Spring and REST

- Spring Support for REST
- Message Converters
- Annotations

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. 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, 2001-2014. 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

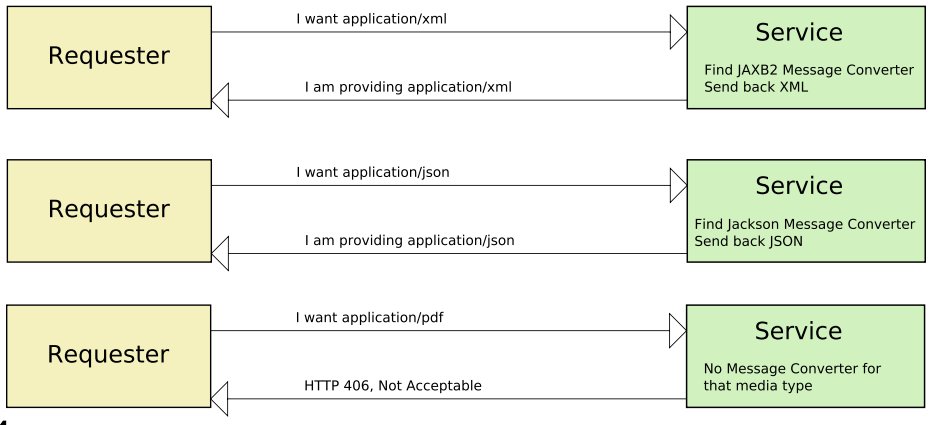
You must configure the JARs for JAXB2 and Jackson on the application's CLASSPATH to use those converters. Note that full JEE application servers like WebSphere include JAXB2 and might also include Jackson. If you're not running on a full JEE server, you can obtain Jackson from http://jackson.codehaus.org 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, 2001-2014. 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, 2001-2014. 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, 2001-2014. 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
  - **@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.

Copyright © Descriptor Systems, 2001-2014. 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, 2001-2014. 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, 2001-2014. 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   }
2 - 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, 2001-2014. 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>
```


Copyright © Descriptor Systems, 2001-2014. 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 use Spring MVC controllers, annotations and message converters to write RESTful CRUD services in Spring.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum