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

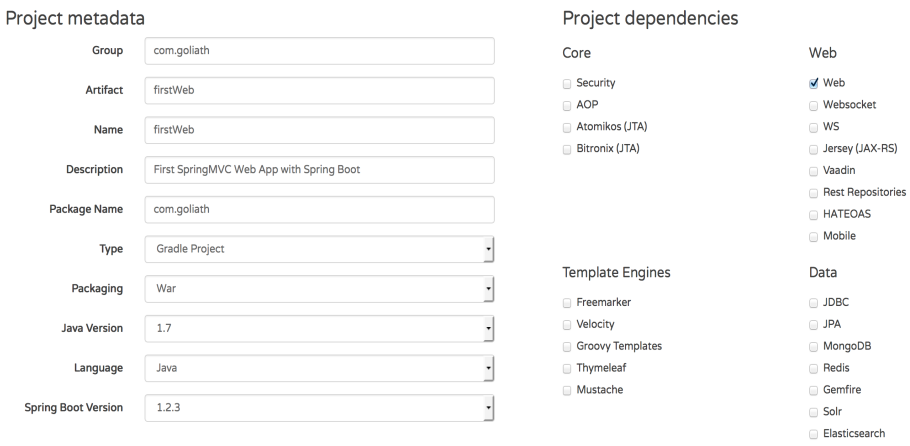
- Spring Boot supports both Maven and Gradle and can generate starter build scripts for both tools
- You can read more about Maven at: https://maven.apache.org/
- You can read more about Gradle at: https://gradle.org/

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



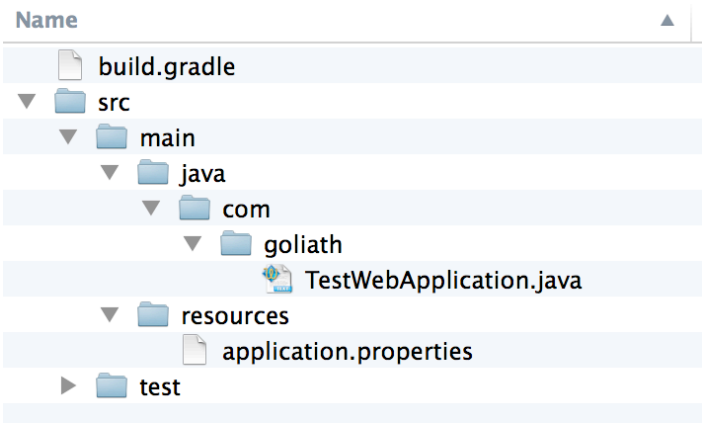
The Intitializr generates a ZIP file that you can download and unzip.

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.

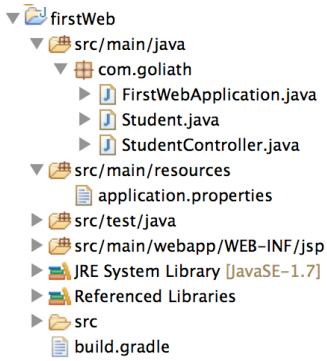
Repositories are sources of dependencies, i.e. libraries. You can use Maven, Ivy or locally defined repositories.

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

Configuring SpringMVC JSP Views

- It's traditional in Spring MVC to place JSPs in the WEB-INF folder
- For this to work with Spring Boot, you can update the **application.properties** to specify the exact location

```
1    spring.mvc.view.prefix: /WEB-INF/jsp/
2    spring.mvc.view.suffix: .jsp
```


Of course, you'll also need to create this directory in your project. You should also ensure that there's no trailing spaces at the end of any of the lines in application.properties.

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

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
- If you specify the *thymeleaf* starter dependency, Spring Boot autoconfigures Thymeleaf and expects view templates to be in the src/main/resources/templates folder



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

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

Complete Example

notes.txt

build.gradle

application.properties

FirstWebApplication.java

Student.java

StudentController.java

studentList.jsp

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

```
- Download, install gradle and put its "bin" directory
  in PATH

- Create the project using Web-based Spring Boot Initializr:
    Metadata:
        Group:  com.goliath
        Artifact: firstWeb
        Name: firstWeb
        Description: First SpringMVC Web App with Spring Boot
        Package Name: com.goliath
        Type: Gradle Project
        Packaging: JAR
        Java Version: 1.7
        Language: Java
        Spring Boot Version: 1.2.3

    Project dependencies:
        Web

- Generate Project, then unzip

- In terminal (command prompt), change to project directory

- Generate Eclipse project:

    gradle eclipse

- Import "existing project" into Eclipse

- In com.goliath package, create class Student.

- In com.goliath package, create StudentController

- Add to application.properties:

spring.view.prefix: /WEB-INF/jsp/
spring.view.suffix: .jsp

- Create source folder: src/main/webapp/WEB-INF/jsp

- In the new folder, create studentList.jsp

- In gradle.build, add plugin:

    apply plugin: 'war'
```

```
46
47   - In gradle.build, add dependencies:
48
49       providedRuntime("org.apache.tomcat.embed:tomcat-embed-jasper")
50       providedRuntime("javax.servlet:jstl")
51
52   - In terminal:
53
54      gradle bootRun
55
56   - In browser, navigate to http://localhost:8080/studentList
57
58
```

```
1   buildscript {
2       ext {
3           springBootVersion = '1.2.3.RELEASE'
4       }
5       repositories {
6           mavenCentral()
7       }
8       dependencies {
9           classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBoot
10          classpath("io.spring.gradle:dependency-management-plugin:0.5.0.RELEASE")
11      }
12  }
13
14  apply plugin: 'java'
15  apply plugin: 'eclipse'
16  apply plugin: 'idea'
17  apply plugin: 'spring-boot'
18  apply plugin: 'io.spring.dependency-management'
19  apply plugin: 'war'
20
21
22  jar {
23      baseName = 'firstWeb'
24      version = '0.0.1-SNAPSHOT'
25  }
26  sourceCompatibility = 1.7
27  targetCompatibility = 1.7
28
29  repositories {
30      mavenCentral()
31  }
32
33
34  dependencies {
35      compile("org.springframework.boot:spring-boot-starter-web")
36      testCompile("org.springframework.boot:spring-boot-starter-test")
37      providedRuntime("org.apache.tomcat.embed:tomcat-embed-jasper")
38      providedRuntime("javax.servlet:jstl")
39  }
40
41
42  eclipse {
43      classpath {
44           containers.remove('org.eclipse.jdt.launching.JRE_CONTAINER')
45           containers 'org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.inter
```

```
46      }
47  }
48
49  task wrapper(type: Wrapper) {
50      gradleVersion = '2.3'
51  }
```

```
1    spring.view.prefix: /WEB-INF/jsp/
2    spring.view.suffix: .jsp
```

```
1    package com.goliath;
2
3    import org.springframework.boot.SpringApplication;
4    import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6    @SpringBootApplication
7    public class FirstWebApplication {
8
9        public static void main(String[] args) {
10           SpringApplication.run(FirstWebApplication.class, args);
11       }
12   }
```

```
1    package com.goliath;
2
3    public class Student
4    {
5        private int studentID;
6        private String name;
7        private double gpa;
8
9        public Student()
10       {
11       }
12
13       public Student(int studentID, String name, double gpa)
14       {
15           this.studentID = studentID;
16           this.name = name;
17           this.gpa = gpa;
18       }
19
20       public int getStudentID()
21       {
22           return studentID;
23       }
24
25       public void setStudentID(int studentID)
26       {
27           this.studentID = studentID;
28       }
29
30       public String getName()
31       {
32           return name;
33       }
34
35       public void setName(String name)
36       {
37           this.name = name;
38       }
39
40       public double getGpa()
41       {
42           return gpa;
43       }
44
45       public void setGpa(double gpa)
```

```
46        {
47            this.gpa = gpa;
48        }
49
50    }
```

```
1    package com.goliath;
2
3    import java.util.ArrayList;
4    import java.util.List;
5
6    import org.springframework.stereotype.Controller;
7    import org.springframework.ui.Model;
8    import org.springframework.web.bind.annotation.RequestMapping;
9    import org.springframework.web.bind.annotation.RequestMethod;
10
11   @Controller
12   public class StudentController
13   {
14
15       @RequestMapping(value = "/studentList", method = RequestMethod.GET)
16       public String getStudents(Model model)
17       {
18           List<Student> students = new ArrayList<Student>();
19
20           students.add(new Student(12, "Sue Smith", 3.44));
21           students.add(new Student(87, "Harry Wolfe", 2.68));
22           students.add(new Student(947, "Jerry Olde", 3.12));
23
24           model.addAttribute("students", students);
25
26           return "studentList";
27       }
28   }
```

```
1    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2    <DOCTYPE html>
3    <html lang="en">
4      <head>
5        <title>Student List</title>
6      </head>
7      <body>
8        <h1>Student List</h1>
9          <ul>
10           <c:forEach items="${students}" var="aStudent">
11             <li>${aStudent.name}</li>
12           </c:forEach>
13         </ul>
14     </body>
15   </html>
```

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

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