programmer does not need to know connection details, only the JNDI name provided by an administrator.

Spring 1.x provided the JndiObjectFactoryBean, but newer applications should use jee:jndi-lookup.

If your client is a servlet, you can define a resource reference in the servlet's Web application deployment descriptor. This is an extra level of indirection that's considered a best practice in JEE since it insulates clients from changes in the actual JNDI name.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring JNDI (Java Config)

- With Java configuration, you can use the **JndiDataSourceLookup** to create and initialize a JNDI-based DataSource

```
1    @Bean
2    public DataSource studentDataSource()
3    {
4      JndiDataSourceLookup dsLookup =
5          new JndiDataSourceLookup();
6      dsLookup.setResourceRef(true);
7      DataSource dataSource = dsLookup.getDataSource(
8          "StudentDS");
9      return dataSource;
10   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a DAO

- DAO classes normally implement a DAO interface, and use injection to obtain the data source

```
1    @Repository
2    public class StudentDAOJdbcImpl implements StudentDAO
3    {
4        private DataSource studentDataSource;
5        private JdbcTemplate template;
6
7        @Autowired
8        public void setDataSource(DataSource ds)
9        {
10          studentDataSource = ds;
11          template = new JdbcTemplate(ds);
12       }
13   . . .
14   }
```

The @Repository annotation is a special case of the @Component annotation designed for use in the data tier. You have to configure the <context:component-scan> element in the XML to include the DAO's package.

Note how we use autowiring to obtain the data source reference on a "set" method, then create the JDBC template that we can use for queries, etc.

Not shown is the interface, StudentDAO, which defines all of the methods that the DAO provides.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JdbcTemplate Query Methods

- The JdbcTemplate provides numerous ways to query for various types

```
<T> List<T> query(String sql, RowMapper<T> rm, Map<String,?> args)
<T> List<T> query(String sql, RowMapper<T> rm, Object... args)
<T> List<T> query(String sql, RowMapper<T> rm, SqlParameterSource args)
int queryForInt(String sql, Map<String,?> args)
int queryForInt(String sql, Object... args)
int queryForInt(String sql, SqlParameterSource args)
List<Map<String,Object>> queryForList(String sql, Map<String,?> args)
List<Map<String,Object>> queryForList(String sql, Object... args)
List<Map<String,Object>> queryForList(String sql, SqlParameterSource args)
long queryForLong(String sql, Map<String,?> args)
long queryForLong(String sql, Object... args)
long queryForLong(String sql, SqlParameterSource args)
Map<String,Object> queryForMap(String sql, Map<String,?> args)
Map<String,Object> queryForMap(String sql, Object... args)
Map<String,Object> queryForMap(String sql, SqlParameterSource args)
<T> T queryForObject(String sql, Class<T> requiredType, Map<String,?> args)
<T> T queryForObject(String sql, Class<T> requiredType, Object... args)
<T> T queryForObject(String sql, Class<T> requiredType, SqlParameterSource args)
<T> T queryForObject(String sql, RowMapper<T> rm, Map<String,?> args)
<T> T queryForObject(String sql, RowMapper<T> rm, Object... args)
<T> T queryForObject(String sql, RowMapper<T> rm, SqlParameterSource args)
```


SimpleJdbcTemplate has quite a few query methods, and its wrapped JdbcTemplate defines even more (recall that you can retrieve the wrapped JdbcTemplate via getJdbcOperations()).

Many of these query methods use Java 5 generics and varargs so that using them is relatively easy.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Querying Row Count

- A common DAO operation is to return the count of rows in a table

```
1    @Repository
2    public class StudentDAOJdbcImpl implements StudentDAO
3    {
4        // initialized in setDataSource()
5        private JdbcTemplate template;
6
7    . . .
8
9      public int getStudentCount()
10     {
11       return template.queryForObject(
12          "SELECT COUNT(*) FROM student", Integer.class);
13     }
14   }
```

Note how we provide the actual SQL for the query, but we don't have to connect to the database, create any statements or handle exceptions (if we don't want to). Spring JDBC handles all of that for us.

NOTE: In Spring prior to 3.2.2, we used queryForInt(), but that is now deprecated.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mapping Rows to Objects

- Spring defines **row mappers**, which are interfaces you can implement so a template can convert a result-set row to an object

```
1    private class StudentRowMapper implements
2      RowMapper<Student>  {
3      public Student mapRow(ResultSet rs, int rowNum)
4       throws SQLException  {
5          Student s = new Student();
6          s.setGpa(rs.getDouble("gpa"));
7          s.setName(rs.getString("name"));
8          s.setStudentID(rs.getInt("studentID"));
9          return s;
10      }
11   }
```

Several of the query methods shown on the last page accept a row-mapper object - the template calls the row-mapper after a query to convert the result(s) to object(s).

Most programmers implement these row-mapper classes as Java "inner classes" within the DAO.

NOTE: Prior to Spring 3.2.2 or later, we used ParameterizedRowMapper instead of RowMapper, which has the same syntax and is functionally equivalent. ParameterizedRowMapper is now deprecated.

In Java 8 or later, instead of writing a separate row mapper class containing only a single method, you could instead use a lambda expression.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Query All

```
1    public class StudentDAOJdbcImpl
2      implements StudentDAO
3    {
4        // initialized in setDataSource()
5        private JdbcTemplate template;
6    . . .
7        private class StudentRowMapper
8           implements RowMapper<Student>
9        { . . . // shown earlier }
10
11       public Collection<Student> findAllStudents()
12       {
13           List<Student> students = template
14              .query("SELECT * FROM student", mapper);
15           return students;
16       }
17   }
```

Here we are using the query method:

```
<T> List<T> query(String sql, RowMapper<T> rm, Object... args)
```

Note that we are passing zero arguments for the last Object... parameter.

The equivalent Java 8 lambda might look like:

```
List<Student> students = jdbcTemplate.query(
  "SELECT * FROM student", (rs, rowNum) ->
   {
      Student s = new Student();
      s.setGpa(rs.getDouble("gpa"));
      s.setName(rs.getString("name"));
      s.setStudentID(rs.getInt("studentID"));
      return s;
   });
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Find Student By ID

```
1    public class StudentDAOJdbcImpl
2      implements StudentDAO
3    {
4
5    . . .
6
7        public Student findStudentByID(int studentID)
8        {
9            return template.queryForObject(
10             "SELECT * FROM student WHERE studentID=?",
11              mapper, studentID);
12       }
13   }
```


Here we are using the query method:

```
<T> List<T> query(String sql, RowMapper<T> rm, Object... args)
```

Note that we are passing a single integer argument for the last Object... parameter. The template uses that integer to supply the value for the '?' in the SQL. If the SQL had more than one '?', we could simply pass as many arguments to the "query()" as was needed.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JdbcTemplate Insert, Update and Delete

```
int update(String sql, Map<String,?> args)
int update(String sql, Object... args)
int update(String sql, SqlParameterSource args)

int[] batchUpdate(String sql, List<Object[]> batchArgs)
int[] batchUpdate(String sql, List<Object[]> batchArgs, int[] argTypes)
int[] batchUpdate(String sql, Map<String,?>[] batchValues)
int[] batchUpdate(String sql, SqlParameterSource[] batchArgs)
```


Note that the non-batch versions of these return an integer, which is the count of rows affected. The batch versions return an array of integers, which are the count of rows affected for each of the batch operations.

According to the Spring docs:

Most JDBC drivers provide improved performance if you batch multiple calls to the same prepared statement. By grouping updates into batches you limit the number of round trips to the database.

There is an example of batching in the Spring docs at:

http://static.springsource.org/spring/docs/2.5.6/reference/jdbc.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example: Insert

```
1    public class StudentDAOJdbcImpl
2      implements StudentDAO
3    {
4        // initialized in setDataSource()
5        private JdbcTemplate template;
6
7    . . .
8
9        public int insertStudent(Student s)
10       {
11         return template.update(
12           "INSERT INTO student
13           (studentId,name,gpa) VALUES(?,?,?)",
14           s.getStudentID(), s.getName(), s.getGpa());
15       }
16   }
```


Here are using the update method of the form:

```
int update(String sql, Object... args)
```

We supply three values for the Object... varargs parameter that correspond to the three '?' in the SQL. Note that we are expecting the caller to provide the primary key (studentID).

The update() method returns the count of rows affected. In this case, we'd expect the count to be one, since we inserted a single row.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete DAO Example

Student.java

StudentDAO.java

StudentDAOJdbcImpl.java

DisplayStudents.java

FindStudent.java

InsertStudent.java

spring.xml

web.xml

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package university;
2
3    public class Student
4    {
5        private int studentID;
6        private String name;
7        private double gpa;
8
9        public double getGpa()
10       {
11           return gpa;
12       }
13
14       public void setGpa(double gpa)
15       {
16           this.gpa = gpa;
17       }
18
19       public String getName()
20       {
21           return name;
22       }
23
24       public void setName(String name)
25       {
26           this.name = name;
27       }
28
29       public int getStudentID()
30       {
31           return studentID;
32       }
33
34       public void setStudentID(int studentID)
35       {
36           this.studentID = studentID;
37       }
38   }
```

```
1    package dao;
2
3    import java.util.Collection;
4
5    import university.Student;
6
7    public interface StudentDAO
8    {
9        public int getStudentCount();
10       public Collection<Student> findAllStudents();
11       public Student findStudentByID(int studentID);
12       public int insertStudent(Student s);
13       public int insertStudentReturnId(Student s);
14   }
```

```
1    package dao;
2
3    import java.sql.ResultSet;
4    import java.sql.SQLException;
5    import java.util.Collection;
6    import java.util.HashMap;
7    import java.util.List;
8    import java.util.Map;
9
10   import javax.sql.DataSource;
11
12   import org.springframework.beans.factory.annotation.Autowired;
13   import org.springframework.jdbc.core.JdbcTemplate;
14   import org.springframework.jdbc.core.RowMapper;
15   import org.springframework.jdbc.core.simple.SimpleJdbcInsert;
16   import org.springframework.stereotype.Repository;
17
18   import university.Student;
19
20   @Repository("studentDAO")
21   public class StudentDAOJdbcImpl implements StudentDAO
22   {
23       private DataSource studentDataSource;
24       private JdbcTemplate template;
25       private StudentRowMapper mapper;
26
27       @Autowired
28       public void setDataSource(DataSource ds)
29       {
30           studentDataSource = ds;
31           template = new JdbcTemplate(ds);
32           mapper = new StudentRowMapper();
33       }
34
35       private class StudentRowMapper implements RowMapper<Student>
36       {
37           public Student mapRow(ResultSet rs, int rowNum) throws SQLException
38           {
39               Student s = new Student();
40               s.setGpa(rs.getDouble("gpa"));
41               s.setName(rs.getString("name"));
42               s.setStudentID(rs.getInt("studentID"));
43               return s;
44           }
45       }
```

```
46
47      public int getStudentCount()
48      {
49          return template.queryForObject("SELECT COUNT(*) FROM student",
50                  Integer.class);
51      }
52
53      public Collection<Student> findAllStudents()
54      {
55          List<Student> students = template.query("SELECT * FROM student",
56                  mapper);
57          return students;
58      }
59
60      public Student findStudentByID(int studentID)
61      {
62          Student s = template.queryForObject(
63                  "SELECT * FROM student WHERE studentID=?",
64                  mapper, studentID);
65          return s;
66      }
67
68      public int insertStudent(Student s)
69      {
70          return template.update(
71                 "INSERT INTO student (name,gpa) VALUES(?,?)",
72                  s.getName(), s.getGpa());
73      }
74
75      // This insertStudent version uses the SimpleJdbcInsert
76      // class to retrieve an autogenerated key
77      public int insertStudentReturnId(Student s)
78      {
79          SimpleJdbcInsert inserter = new SimpleJdbcInsert(studentDataSource);
80          inserter.setTableName("student");
81          inserter.usingGeneratedKeyColumns("StudentID");
82          Map<String, Object> parms = new HashMap<String, Object>();
83          parms.put("name", s.getName());
84          parms.put("gpa", s.getGpa());
85
86          Number id = inserter.executeAndReturnKey(parms);
87          return id.intValue();
88      }
89  }
```

```
1    package servlets;
2
3    import java.io.IOException;
4    import java.io.PrintWriter;
5    import java.util.Collection;
6
7    import javax.servlet.ServletContext;
8    import javax.servlet.ServletException;
9    import javax.servlet.annotation.WebServlet;
10   import javax.servlet.http.HttpServletRequest;
11   import javax.servlet.http.HttpServletResponse;
12
13   import org.springframework.web.context.WebApplicationContext;
14   import org.springframework.web.context.support.WebApplicationContextUtils;
15
16   import university.Student;
17   import dao.StudentDAO;
18
19
20   @SuppressWarnings("serial")
21   @WebServlet("/displayStudents")
22   public class DisplayStudents extends javax.servlet.http.HttpServlet
23   {
24       @Override
25       protected void doGet(HttpServletRequest request, HttpServletResponse response)
26       {
27           response.setContentType("text/html");
28           PrintWriter out = response.getWriter();
29
30           ServletContext servletContext = getServletContext();
31           WebApplicationContext ctx =
32               WebApplicationContextUtils.getRequiredWebApplicationContext(
33                   servletContext);
34
35           try
36           {
37               StudentDAO dao = (StudentDAO)ctx.getBean("studentDAO");
38               out.println("There are " + dao.getStudentCount() + " students");
39
40               Collection<Student> students = dao.findAllStudents();
41               out.println("<table border='1'>");
42               for(Student s : students)
43               {
44                   out.println("<tr>");
45                   out.println("<td>" + s.getStudentID() + "</td>");
```

```
46                  out.println("<td>" + s.getName() + "</td>");
47                  out.println("<td>" + s.getGpa() + "</td>");
48                  out.println("</tr>");
49              }
50              out.println("</table>");
51          }
52          catch (Exception e)
53          {
54              out.println("Something went wrong: " + e);
55          }
56
57  //        Connection con = null;
58  //        try
59  //        {
60  //            Context ctx = new InitialContext();
61  //            DataSource ds =
62  //                (DataSource)ctx.lookup("java:comp/env/studentDS");
63  //            con = ds.getConnection();
64  //            Statement st = con.createStatement();
65  //            ResultSet rs = st.executeQuery("SELECT COUNT(*) FROM student");
66  //            rs.next();
67  //            out.println("There are " + rs.getInt(1) + " students");
68  //        }
69  //        catch (NamingException e)
70  //        {
71  //            e.printStackTrace();
72  //        }
73  //        catch (SQLException e)
74  //        {
75  //            e.printStackTrace();
76  //        }
77  //        finally
78  //        {
79  //            try
80  //            {
81  //               if (con != null) con.close();
82  //            } catch (Exception e1) {}
83  //        }
84
85
86      }
87  }
```

```
1    package servlets;
2
3    import java.io.IOException;
4    import java.io.PrintWriter;
5
6    import javax.servlet.ServletContext;
7    import javax.servlet.ServletException;
8    import javax.servlet.annotation.WebServlet;
9    import javax.servlet.http.HttpServletRequest;
10   import javax.servlet.http.HttpServletResponse;
11
12   import org.springframework.dao.EmptyResultDataAccessException;
13   import org.springframework.web.context.WebApplicationContext;
14   import org.springframework.web.context.support.WebApplicationContextUtils;
15
16   import university.Student;
17   import dao.StudentDAO;
18
19   @SuppressWarnings("serial")
20   @WebServlet("/findStudent")
21   public class FindStudent extends javax.servlet.http.HttpServlet
22   {
23       protected void doGet(HttpServletRequest request,
24               HttpServletResponse response) throws ServletException, IOException
25       {
26           response.setContentType("text/html");
27           PrintWriter out = response.getWriter();
28
29           ServletContext servletContext = getServletContext();
30           WebApplicationContext ctx = WebApplicationContextUtils
31                   .getRequiredWebApplicationContext(servletContext);
32
33           try
34           {
35               int studentID = Integer.parseInt(request.getParameter("studentID"));
36               StudentDAO dao = (StudentDAO) ctx.getBean("studentDAO");
37               Student s = dao.findStudentByID(studentID);
38               if (s != null)
39               {
40                   out.println("<p>Student ID: " + s.getStudentID() + "</p>");
41                   out.println("<p>Name: " + s.getName() + "</p>");
42                   out.println("<p>GPA: " + s.getGpa() + "</p>");
43               }
44           }
45           catch (NumberFormatException e)
```

```
46          {
47              out.println("Invalid number entered. Press 'Back' and try again.");
48          }
49          catch (EmptyResultDataAccessException e)
50          {
51              out.println("Student not found");
52          }
53          catch (Exception e)
54          {
55              out.println("Something went wrong: " + e);
56          }
57      }
58  }
```

```
1    package servlets;
2
3    import java.io.IOException;
4    import java.io.PrintWriter;
5
6    import javax.servlet.ServletContext;
7    import javax.servlet.ServletException;
8    import javax.servlet.annotation.WebServlet;
9    import javax.servlet.http.HttpServletRequest;
10   import javax.servlet.http.HttpServletResponse;
11
12   import org.springframework.web.context.WebApplicationContext;
13   import org.springframework.web.context.support.WebApplicationContextUtils;
14
15   import university.Student;
16   import dao.StudentDAO;
17
18   @SuppressWarnings("serial")
19   @WebServlet("/insertStudent")
20   public class InsertStudent extends javax.servlet.http.HttpServlet
21   {
22       protected void doPost(HttpServletRequest request,
23               HttpServletResponse response) throws ServletException, IOException
24       {
25           response.setContentType("text/html");
26           PrintWriter out = response.getWriter();
27
28           ServletContext servletContext = getServletContext();
29           WebApplicationContext ctx = WebApplicationContextUtils
30                   .getRequiredWebApplicationContext(servletContext);
31
32           String name = request.getParameter("name");
33           String s = request.getParameter("gpa");
34
35           double gpa = Double.parseDouble(s);
36
37           try
38           {
39               StudentDAO dao = (StudentDAO) ctx.getBean("studentDAO");
40
41               Student student = new Student();
42               student.setGpa(gpa);
43               student.setName(name);
44               int rows = dao.insertStudent(student);
45               out.println("Insert successful, " + rows + " inserted.");
```

```
46            }
47            catch (Exception e)
48            {
49                out.println("Something went wrong: " + e);
50            }
51        }
52    }
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:lang="http://www.springframework.org/schema/lang"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-4.1.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.1.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-4.1.xsd
        http://www.springframework.org/schema/lang
        http://www.springframework.org/schema/lang/spring-lang-4.1.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-4.1.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-4.1.xsd">

    <context:component-scan base-package="dao" />

    <bean id="studentDataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
        <property name="url" value="jdbc:hsqldb:hsql://localhost/" />
        <property name="username" value="sa" />
        <property name="password" value="" />
    </bean>

</beans>
```

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.
3      <display-name>SpringJdbcTemplateWeb</display-name>
4        <context-param>
5            <param-name>contextConfigLocation</param-name>
6            <param-value>/WEB-INF/spring.xml</param-value>
7        </context-param>
8        <listener>
9            <display-name>ContextLoaderListener</display-name>
10           <listener-class>
11               org.springframework.web.context.ContextLoaderListener
12           </listener-class>
13       </listener>
14     <welcome-file-list>
15       <welcome-file>index.html</welcome-file>
16       <welcome-file>index.htm</welcome-file>
17       <welcome-file>index.jsp</welcome-file>
18       <welcome-file>default.html</welcome-file>
19       <welcome-file>default.htm</welcome-file>
20       <welcome-file>default.jsp</welcome-file>
21     </welcome-file-list>
22   </web-app>
```

16 - 21L : web.xml

Chapter Summary

In this chapter, you learned:

- The basics of Spring's JDBC support
- How to use SimpleJdbcTemplate and SimpleJdbcDaoSupport

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Web Services

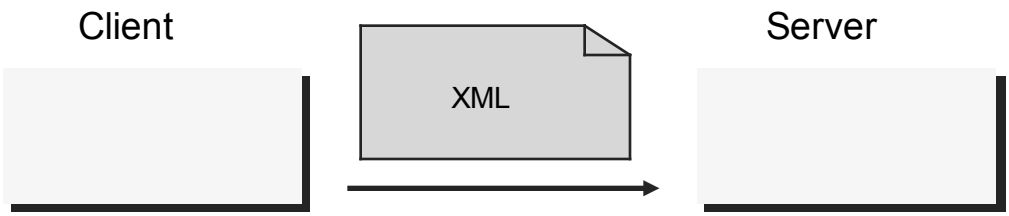
- What are Web Services?
- SOAP
- WSDL
- UDDI

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What are Web Services?

- Web services let you create **interoperable** distributed applications that use XML



Web services are an emerging technology that promote interoperability between dissimilar systems. Web services use XML, which is an industry standard to enable platform, operating system and programming language neutrality.

Most of the major software companies have thrown their weight behind web services, including IBM, Microsoft, Sun, BEA and Oracle.

In addition, web services are an integral part of initiatives such as Microsoft .NET and Sun ONE.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Benefits of Web Services

- Interoperability between heterogeneous systems
- Loosely-coupled applications that use standard tools, formats and protocols
- Wide industry support for runtimes and tools

As an example of the application use of the Internet, the Google search engine has a web services-based API that lets you issue web searches from a program rather than from a browser. See http://www.google.com/apis/ for details.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

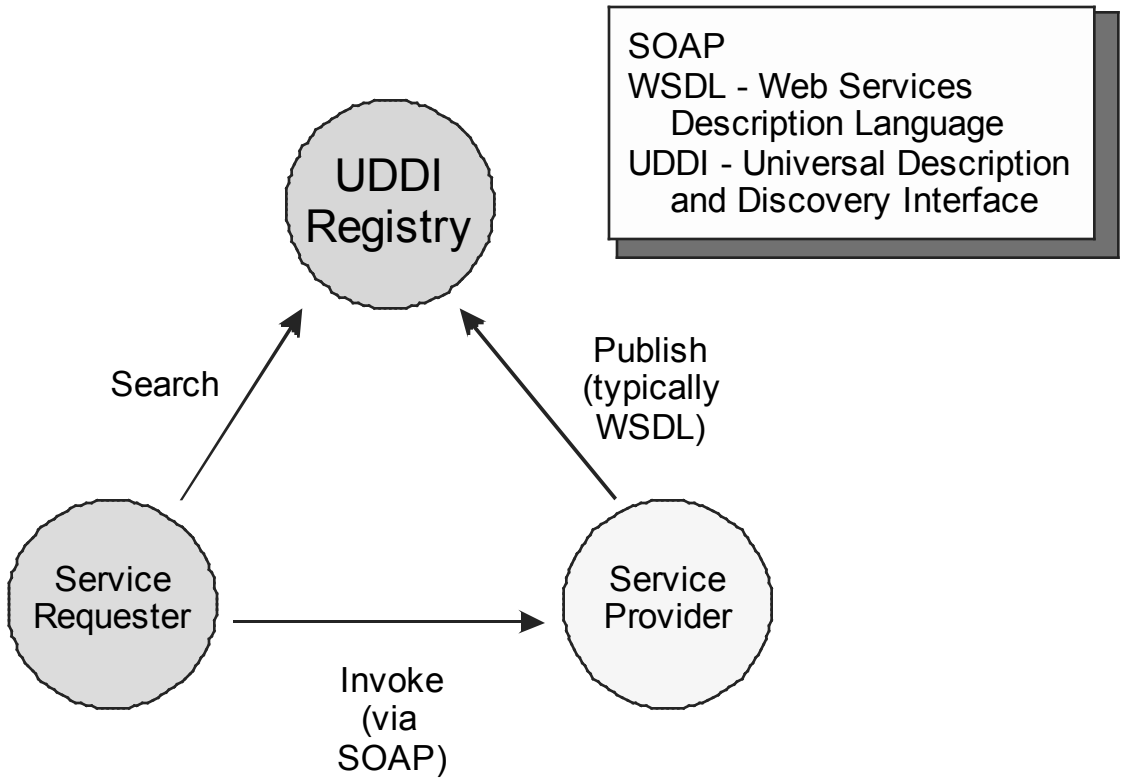
Web Services Challenges

- Interoperability can still be difficult
- Evolving standards (e.g. SOAP 1.1 to SOAP 1.2)
- Complexity due to many WS-* standards (e.g. WS-Security, WS-Addressing, WS-Policy, etc.)

Web services have been widely adopted, but issues remain.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Service Oriented Architectures



The triad of technologies, SOAP, WSDL and UDDI, provide the infrastructure for dynamic, loosely coupled applications that work across platforms, operating systems and programming languages.

The service provider exposes one or more web services, which the requester invokes via SOAP messages. To make it easier to find the services, the provider can create XML WSDL documents that describe the service and publish them to a registry. The requester can then retrieve the service descriptions from the registry and use the description to build the SOAP request.

Note that a web services application does not have to use all of the these - for example, if a service requester already knows everything there is to know about a service, it doesn't need either WSDL or UDDI.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is SOAP?

- SOAP is a low-level XML-based protocol for exchanging messages
- SOAP is transport-neutral, but many implementations use HTTP
- SOAP is administered by the World Wide Web Consortium (http://www.w3.org)

As of version 1.2, SOAP no longer means "Simple Object Access Protocol" - now it's just a term, not an acronym.

SOAP is related to XML-RPC, which is a lighter-weight protocol still in use. SOAP was originally developed by Dave Winer and Microsoft. See http://www.fawcette.com/xmlmag/2001_12/magazine/features/interview2/page3.asp for a history lesson.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example SOAP Message

- This example uses the SOAP 1.1 specification, which is currently in most wide use

```
1    <soapenv:Envelope
2       xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
3       xmlns:q0="http://ws.cdyne.com/WeatherWS/">
4      <soapenv:Body>
5        <q0:GetCityWeatherByZIP>
6          <q0:ZIP>52403</q0:ZIP>
7        </q0:GetCityWeatherByZIP>
8      </soapenv:Body>
9    </soapenv:Envelope>
```

This SOAP message invokes a web service that reports the weather at the specified ZIP code . The web service endpoint is: http://ws.cdyne.com/WeatherWS/Weather.asmx.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Example RPC Response

- This example shows the SOAP 1.1 response to the **GetCityWeatherByZIP** message

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetCityWeatherByZIPResponse xmlns="http://ws.cdyne.com/WeatherWS/">
      <GetCityWeatherByZIPResult>
        <Success>true</Success>
        <ResponseText>City Found</ResponseText>
        <State>IA</State>
        <City>Cedar Rapids</City>
        <WeatherStationCity>Cedar Rapids</WeatherStationCity>
        <WeatherID>9</WeatherID>
        <Description>Partly Sunny</Description>
        <Temperature>1</Temperature>
        <RelativeHumidity>85</RelativeHumidity>
        <Wind>NW15</Wind>
        <Pressure>30.39F</Pressure>
      </GetCityWeatherByZIPResult>
    </GetCityWeatherByZIPResponse>
  </soap:Body>
</soap:Envelope>
```


This listing shows the SOAP response for the GetCityWeatherByZIP message shown earlier.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is WSDL?

- The Web Services Description Language is an XML vocabulary for describing the interface to a web service
- WSDL is currently at version 2, but WSDL 1.1 is the most widely used version
- Like SOAP, WSDL is administered by the W3C
- The WSDL for a web service should contain enough detail so a client can issue a request to the web service

weather.wsdl

WSDL is designed so that it contains enough information so that a client program can issue a call to the web service.

There are several ways to "publish" WSDL so that it's available to clients. First, you could put it on your website or email it to the client programmers. Second, there's an emerging convention where providers put the WSDL file at the same URL as the web service endpoint, with "?WSDL" appended (e.g. http://mycompany.com/myservice?WSDL". Finally, you could publish the WSDL to a UDDI registry.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<wsdl:definitions xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://ws.cdyne.com/
      <s:element name="GetWeatherInformation">
        <s:complexType />
      </s:element>
      <s:element name="GetWeatherInformationResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetWeatherInformationResu
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfWeatherDescription">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="WeatherDescription"
        </s:sequence>
      </s:complexType>
      <s:complexType name="WeatherDescription">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="WeatherID" type="s:short" /
          <s:element minOccurs="0" maxOccurs="1" name="Description" type="s:string
          <s:element minOccurs="0" maxOccurs="1" name="PictureURL" type="s:string"
        </s:sequence>
      </s:complexType>
      <s:element name="GetCityForecastByZIP">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="ZIP" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetCityForecastByZIPResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetCityForecastByZIPResul
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ForecastReturn">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="Success" type="s:boolean" /
          <s:element minOccurs="0" maxOccurs="1" name="ResponseText" type="s:strin
          <s:element minOccurs="0" maxOccurs="1" name="State" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="City" type="s:string" />
```

```
46            <s:element minOccurs="0" maxOccurs="1" name="WeatherStationCity" type="s
47            <s:element minOccurs="0" maxOccurs="1" name="ForecastResult" type="tns:A
48          </s:sequence>
49        </s:complexType>
50        <s:complexType name="ArrayOfForecast">
51          <s:sequence>
52            <s:element minOccurs="0" maxOccurs="unbounded" name="Forecast" nillable=
53          </s:sequence>
54        </s:complexType>
55        <s:complexType name="Forecast">
56          <s:sequence>
57            <s:element minOccurs="1" maxOccurs="1" name="Date" type="s:dateTime" />
58            <s:element minOccurs="1" maxOccurs="1" name="WeatherID" type="s:short" /
59            <s:element minOccurs="0" maxOccurs="1" name="Desciption" type="s:string"
60            <s:element minOccurs="1" maxOccurs="1" name="Temperatures" type="tns:tem
61            <s:element minOccurs="1" maxOccurs="1" name="ProbabilityOfPrecipiation"
62          </s:sequence>
63        </s:complexType>
64        <s:complexType name="temp">
65          <s:sequence>
66            <s:element minOccurs="0" maxOccurs="1" name="MorningLow" type="s:string"
67            <s:element minOccurs="0" maxOccurs="1" name="DaytimeHigh" type="s:string
68          </s:sequence>
69        </s:complexType>
70        <s:complexType name="POP">
71          <s:sequence>
72            <s:element minOccurs="0" maxOccurs="1" name="Nighttime" type="s:string"
73            <s:element minOccurs="0" maxOccurs="1" name="Daytime" type="s:string" />
74          </s:sequence>
75        </s:complexType>
76        <s:element name="GetCityWeatherByZIP">
77          <s:complexType>
78            <s:sequence>
79              <s:element minOccurs="0" maxOccurs="1" name="ZIP" type="s:string" />
80            </s:sequence>
81          </s:complexType>
82        </s:element>
83        <s:element name="GetCityWeatherByZIPResponse">
84          <s:complexType>
85            <s:sequence>
86              <s:element minOccurs="1" maxOccurs="1" name="GetCityWeatherByZIPResult
87            </s:sequence>
88          </s:complexType>
89        </s:element>
90        <s:complexType name="WeatherReturn">
```

```
91          <s:sequence>
92            <s:element minOccurs="1" maxOccurs="1" name="Success" type="s:boolean" /
93            <s:element minOccurs="0" maxOccurs="1" name="ResponseText" type="s:strin
94            <s:element minOccurs="0" maxOccurs="1" name="State" type="s:string" />
95            <s:element minOccurs="0" maxOccurs="1" name="City" type="s:string" />
96            <s:element minOccurs="0" maxOccurs="1" name="WeatherStationCity" type="s
97            <s:element minOccurs="1" maxOccurs="1" name="WeatherID" type="s:short" /
98            <s:element minOccurs="0" maxOccurs="1" name="Description" type="s:string
99            <s:element minOccurs="0" maxOccurs="1" name="Temperature" type="s:string
100           <s:element minOccurs="0" maxOccurs="1" name="RelativeHumidity" type="s:s
101           <s:element minOccurs="0" maxOccurs="1" name="Wind" type="s:string" />
102           <s:element minOccurs="0" maxOccurs="1" name="Pressure" type="s:string" /
103           <s:element minOccurs="0" maxOccurs="1" name="Visibility" type="s:string"
104           <s:element minOccurs="0" maxOccurs="1" name="WindChill" type="s:string"
105           <s:element minOccurs="0" maxOccurs="1" name="Remarks" type="s:string" />
106         </s:sequence>
107       </s:complexType>
108       <s:element name="ArrayOfWeatherDescription" nillable="true" type="tns:ArrayO
109       <s:element name="ForecastReturn" nillable="true" type="tns:ForecastReturn" /
110       <s:element name="WeatherReturn" type="tns:WeatherReturn" />
111     </s:schema>
112   </wsdl:types>
113   <wsdl:message name="GetWeatherInformationSoapIn">
114     <wsdl:part name="parameters" element="tns:GetWeatherInformation" />
115   </wsdl:message>
116   <wsdl:message name="GetWeatherInformationSoapOut">
117     <wsdl:part name="parameters" element="tns:GetWeatherInformationResponse" />
118   </wsdl:message>
119   <wsdl:message name="GetCityForecastByZIPSoapIn">
120     <wsdl:part name="parameters" element="tns:GetCityForecastByZIP" />
121   </wsdl:message>
122   <wsdl:message name="GetCityForecastByZIPSoapOut">
123     <wsdl:part name="parameters" element="tns:GetCityForecastByZIPResponse" />
124   </wsdl:message>
125   <wsdl:message name="GetCityWeatherByZIPSoapIn">
126     <wsdl:part name="parameters" element="tns:GetCityWeatherByZIP" />
127   </wsdl:message>
128   <wsdl:message name="GetCityWeatherByZIPSoapOut">
129     <wsdl:part name="parameters" element="tns:GetCityWeatherByZIPResponse" />
130   </wsdl:message>
131   <wsdl:message name="GetWeatherInformationHttpGetIn" />
132   <wsdl:message name="GetWeatherInformationHttpGetOut">
133     <wsdl:part name="Body" element="tns:ArrayOfWeatherDescription" />
134   </wsdl:message>
135   <wsdl:message name="GetCityForecastByZIPHttpGetIn">
```

```
136       <wsdl:part name="ZIP" type="s:string" />
137     </wsdl:message>
138     <wsdl:message name="GetCityForecastByZIPHttpGetOut">
139       <wsdl:part name="Body" element="tns:ForecastReturn" />
140     </wsdl:message>
141     <wsdl:message name="GetCityWeatherByZIPHttpGetIn">
142       <wsdl:part name="ZIP" type="s:string" />
143     </wsdl:message>
144     <wsdl:message name="GetCityWeatherByZIPHttpGetOut">
145       <wsdl:part name="Body" element="tns:WeatherReturn" />
146     </wsdl:message>
147     <wsdl:message name="GetWeatherInformationHttpPostIn" />
148     <wsdl:message name="GetWeatherInformationHttpPostOut">
149       <wsdl:part name="Body" element="tns:ArrayOfWeatherDescription" />
150     </wsdl:message>
151     <wsdl:message name="GetCityForecastByZIPHttpPostIn">
152       <wsdl:part name="ZIP" type="s:string" />
153     </wsdl:message>
154     <wsdl:message name="GetCityForecastByZIPHttpPostOut">
155       <wsdl:part name="Body" element="tns:ForecastReturn" />
156     </wsdl:message>
157     <wsdl:message name="GetCityWeatherByZIPHttpPostIn">
158       <wsdl:part name="ZIP" type="s:string" />
159     </wsdl:message>
160     <wsdl:message name="GetCityWeatherByZIPHttpPostOut">
161       <wsdl:part name="Body" element="tns:WeatherReturn" />
162     </wsdl:message>
163     <wsdl:portType name="WeatherSoap">
164       <wsdl:operation name="GetWeatherInformation">
165         <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Gets Infor
166         <wsdl:input message="tns:GetWeatherInformationSoapIn" />
167         <wsdl:output message="tns:GetWeatherInformationSoapOut" />
168       </wsdl:operation>
169       <wsdl:operation name="GetCityForecastByZIP">
170         <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you
171         <wsdl:input message="tns:GetCityForecastByZIPSoapIn" />
172         <wsdl:output message="tns:GetCityForecastByZIPSoapOut" />
173       </wsdl:operation>
174       <wsdl:operation name="GetCityWeatherByZIP">
175         <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you
176         <wsdl:input message="tns:GetCityWeatherByZIPSoapIn" />
177         <wsdl:output message="tns:GetCityWeatherByZIPSoapOut" />
178       </wsdl:operation>
179     </wsdl:portType>
180     <wsdl:portType name="WeatherHttpGet">
```

```
181     <wsdl:operation name="GetWeatherInformation">
182       <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Gets Infor
183       <wsdl:input message="tns:GetWeatherInformationHttpGetIn" />
184       <wsdl:output message="tns:GetWeatherInformationHttpGetOut" />
185     </wsdl:operation>
186     <wsdl:operation name="GetCityForecastByZIP">
187       <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you
188       <wsdl:input message="tns:GetCityForecastByZIPHttpGetIn" />
189       <wsdl:output message="tns:GetCityForecastByZIPHttpGetOut" />
190     </wsdl:operation>
191     <wsdl:operation name="GetCityWeatherByZIP">
192       <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you
193       <wsdl:input message="tns:GetCityWeatherByZIPHttpGetIn" />
194       <wsdl:output message="tns:GetCityWeatherByZIPHttpGetOut" />
195     </wsdl:operation>
196   </wsdl:portType>
197   <wsdl:portType name="WeatherHttpPost">
198     <wsdl:operation name="GetWeatherInformation">
199       <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Gets Infor
200       <wsdl:input message="tns:GetWeatherInformationHttpPostIn" />
201       <wsdl:output message="tns:GetWeatherInformationHttpPostOut" />
202     </wsdl:operation>
203     <wsdl:operation name="GetCityForecastByZIP">
204       <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you
205       <wsdl:input message="tns:GetCityForecastByZIPHttpPostIn" />
206       <wsdl:output message="tns:GetCityForecastByZIPHttpPostOut" />
207     </wsdl:operation>
208     <wsdl:operation name="GetCityWeatherByZIP">
209       <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Allows you
210       <wsdl:input message="tns:GetCityWeatherByZIPHttpPostIn" />
211       <wsdl:output message="tns:GetCityWeatherByZIPHttpPostOut" />
212     </wsdl:operation>
213   </wsdl:portType>
214   <wsdl:binding name="WeatherSoap" type="tns:WeatherSoap">
215     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
216     <wsdl:operation name="GetWeatherInformation">
217       <soap:operation soapAction="http://ws.cdyne.com/WeatherWS/GetWeatherInformat
218       <wsdl:input>
219         <soap:body use="literal" />
220       </wsdl:input>
221       <wsdl:output>
222         <soap:body use="literal" />
223       </wsdl:output>
224     </wsdl:operation>
225     <wsdl:operation name="GetCityForecastByZIP">
```

```
226       <soap:operation soapAction="http://ws.cdyne.com/WeatherWS/GetCityForecastByZ
227       <wsdl:input>
228         <soap:body use="literal" />
229       </wsdl:input>
230       <wsdl:output>
231         <soap:body use="literal" />
232       </wsdl:output>
233     </wsdl:operation>
234     <wsdl:operation name="GetCityWeatherByZIP">
235       <soap:operation soapAction="http://ws.cdyne.com/WeatherWS/GetCityWeatherByZI
236       <wsdl:input>
237         <soap:body use="literal" />
238       </wsdl:input>
239       <wsdl:output>
240         <soap:body use="literal" />
241       </wsdl:output>
242     </wsdl:operation>
243   </wsdl:binding>
244   <wsdl:binding name="WeatherSoap12" type="tns:WeatherSoap">
245     <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
246     <wsdl:operation name="GetWeatherInformation">
247       <soap12:operation soapAction="http://ws.cdyne.com/WeatherWS/GetWeatherInform
248       <wsdl:input>
249         <soap12:body use="literal" />
250       </wsdl:input>
251       <wsdl:output>
252         <soap12:body use="literal" />
253       </wsdl:output>
254     </wsdl:operation>
255     <wsdl:operation name="GetCityForecastByZIP">
256       <soap12:operation soapAction="http://ws.cdyne.com/WeatherWS/GetCityForecastB
257       <wsdl:input>
258         <soap12:body use="literal" />
259       </wsdl:input>
260       <wsdl:output>
261         <soap12:body use="literal" />
262       </wsdl:output>
263     </wsdl:operation>
264     <wsdl:operation name="GetCityWeatherByZIP">
265       <soap12:operation soapAction="http://ws.cdyne.com/WeatherWS/GetCityWeatherBy
266       <wsdl:input>
267         <soap12:body use="literal" />
268       </wsdl:input>
269       <wsdl:output>
270         <soap12:body use="literal" />
```

```
271       </wsdl:output>
272     </wsdl:operation>
273   </wsdl:binding>
274   <wsdl:binding name="WeatherHttpGet" type="tns:WeatherHttpGet">
275     <http:binding verb="GET" />
276     <wsdl:operation name="GetWeatherInformation">
277       <http:operation location="/GetWeatherInformation" />
278       <wsdl:input>
279         <http:urlEncoded />
280       </wsdl:input>
281       <wsdl:output>
282         <mime:mimeXml part="Body" />
283       </wsdl:output>
284     </wsdl:operation>
285     <wsdl:operation name="GetCityForecastByZIP">
286       <http:operation location="/GetCityForecastByZIP" />
287       <wsdl:input>
288         <http:urlEncoded />
289       </wsdl:input>
290       <wsdl:output>
291         <mime:mimeXml part="Body" />
292       </wsdl:output>
293     </wsdl:operation>
294     <wsdl:operation name="GetCityWeatherByZIP">
295       <http:operation location="/GetCityWeatherByZIP" />
296       <wsdl:input>
297         <http:urlEncoded />
298       </wsdl:input>
299       <wsdl:output>
300         <mime:mimeXml part="Body" />
301       </wsdl:output>
302     </wsdl:operation>
303   </wsdl:binding>
304   <wsdl:binding name="WeatherHttpPost" type="tns:WeatherHttpPost">
305     <http:binding verb="POST" />
306     <wsdl:operation name="GetWeatherInformation">
307       <http:operation location="/GetWeatherInformation" />
308       <wsdl:input>
309         <mime:content type="application/x-www-form-urlencoded" />
310       </wsdl:input>
311       <wsdl:output>
312         <mime:mimeXml part="Body" />
313       </wsdl:output>
314     </wsdl:operation>
315     <wsdl:operation name="GetCityForecastByZIP">
```

```
316       <http:operation location="/GetCityForecastByZIP" />
317       <wsdl:input>
318         <mime:content type="application/x-www-form-urlencoded" />
319       </wsdl:input>
320       <wsdl:output>
321         <mime:mimeXml part="Body" />
322       </wsdl:output>
323     </wsdl:operation>
324     <wsdl:operation name="GetCityWeatherByZIP">
325       <http:operation location="/GetCityWeatherByZIP" />
326       <wsdl:input>
327         <mime:content type="application/x-www-form-urlencoded" />
328       </wsdl:input>
329       <wsdl:output>
330         <mime:mimeXml part="Body" />
331       </wsdl:output>
332     </wsdl:operation>
333   </wsdl:binding>
334   <wsdl:service name="Weather">
335     <wsdl:port name="WeatherSoap" binding="tns:WeatherSoap">
336       <soap:address location="http://ws.cdyne.com/WeatherWS/Weather.asmx" />
337     </wsdl:port>
338     <wsdl:port name="WeatherSoap12" binding="tns:WeatherSoap12">
339       <soap12:address location="http://ws.cdyne.com/WeatherWS/Weather.asmx" />
340     </wsdl:port>
341     <wsdl:port name="WeatherHttpGet" binding="tns:WeatherHttpGet">
342       <http:address location="http://ws.cdyne.com/WeatherWS/Weather.asmx" />
343     </wsdl:port>
344     <wsdl:port name="WeatherHttpPost" binding="tns:WeatherHttpPost">
345       <http:address location="http://ws.cdyne.com/WeatherWS/Weather.asmx" />
346     </wsdl:port>
347   </wsdl:service>
348 </wsdl:definitions>
```

What is UDDI?

- The Universal Description and Discovery Interface defines a registry so that web service providers can publish their services and web service consumers can locate the services
- UDDI registries come in two flavors: the public UDDI Business Registry (UBR) and private registries
- UDDI is administered by the OASIS standards body

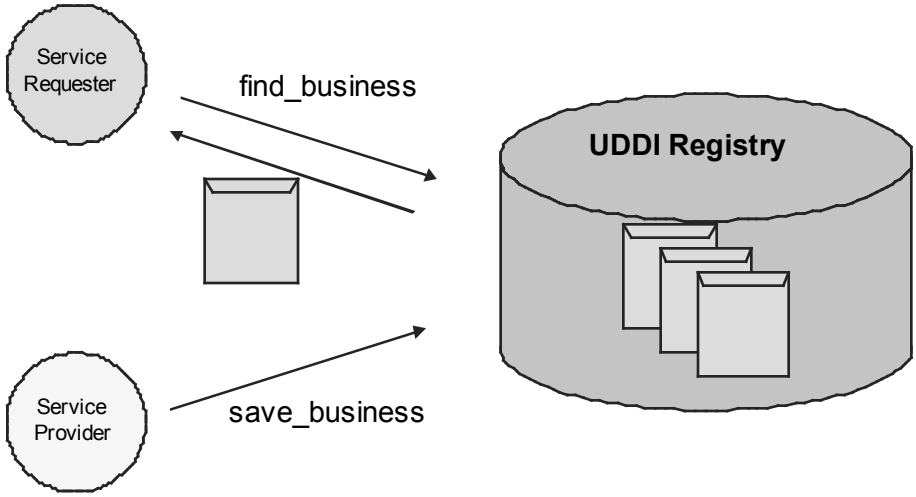
UDDI descends from proprietary IBM and Microsoft specifications and is now an industry standard administered by OASIS (www.uddi.org).

The UDDI Business Registry is administered by IBM, Microsoft, SAP and NTT Communications. Additions to any of the sites maintained by each of these partners is replicated to all other UDDI servers. UDDI is currently at version 3.0, but most implementations still are at version 2.0.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using UDDI

- UDDI defines a set of web services that let providers publish their service and consumers search for services
- It's possible (and likely) that WSDL is stored in the registry, but it's not required



The UDDI "find_business" API lets a service requester search for a business name stored in a registry - this is much like the white pages in a phone book. UDDI also lets requesters search by service category (yellow pages) and by service description (so-called green pages). Note that the public UBR sites provide an HTML front end to the public registry so humans can perform these searches.

The UDDI "save_business" API lets a service provider create or update business entry in the registry. Unlike the "find_business" API, this operation is restricted to authorized programs (using a login and password). There are additional APIs to let a business create entries for the services they can provide.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Java APIs for Web Services

Sun has defined several APIs so that Java programs can work with web services:

- JAX-RPC - XML Remote Procedure Calls
- JAX-WS - The successor to JAX-RPC in JEE Version 5
- SAAJ - SOAP API for Attachments in Java
- JAX-M - XML Messaging
- JAX-R - XML Registries
- JAX-RS - The Java API for RESTful Web Services

The APIs defined by Sun let Java programs perform SOAP RPC (JAX-RPC), SOAP document-oriented messaging (JAX-M) and access registries including UDDI (JAX-R). See http://java.sun.com/webservices/nutshell.html for more information on these APIs.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Toolkits for Web Services

There are numerous Web services programming toolkits for Java and other languages, including:

- Sun Glassfish (Java)
- Apache Axis (Java)
- IBM Rational Application Developer (RAD)
- BEA WebLogic Server and Workshop
- Spring Framework Web Services
- Microsoft Visual Studio.NET

Most of these toolkits not only provide support for creating web services, but also include tools to let you create/edit WSDL, browse UDDI and so forth. Most will also generate "starter" client programs given the WSDL for a web service.

Remember that a big goal of web services is interoperability, so you should pick your tool based on factors such as price, features, support and ease of use.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The basics of web services and service-oriented architectures
- About SOAP, WSDL and UDDI

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to REST

- What is Representational State Transfer?
- Principles of REST
- Introduction to Java REST frameworks

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is REST?

- **Representational State Transfer** is an architecture for distributed systems that emphasizes statelessness
- REST was introduced by Roy Fielding, who was one of the authors of HTTP

The World Wide Web itself exhibits many REST principles.

Developers that use the REST style are sometimes humorously referred to as "RESTafarians".

Fielding's REST dissertation is available at:

`http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.html`

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

REST vs SOAP

- REST provides a straightforward, simple architecture for providing highly scalable services (note that the Web itself is RESTful)
- SOAP provides a more complex but perhaps more enterprise-class architecture that includes additional standards such as WSDL, WS-Security and so forth

This is a highly contested argument -- both approaches have advantages.

Note that there is a research project from Sun for describing RESTful services in a similar fashion to WSDL: Web Application Description Language (WADL).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Principles of REST

- Every resource should have a unique ID
- Resources can have multiple representations
- Use links to connect resources
- Use standard HTTP methods
- Clients and servers communicate statelessly

These principles are adapted from an article by Stefan Tilkov, which can be found at:

http://www.infoq.com/articles/rest-introduction

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Resource IDs

- The Internet provides a resource-identification technique known as a **Uniform Resource Identifier** (URI), a.k.a URL
- RESTful applications use URIs to assign unique IDs to application resources
- Examples:
    - http://www.mydomain.com/students/5847
    - http://www.mydomain.com/students?id=44
    - http://www.mydomain.com/id=17

A URL is actually a type of URI, the other category being Uniform Resource Names (URN).

Note that the resources exposed by URIs are generally fairly abstract and don't necessarily exactly map to a database entry.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Resources With Multiple Representations

- For flexibility, applications that provide data can support different data formats, e.g. HTML, XML, CSV, JSON
- Clients can then request the resource in the format they like best, perhaps by specifying an HTTP **accept** header

```
//XML representation
<student>
  <ID>17</ID>
  <name>Sue Smith</student>
  <gpa>3.45</gpa>
</student>

//JSON representation
{"student":{"id":17, "name":"Sue Smith, "gpa":3.45}}
```

JSON is the JavaScript Object Notation. JSON is a lightweight data-interchange format that's easy for humans and programs to process.

You can find out more about JSON at:

http://www.json.org

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Use Links to Connect Resources

- HTML supports **hyperlinking** so that resources can reference other resources
- You can also perform linking in other data formats such as XML

```
<student xmlns:xlink="http://www.w3.org/1999/xlink">
  <ID>17</ID>
  <name>Sue Smith</student>
  <gpa>3.45</gpa>
  <advisor
    xlink:href="http://www.mydomain.com/advisors/156"/>
</student>
```


XML supports the Xlink standard for linking between documents.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Use Standard HTTP Methods

- HTTP supports **methods** that map to the four standard database Create, Retrieve, Update and Delete (CRUD) actions
- Note that only GET and POST can be sent directly from a browser so PUT, DELETE actions must be done programmatically (e.g. via AJAX)

HTTP Method	Database CRUD Operation
POST	CREATE
GET	SELECT (Retrieve)
PUT	UPDATE or CREATE
DELETE	DELETE

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Stateless Communication

- RESTful applications do **not** require the server to maintain communication state based on client requests
- This makes servers more scalable and allows the server to efficiently use techniques like caching, clustering and instance pooling

Most SOAP-based services are stateless, too.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

REST in Java

- JEE developers can use two basic approaches for writing RESTful services:
  - Write basic servlets
  - Use a REST framework
- Java RESTful frameworks include:
  - Restlets
  - JAX-WS REST
  - JAX-RS
  - Spring MVC REST

It's actually straightforward to write basic servlets that act as RESTful services, since servlets can override doGet(), doPost() and so forth. However, many developers prefer to use a framework that automates some of the low-level details.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Restlet?

- Restlet is a lightweight, open-source REST framework for Java
- Restlet applications can run within a servlet container or as standalone Java applications
- Restlet was created by Jerome Louvel in 2005

The homepage for the Restlet framework is:

`http://www.restlet.org`

Restlets also have been ported to the Google Web Toolkit (GWT).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Restlet Resource

```
1    public class HelloServerResource extends ServerResource
2    {
3        @Get
4        public String represent()
5        {
6            return "hello, world";
7        }
8    }
```


Imports not shown for brevity.

By default, Restlet provides a plain text representation.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JAX-WS and REST

- JAX-WS is the standard Web services API for Java Enterprise Edition 5 and later
- JAX-WS provides support for creating Web services with no SOAP envelope, i.e. RESTful services

```
1    @WebServiceProvider(serviceName = "HWService")
2    @BindingType(value = HTTPBinding.HTTP_BINDING)
3    public class MyProvider implements Provider<Source>
4    {
5      public Source invoke(Source source)
6      {
7         String replyElement =
8            new String("<p>hello world</p>");
9         StreamSource reply = new StreamSource(
10            new StringReader(replyElement));
11        return reply;
12     }
13   }
```


Some developers believe that the JAX-WS support for REST is not all it could be.

Here we show a JAX-WS endpoint that acts as a RESTful resource, returning the proverbial Hello, world string in HTML. Note the @BindingType annotation, which tells the JAX-WS runtime that this service uses HTTP binding rather than a SOAP envelope.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JAX-RS

- JAX-RS is an API for providing RESTful services in Java
- JAX-RS went through the Java Community Process as JSR-311 and is a part of JEE Version 6



The primary developer of Restlets was heavily involved in JSR-311 so there are similarities between the two frameworks.

If you don't have a JEE V6 container, you can use the reference implementation of JAX-RS, Jersey, which you can download and read about at:

`https://jersey.dev.java.net/`

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A JAX-RS Service

- To use JAX-RS, you create **resource** classes, using annotations to configure them

```
1    @Path("/helloworld")
2    public class HelloWorldResource
3    {
4        @GET
5        @Produces("text/plain")
6        public String getMessage(
7                @DefaultValue("")
8                @QueryParam("name") String name)
9        {
10           return "Hello World, " + name;
11       }
12   }
```


This service would be available at:

`http://mydomain.com/myWebApp/helloworld?name=bill`

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring MVC REST

- The popular Spring Framework's MVC module includes support for RESTful services

```
1    @Controller
2    @RequestMapping("rest")
3    public class StudentController
4    {
5      @RequestMapping(value = "/student/{id}",
6        method = RequestMethod.GET)
7        public @ResponseBody
8         Student getStudent(@PathVariable("id") int id)
9      {
10       Student s = findStudentByID(id);
11
12       return s;
13     }
14   }
18 - 16
```

This Spring class returns a JSON or XML representation of a Java "Student" object (source for Student not shown).

The service responds to URLs that have a student ID at the end of the URL:

`http:// . . . /student/12`

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About the fundamentals of representational state transfer
- The basics of Java REST frameworks

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JAX-RS

- Resource Classes
- Annotations
- Paths
- Context

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JAX-RS

- JAX-RS is an API for providing RESTful services in Java
- JAX-RS went through the Java Community Process as JSR-311 and is a part of JEE Version 6



If you don't have a JEE V6 container, you can use the reference implementation of JAX-RS, Jersey, which you can download and read about at:

`http://jersey.java.net`

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JAX-RS Annotations

```
@Path specifies a relative URI for a resource class or method
@ApplicationPath specifies a relative URI for application class
@GET, @PUT, @POST, @DELETE and @HEAD specify the HTTP method
@Produces specifies response Media Type(s)
@Consumes specifies accepted request Media Type(s)
@PathParam binds a parameter based on URI
@QueryParam binds a parameter based on HTTP query parameter
@MatrixParam binds a parameter based on HTTP matrix parameter
@HeaderParam binds a parameter based on HTTP header
@CookieParam binds a parameter based on cookie
@FormParam binds a parameter based on form value
@DefaultValue specifies a default value
@Context binds a field or parameter to an environmental value
```


All of these annotations are from the javax.ws.rs package.

We will see examples of many of these annotations in this chapter.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Resource Class

- A **resource class** defines methods that respond to RESTful GET, POST or PUT methods, decorated with JAX-RS annotations
- Resource classes must provide a public constructor that JAX-RS can use, either a zero-argument constructor or a constructor with arguments annotated with @Context

```
1    @Path("/student")
2    public class RosterService
3    {
4      @GET
5      @Produces("text/xml")
6      public String studentAsXml(
7       @QueryParam("studentID") String id)
8      {
9       . . .
10     }
11   }
```

According to the JAX-RS specification, implementations will instantiate a separate resource object for each request, at least by default for non-singleton objects.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing an Application Class

- An **application class** configures the resource classes (and providers) that make up a JAX-RS application
- Such classes extend **javax.ws.rs.core.Application** and typically override the **getClasses** method

```
1    @ApplicationPath("/roster/")
2    public class RosterApplication extends Application
3    {
4      public Set<Class<?>> getClasses()
5      {
6        Set<Class<?>> s = new HashSet<Class<?>>();
7        s.add(RosterService.class);
8        return s;
9      }
10   }
```

The JAX-RS specification notes that implementations can provide other ways of configuration, but the Application subclass is the only portable technique.

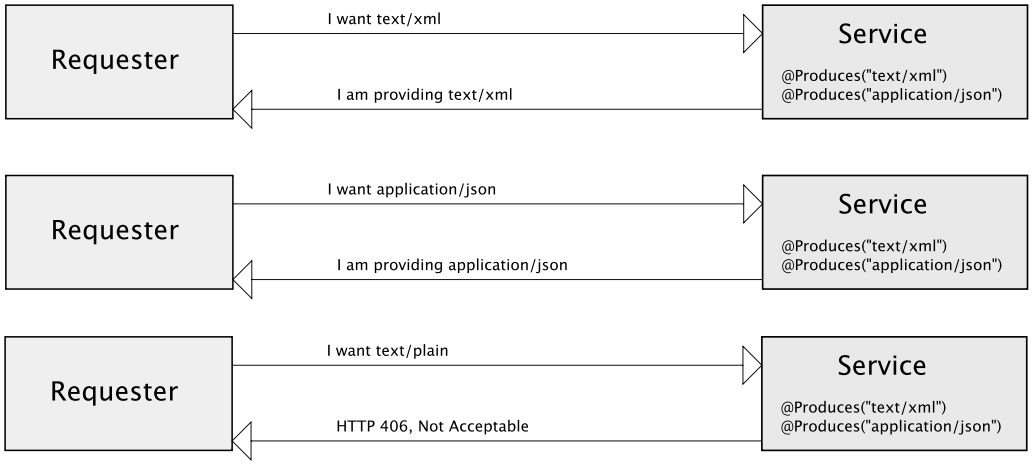
Instead (or in addition to) overriding getClasses, you can override getSingletons and return a Set of objects. If you do so, JAX-RS will not create a separate instance for the resource class for each request.

"Providers" are optional classes that you can write to extend the JAX-RS framework. For example, you can write an "entity provider" that changes how JAX-RS maps between Java types and resource representations. See the JAX-RS specification for more details.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation

- The client and server must agree on a data **representation**, which is usually a standard format such as text, XML or JSON using a **MediaType**
- Ultimately, the service decides on which representations it will provide



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Negotiation via HTTP Accept Header

- The HTTP specification allows for an **Accept** header, in which the user agent client provide a list of MIME types
- For GET requests, you use the **@Produces** annotation and the **@Consumes** annotation for POST and PUT requests
- JAX-RS will map requests to methods based on the Accept header and the annotations

```
curl –H "Accept: application/json, application/xml"
 http://localhost:8080/TestRESTWeb/rest/advisor/12

{"id":12,"name":"Sue Joens","salary":50000.0}
```


cURL is a popular and powerful tool for sending network requests - it's very useful for testing RESTful services. cURL is included with many operating systems, but not Windows. You can download it from;

http://curl.haxx.se/download.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation Example #1

```
1    // JavaScript Ajax requester
2
3    req.open("GET", url, true);
4    req.setRequestHeader("Accept","application/json");
5    req.onreadystatechange = callback;
6    req.send(null);
```

```
1    // JAX-RS resource class
2
3    @GET
4    @Produces("application/json")
5    public String studentAsJSON(
6       @QueryParam("studentID") String id)
7    {
8    . . .
9    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation Example #2

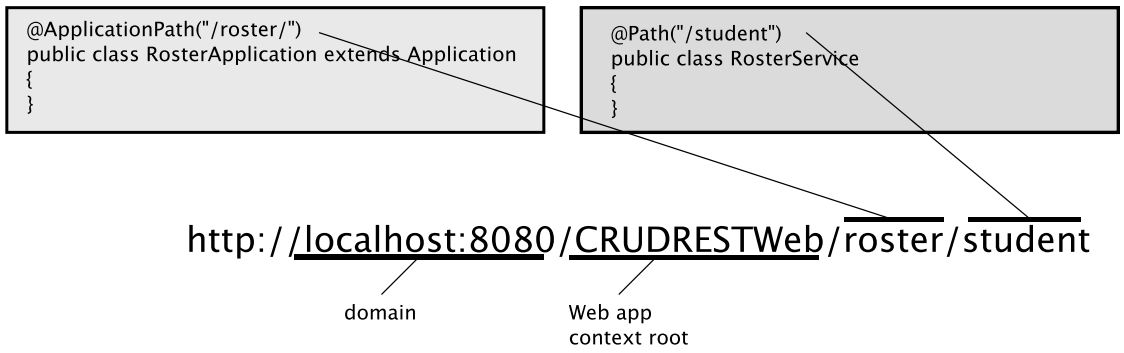
```
1    // JavaScript Ajax requester
2
3    req.open("POST", url, true);
4    req.setRequestHeader("Content-type","text/xml");
5    req.onreadystatechange = callback;
6    req.send(xml);
```

```
1    // JAX-RS resource class
2
3    @POST
4    @Consumes("text/xml")
5    public String createStudentFromXml(String xml)
6    {
7    . . .
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Assigning a URI Path

- To make methods available to respond to REST requests, you use the **Path** annotation to assign URIs
- You also can assign an overall application URI



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing Request Parameters

- HTTP **GET** requests can contain parameters as part of the URL
- JAX-RS methods can bind to such parameters using the **@QueryParam** annotation

```
1    // JavaScript Ajax client
2
3    // url like http://...../roster/student?studentID=12
4    var url="roster/student?studentID=" + escape(studentID);
```

```
1    // JAX-RS resource
2    @GET
3    @Produces("text/xml, application/json")
4    public String studentAsXml(
5       @QueryParam("studentID") String id)
6    {
7     . . .
8    }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing URI Parameters

- Instead of, or in addition to, HTTP GET request parameters, RESTful applications can use parameters from the URI itself
- JAX-RS methods can bind to such parameters using the **@PathParam** annotation

```
1    // JAX-RS resource
2    @Path("/student/{studentId}")
3    public class StudentService
4    {
5        @GET
6        @Produces("text/xml, application/json")
7        public Response find(@PathParam("studentId")int id)
8        {
9           . . .
10       }
11   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Context Annotation

- You can use the **javax.ws.rs.core.Context** annotation to decorate fields or method parameters to **inject** context information of the following types:
  - **Application** - Reference to the Application subclass
  - **UriInfo** - Request URI information
  - **HttpHeaders** - HTTP headers
  - **Request** - Content negotiation information
  - **SecurityContext** - User principal, etc.
  - **ServletConfig**, **ServletContext**, **HttpServletRequest**, **HttpServletResponse** - Standard JEE objects

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

@Context Example

```
1    @Path("/student")
2    public class RosterService
3    {
4      @Context
5       HttpServletRequest request;
6
7      @GET
8      @Produces("text/xml, application/json")
9      public String studentAsXml(
10            @Context HttpHeaders headers)
11     {
12       System.out.println("User-agent: " +
13          headers.getRequestHeader("user-agent"));
14       HttpSession session = request.getSession();
15
16       . . .
17      }
18   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mapping Java Objects to Content Types

- Many JAX-RS implementations, including Jersey, can automatically map Java objects to/from common types such as XML or JSON
- You must decorate the Java classes with **JAX-B** annotations

```
1    // JAX-RS resource
2    @GET
3    @Produces("text/xml, application/json")
4    public Response find(@PathParam("studentId")int id)
5    {
6        Student s = . . .;
7        return Response.ok(s).build();
8    }
9
10   @XmlRootElement
11   public class Student{. . .}
```

The javax.ws.rs.core.Response lets you build responses with well known HTTP status codes and response body. Here we send back HTTP 200 OK and the response body with a Student object serialized into XML.

Note for this to work in Jersey, the application's CLASSPATH must include optional JARs including jersey-json*.jar and jackson-*.jar.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a JAX-RS Application

- Create one or more **resource** classes that produce or consume RESTful data and annotate them with JAX-RS annotations
- Create a class that extends **javax.ws.rs.core.Application** and overrides the **getClasses** method
- Configure JAX-RS libraries into Web application (if not using a full JEE application server)
- Deploy Web application
- Write client(s) that invoke the service

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example

Student.java

Roster.java

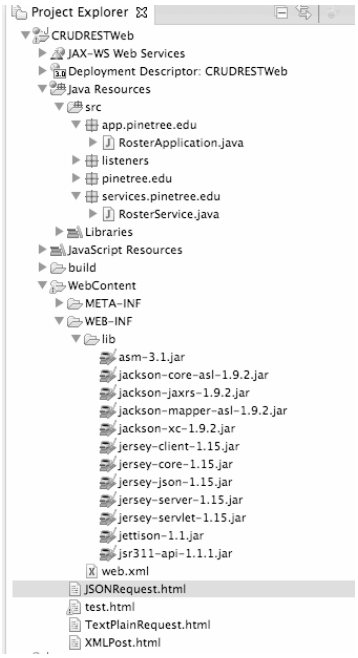
MyContextListener.java

RosterApplication.java

RosterService.java

GETRequest.html

XMLPost.html



Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package pinetree.edu;
2
3    import javax.xml.bind.annotation.XmlRootElement;
4
5    @XmlRootElement
6    public class Student
7    {
8        private int studentID;
9        private String name;
10       private double gpa;
11
12       @Override
13       public int hashCode()
14       {
15           final int prime = 31;
16           int result = 1;
17           result = prime * result + studentID;
18           return result;
19       }
20
21       @Override
22       public boolean equals(Object obj)
23       {
24           if (this == obj)
25               return true;
26           if (obj == null)
27               return false;
28           if (getClass() != obj.getClass())
29               return false;
30           Student other = (Student) obj;
31           if (studentID != other.studentID)
32               return false;
33           return true;
34       }
35
36       public int getStudentID()
37       {
38           return studentID;
39       }
40
41       public void setStudentID(int studentID)
42       {
43           this.studentID = studentID;
44       }
45
```

```
46      public String getName()
47      {
48          return name;
49      }
50
51      public void setName(String name)
52      {
53          this.name = name;
54      }
55
56      public double getGpa()
57      {
58          return gpa;
59      }
60
61      public void setGpa(double gpa)
62      {
63          this.gpa = gpa;
64      }
65
66      @Override
67      public String toString()
68      {
69          return "Student [studentID=" + studentID + ", name=" + name + ", gpa="
70                  + gpa + "]";
71      }
72
73  }
```

```
1    package pinetree.edu;
2
3    import java.util.ArrayList;
4    import java.util.List;
5
6    import javax.xml.bind.annotation.XmlRootElement;
7
8    @XmlRootElement
9    public class Roster
10   {
11       private List<Student> students = new ArrayList<Student>();
12
13       public List<Student> getStudents()
14       {
15           return students;
16       }
17
18       public void setStudents(List<Student> students)
19       {
20           this.students = students;
21       }
22   }
```

```
1    package listeners;
2
3    import javax.servlet.ServletContextEvent;
4    import javax.servlet.ServletContextListener;
5
6    import pinetree.edu.Roster;
7    import pinetree.edu.Student;
8
9    public class MyContextListener implements ServletContextListener
10   {
11       @Override
12       public void contextDestroyed(ServletContextEvent evt)
13       {
14       }
15
16       @Override
17       public void contextInitialized(ServletContextEvent evt)
18       {
19           Roster r = new Roster();
20           Student s = new Student();
21           s.setGpa(3.44);
22           s.setName("Sam B. Jones");
23           s.setStudentID(12);
24           r.getStudents().add(s);
25
26           s = new Student();
27           s.setGpa(3.45);
28           s.setName("Susan Q. Smith");
29           s.setStudentID(13);
30           r.getStudents().add(s);
31
32           s = new Student();
33           s.setGpa(2.98);
34           s.setName("Xiang Lu");
35           s.setStudentID(14);
36           r.getStudents().add(s);
37
38           System.out.println("Context initialized");
39
40           evt.getServletContext().setAttribute("roster", r);
41       }
42   }
```

```
1    package app.pinetree.edu;
2
3    import java.util.HashSet;
4    import java.util.Set;
5
6    import javax.ws.rs.ApplicationPath;
7    import javax.ws.rs.core.Application;
8
9    import services.pinetree.edu.RosterService;
10
11   @ApplicationPath("/roster/")
12   public class RosterApplication extends Application
13   {
14       @Override
15       public Set<Class<?>> getClasses()
16       {
17           Set<Class<?>> s = new HashSet<Class<?>>();
18           s.add(RosterService.class);
19           return s;
20       }
21
22   }
```

```
1    package services.pinetree.edu;
2
3    import java.net.InetAddress;
4    import java.net.URI;
5
6    import javax.servlet.ServletContext;
7    import javax.servlet.http.HttpServletRequest;
8    import javax.ws.rs.Consumes;
9    import javax.ws.rs.GET;
10   import javax.ws.rs.HeaderParam;
11   import javax.ws.rs.POST;
12   import javax.ws.rs.Path;
13   import javax.ws.rs.Produces;
14   import javax.ws.rs.QueryParam;
15   import javax.ws.rs.core.Context;
16   import javax.ws.rs.core.Response;
17
18   import pinetree.edu.Roster;
19   import pinetree.edu.Student;
20
21   @Path("/student")
22   public class RosterService
23   {
24       @Context
25       ServletContext context;
26
27       @GET
28       @Produces("text/xml, application/json")
29       public Response getStudent(@QueryParam("studentID")
30          String id, @HeaderParam("Accept") String acceptHeader)
31       {
32           System.out.println("Received GET request for student: " +
33              id + " requested type: " + acceptHeader);
34           Response retVal = null;
35
36           retVal = findStudent(id);
37
38           return retVal;
39       }
40
41       private Response findStudent(String id)
42       {
43           Response retVal;
44           int studentID = Integer.parseInt(id);
45           Roster roster = (Roster)context.getAttribute("roster");
```

```
46
47          Student found = null;
48          for (Student s : roster.getStudents())
49          {
50              if (s.getStudentID() == studentID)
51              {
52                  System.out.println("Found a student: " + s);
53                  found = s;
54                  break;
55              }
56          }
57
58          if (found == null)
59          {
60              System.out.println("No student found");
61              retVal =
62                Response.status(Response.Status.NOT_FOUND).build();
63          }
64          else
65          {
66              retVal = Response.ok(found).build();
67          }
68          return retVal;
69      }
70
71      @POST
72      @Consumes("text/xml")
73      public Response createStudentFromXml(Student s,
74              @Context final HttpServletRequest request)
75          throws Exception
76      {
77          Response retVal =
78            Response.status(Response.Status.BAD_REQUEST).build();
79
80          Roster roster = (Roster) context.getAttribute("roster");
81
82          if (s != null)
83          {
84              System.out.println("Received student XML " + s);
85              if (!roster.getStudents().contains(s))
86              {
87                  roster.getStudents().add(s);
88
89                  String newStudentURI = "http://" +
90                    InetAddress.getLocalHost().getHostName() +
```

```
91                    ":" + request.getLocalPort() + request.getContextPath() +
92                    "/roster/student?studentID=" + s.getStudentID();
93
94                 System.out.println(
95                    "Created student resource with URI: " +
96                         newStudentURI);
97
98                 // the Response.created() method sets the HTTP
99                 // Location header to the given URI
100                retVal = Response.created(
101                   URI.create(newStudentURI)).build();
102             }
103         }
104
105         return retVal;
106     }
107 }
```

```
1    <html>
2      <head>
3        <title>XML POST</title>
4        <script>
5
6          function sendRequest()
7          {
8            var url = "roster/student";
9            sendAsyncPostRequest(url);
10         }
11
12         function callback()
13         {
14           if (req.readyState == 4)
15           {
16               if (req.status == 200)
17               {
18               var text = req.responseText;
19               alert(text);
20               }
21               else
22               {
23                   alert("Problem: " + req.status + ", " + req.statusText);
24               }
25           }
26         }
27
28         function sendAsyncPostRequest(url)
29         {
30           req = getXMLHttpRequest();
31           if (req)
32           {
33           var studentID = document.getElementById("studentid").value;
34           var name = document.getElementById("name").value;
35           var gpa = document.getElementById("gpa").value;
36           var xml = "<student><name>" + name + "</name>" +
37                               "<studentID>" + studentID + "</studentID>" +
38                               "<gpa>" + gpa + "</gpa>" +
39                     "</student>";
40
41             req.open("POST", url, true);
42             req.setRequestHeader("Content-type","text/xml");
43             req.onreadystatechange = callback;
44             req.send(xml);
45           }
```

```
46          }
47
48          function getXMLHttpRequest()
49          {
50            var xRequest=null;
51            if (window.XMLHttpRequest)
52            {
53                xRequest=new XMLHttpRequest();
54            }
55            else if (typeof ActiveXObject != "undefined")
56            {
57                xRequest=new ActiveXObject("Microsoft.XMLHTTP");
58            }
59            return xRequest;
60          }
61        </script>
62      </head>
63      <body>
64        <form>
65          <p>
66            Student ID:   <input type="text" name="studentID" id="studentid"
67          </p>
68          <p>
69            Name:   <input type="text" name="name" id="name">
70          </p>
71          <p>
72            GPA:   <input type="text" name="gpa" id="gpa">
73          </p>
74          <p>
75            <input type="button" value="Insert Student" onclick="sendRequest()">
76          </p>
77        </form>
78      </body>
79    </html>
```

```
<html>
  <head>
    <title>JSON Request</title>
    <script>

      function sendRequest()
      {
        var studentID =
            document.getElementById("studentid").value;
        var mediaType =
            document.getElementById("mediaType").value;
        var url = "roster/student?studentID=" +
            escape(studentID);
        sendAsyncGetRequest(url, mediaType);
      }

      function showResponse(text)
      {
        if (text.trim().length == 0)
          text = "No student with that ID";

        var resp = document.getElementById('output');

        resp.firstChild.nodeValue=text;
      }

      function callback()
      {
        if (req.readyState == 4)
        {
            if (req.status == 200)
            {
                var text = req.responseText;

                showResponse(text);
            }
            else
            {
                alert("Problem: " + req.status + ", " +
                   req.statusText);
            }
        }
      }

      function sendAsyncGetRequest(url, mediaType)
```

```
46          {
47            req = getXMLHttpRequest();
48            if (req)
49            {
50              req.open("GET", url, true);
51              req.setRequestHeader("Accept",mediaType);
52              req.onreadystatechange = callback;
53              req.send(null);
54            }
55          }
56
57          function getXMLHttpRequest()
58          {
59            var xRequest=null;
60            if (window.XMLHttpRequest)
61            {
62                xRequest=new XMLHttpRequest();
63            }
64            else if (typeof ActiveXObject != "undefined")
65            {
66                xRequest=new ActiveXObject("Microsoft.XMLHTTP");
67            }
68            return xRequest;
69          }
70        </script>
71      </head>
72      <body>
73        <h3>Service Response</h3>
74        <div id="output">Nothing yet</div>
75        <form>
76          <p>
77            Student ID:
78            <input type="text" name="studentID" id="studentid">
79          </p>
80          <p>
81            Media Type (text/xml or application/json):
82            <input type="text" name="mediaType" id="mediaType">
83          </p>
84          <p>
85            <input type="button" value="Send Request"
86               onclick="sendRequest()">
87          </p>
88        </form>
89      </body>
90    </html>
```

Chapter Summary

In this chapter, you learned:

- The fundamentals of the JAX-RS framework
- How to write **resource** classes and **application** classes
- How JAX-RS applications negotiate content representation types

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

RESTful CRUD Services

- Create
- Retrieve
- Update
- Delete

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Richardson Maturity Model

- According to the **Richardson Maturity Model**, which evaluates how RESTful an application is, apps that correctly implement CRUD are at **Level 2**

```
Level 0  Use a single verb, typically POST, and a single URI
         for all interactions. Most SOAP based services are
         like this

Level 1  Use a single verb, typically GET, but multiple URIs.
         These types of services often violate the idempotency
         of GET

Level 2  Use multiple verbs and multiple URIs. This includes
         CRUD RESTful apps

Level 3  Use Hypermedia as the Engine of Application State
```

You can read more about the RMM at:

http://martinfowler.com/articles/richardsonMaturityModel.html

According to the Wikipedia entry on "idempotence":

In computer science, the term idempotent is used more comprehensively to describe an operation that will produce the same results if executed once or multiple times.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

RESTful CRUD Service Overview

- Many applications use the four **CRUD** operations: **Create**, **Retrieve**, **Update** and **Delete**
- The REST community has come up with guidelines for how to implement CRUD using standard HTTP facilities

The Rest in Practice book by Leonard Richardson is a very good reference for these guidelines.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTTP Review

- HTTP provides several **methods** as well as standard **return codes** and **headers**

```
HTTP Methods
------------
GET, POST,
PUT, DELETE,
HEAD, PATCH,
OPTIONS, TRACE,
CONNECT

HTTP Headers
------------
Accept
Content-Type
Status
...
```

```
HTTP Status Codes
-----------------
200 OK
201 Created
204 No Content
400 Bad Request
404 Not Found
409 Conflict
500 Internal Server Error
...
```

For more information on HTTP, see the specification:

http://www.w3.org/Protocols/rfc2616/rfc2616.html

Note that this is not a complete listing of status codes and headers, only those commonly used by RESTful services.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Media Types

- The IANA defines **media types (MIME)** that services can use to specify data types

```
Common Media Types
------------------

application/json
text/plain
text/xml
application/vnd.*     -- vendor specific
```


For more information on the Internet Assigned Numbers Authority (IANA) and media types, see:

https://www.iana.org/assignments/media-types

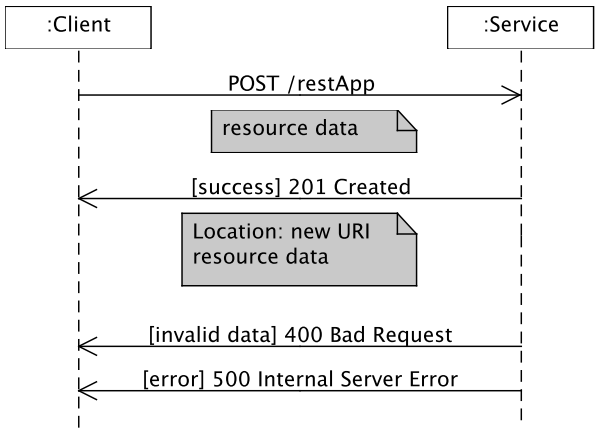
There is a lot of discussion in the REST community on whether to use standard media types like text/xml or application/json or to define custom media types for the application. See this article for a discussion:

http://www.infoq.com/news/2010/01/subbu-custom-media-types

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing Create

- Client should **POST** a resource representation and will receive a possibly modified resource representation
- Service returns response code **201 Created** on success or **400 Bad Request** or **500 Internal Server Error**



The response message also contains an HTTP Location header that specifies the newly created resource's RESTful URI.

The body of the response contains a possibly different resource representation than was passed, since it might contain auto-generated data like a database primary key.

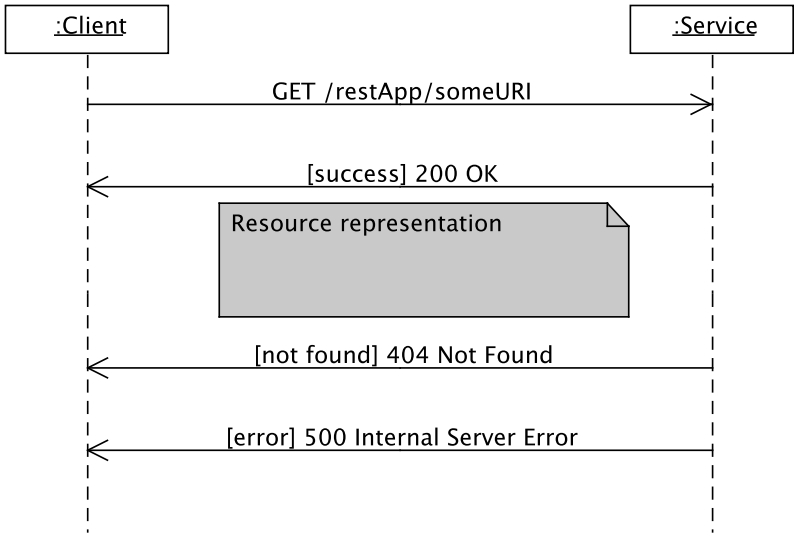
This page discusses using HTTP 201 vs 202:

http://benramsey.com/blog/2008/04/http-status-201-created-vs-202-accepted/

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing Retrieve

- Client should **GET** to a URI for the requested resource
- Service returns requested resource and **200 OK** or **404 Not Found** or **500 Internal Server Error**

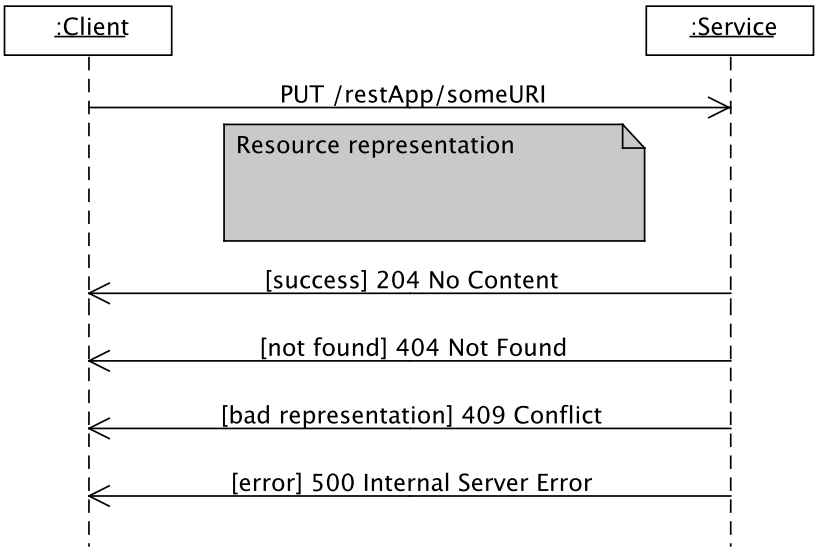


Even though the client can expect that the GET request itself doesn't change state on the server, it's not necessarily true that multiple GETs return the same data. For example, consider an Order-status request - the status might initially be "in progress" and eventually "complete".

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing Update

- Client should **PUT** a complete resource representation to the resource's URI
- Service updates the requested resource and returns **204 No Content**, **404 Not Found**, **409 Conflict** or **500 Internal Server Error**



Many RESTful applications follow this convention (established by Richardson and Ruby): <pre>- Use POST to create a resource identified by a service-generated URI. - Use POST to append a resource to a collection identified by a service-generated URI. - Use PUT to create or overwrite a resource identified by a URI computed by the client. </pre> The service could also return 200 OK, but the HTTP specification requires that such responses contain a response body, while 204 does not.

The service can return 409 Conflict if the supplied resource representation is incorrect, for example ill formed XML.

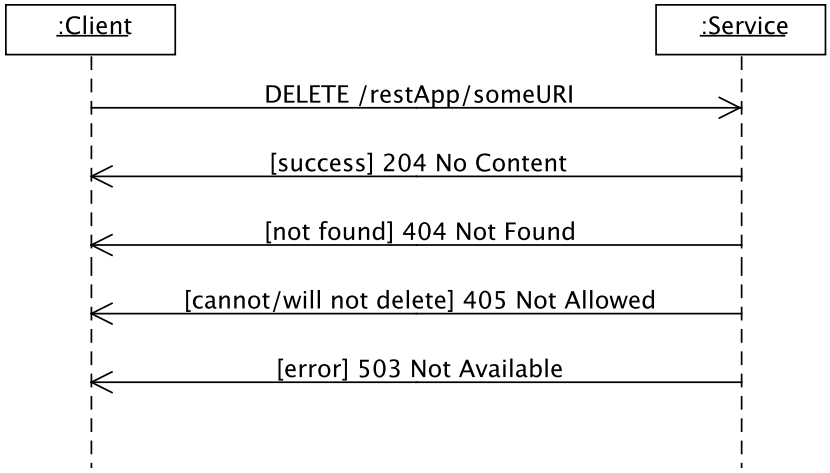
Note that the client must send complete resource representation. There is an emerging concept of using HTTP PATCH to provide just a delta to update.

Note that PUT should be idempotent since the client provides the entire resource. So if the service returns 500, it should be OK for the client to resend the request.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing Delete

- Client uses HTTP **DELETE** specifying the resource's URI
- Service returns **204 No Content** on success or **404 Not Found**, **405 Method Not Allowed** or **503 Service Unavailable**



One reason the service might return 405 is if it refuses to delete for some reason. For example, the service might have a policy that once an order is fully complete, it cannot be deleted.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete JAX-RS CRUD Example

Student.java

MyContextListener.java

StudentApplication.java

StudentService.java

GetClient.java

PostClient.java

PutClient.java

DeleteClient.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```java
package edu.bigcollege;

import java.io.Serializable;

import javax.xml.bind.annotation.XmlRootElement;

@SuppressWarnings("serial")
@XmlRootElement
public class Student implements Serializable
{
    private int studentId;
    private String name;
    private double gpa;

    public Student()
    {
    }

    public Student(int studentId, String name, double gpa)
    {
        super();
        this.studentId = studentId;
        this.name = name;
        this.gpa = gpa;
    }

    public int getStudentId()
    {
        return studentId;
    }

    public void setStudentId(int studentId)
    {
        this.studentId = studentId;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }
```

```
46
47      public double getGpa()
48      {
49          return gpa;
50      }
51
52      public void setGpa(double gpa)
53      {
54          this.gpa = gpa;
55      }
56
57      @Override
58      public int hashCode()
59      {
60          final int prime = 31;
61          int result = 1;
62          result = prime * result + studentId;
63          return result;
64      }
65
66      @Override
67      public boolean equals(Object obj)
68      {
69          if (this == obj)
70              return true;
71          if (obj == null)
72              return false;
73          if (getClass() != obj.getClass())
74              return false;
75          Student other = (Student) obj;
76          if (studentId != other.studentId)
77              return false;
78          return true;
79      }
80
81      @Override
82      public String toString()
83      {
84          StringBuilder builder = new StringBuilder();
85          builder.append("Student [studentId=").append(studentId)
86                  .append(", name=").append(name).append(", gpa=").append(gpa)
87                  .append("]");
88          return builder.toString();
89      }
90  }
```

```java
package edu.bigcollege.listeners;

import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import edu.bigcollege.Student;

public class MyContextListener implements ServletContextListener
{

    @Override
    public void contextDestroyed(ServletContextEvent ctx)
    {
    }

    @Override
    public void contextInitialized(ServletContextEvent ctx)
    {
        System.out.println("Context initialized");
        List<Student> students = new ArrayList<Student>();

        Student c = new Student();
        c.setGpa(3.44);
        c.setName("Sue Smith");
        c.setStudentId(12);
        students.add(c);

        c = new Student();
        c.setGpa(3.87);
        c.setName("Bill Jones");
        c.setStudentId(44);
        students.add(c);

        ctx.getServletContext().setAttribute("students", students);
    }

}
```

```
1    package edu.bigcollege.service;
2
3    import java.util.HashSet;
4    import java.util.Set;
5
6    import javax.ws.rs.ApplicationPath;
7    import javax.ws.rs.core.Application;
8
9    @ApplicationPath("/rest/")
10   public class StudentApplication extends Application
11   {
12       @Override
13       public Set<Class<?>> getClasses()
14       {
15           Set<Class<?>> s = new HashSet<Class<?>>();
16           s.add(StudentService.class);
17
18           return s;
19       }
20   }
```

```
1    package edu.bigcollege.service;
2
3    import java.net.URI;
4    import java.net.URISyntaxException;
5    import java.util.List;
6    import java.util.Random;
7
8    import javax.servlet.ServletContext;
9    import javax.ws.rs.Consumes;
10   import javax.ws.rs.DELETE;
11   import javax.ws.rs.GET;
12   import javax.ws.rs.POST;
13   import javax.ws.rs.PUT;
14   import javax.ws.rs.Path;
15   import javax.ws.rs.PathParam;
16   import javax.ws.rs.Produces;
17   import javax.ws.rs.core.Context;
18   import javax.ws.rs.core.Response;
19   import javax.ws.rs.core.Response.ResponseBuilder;
20   import javax.ws.rs.core.Response.Status;
21
22   import edu.bigcollege.Student;
23
24   @Path("students")
25   public class StudentService
26   {
27       @Context
28       ServletContext servletContext;
29
30       @SuppressWarnings("unchecked")
31       @DELETE
32       @Path("{id}")
33       public Response delete(@PathParam("id") int id)
34       {
35           ResponseBuilder rb = null;
36
37           System.out.println("In delete: " + id);
38
39           Student found = findStudent(id);
40           if(found != null)
41           {
42               List<Student> students = (List<Student>) servletContext
43                       .getAttribute("students");
44
45               students.remove(found);
```

```
46              rb = Response.noContent();
47          }
48          else
49          {
50              rb = Response.status(Status.NOT_FOUND);
51          }
52
53          return rb.build();
54      }
55
56      @PUT
57      @Consumes({ "text/xml", "application/json" })
58      public Response update(Student s)
59      {
60          System.out.println("In update: " + s);
61          ResponseBuilder rb = null;
62
63          if(s != null)
64          {
65              Student found = findStudent(s.getStudentId());
66
67              if(found != null)
68              {
69                  found.setGpa(s.getGpa());
70                  found.setName(s.getName());
71                  rb = Response.noContent();
72              }
73              else
74              {
75                  rb = Response.status(Status.NOT_FOUND);
76              }
77          }
78          else
79          {
80              rb = Response.status(Status.CONFLICT);
81          }
82
83          return rb.build();
84      }
85
86
87      @SuppressWarnings("unchecked")
88      @POST
89      @Consumes({ "text/xml", "application/json" })
90      // need @Produces since we send back updated Student
```

```
91      // representation in response body
92      @Produces({ "text/xml", "application/json" })
93      public Response insert(Student s)
94      {
95          System.out.println("In insert: " + s);
96          ResponseBuilder rb = null;
97
98          if(s != null)
99          {
100             List<Student> students = (List<Student>) servletContext
101                     .getAttribute("students");
102
103             Random r = new Random();
104             s.setStudentId(Math.abs(r.nextInt()));
105
106             students.add(s);
107
108             try
109             {
110                 rb = Response.created(new URI("/students/" +
111                     s.getStudentId()));
112                 // marshal the student into the response body
113                 rb.entity(s);
114             }
115             catch (URISyntaxException e)
116             {
117                 System.out.println("ERROR inserting: " +
118                   e.getMessage());
119                 rb = Response.serverError();
120             }
121         }
122         else
123         {
124             rb = Response.status(Status.BAD_REQUEST);
125         }
126
127         return rb.build();
128     }
129
130     @SuppressWarnings("unchecked")
131     @GET
132     @Path("{id}")
133     @Produces({ "text/xml", "application/json" })
134     public Response getStudent(@PathParam("id") int id)
135     {
```

```
136         Response ret = null;
137
138         List<Student> students = (List<Student>) servletContext
139                 .getAttribute("students");
140
141         Student theStudent = null;
142         for (Student c : students)
143         {
144             if (c.getStudentId() == id)
145             {
146                 theStudent = c;
147                 break;
148             }
149         }
150
151         if (theStudent != null)
152         {
153             ret = Response.ok(theStudent).build();
154         }
155         else
156         {
157             ret = Response.status(Response.Status.NOT_FOUND).build();
158         }
159
160         return ret;
161     }
162
163     @SuppressWarnings("unchecked")
164     private Student findStudent(int id)
165     {
166         List<Student> students = (List<Student>) servletContext
167                 .getAttribute("students");
168
169         Student found = null;
170         for(Student temp : students)
171         {
172             if(temp.getStudentId() == id)
173             {
174                 found = temp;
175                 break;
176             }
177         }
178         return found;
179     }
180
```

```
181   }
```

```
1    package client;
2
3    import java.io.BufferedReader;
4    import java.io.InputStreamReader;
5    import java.net.HttpURLConnection;
6    import java.net.URL;
7
8    public class GetClient
9    {
10
11       public static void main(String[] args) throws Exception
12       {
13           URL url = new URL(
14             "http://localhost:9081/TestRestCRUDWeb/rest/students/12");
15           HttpURLConnection conn =
16             (HttpURLConnection) url.openConnection();
17           conn.setRequestMethod("GET");
18           conn.setRequestProperty("Accept", "application/json");
19
20           if (conn.getResponseCode() != 200)
21           {
22               throw new RuntimeException("Failed : HTTP error code : "
23                       + conn.getResponseCode());
24           }
25
26           BufferedReader br = new BufferedReader(new InputStreamReader(
27                   (conn.getInputStream())));
28
29           String output;
30           System.out.println("Output from Server .... \n");
31           while ((output = br.readLine()) != null)
32           {
33               System.out.println(output);
34           }
35
36           conn.disconnect();
37
38       }
39   }
```

```
1    package client;
2
3    import java.io.BufferedReader;
4    import java.io.InputStreamReader;
5    import java.io.OutputStream;
6    import java.net.HttpURLConnection;
7    import java.net.URL;
8
9    public class PostClient
10   {
11       public static void main(String[] args) throws Exception
12       {
13           URL url =
14             new URL(
15              "http://localhost:9081/TestRestCRUDWeb/rest/students");
16           HttpURLConnection conn =
17             (HttpURLConnection) url.openConnection();
18           conn.setDoOutput(true);
19           conn.setRequestMethod("POST");
20           conn.setRequestProperty("Content-Type", "application/json");
21
22           String input = "{\"name\":\"Harry Wolfe\",\"gpa\":2.22}";
23
24           OutputStream os = conn.getOutputStream();
25           os.write(input.getBytes());
26           os.flush();
27
28           if (conn.getResponseCode() != HttpURLConnection.HTTP_CREATED)
29           {
30               throw new RuntimeException("Failed : HTTP error code : "
31                       + conn.getResponseCode());
32           }
33
34           BufferedReader br = new BufferedReader(new InputStreamReader(
35                   (conn.getInputStream())));
36
37           String output;
38           System.out.println("Output from Server .... \n");
39           while ((output = br.readLine()) != null)
40           {
41               System.out.println(output);
42           }
43
44           System.out.println("Location header: "
45                   + conn.getHeaderField("Location"));
```

```
46
47              conn.disconnect();
48          }
49      }
```

```
1    package client;
2
3    import java.io.OutputStream;
4    import java.net.HttpURLConnection;
5    import java.net.URL;
6
7    public class PutClient
8    {
9        public static void main(String[] args) throws Exception
10       {
11           URL url =
12             new URL(
13               "http://localhost:9081/TestRestCRUDWeb/rest/students");
14           HttpURLConnection conn =
15             (HttpURLConnection) url.openConnection();
16           conn.setDoOutput(true);
17           conn.setRequestMethod("PUT");
18           conn.setRequestProperty("Content-Type", "application/json");
19
20           String input =
21             "{\"name\":\"Joe Blow\",\"gpa\":4.56, \"studentId\":12}";
22
23           OutputStream os = conn.getOutputStream();
24           os.write(input.getBytes());
25           os.flush();
26
27           if (conn.getResponseCode() !=
28              HttpURLConnection.HTTP_NO_CONTENT)
29           {
30               throw new RuntimeException("Failed : HTTP error code : "
31                       + conn.getResponseCode());
32           }
33
34           conn.disconnect();
35       }
36   }
```

```
1    package client;
2
3    import java.net.HttpURLConnection;
4    import java.net.URL;
5
6    public class DeleteClient
7    {
8        public static void main(String[] args) throws Exception
9        {
10           URL url =
11             new URL(
12               "http://localhost:9081/TestRestCRUDWeb/rest/students/44");
13           HttpURLConnection conn =
14             (HttpURLConnection) url.openConnection();
15           conn.setRequestMethod("DELETE");
16
17           conn.connect();
18
19           if (conn.getResponseCode() != HttpURLConnection.HTTP_NO_CONTENT)
20           {
21               throw new RuntimeException("Failed : HTTP error code : "
22                       + conn.getResponseCode());
23           }
24
25           conn.disconnect();
26       }
27   }
```

Chapter Summary

In this chapter, you learned:

- The fundamentals of REST applications that create, retrieve, update and delete resources
- How to implement a RESTful CRUD application using JAX-RS

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and REST

- Spring Support for REST
- Message Converters
- Annotations

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Support for REST

- Starting in Spring 3, Spring makes it easy to write SpringMVC **controllers** that work with REST resources
- If you are writing your application with Spring, than using Spring REST is easier and more natural than using JAX-RS

JAX-RS is the JEE standard for working with RESTful resources. Spring REST is similar in many ways, but integrates with the Spring infrastructure better than JAX-RS.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Message Converters

- Spring uses **message converters** to convert Java objects to/from a **resource representation**, including the following:
    - **SpringHttpMessageConverter** - transforms String objects to/from *text/plain*
    - **Jaxb2RootElementHttpMessageConverter** - transforms JAXB-annotated objects to/from *application/xml*
    - **MappingJacksonHttpMessageConverter** - transforms object to/from *application/json*

There are several more message converters available - see:

http://docs.spring.io/spring/docs/3.1.x/spring-framework-reference/html/remoting.html#rest-message-c

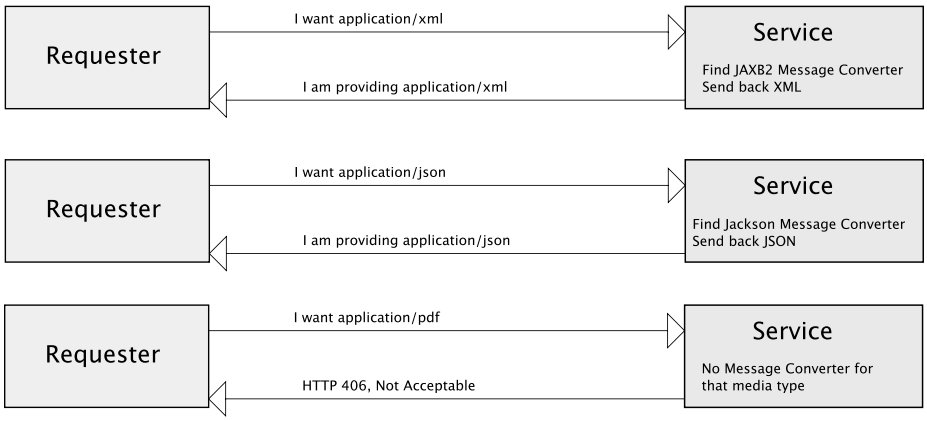
You must configure the JARs for JAXB2 and Jackson on the application's CLASSPATH to use those converters. Note that full JEE application servers like WebSphere include JAXB2 and might also include Jackson. If you're not running on a full JEE server, you can obtain Jackson from http://jackson.codehaus.org and JAXB2 from https://jaxb.java.net/.

Spring REST doesn't require you to use message converters. You could instead use the normal Spring MVC view-based infrastructure to send back JSON or XML to the client. Spring provides the ContentNegotiatingViewResolver for that purpose - see the Spring documentation for details.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation

- The client and server must agree on a data **representation**, which is usually a standard format such as text, XML or JSON using a **MediaType**
- Ultimately, the service decides on which representations it will provide



Media types are also known as MIME types. When the client requests a particular media type, Spring looks for a registered message converter for that type and uses it to convert the service's data.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Negotiation via HTTP Accept Header

- The HTTP specification allows for an **Accept** header, in which the user agent client provide a list of media types
- Spring parses the Accept header and then finds a message converter that can work with that media type

```
 curl -H "Accept: application/xml, application/json"
  http://localhost:9080/TestSpringRESTWeb/rest/advisor/12

<!-- response -->
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<advisor>
  <id>12</id>
  <name>Sue Joens</name>
  <salary>50000.0</salary>
</advisor>
```


---

cURL is a popular and powerful tool for sending network requests - it's very useful for testing RESTful services. cURL is included with many operating systems, but not Windows. You can download it from;

http://curl.haxx.se/download.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring REST Annotations

- Spring provides several annotations to configure controllers to work with REST:
  - **@RequestMapping** - assigns a URL and an HTTP method to a processing method
  - **@ResponsStatus** - sets the HTTP status code
  - **@RequestBody** - transforms REST request into Java object(s)
  - **@ResponseBody** - transforms Java object into a REST response
  - **@RestController** - combines @Controller and @ResponseBody
  - **@PathVariable** - initializes a method parameter from part of the request URL
  - **@RequestParam** - initializes a method parameter from an HTTP *request parameter*

Controllers can use other standard Spring MVC annotations such as @Valid, @Autowire and so forth.

In Spring 4.3 and later, instead of using @RequestMapping and using an option to specify the HTTP method, you can use so-called "composed" annotations like @GetMapping that do the same thing but are more descriptive.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple RESTful Controller

- This controller method returns either XML or JSON depending on the client's **Accept** header

```
1    @Controller
2    @RequestMapping("rest")
3    public class MyService
4    {
5      // respond to URLs like ..../rest/advisor/12
6      @RequestMapping(value = "/advisor/{id}",
7        method = RequestMethod.GET)
8      public @ResponseBody
9        Advisor getAdvisor(@PathVariable("id") int id)
10     {
11       Advisor a = findAdvisorByID(id);
12       if(a == null) throw new ResourceNotFoundException():
13       return a;
14     }
15   }
```

The ResourceNotFoundException class looks like:

```
@ResponseStatus( value = HttpStatus.NOT_FOUND )
public class ResourceNotFoundException extends RuntimeException{}
```

Note that for XML conversion to work, the class of the returned object must be annotated with JAXB:

```
@XmlRootElement
public class Advisor
{
. . .
}
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Responding to POST Requests

- POST requests cause the REST service to create the resource and assign its RESTful URL

```
1    @RequestMapping(value = "/advisor",
2      method=RequestMethod.POST)
3    @ResponseStatus(HttpStatus.CREATED)
4    public @ResponseBody Advisor createAdvisor(
5      @RequestBody Advisor a, HttpServletResponse response)
6    {
7        int newID = insertAdvisor(a);
8        a.id = newID;
9
10       response.setHeader("Location", "/advisor/" + newID);
11       return a;
12   }
```


curl -v -X POST -H "Content-type: application/json" -H "Accept: application/json" -d "{\"id\":0,\"name\":\"Sue Joens\",\"salary\":150000.0}" http://localhost:9080/TestSpringRESTWeb/rest/advisor

Return might look like:

```
* upload completely sent off: 45 out of 45 bytes
< HTTP/1.1 201 Created
< X-Powered-By: Servlet/3.0
< Location: /advisor/56
< Content-Type: application/json;charset=UTF-8
< Content-Language: en-US
< Transfer-Encoding: chunked
< Date: Thu, 12 Jun 2014 16:12:23 GMT
< Server: WebSphere Application Server/8.0
<
{"id":56,"name":"Sue Joens","salary":150000.0}*
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Responding to PUT Requests

- PUT requests cause the REST service to update the RESTful resource

```
1    @RequestMapping(value = "/advisor/{id}",
2      method = RequestMethod.PUT)
3    public ResponseEntity<String> putAdvisor(
4      @PathVariable("id") long id, @RequestBody Advisor a)
5    {
6      ResponseEntity<String> retVal =
7        new ResponseEntity<String>(HttpStatus.NO_CONTENT);
8
9        boolean ok = updateAdvisor(a);
10
11       if(!ok)
12         retVal = new
13           ResponseEntity<String>(HttpStatus.NOT_FOUND);
14
15       return retVal;
16   }
21 - 9
```

---

curl -v -X PUT -H "Content-Type: application/json" -H "Accept:application/json" -d "{\"id\":12,\"name\":\"Sue\",\"salary\":345}" http://localhost:9081/TestSpringRESTWeb/rest/advisor/12

Response:

```
* upload completely sent off: 35 out of 35 bytes
< HTTP/1.1 204 No Content
< X-Powered-By: Servlet/3.0
< Content-Language: en-US
< Content-Length: 0
< Date: Thu, 12 Jun 2014 16:23:17 GMT
< Server: WebSphere Application Server/8.0
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a Spring REST Application

- You configure a RESTful Spring application in the same fashion as for a "normal" Spring MVC application

```
1    <!-- springmvc-servlet.xml -->
2    <beans . . .>
3
4      <context:component-scan
5         base-package="com.goliath" />
6
7      <mvc:annotation-driven />
8    . . .
9    </beans>
```

MyConfig.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package com.goliath.config;
2
3    import org.springframework.context.annotation.ComponentScan;
4    import org.springframework.context.annotation.Configuration;
5    import org.springframework.web.servlet.config.annotation.DefaultServletHandlerConf
6    import org.springframework.web.servlet.config.annotation.EnableWebMvc;
7    import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
8
9    @Configuration
10   @EnableWebMvc
11   @ComponentScan("com.goliath.services")
12   public class MyConfig extends WebMvcConfigurerAdapter
13   {
14       @Override
15       public void configureDefaultServletHandling(
16               DefaultServletHandlerConfigurer configurer)
17       {
18           configurer.enable();
19       }
20   }
```

21 - 10A : MyConfig.java

Chapter Summary

In this chapter, you learned:

- How to use Spring MVC controllers, annotations and message converters to write RESTful CRUD services in Spring.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Additional Spring Boot Topics

- Starter Dependencies
- Auto Configuration
- Deploying a WAR

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dependent Libraries

- Most large-scale Java applications use third-party libraries, which are typically packaged as JAR files
- Many libraries themselves depend on other libraries, leading to **transitive dependencies**
- Build tools such as Gradle and Maven can satisfy transitive dependencies

The Apache Ant build tool can also handle dependencies if you extend it with the Ivy tool.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Traditional Approaches for Dependencies

- Libraries typically have multiple versions, with newer versions having additional functionality and/or bug fixes
- Gradle and Maven let you specify specific library version numbers
- One challenge with transitive dependencies is ensuring that the versions "play well together" - traditionally, this job falls to the developer

```
compile("org.springframework:spring-web:4.1.2.RELEASE")
compile("org.springframework.data:spring-data-jpa:1.7.0.RELEASE")
compile("org.hibernate:hibernate-entitymanager:jar:4.2.2.Final")
runtime("com.h2database:h2:1.3.64")
```


The above shows a fragment of a Gradle script that explicitly specific versions of a few dependencies.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Boot Starter Dependencies

- Spring Boot provides **starter dependencies** for both Gradle and Maven that are curated versions of common libraries that work well together
- Instead of specifying individual library versions, you specify a Spring Boot version and Spring Boot provides the dependent versions

```
compile('org.springframework.boot:spring-boot-starter-jdbc')
compile('org.springframework.boot:spring-boot-starter-web')
runtime('com.h2database.h2')
```

- You can see a list of available starters using the CLI's *spring init --list* command

---

The above shows a fragment of a Gradle build script. Note how there's no version numbers! The 'buildscript' portion of the script might look like:

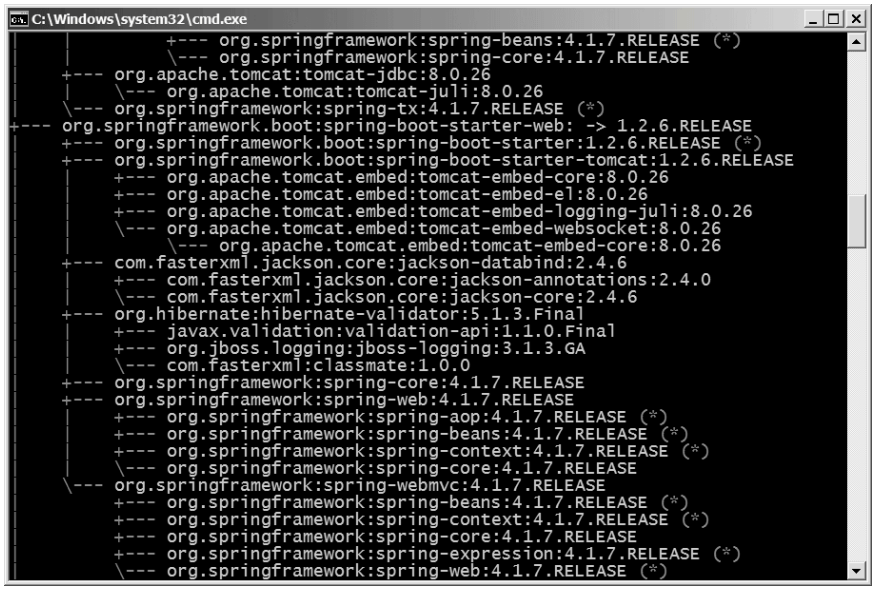
```
buildscript {
    ext {
        springBootVersion = '1.2.6.RELEASE'
    }
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersi
        classpath('io.spring.gradle:dependency-management-plugin:0.5.2.RELEASE')
    }
}
```

The key is the dependencies in the buildscript that enable Spring Boot starter dependencies based on the Spring Boot version.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Viewing Dependencies

- Gradle and Maven scripts let you view libraries so you can see what the starter dependencies brought in and the library versions



The above screenshot used this command to generate the output:

```
gradlew dependencies
```

Maven provides the the dependency:tree goal that acts in a similar fashion to the Gradle output shown above.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Eliminating Starter Dependencies

- Starter dependencies bring in commonly used transitive dependencies
- For example, **starter-web** brings in Spring MVC and embedded Tomcat
- You can remove dependencies that your app doesn't need to reduce the size of the executable JAR or WAR

```
compile('org.springframework.boot:spring-boot-starter-web') {
//     exclude group: 'org.hibernate'
       exclude module: 'hibernate-validator'
}
```


The full name of a dependency includes group, module and version, e.g.:

```
org.hibernate:hibernate-validator:5.1.3.Final
```

You can exclude using either the group, which eliminates all modules in that group, or via the module name. Note that you can specify both the group and module name to be more specific.

In Maven, you can use the "exclusions" element to eliminate transitive dependencies.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Specific Library

- Due to organizational policy, you might wish to use specific versions of some libraries other than what starters provide
- In Gradle, to use a newer version, simply specify it, but to use an older version, first exclude the older version, then specify the newer

```
compile('org.springframework.boot:spring-boot-starter-jdbc') {
  exclude module: 'snakeyaml'
}

compile('org.yaml:snakeyaml:1.8')
```

```
C:\Users\jbarnum\advspring\workspace\lab02>gradlew dependencies | find "snake"
|    |    |    \--- org.yaml:snakeyaml:1.14 -> 1.8
|    |    \--- org.yaml:snakeyaml:1.14 -> 1.8
```


One reason you might need to use specific versions is for security reasons - your organization might vet libraries to ensure they have no vulnerabilities that could affect your application.

Note that if you override the choices made by Spring Boot's starters, the onus is on you to ensure that the versions you choose work well together. One of the big benefits of the starters is that you know that the Spring Boot team has tested the chosen versions together.

In Maven, you just need to specify the specific library version using with a "dependency" element.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Auto Configuration

- In traditional Spring applications, developers needed to provide extensive configuration either using XML, annotations or Java configuration
- Spring Boot eliminates the need for most configuration, freeing the developer to concentrate on application code

```
1    <bean id="studentDataSource"
2      class=
3    "org.springframework.jdbc.datasource.DriverManagerDataSource">
4      <property name="driverClassName"
5        value="org.hsqldb.jdbcDriver"/>
6      <property name="url"
7        value="jdbc:hsqldb:hsql://localhost/"/>
8      <property name="username"
9        value="SA"/>
10     <property name="password"
11       value=""/>
12   </bean>
```


Over the years, Spring has striven to reduce the amount of configuration required using annotations and auto-wiring. Spring Boot continues that trend.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

How Auto Configuration Works

- Essentially, Spring Boot makes configuration decisions based on what it finds on the application's **classpath**
- For example, if you have a dependency for the HSQLDB database, Spring Boot will automatically configure a database connection
- For auto config to work, your application's "main" class should be annotated with **@SpringBootApplication**

The @SpringBootApplication annotation combines @Configuration, @ComponentScan and @EnableAutoConfiguration.

Behind the scenes, auto configuration works based on Spring Boot annotations such as @ConditionalOnMissingBean. For a good explanation on how this works, see: https://geowarin.github.io/understanding-spring-boot.html

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Overriding Auto Config Externally

- You can override many auto configuration defaults by adding to the **application.properties** file
- For example, you can specify database connection information:

```
spring.datasource.url=jdbc:hsqldb:hsql://localhost/
spring.datasource.username=SA
spring.datasource.password=
spring.datasource.driver-class-name=org.hsqldb.jdbcDriver
```


Instead of writing simple Java properties files, you can also use a more readable format known as YAML (Yaml Ain't a Markup Language). You can read more about YAML at: http://yaml.org. Note that you need to have the Snake YAML library as a dependency for YAML to work.

You can see a list of common configuration properties at: http://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html.

You can also override auto configuration defaults by writing your own Java configuration classes. For an example of providing database connection info, see: http://docs.spring.io/spring-boot/docs/current/reference/html/howto-data-access.html.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Executable JAR Deployment

- By default, *Initializr* generates build scripts that create a "fat" executable JAR that you can run to bootstrap the application
- If it's a Web application, executing the JAR launches an embedded Tomcat
- You can configure many server and Tomcat options via properties files

```
server.port=80

# enable SSL
server.port=443
server.ssl.key-store=classpath:mykeystore.jks
server.ssl.key-store-password=duke
server.ssl.key-password=mypassword
```


Executable JAR deployment is a good choice if you plan to run the application with a 'cloud' provider such as Heroku, or Docker.

You can also configure Spring Boot to use Jetty instead of Tomcat.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Building a WAR

- Executable JARs are easy, but some organizations use a traditional **application server** such as WebSphere or WebLogic
- Gradle scripts can use the **war** plugin to generate a WAR that you can deploy to an app server

```
apply plugin: 'war'

war {
    baseName = 'myapp'
    version = '0.0.1-SNAPSHOT'
}
```

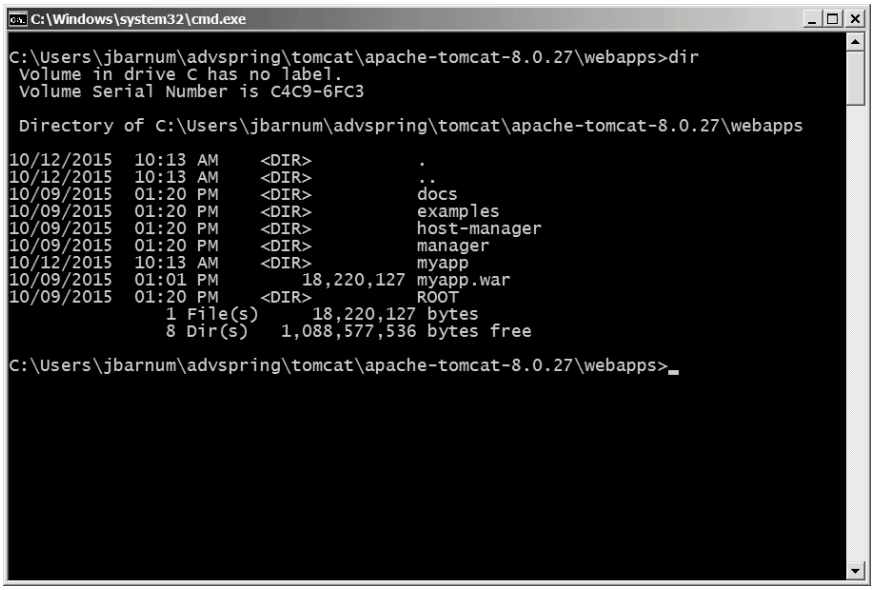

Maven scripts can change the "packaging" element from jar to war.

If you need to deploy an EAR instead of a WAR, it's a simple matter to package the WAR and an application.xml deployment descriptor into an EAR.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Deploying the WAR

- Deploying WARs is application-server specific - for Tomcat, just drop the WAR into TOMCAT_HOME/webapps



Note that depending on your application, you might need to do more work. For example, if you're using Spring MVC, Spring Boot executable JARs automatically enable the Spring 'dispatcher servlet', but if you're deploying a WAR, you will need to configure the servlet yourself, perhaps by writing entries in the web app's web.xml deployment descriptor.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About Spring Boot starter dependencies and how to override them
- About Spring Boot auto configuration and how to override it
- How to deploy Spring Boot applications as executable JARs or WARs

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transactions in Spring

- What is a Transaction?
- Transaction Scope and Propagation

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Transactions

- Transactions let programs define units of work consisting of steps that must all complete or none complete
- If you use transactions properly, your programs will be more robust and reliable

The notion of transactions has been around for quite awhile and is well known in enterprise systems. Many databases and other back-end resource managers implement transactions and can be "enlisted" by an Spring container.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Program with No Transactions

- This program does not use transactions and is subject to integrity problems if something goes wrong

```
1    // transfer $100 from savings to checking
2
3    long balance = savings.getBalance();
4    balance -= 100;
5    savings.setBalance ( balance );
6
7    balance = checking.getBalance();
8    balance += 100;
9    checking.setBalance()
```


This is the classic program that illustrates the need for transactions. Suppose the system crashes in line 6 -- will the customer be happy that the program "lost" $100?

Note that in the interest of illustrating the point, we are doing the balance transfer using primitive "get" and "set" methods. A real program would probably not work exactly like this, but the concept is still relevant.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Program with Transactions

- By adding calls to a *transaction service*, the system will not be corrupted if something goes wrong
- This is an example of program-demarcated transactions

```
1    // transfer $100 from savings to checking
2
3    ts.begin();
4    long balance = savings.getBalance();
5    balance -= 100;
6    savings.setBalance ( balance );
7
8    balance = checking.getBalance();
9    balance += 100;
10   checking.setBalance()
11   ts.commit();
```

This is the same program, now with calls to some "transaction service", which manages the transaction. Now if the program crashes in line 7, the transaction service can rollback the system so that no money is lost.

Note that this is the traditional transaction model -- EJBs support this, but also provide a technique in which the container manages transactions for you.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

ACID Transactions

Atomic Transactions should complete entirely or not at all

Consistent The system state after the transaction should meet "invariants", for example `balance > 0` or `customerID is unique`

Isolated Other transactions cannot "see" intermediate results. Implies locking

Durable Updates to data should survive a system crash, perhaps use transaction logs to recover after a restart

The literature on transactions defines these four properties as attributes of a good transaction, sometimes referred to by the acronym ACID</i>

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

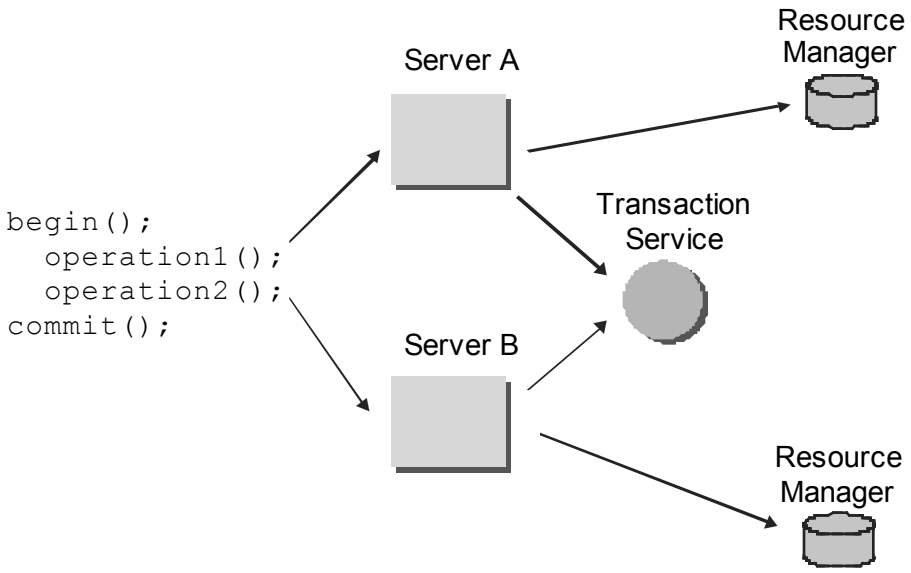
Global and Local Transactions

- **Local** transactions are those associated with a single resource manager, e.g. a relational database
- **Global** transactions are under the control of a **transaction manager**, typically provided an application server
- Global transactions allow for a distributed, two-phase commit across multiple transaction managers
- Unfortunately, local and global transactions have different programing models in Java, thus making it difficult to switch

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Distributed Transactions

- Some transaction managers allow for transactions to be work across different resource managers that adhere to industry standards for transactions



Distributed transactions are more complex than simple transactions and typically require a separate transaction manager that coordinates the individual resource managers' transaction services.

To prevent data corruption, distributed transactions require a process referred to as a two-phase commit</i>. In a two-phase commit, the transaction manager coordinates commits to ensure that all resource managers can successfully commit before committing the transaction. If they can't all commit, the transaction manager rolls back the transaction.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring and Transactions

- Spring provides the following support for transactions:
    - **Transaction managers** that abstract the underlying transaction architecture (i.e. local vs global)
    - An exception hierarchy based on **unchecked** exceptions
    - Either **programmatic** or **declarative** transaction demarcation

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Transaction Managers

- Spring provides **transaction managers** that you can configure via dependency injection, thus insulating applications from the actual transaction implementation and programing model
- The Spring transaction managers delegate to the underlying resource manager
- Spring provides several transaction managers, including:
    - DataSourceTransactionManager (basic JDBC)
    - HibernateTransactionManager
    - JpaTransactionManager
    - JtaTransactionManager
    - WebLogicJtaTransactionManger
    - WebSphereUowTransactionManager

Since you configure the transaction manager in configuration, it's easy to switch between them without changing your application code.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a DataSource Transaction Manager (XML)

- This is a local transaction manager that works with a database

```
1    <jee:jndi-lookup id="acctDataSource"
2      jndi-name="accountDS"
3      resource-ref="true" />
4
5    <bean id="transactionManager"
6      class=
7    "org.springframework.jdbc.datasource.DataSourceTransactionManager">
8      <property name="dataSource" ref="acctDataSource" />
9    </bean>
```


You can use this transaction manager in applications where all of the transactional data resides in a single database -- it doesn't support distributed transactions. This transaction manager basically just delegates to the database manager's own JDBC-based transaction service.

Note that you configure the transaction manager with the data source it needs to begin, commit and rollback transactions.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a DataSource Transaction Manager (Java Config)

```
1    @Configuration
2    @EnableTransactionManagement
3    public class MyConfig
4    {
5      @Bean
6      public DataSource acctDataSource()
7      {
8        JndiDataSourceLookup dsLookup = new JndiDataSourceLookup();
9        dsLookup.setResourceRef(true);
10       return  dsLookup.getDataSource("AccountDS");
11     }
12
13     @Bean
14     public DataSourceTransactionManager transactionManager()
15     {
16       return new DataSourceTransactionManager(acctDataSource());
17     }
18   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring a JTA Transaction Manager (XML)

- This transaction manager uses a application server's JTA service and thus typically supports global transactions (e.g. two-phase commit)
- This transaction manager can **enlist** multiple underlying resource managers (e.g. DataSource, Hibernate or JPA transaction managers)

```
<bean id="transactionManager" class=
"org.springframework.transaction.jta.JtaTransactionManager" />
```


This only works if the Spring application itself is deployed onto a JEE application server. The JTA transaction manager looks up the app server's JTA reference using a well known JNDI name, and then delegates to it. Since JTA can enlist resource managers, no other configuration is required.

Not all application servers support distributed transactions, but most do.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

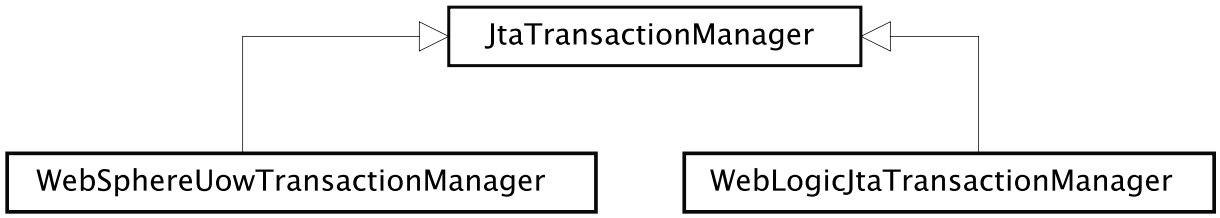
Configuring a JTA Transaction Manager (Java Config)

```
1    @Configuration
2    @EnableTransactionManagement
3    public class MyConfig
4    {
5      . . .
6
7      @Bean
8      public JtaTransactionManager transactionManager()
9      {
10       return new JtaTransactionManager();
11     }
12   }
```


Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Vendor-Specific Transaction Manager

- If your Spring application is running on BEA (Oracle) WebLogic or IBM WebSphere, Spring provides specialized transaction managers tailored for those application servers
- The tailored transaction managers support the full range of Spring transactional capability including transaction suspension



If you are using WebSphere, be sure to install the latest fixpacks.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Demarcation and Management

- Applications that use transactions must define where a transaction should start and where it should commit or rollback
- We refer to that process as **transaction demarcation**
- Spring lets you demarcate and configure transactions via configuration (declarative) or in the code itself (programmatic)
- This chapter covers only **declarative** transaction management

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Declarative Transaction Management

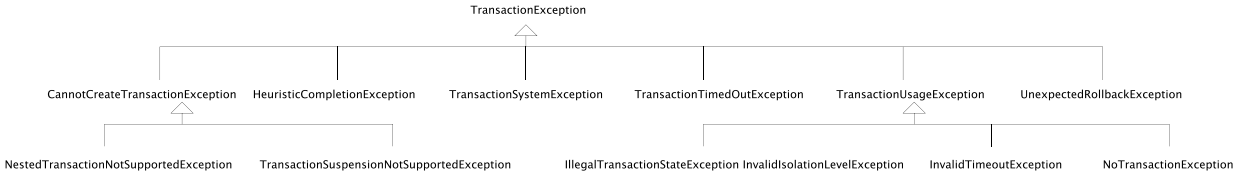
- According to the Spring docs, most Spring applications use declarative transactions since it affects code the least
- Spring lets you configure transactions either via XML configuration or via annotations in the code
- Behind the scenes, Spring uses **aspect oriented programming (AOP) proxies** to implement declarative transactions

AOP works behind the scenes to install "proxies" in front of transactional objects – the AOP proxies automatically begin and commit transactions so that your application code doesn't need to.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Transaction Exception Handling

- Since transaction exceptions are generally not recoverable, Spring provides **unchecked** exceptions that you can catch at a global level in your program and report the problem
- This saves you from writing messy try-catch blocks when using programmatic transaction management

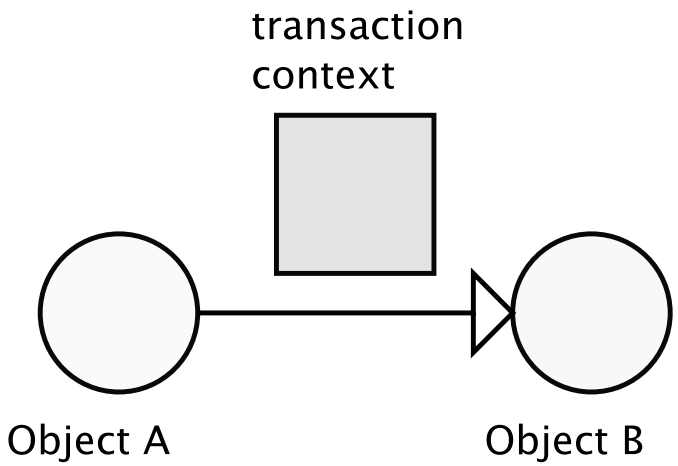


Using unchecked exceptions for unrecoverable errors is a general philosphy in Spring. It makes for more readable, maintainable code that's not cluttered with a lot of try-catch blocks.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Propagation

- Transaction **propagation** governs on how transactions flow from object to object
- Spring provides several **propagation attributes**, based on a similar notion in Enterprise JavaBeans



Objects typically call methods in other objects. The question is: when that happens, does the called method run in the same transaction as the caller? Spring provides "propagation attributes" so you can configure how called methods participate in transactions.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Propagation Attributes

Required If there is no existing transaction, start one. If transaction already exists, use existing

RequiresNew If there is no existing transaction, start one. If transaction exists, suspend it and start new

Mandatory If there is no existing transaction, throw exception. If transaction exists, use it

Supports If there is no existing transaction, run method without transaction. If transaction exists, use it

NotSupported If there is no existing transaction, run method without a transaction. If transaction exists, suspend existing transaction then run method with no transaction

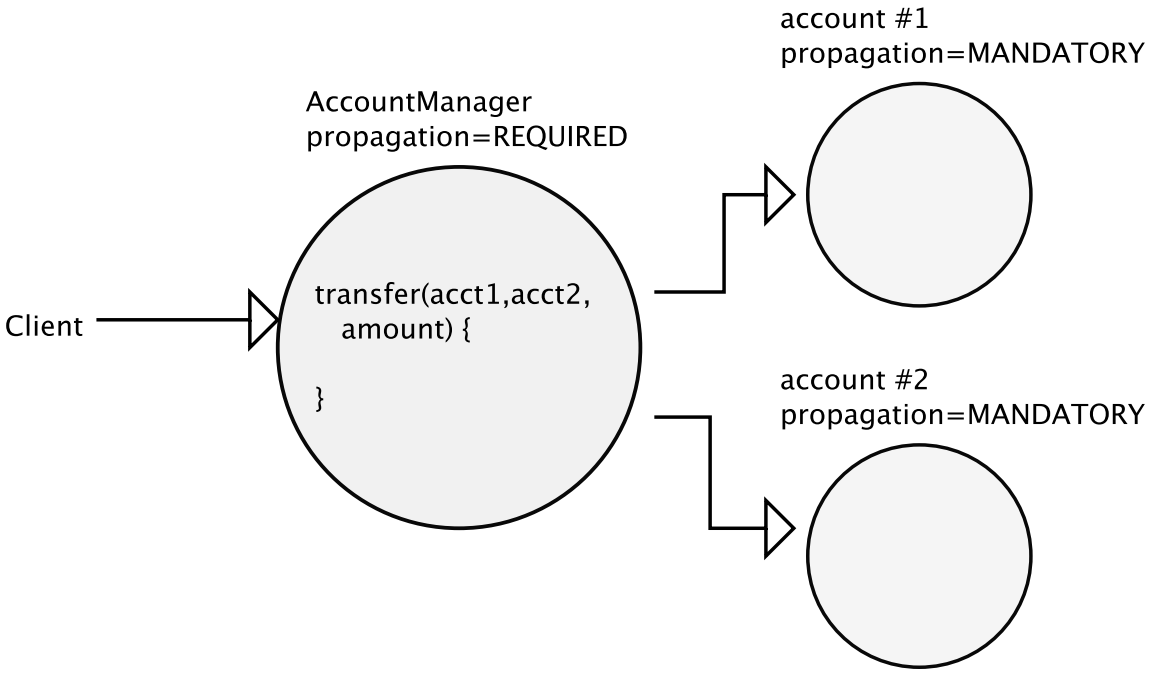
Never If there is no existing transaction, run method without a transaction. If transaction exists, throw RemoteException

---

Spring also defines a "Nested" transactional propagation attribute, but it's not well supported by resource managers, so it's not often used.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Propagation Example

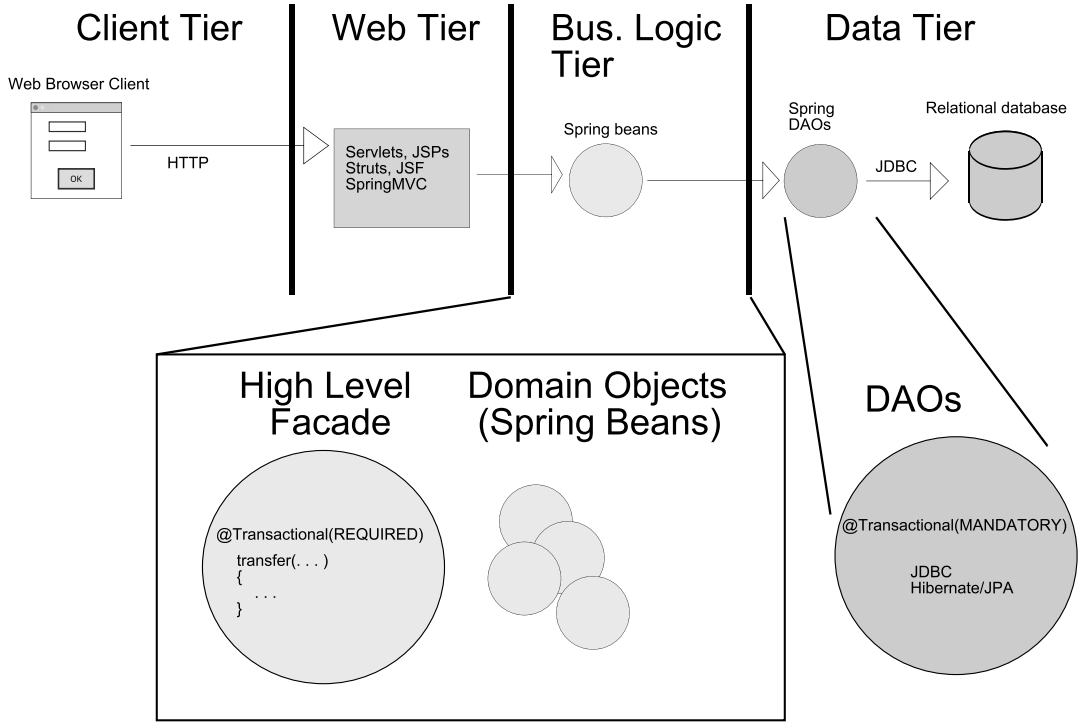


In this example, the "front" bean, AccountManager's transfer() method is configured with the REQUIRED attribute, so when a client calls it, the Spring transaction manager starts a new transaction. Then when the transfer method calls methods in the Account beans that are configured with MANDATORY, Spring enlists those objects in the running transaction.

The transaction commits when the transfer() method returns or rolls back automatically if an unchecked exception occurs.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Typical Architecture



When the Web tier calls methods in the facade, since it's annotated as REQUIRED, Spring starts a transaction and commits it when the method returns.

The facade delegates to Spring domain beans and DAOs which are annotated with MANDATORY, thus propogating the transaction.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Exceptions and Rolling Back

- Spring automatically rolls back a transaction if an **unchecked** exception occurs, but not for **checked** exceptions
- You can change this behavior on a method-by-method basis during configuration

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Timeouts

- Most resource managers define a **timeout** after which a running transaction is automatically aborted (rolled back)
- Spring lets you configure this timeout value
- The default is whatever default the resource manager has, or zero (no timeout) for resource managers that don't support timeouts

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Isolation

- Completely isolated transactions are robust but complete isolation can impact performance
- Spring lets you set an **isolation level** that relaxes isolation at the risk of allowing transactions to "see" intermediate results of other uncommitted transactions

The problems that can occur are:

```
Dirty reads
Phantom reads
Non-repeatable reads
```

The Wikipedia has a nice discussion of these problems at:

```
http://en.wikipedia.org/wiki/Transaction_isolation
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Isolation Levels

- The TransactionDefinition interface defines several constants for isolation:

- **ISOLATION_DEFAULT** -- Use resource manager default
- **ISOLATION_READ_COMMITED** -- *Phantom Read* and *Non-repeatable Read* inconsistencies may occur
- **ISOLATION_READ_UNCOMMITED** -- *Dirty Read*, *Phantom Read* and *Non-repeatable Read* inconsistencies may occur
- **ISOLATION_REPEATABLE_READ** -- *Phantom Read* inconsistencies may occur
- **ISOLATION_SERIALIZABLE** -- Transaction is completed isolated

This interface is in the org.springframework.transaction package.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Read Only Transactions

- In some cases, a resource manager can optimize access if the transaction will not modify data
- Spring lets you configure transactions as **read-only** as a hint to the resource manager

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Specifying Transaction Attributes

- Spring lets you specify transaction propagation attributes either via:
    - Annotations (Spring 2.x or later)
    - XML configuration
- It's generally simpler to use annotations, so that's what we cover in this chapter

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transaction Annotations

- Instead of, or in addition to using XML configuration, Spring 2.x supports configuring transactions by annotations in the code itself
- You must use the **annotation-driven** element in the configuration file for transactional annotations to work

```
1    Spring configuration file:
2    -------------------------
3    <beans ...
4      xmlns:tx="http://www.springframework.org/schema/tx"
5       ...>
6
7      <tx:annotation-driven
8        transaction-manager="transactionManager" />
9
10   </beans>
```


The "annotation-driven" element instructs Spring to process annotations in your source files. If you forget this line in the configuration file, your annotations will have no effect.

If you are using Java configuration, you can accomplish the same goal using the EnableTransactionManagement annotation on your configuration class.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Transactional Annotation

```
1    @Transactional(
2      propagation=Propagation.MANDATORY)
3    public class MyBean
4    {
5      @Transactional(readOnly=true,
6        rollbackFor=MyException.class)
7      public void myMethod() throws MyException
8      {}
9    }
```

Element	Type	Default
propagation	enum on Propagation type	PROPAGATION_REQUIRED
isolation	enum on Isolation type	ISOLATION_DEFAULT
readOnly	boolean	false
timeout	int (in seconds)	resource manager default
rollbackFor	array of Class objects	none
noRollbackFor	array of Class objects	none

@Transactional is the workhorse annotation for transactions in Spring. You can place on a class definition, in which case it applies to all of the public methods in the class, or individually on public methods to configure each method separately. Note that you can annotation on both a class and a method - if there's a conflict, the method takes precedence.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Declarative Transaction Example

Account.java

AccountManager.java

AccountManagerImpl.java

AccountDAO.java

JdbcAccountDAO.java

NoMoneyException.java

TestTransactions.java

web.xml

AccountConfig.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package acct;
2
3    public class Account
4    {
5        private Long accountNumber;
6        private double balance;
7        private String holderName;
8
9        public Long getAccountNumber()
10       {
11           return accountNumber;
12       }
13       public void setAccountNumber(Long accountNumber)
14       {
15           this.accountNumber = accountNumber;
16       }
17       public double getBalance()
18       {
19           return balance;
20       }
21       public void setBalance(double balance)
22       {
23           this.balance = balance;
24       }
25       public String getHolderName()
26       {
27           return holderName;
28       }
29       public void setHolderName(String holderName)
30       {
31           this.holderName = holderName;
32       }
33
34       @Override
35       public String toString()
36       {
37           return "Acct#: " + accountNumber + ", holder: " + holderName + ", balance:
38       }
39   }
```

```
1    package acct;
2
3    public interface AccountManager
4    {
5        public void transfer(int acct1, int acct2, double amount)
6            throws NoMoneyException;
7        public String getAccountInfo(int accountNumber);
8    }
```

```
1    package acct;
2
3    import org.springframework.beans.factory.annotation.Autowired;
4    import org.springframework.stereotype.Component;
5    import org.springframework.transaction.annotation.Propagation;
6    import org.springframework.transaction.annotation.Transactional;
7
8    @Component("accountMgr")
9    public class AccountManagerImpl
10       implements AccountManager
11   {
12       private AccountDAO accountDAO;
13
14       @Autowired
15       public void setAccountDAO(AccountDAO accountDAO)
16       {
17           this.accountDAO = accountDAO;
18       }
19
20       @Transactional(rollbackFor=NoMoneyException.class, timeout=10,
21               propagation=Propagation.REQUIRED)
22       public void transfer(int acct1, int acct2, double amount)
23           throws NoMoneyException
24       {
25           Account a1 = accountDAO.getAccountByID(acct1);
26           Account a2 = accountDAO.getAccountByID(acct2);
27
28           double a1Balance = a1.getBalance();
29           if (a1Balance - amount < 0)
30               throw new NoMoneyException();
31
32           double a2Balance = a2.getBalance();
33           a1Balance -= amount;
34           a2Balance += amount;
35           a1.setBalance(a1Balance);
36           a2.setBalance(a2Balance);
37           accountDAO.updateAccount(a1);
38           accountDAO.updateAccount(a2);
39       }
40
41       @Transactional(readOnly=true)
42       public String getAccountInfo(int accountNumber)
43       {
44           Account a = accountDAO.getAccountByID(accountNumber);
45           return a.toString();
```

```
46          }
47      }
```

```
1    package acct;
2
3    public interface AccountDAO
4    {
5        public Account getAccountByID(int accountNumber);
6        public void updateAccount(Account a);
7    }
```

```
1    package acct;
2
3    import java.sql.ResultSet;
4    import java.sql.SQLException;
5
6    import javax.sql.DataSource;
7
8    import org.springframework.beans.factory.annotation.Autowired;
9    import org.springframework.jdbc.core.JdbcTemplate;
10   import org.springframework.jdbc.core.RowMapper;
11   import org.springframework.stereotype.Repository;
12   import org.springframework.transaction.annotation.Propagation;
13   import org.springframework.transaction.annotation.Transactional;
14
15   @Repository("accountDao")
16   public class JdbcAccountDAO implements AccountDAO
17   {
18       private static class AccountRowMapper implements
19               RowMapper<Account>
20       {
21           public Account mapRow(ResultSet rs, int index) throws SQLException
22           {
23               Account acct = new Account();
24               acct.setAccountNumber(rs.getLong("ACCT_NUM"));
25               acct.setBalance(rs.getDouble("BALANCE"));
26               acct.setHolderName(rs.getString("HOLDERNAME"));
27               return acct;
28           }
29       }
30
31       private JdbcTemplate template;
32       private AccountRowMapper mapper;
33
34       @Autowired
35       public void setDataSource(DataSource ds)
36       {
37           template = new JdbcTemplate(ds);
38           mapper = new AccountRowMapper();
39       }
40
41       @Transactional(propagation=Propagation.MANDATORY)
42       public void updateAccount(Account a)
43       {
44           String SQL = "UPDATE account SET balance=?,holdername=? WHERE acct_num=?";
45           template.update(SQL, a.getBalance(), a.getHolderName(),
```

```
46                      a.getAccountNumber());
47          }
48
49          @Transactional(propagation=Propagation.MANDATORY)
50          public Account getAccountByID(int accountNumber)
51          {
52              String SQL = "SELECT * FROM account WHERE ACCT_NUM=?";
53              return template.queryForObject(SQL,
54                      mapper, accountNumber);
55          }
56      }
```

```
1    package acct;
2
3    @SuppressWarnings("serial")
4    public class NoMoneyException extends Exception
5    {
6        public NoMoneyException()
7        {
8            super("Insufficient funds for transfer");
9        }
10   }
```

```
1   package servlets;
2
3   import java.io.IOException;
4   import java.io.PrintWriter;
5
6   import javax.servlet.ServletContext;
7   import javax.servlet.ServletException;
8   import javax.servlet.annotation.WebServlet;
9   import javax.servlet.http.HttpServletRequest;
10  import javax.servlet.http.HttpServletResponse;
11
12  import org.springframework.web.context.WebApplicationContext;
13  import org.springframework.web.context.support.WebApplicationContextUtils;
14
15  import acct.AccountManager;
16
17  @SuppressWarnings("serial")
18  @WebServlet("/TestTransactions")
19  public class TestTransactions extends javax.servlet.http.HttpServlet
20  {
21      protected void doGet(HttpServletRequest request,
22              HttpServletResponse response) throws ServletException, IOException
23      {
24          response.setContentType("text/html");
25          PrintWriter out = response.getWriter();
26
27          ServletContext servletContext = getServletContext();
28          WebApplicationContext ctx = WebApplicationContextUtils
29                  .getRequiredWebApplicationContext(servletContext);
30          try
31          {
32              AccountManager mgr = (AccountManager)ctx.getBean("accountMgr");
33
34              out.println("Before transfer: <br>");
35              out.println(mgr.getAccountInfo(1) + "<br>");
36              out.println(mgr.getAccountInfo(2) + "<br>");
37
38              mgr.transfer(1, 2, 1);
39
40              out.println("After transfer: <br>");
41              out.println(mgr.getAccountInfo(1) + "<br>");
42              out.println(mgr.getAccountInfo(2) + "<br>");
43          }
44          catch (Exception e)
45          {
```

```
46                out.println("Something went wrong: " + e);
47            }
48        }
49    }
```

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns
  <display-name>TestTransactionsTomcatWeb</display-name>
  <resource-ref>
    <res-ref-name>jdbc/account</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listene
  </listener>
  <context-param>
    <param-name>contextClass</param-name>
    <param-value>org.springframework.web.context.support.AnnotationConfigWebApplic
  </context-param>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>config.AccountConfig</param-value>
  </context-param>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

```
1    package config;
2
3    import javax.sql.DataSource;
4
5    import org.springframework.context.annotation.Bean;
6    import org.springframework.context.annotation.ComponentScan;
7    import org.springframework.context.annotation.Configuration;
8    import org.springframework.jdbc.datasource.DataSourceTransactionManager;
9    import org.springframework.jdbc.datasource.lookup.JndiDataSourceLookup;
10   import org.springframework.transaction.annotation.EnableTransactionManagement;
11
12   @Configuration
13   @ComponentScan("acct")
14   @EnableTransactionManagement
15   public class AccountConfig
16   {
17       @Bean
18       public DataSource acctDataSource()
19       {
20         JndiDataSourceLookup dsLookup =
21             new JndiDataSourceLookup();
22         dsLookup.setResourceRef(true);
23         DataSource dataSource = dsLookup.getDataSource(
24             "jdbc/account");
25         return dataSource;
26       }
27
28         @Bean
29         public DataSourceTransactionManager transactionManager()
30         {
31           return new DataSourceTransactionManager(acctDataSource());
32         }
33   }
```

Chapter Summary

In this chapter, you learned:

- The basics of how transactions work
- About Spring transaction managers
- How to configure and control transactions in Spring

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

WSDL Fundamentals

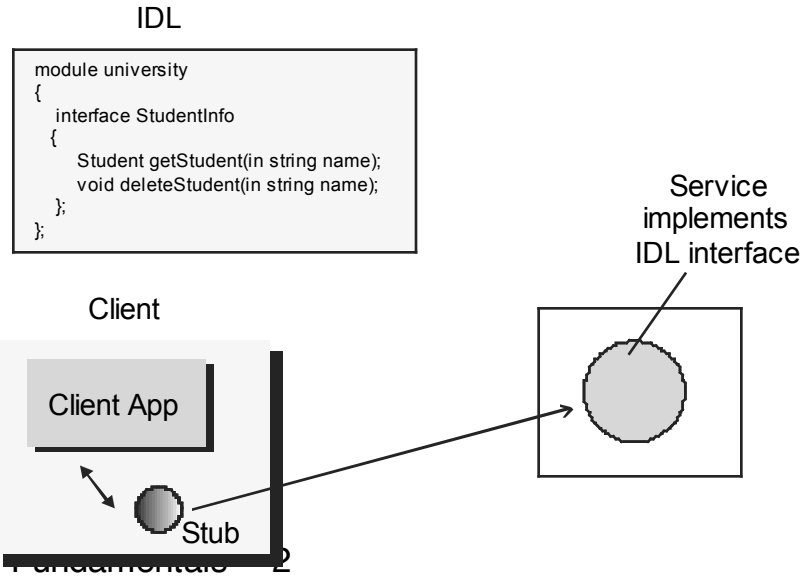
- What is WSDL?
- WSDL Structure
- WSDL and Code Generation

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is an IDL?

- An **interface definition language** lets you describe the function of a software system without exposing anything about its implementation
- Well-known IDLs include CORBA IDL and Microsoft IDL



IDLs are most often used with distributed computing systems, where a caller needs to understand how to invoke a service, but shouldn't depend on the service's implementation. Note that if the IDL itself is not tied to any programming language, then there's at leas the possibility that the client and sender could be written in different programming languages.

Java itself doesn't define an IDL, but the RMI infrastructure requires you to write a Java interface that plays the same role as an IDL, though not in a language-independent fashion.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is WSDL?

- The **Web Services Description Language** is the standard IDL for describing how to invoke a Web service
- WSDL is a W3C recommendation at version 2.0, but version 1.1 is still the most widely used
- This chapter covers WSDL 1.1



WSDL files are XML documents that follow the XML schema defined by the W3C.

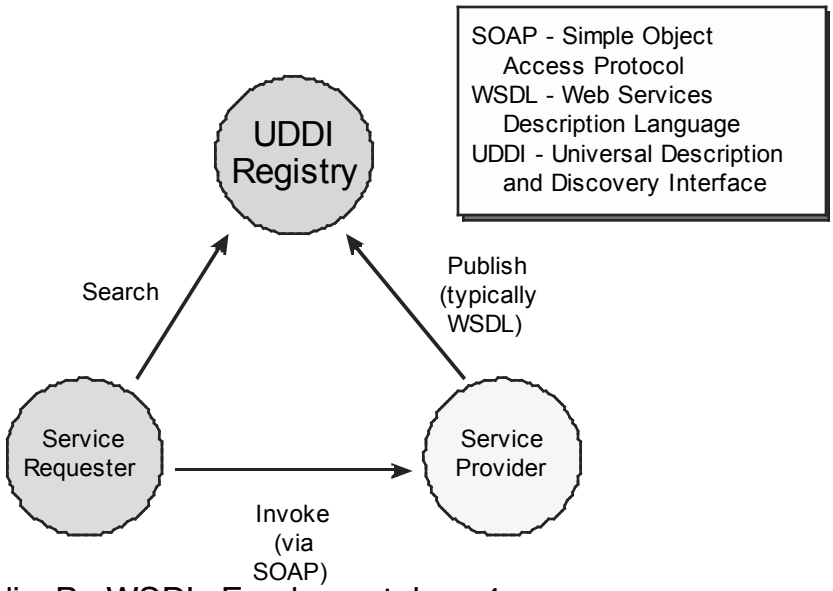
From the 1.1 draft:

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

WSDL and SOA

- WSDL is a key part of using Web services for SOA -- WSDL lets a client determine how to issue the SOAP messages to invoke the service



Appendix B: WSDL Fundamentals - 4

As we've already seen, WSDL is an integral component of a service-oriented architecture using Web services.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

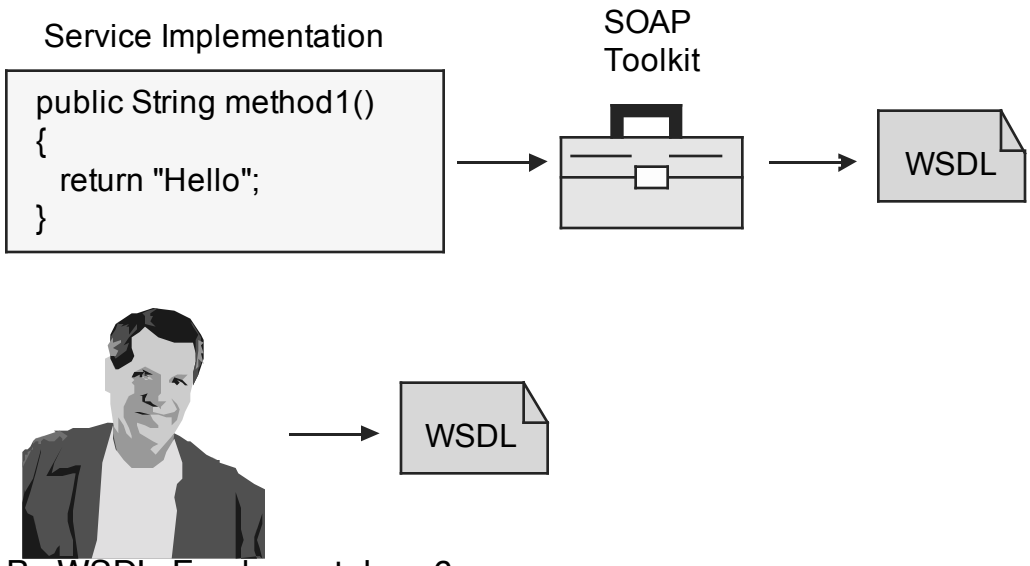
History of WSDL

- IBM created a language known as Network Accessible Service Specification Language (NASSL) and Microsoft created the SOAP Contract Language (SCL)
- The community worked together to combine the two to create WSDL

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating WSDL

- In most cases, WSDL is generated by tools when you create a Web service
- In a top-down approach, an architect could write WSDL to define the interface to a service

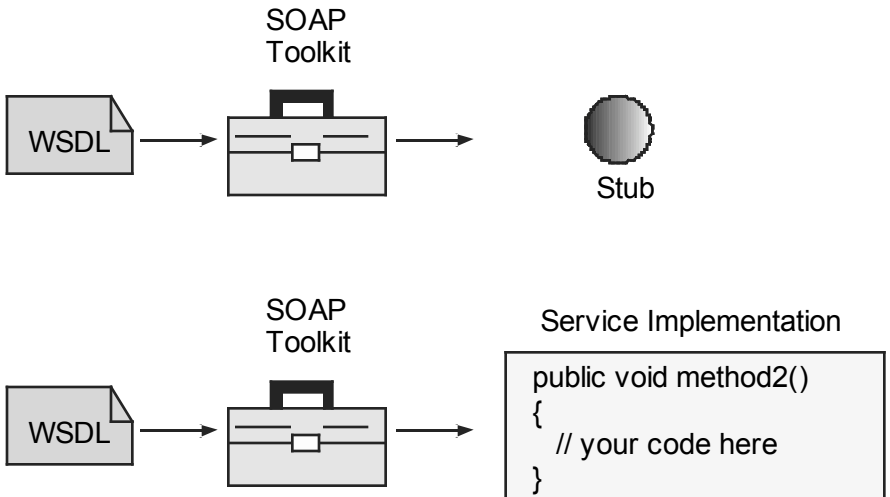


It is possible to write WSDL from scratch, but in many cases WSDL will be generated and consumed by programs.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using WSDL

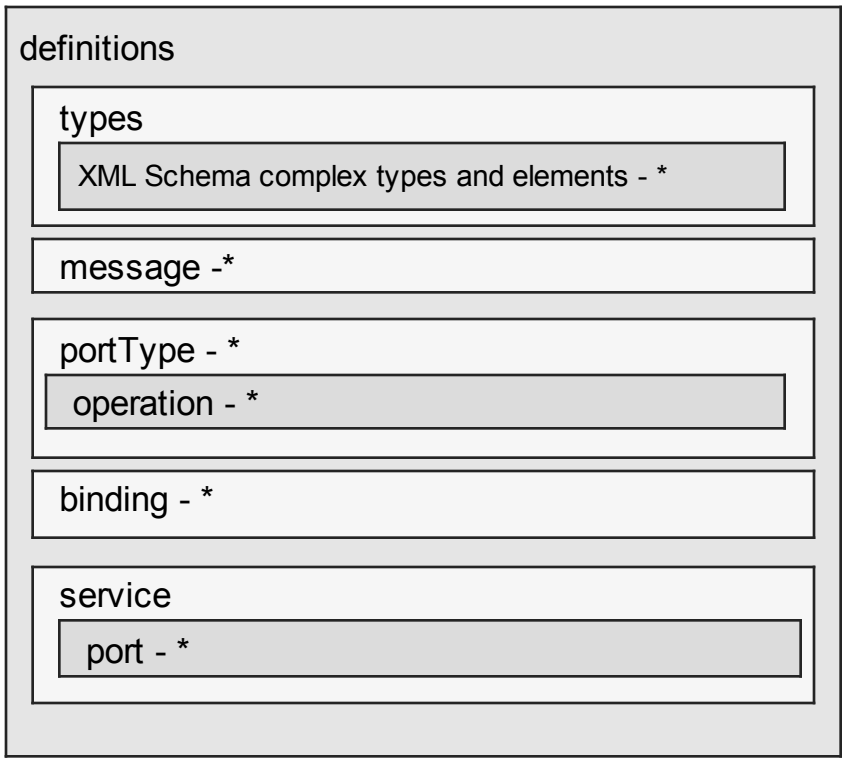
- From WSDL, most Web service toolkits can generate:
  - Client bindings (a.k.a. a **stubs** or **proxies**) in the client's programming language
  - A starter service (top-down approach) in the implementor's programming language



Most Web service toolkits contain either command-line utilities or GUI "wizards" that can generate code from WSDL.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Structure of a WSDL Document



A WSDL's root element is "definitions". The "definitions" element has several direct child elements as described below.

The "types" element is optional and contains zero or XML schema definitions of complex types or schema elements. You only need the "types" section if the messages use something other than the predefined XML schema data types.

The WSDL defines a "message" element for each of the messages for the Web service.

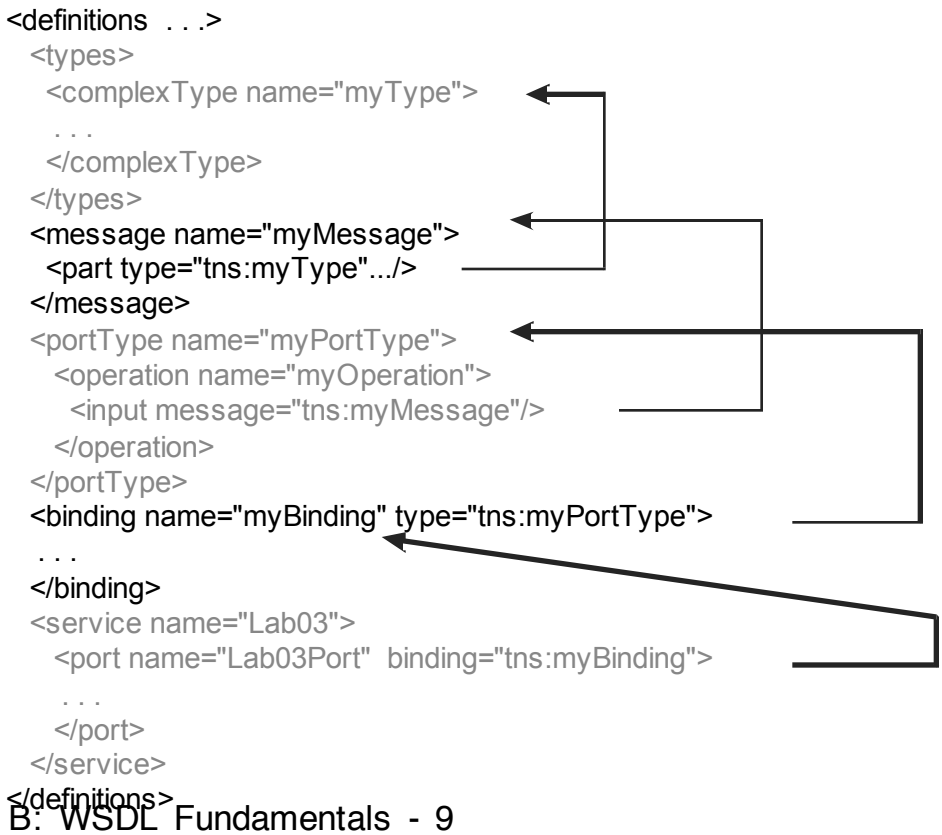
The "portType" element is like a Java interface in that it defines one or more "operation" elements, which are similar to Java methods.

The "binding" element specifies how to invoke the operations (e.g. which transport to use). A single service can have multiple bindings (e.g. one for HTTP, one for JMS).

The "service" element defines a "port" for each binding -- the "port" specifies the endpoint of the service (its invocation address).

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Structure of a WSDL Document, cont'd



This figure shows how the sections of the WSDL reference each other. Within a "message" element, a "part" element can point to a complex type defined in the "types" section. (The "part" element is similar to a parameter on a Java method.)

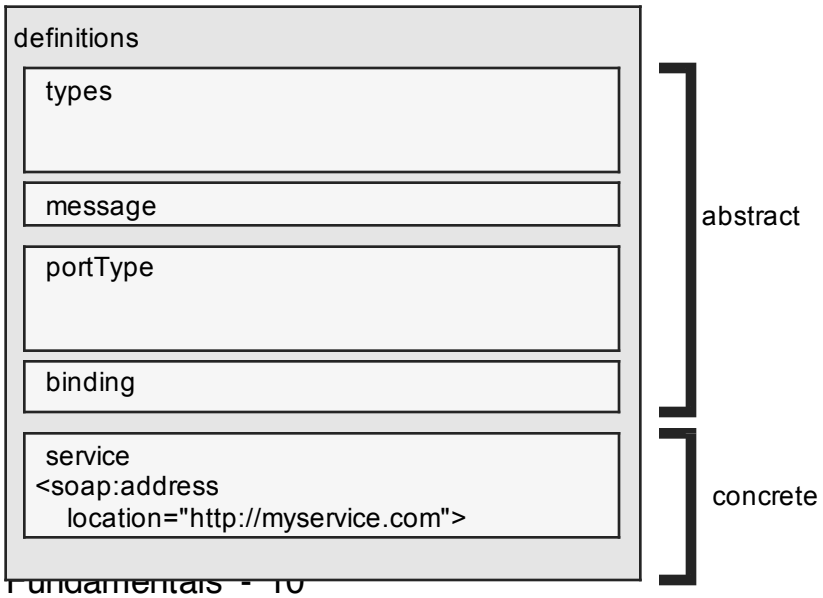
Likewise, each "operation" element gathers one or more "message" elements. In most cases, especially in RPC, an operation will have both an "input" and a "output" message.

Each "binding" references a "portType", and each "port" within the "service" references a "binding".

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Abstract versus Concrete WSDL

- The **service** section of the specifies the endpoint and thus is considered concrete, while the rest of the WSDL is considered an abstract description of the service



This distinction between abstract and concrete works as follows: The abstract portion defines the interface to the service (the "portType") and the specifics of invoking the service (the "binding"), but doesn't specify the invocation address of the service. That address resides in the "service" element, which represents a concrete implementation of the abstract service.

This distinction becomes useful when you think about having multiple service providers implementing some standard service -- they would all use the same interface, but have different endpoints.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Definitions Element

- The **definitions** element is the root element of the WSDL and typically defines several namespaces

```
1    <definitions  xmlns:tns="http://example.com/Lab03"
2      xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
3      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
4      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
5      xmlns="http://schemas.xmlsoap.org/wsdl/"
6      targetNamespace="http://example.com/Lab03">
7
8      . . .
9    </definitions>
```


The "definitions" element acts as the root element of the WSDL and defines the namespace prefixes used within the document. Note that it defines a prefix, "tns" (a.k.a "this name space"), and a "targetnamespace", which normally maps to the namespace of the service itself.

Note that in this example, the default namespace is the WSDL namespace.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Types Section

- The **types** section defines any special types that are not defined in the standard XML Schema Datatypes specification

```
1     <types>
2      <xsd:schema
3        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4        xmlns:stns="java:lab03"
5        elementFormDefault="qualified"
6        attributeFormDefault="qualified"
7        targetNamespace="java:lab03">
8       <xsd:complexType     name="Person">
9        <xsd:sequence>
10        <xsd:element       type="xsd:string"
11          name="name" minOccurs="1" maxOccurs="1">
12        </xsd:element>
13       </xsd:sequence>
14      </xsd:complexType>
15     </xsd:schema>
16    </types>
```

The "types" section defines the types used in messages. In WSDL 1.1, you don't need a "types" section if all of the types are built-in XML Schema datatypes.

If the service is for an RPC, then this section defines the types of parameters and return values. If it's a document-style service, then the format of the documents are defined here.

It's also possible, though not shown here, for the WSDL to "import" external schemas for use in the WSDL.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Messages Section

- The **message** elements define all of the messages sent or received by the Web service

```
1     <message   name="findSam">
2     </message>
3     <message   name="findSamResponse">
4      <part    xmlns:partns="java:lab03"
5        type="partns:Person"
6        name="result">
7      </part>
8     </message>
9     <message   name="NoSuchPersonException">
10     <part    xmlns:partns="java:lab03"
11       type="partns:NoSuchPersonException"
12       name="NoSuchPersonException">
13     </part>
14    </message>
```

For messages for RPC, the "part" child elements are analogous to parameters to a Java method.

In WSDL 2.0, there are no "message" elements -- instead, the messages are defined in the "types" element.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The PortType Section

- The **portType** section is similar in concept to a Java interface -- it defines a set of **operations** and the messages that comprise them

```
1     <portType   name="Lab03Port">
2      <operation    name="findSam">
3       <input     message="tns:findSam">
4       </input>
5       <output     message="tns:findSamResponse">
6       </output>
7       <fault     name="NoSuchPersonException"
8         message="tns:NoSuchPersonException">
9       </fault>
10     </operation>
11    </portType>
```

```
Person findSam()
    throws NoSuchPersonException
```


As you learn to read WSDL, you will become adept at mentally translating WSDL portTypes into Java interfaces.

In WSDL 2.0, the name of this element is "interface", rather than "portType".

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Bindings Section

- The **bindings** section describes the "how" of invoking the service -- it specifies the transport and the **style** and **use**

```
1     <binding   type="tns:Lab03Port"
2       name="Lab03Port">
3      <soap:binding    style="rpc"
4        transport=
5          "http://schemas.xmlsoap.org/soap/http">
6      </soap:binding>
7      <operation    name="findSam">
8       <soap:operation     style="rpc"
9         soapAction="">
10     </soap:operation>
```


The "binding" specifies the details on invoking a "portType" -- in particular, it defines whether it's a document or RPC-style service and specifies a URI that identifies the transport for the service -- in this case, HTTP.

The "binding" then continues for each operation, specifying details about the operation.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Bindings Section, cont'd

```
11      <input>
12       <soap:body namespace="http://example.com/Lab03"
13         encodingStyle=
14           "http://schemas.xmlsoap.org/soap/encoding/"
15         use="encoded">
16       </soap:body>
17      </input>
18      <output>
19       <soap:body namespace="http://example.com/Lab03"
20         encodingStyle=
21           "http://schemas.xmlsoap.org/soap/encoding/"
22         use="encoded">
23       </soap:body>
24      </output>
```


Continuing the "binding" for an operation ("findSam"), the WSDL describes whether the message is encoded or literal, and if encoded, as this one is, a URI for the encoding scheme used. In this case, the standard SOAP "Section 5" encoding scheme is in use.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Bindings Section, cont'd

```
21      <fault name="NoSuchPersonException">
22       <soap:fault namespace="http://example.com/Lab03"
23         name="NoSuchPersonException"
24         encodingStyle=
25           "http://schemas.xmlsoap.org/soap/encoding/"
26         use="encoded">
27       </soap:fault>
28      </fault>
29     </operation>
30    </binding>
```


Finishing the "binding" for the "findSam" operation, this WSDL fragment indicates that the fault message is encoded using SOAP "Section 5" encoding.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Service Section

- The **service** section defines a **port** and assigns an endpoint to the service

```
1     <service   name="Lab03">
2      <port    name="Lab03Port"
3        binding="tns:Lab03Port">
4       <soap:address
5          location="http://localhost:7001/Lab03/Lab03">
6       </soap:address>
7      </port>
8     </service>
```

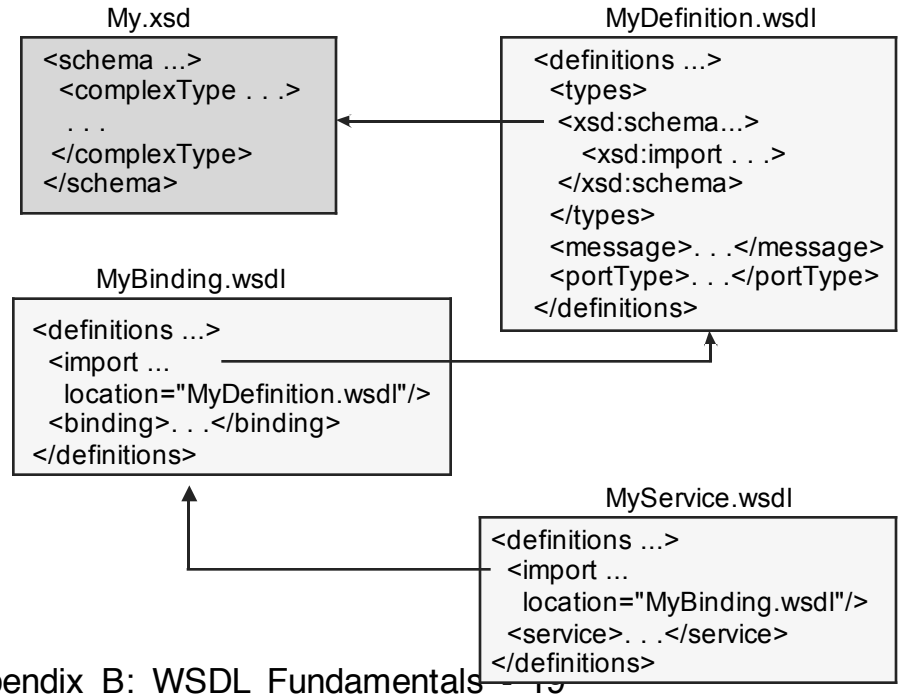

The "service" element defines the concrete portion of the WSDL since it specifies the actual endpoint of the service. Note that in this case, the endpoint is an HTTP URL, which matches the transport specified in the "binding".

In WSDL 2.0, the "port" element is renamed to "endpoint".

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Factoring WSDL

- To allow re-use of WSDL elements, especially types, WSDL defines an **import** facility



So you can re-use parts of the WSDL (and to effectively store WSDL in a UDDI registry), the WSDL defines an "import" element that lets you reference types or definitions defined externally.

Note that for compliance with the WS-Interoperability specification, you should use the schema import facility rather than the WSDL import to reference external schema definitions.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

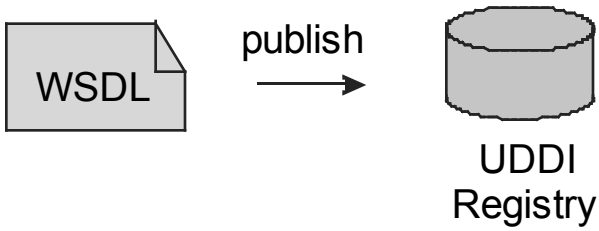
WSDL 2.0 Overview

- WSDL 2.0 was originally named WSDL 1.2, but the W3C considered it different enough that they incremented the version number
- Some of the changes from version 1.1:
    - The **targetNamespace** attributes is required
    - No **message** elements; replaced by XML Schema *types*
    - The **portType** element renamed to **interface**
    - The **port** element renamed to **endpoint**

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using WSDL with UDDI

- The OASIS standards body that administers UDDI has published a "best practice" document that details how to store WSDL in a UDDI registry (it's not as straightforward as you might think)



In a later chapter, we will discuss the issues involved in publishing WSDL to UDDI, especially for standard services that will be implemented by multiple providers.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

WS-Policy Overview

- WSDL describes the basic interaction between a client and a service via SOAP, but doesn't cover things like:
- How security works
- Whether the service implements a reliable messaging protocol
- The **WS-Policy** specification provides an extensible framework for documenting these sorts of policies

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The fundamentals of IDLs and WSDL
- How to generate code from WSDL
- About the syntax of WSDL 1.1

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum