Spring MVC

Following the life of a request

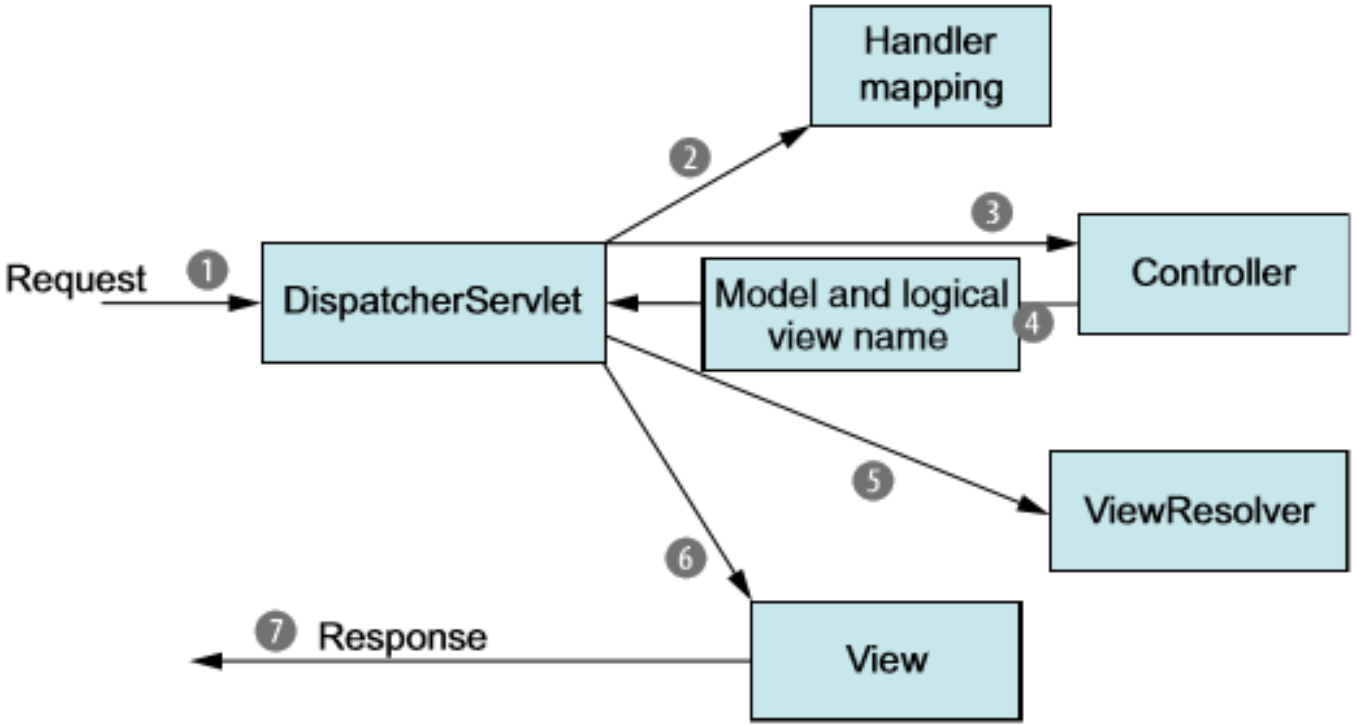


Figure 5.1 A request couriers information to several stops on its way to producing the desired results.

Setting up Spring MVC

Listing 5.1 Configuring DispatcherServlet

```
package spittr.config;
import org.springframework.web.servlet.support.
             AbstractAnnotationConfigDispatcherServletInitializer;

public class SpittrWebAppInitializer
       extends AbstractAnnotationConfigDispatcherServletInitializer {

  @Override
  protected String[] getServletMappings() {        <-- Map DispatcherServlet to /
    return new String[] { "/" };
  }

  @Override
  protected Class<?>[] getRootConfigClasses() {
    return new Class<?>[] { RootConfig.class };
  }

  @Override
  protected Class<?>[] getServletConfigClasses() {    <-- Specify configuration class
    return new Class<?>[] { WebConfig.class };
  }

}
```

Enabling Spring MVC

```
package spittr.config;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

@Configuration
@EnableWebMvc
public class WebConfig {
}
```

This will work, and it will enable Spring MVC. But it leaves a lot to be desired:

- No view resolver is configured. As such, Spring will default to using `BeanNameViewResolver`, a view resolver that resolves views by looking for beans whose ID matches the view name and whose class implements the `View` interface.
- Component-scanning isn't enabled. Consequently, the only way Spring will find any controllers is if you declare them explicitly in the configuration.
- As it is, `DispatcherServlet` is mapped as the default servlet for the application and will handle *all* requests, including requests for static resources, such as images and stylesheets (which is probably not what you want in most cases).

Enabling Spring MVC

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.
                                        DefaultServletHandlerConfigurer;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.
                                        WebMvcConfigurerAdapter;
import org.springframework.web.servlet.view.
InternalResourceViewResolver;
```

```
@Configuration
@EnableWebMvc                                   <-- Enable Spring MVC
@ComponentScan("spitter.web")                   <-- Enable component-scanning
public class WebConfig
        extends WebMvcConfigurerAdapter {

  @Bean
  public ViewResolver viewResolver() {          <-- Configure a JSP view resolver
    InternalResourceViewResolver resolver =
            new InternalResourceViewResolver();
    resolver.setPrefix("/WEB-INF/views/");
    resolver.setSuffix(".jsp");
    resolver.setExposeContextBeansAsAttributes(true);
    return resolver;
  }

  @Override                                     <-- Configure static content handling
  public void configureDefaultServletHandling(
        DefaultServletHandlerConfigurer configurer) {
    configurer.enable();
  }

}
```

Enabling Spring MVC

```
package spittr.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.ComponentScan.Filter;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.FilterType;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

@Configuration
@ComponentScan(basePackages={"spitter"},
    excludeFilters={
        @Filter(type=FilterType.ANNOTATION, value=EnableWebMvc.class)
    })
public class RootConfig {
}
```

Writing a simple controller

Listing 5.3 `HomeController`: an example of an extremely simple controller

```
package spittr.web;
import static org.springframework.web.bind.annotation.RequestMethod.*;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller                                  <-- Declared to be a controller
public class HomeController {

  @RequestMapping(value="/", method=GET)     <-- Handle GET requests for /
  public String home() {
    return "home";                           <-- View name is home
  }

}
```

Writing a simple controller

Listing 5.4 Spittr home page, defined as a simple JSP

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
  <head>
    <title>Spittr</title>
    <link rel="stylesheet"
          type="text/css"
          href="<c:url value="/resources/style.css" />" >
  </head>
  <body>
    <h1>Welcome to Spittr</h1>

    <a href="<c:url value="/spittles" />">Spittles</a> |
    <a href="<c:url value="/spitter/register" />">Register</a>
  </body>
</html>
```

Testing the controller

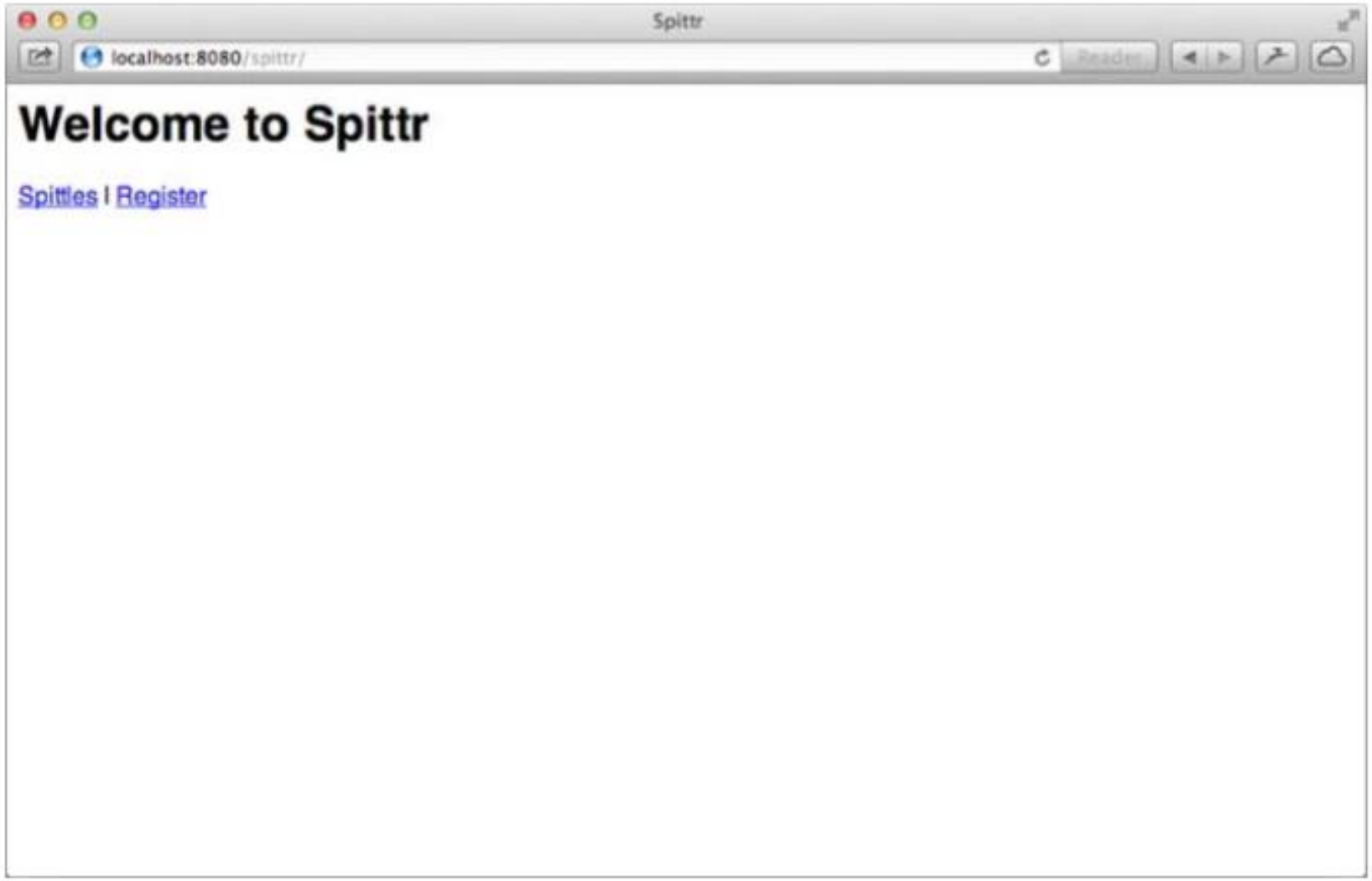


Figure 5.2 The Spittr home page in action

Testing the controller

Listing 5.5 HomeControllerTest: tests HomeController

```
package spittr.web;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
import spittr.web.HomeController;

public class HomeControllerTest {
  @Test
  public void testHomePage() throws Exception {
    HomeController controller = new HomeController();
    assertEquals("home", controller.home());
  }
}
```

Testing the controller

Listing 5.6 Revised HomeControllerTest

```
package spittr.web;
import static
  org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static
  org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
import static
  org.springframework.test.web.servlet.setup.MockMvcBuilders.*;
import org.junit.Test;
import org.springframework.test.web.servlet.MockMvc;
import spittr.web.HomeController;

public class HomeControllerTest {
  @Test
  public void testHomePage() throws Exception {
    HomeController controller = new HomeController();
    MockMvc mockMvc =                                    <-- Set up MockMvc
        standaloneSetup(controller).build();

    mockMvc.perform(get("/"))                  <-- Perform GET /
           .andExpect(view().name("home"));              <-- Expect home view
  }
}
```

Defining class-level request handling

Listing 5.7 Splitting the @RequestMapping in HomeController

```
package spittr.web;
import static org.springframework.web.bind.annotation.RequestMethod.*;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/")                          <-- Map controller to /
public class HomeController {

  @RequestMapping(method=GET)                 <-- Handle GET requests
  public String home() {
    return "home";                            <-- View name is home
   }

}
```

Passing model data to the view

```
package spittr.data;
import java.util.List;
import spittr.Spittle;

public interface SpittleRepository {
  List<Spittle> findSpittles(long max, int count);
}
```

In order to get the 20 most recent `Spittle` objects, you can call `findSpittles()` like this:

```
List<Spittle> recent =
        spittleRepository.findSpittles(Long.MAX_VALUE, 20);
```

Passing model data to the view

and Getter 

Listing 5.8 `Spittle` class: carries a message, a timestamp, and a location

```
package spittr;
import java.util.Date;

public class Spittle {
  private final Long id;
  private final String message;
  private final Date time;
  private Double latitude;
  private Double longitude;

  public Spittle(String message, Date time) {
    this(message, time, null, null);
  }

  public Spittle(
      String message, Date time, Double longitude, Double latitude) {
    this.id = null;
    this.message = message;
    this.time = time;
    this.longitude = longitude;
    this.latitude = latitude;
  }

  @Override
  public boolean equals(Object that) {
    return EqualsBuilder.reflectionEquals(this, that, "id", "time");
  }

  @Override
  public int hashCode() {
    return HashCodeBuilder.reflectionHashCode(this, "id", "time");
  }
```

Passing model data to the view

Listing 5.9 Testing that `SpittleController` handles `GET` requests for `/spittles`

```
@Test
public void shouldShowRecentSpittles() throws Exception {
  List<Spittle> expectedSpittles = createSpittleList(20);
  SpittleRepository mockRepository =                    <-- Mock repository
      mock(SpittleRepository.class);
  when(mockRepository.findSpittles(Long.MAX_VALUE, 20))
      .thenReturn(expectedSpittles);

  SpittleController controller =
        new SpittleController(mockRepository);

  SpittleController controller =
       new SpittleController(mockRepository);
  MockMvc mockMvc = standaloneSetup(controller)         <-- Mock Spring MVC
      .setSingleView(
          new InternalResourceView("/WEB-INF/views/spittles.jsp"))
      .build();

  mockMvc.perform(get("/spittles"))                     <-- GET /spittles
    .andExpect(view().name("spittles"))
    .andExpect(model().attributeExists("spittleList"))
    .andExpect(model().attribute("spittleList",         <-- Assert expectations
        hasItems(expectedSpittles.toArray())));
 }

private List<Spittle> createSpittleList(int count) {
  List<Spittle> spittles = new ArrayList<Spittle>();
  for (int i=0; i < count; i++) {
    spittles.add(new Spittle("Spittle " + i, new Date()));
  }
  return spittles;
}
```

Passing model data to the view

Listing 5.10 SpittleController: places a list of recent spittles in the model

```
package spittr.web;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import spittr.Spittle;
import spittr.data.SpittleRepository;

@Controller
@RequestMapping("/spittles")
public class SpittleController {

  private SpittleRepository spittleRepository;

  @Autowired
  public SpittleController(                          <-- Inject SpittleRepository
      SpittleRepository spittleRepository) {
    this.spittleRepository = spittleRepository;
  }

  @RequestMapping(method=RequestMethod.GET)
  public String spittles(Model model) {
    model.addAttribute(                              <-- Add spittles to model
        spittleRepository.findSpittles(
            Long.MAX_VALUE, 20));
    return "spittles";                               <-- Return view name
  }

}
```

Passing model data to the view

Now that there's data in the model, how does the JSP access it? As it turns out, when the view is a JSP, the model data is copied into the request as request attributes. Therefore, the spittles.jsp file can use JavaServer Pages Standard Tag Library's (JSTL) `<c:forEach>` tag to render the list of spittles:

```
<c:forEach items="${spittleList}" var="spittle" >
  <li id="spittle_<c:out value="spittle.id"/>">
    <div class="spittleMessage">
      <c:out value="${spittle.message}" />
    </div>
    <div>
      <span class="spittleTime"><c:out value="${spittle.time}" /></span>
      <span class="spittleLocation">
          (<c:out value="${spittle.latitude}" />,
          <c:out value="${spittle.longitude}" />)</span>
    </div>
  </li>
</c:forEach>
```

Taking query parameters

Listing 5.11 New method to test for a paged list of spittles

```
@Test
public void shouldShowPagedSpittles() throws Exception {
  List<Spittle> expectedSpittles = createSpittleList(50);
  SpittleRepository mockRepository = mock(SpittleRepository.class);
  when(mockRepository.findSpittles(238900, 50))
      .thenReturn(expectedSpittles);

  SpittleController controller =
      new SpittleController(mockRepository);
  MockMvc mockMvc = standaloneSetup(controller)
      .setSingleView(
          new InternalResourceView("/WEB-INF/views/spittles.jsp"))
      .build();

  mockMvc.perform(get("/spittles?max=238900&count=50"))
    .andExpect(view().name("spittles"))
    .andExpect(model().attributeExists("spittleList"))
    .andExpect(model().attribute("spittleList",
                hasItems(expectedSpittles.toArray())));
}
```

Expect max and count parameters

Pass max and count parameters

Taking query parameters

```
@RequestMapping(method=RequestMethod.GET)
public List<Spittle> spittles(
    @RequestParam(value="max",
                  defaultValue=MAX_LONG_AS_STRING) long max,
    @RequestParam(value="count", defaultValue="20") int count) {
  return spittleRepository.findSpittles(max, count);
}
```

Taking input via path parameters

```
@RequestMapping(value="/show", method=RequestMethod.GET)
public String showSpittle(
    @RequestParam("spittle_id") long spittleId,
    Model model) {
  model.addAttribute(spittleRepository.findOne(spittleId));
  return "spittle";
}
```

Taking input via path parameters

Listing 5.12 Testing a request for a `Spittle` with ID specified in a path variable

```
@Test
public void testSpittle() throws Exception {
  Spittle expectedSpittle = new Spittle("Hello", new Date());
  SpittleRepository mockRepository = mock(SpittleRepository.class);
  when(mockRepository.findOne(12345)).thenReturn(expectedSpittle);

  SpittleController controller = new SpittleController(mockRepository);
  MockMvc mockMvc = standaloneSetup(controller).build();

  mockMvc.perform(get("/spittles/12345"))
    .andExpect(view().name("spittle"))
    .andExpect(model().attributeExists("spittle"))
    .andExpect(model().attribute("spittle", expectedSpittle));
}
```

Request resource via path

Taking input via path parameters

Here's a handler method that uses placeholders to accept a `Spittle` ID as part of the path:

```
@RequestMapping(value="/{spittleId}", method=RequestMethod.GET)
public String spittle(
    @PathVariable("spittleId") long spittleId,
    Model model) {
  model.addAttribute(spittleRepository.findOne(spittleId));
  return "spittle";
}
```

Taking input via path parameters

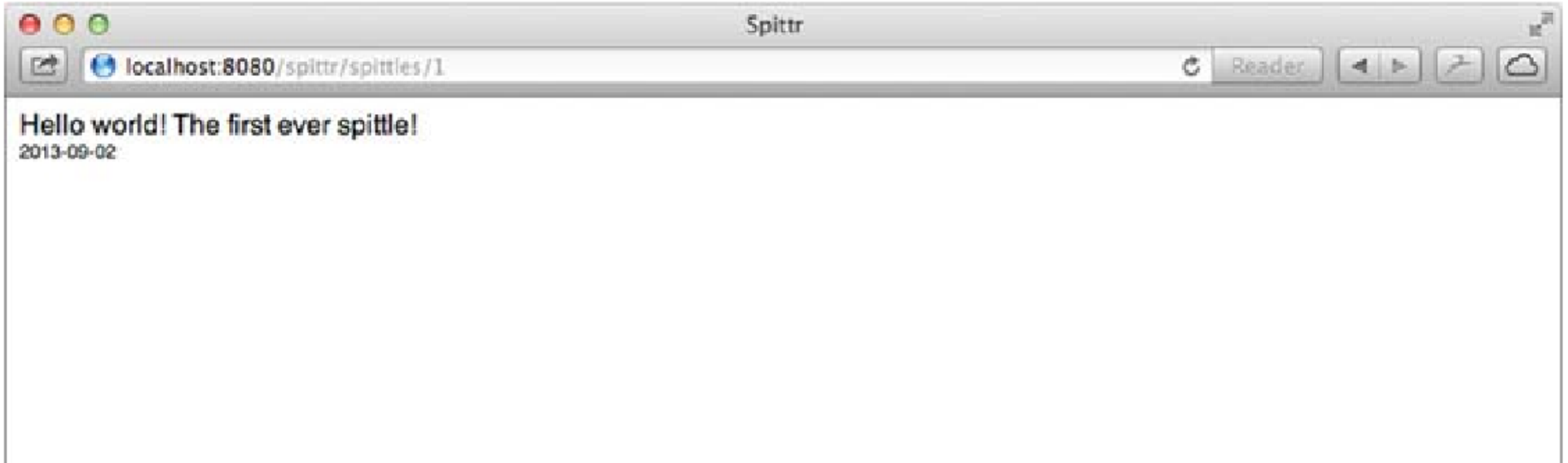
Because the method parameter name happens to be the same as the placeholder name, you can optionally omit the `value` parameter on `@PathVariable`:

```
@RequestMapping(value="/{spittleId}", method=RequestMethod.GET)
public String spittle(@PathVariable long spittleId, Model model) {
  model.addAttribute(spittleRepository.findOne(spittleId));
  return "spittle";
}
```

Taking input via path parameters

The data in the `Spittle` object can then be rendered in the view by referring to the request attribute whose key is `spittle` (the same as the model key). Here's a snippet of a JSP view that renders the `Spittle`:

```
<div class="spittleView">
  <div class="spittleMessage"><c:out value="${spittle.message}" /></div>
  <div>
    <span class="spittleTime"><c:out value="${spittle.time}" /></span>
  </div>
</div>
```



Processing forms

Listing 5.13 `SpitterController`: displays a form for users to sign up with the app

```
package spittr.web;
import static org.springframework.web.bind.annotation.RequestMethod.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import spittr.Spitter;
import spittr.data.SpitterRepository;

@Controller
@RequestMapping("/spitter")
public class SpitterController {

@RequestMapping(value="/register", method=GET)
public String showRegistrationForm() {
  return "registerForm";
}

}
```

Handle GET requests for /spitter/register

Processing forms

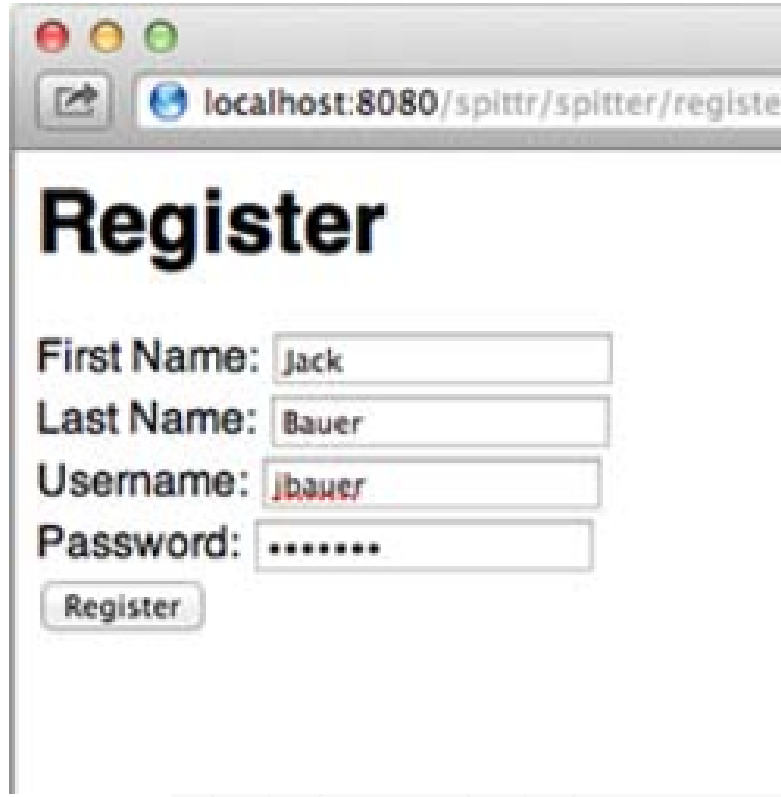
Listing 5.14 Testing a form-displaying controller method

```
@Test
public void shouldShowRegistration() throws Exception {
  SpitterController controller = new SpitterController();
  MockMvc mockMvc = standaloneSetup(controller).build();     <-- Set up MockMvc

  mockMvc.perform(get("/spitter/register"))
         .andExpect(view().name("registerForm"));     <-- Assert registerForm view
}
```

Processing forms

Listing 5.15 JSP to render a registration form

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
  <head>
    <title>Spittr</title>
    <link rel="stylesheet" type="text/css"
          href="<c:url value="/resources/style.css" />" >
  </head>
  <body>
    <h1>Register</h1>

    <form method="POST">
      First Name: <input type="text" name="firstName" /><br/>
      Last Name: <input type="text" name="lastName" /><br/>
      Username: <input type="text" name="username" /><br/>
      Password: <input type="password" name="password" /><br/>
       <input type="submit" value="Register" />
     </form>
   </body>
 </html>
```



Writing a form-handling controller

Listing 5.16 Testing form-handling controller methods

```
@Test
public void shouldProcessRegistration() throws Exception {
  SpitterRepository mockRepository =
           mock(SpitterRepository.class);        <-- Set up mock repository
  Spitter unsaved =
          new Spitter("jbauer", "24hours", "Jack", "Bauer");
  Spitter saved =
          new Spitter(24L, "jbauer", "24hours", "Jack", "Bauer");
  when(mockRepository.save(unsaved)).thenReturn(saved);

  SpitterController controller =
          new SpitterController(mockRepository);
  MockMvc mockMvc = standaloneSetup(controller).build();    <-- Set up MockMvc

  mockMvc.perform(post("/spitter/register")          <-- Perform request
         .param("firstName", "Jack")
         .param("lastName", "Bauer")
         .param("username", "jbauer")
         .param("password", "24hours"))
         .andExpect(redirectedUrl("/spitter/jbauer"));

  verify(mockRepository, atLeastOnce()).save(unsaved);        <-- Verify save
}
```

Writing a form-handling controller

```
import static org.springframework.web.bind.annotation.RequestMethod.*;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

import spittr.Spitter;
import spittr.data.SpitterRepository;

@Controller
@RequestMapping("/spitter")
public class SpitterController {
  private SpitterRepository spitterRepository;

  @Autowired
  public SpitterController(                          <-- Inject SpitterRepository
      SpitterRepository spitterRepository) {
    this.spitterRepository = spitterRepository;
  }

  @RequestMapping(value="/register", method=GET)
  public String showRegistrationForm() {
    return "registerForm";
  }

  @RequestMapping(value="/register", method=POST)
  public String processRegistration(Spitter spitter) {
    spitterRepository.save(spitter);                 <-- Save a Spitter

    return "redirect:/spitter/" +                    <-- Redirect to profile page
           spitter.getUsername();
  }
}
```

Writing a form-handling controller

```
@RequestMapping(value="/{username}", method=GET)
public String showSpitterProfile(
      @PathVariable String username, Model model) {
  Spitter spitter = spitterRepository.findByUsername(username);
  model.addAttribute(spitter);
  return "profile";
}
```

```
<h1>Your Profile</h1>
<c:out value="${spitter.username}" /><br/>
<c:out value="${spitter.firstName}" />
    <c:out value="${spitter.lastName}" />
```

Validating forms

Annotation	Description
`@AssertFalse`	The annotated element must be a Boolean type and be `false`.
`@AssertTrue`	The annotated element must be a Boolean type and be `true`.
`@DecimalMax`	The annotated element must be a number whose value is less than or equal to a given `BigDecimalString` value.
`@DecimalMin`	The annotated element must be a number whose value is greater than or equal to a given `BigDecimalString` value.
`@Digits`	The annotated element must be a number whose value has a specified number of digits.
`@Future`	The value of the annotated element must be a date in the future.
`@Max`	The annotated element must be a number whose value is less than or equal to a given value.
`@Min`	The annotated element must be a number whose value is greater than or equal to a given value.
`@NotNull`	The value of the annotated element must not be `null`.
`@Null`	The value of the annotated element must be `null`.
`@Past`	The value of the annotated element must be a date in the past.
`@Pattern`	The value of the annotated element must match a given regular expression.

Validating forms

Listing 5.18 SpittleForm: carries only fields submitted in a SpittlePOST request

```
package spittr;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import org.apache.commons.lang3.builder.EqualsBuilder;
import org.apache.commons.lang3.builder.HashCodeBuilder;

public class Spitter {

  private Long id;

  @NotNull
  @Size(min=5, max=16)
  private String username;

  @NotNull
  @Size(min=5, max=25)
  private String password;


  @NotNull
  @Size(min=2, max=30)
  private String firstName;

  @NotNull
  @Size(min=2, max=30)
  private String lastName;

  ...
```

Not null, from 5 to 16 characters

Not null, from 5 to 25 characters

Not null, from 2 to 30 characters

Not null, from 2 to 30 characters

Validating forms

Listing 5.19 `processRegistration()`: ensures that data submitted is valid

```
@RequestMapping(value="/register", method=POST)
public String processRegistration(
    @Valid Spitter spitter,                    <-- Validate Spitter input
     Errors errors) {

  if (errors.hasErrors()) {                    <-- Return to form on validation errors
    return "registerForm";
   }

  spitterRepository.save(spitter);
  return "redirect:/spitter/" + spitter.getUsername();
}
```

Rendering web views

Understanding view resolution

Spring MVC defines an interface named `ViewResolver` that looks a little something like this:

```
public interface ViewResolver {
  View resolveViewName(String viewName, Locale locale)
                       throws Exception;
}
```

The `resolveViewName()` method, when given a view name and a `Locale`, returns a `View` instance. `View` is another interface that looks like this:

```
public interface View {
  String getContentType();
  void render(Map<String, ?> model,
              HttpServletRequest request,
              HttpServletResponse response) throws Exception;
}
```

Understanding view resolution

View resolver	Description
`BeanNameViewResolver`	Resolves views as beans in the Spring application context whose ID is the same as the view name.
`ContentNegotiatingViewResolver`	Resolves views by considering the content type desired by the client and delegating to another view resolver that can produce that type.
`FreeMarkerViewResolver`	Resolves views as FreeMarker templates.
`InternalResourceViewResolver`	Resolves views as resources internal to the web application (typically JSPs).
`JasperReportsViewResolver`	Resolves views as JasperReports definitions.
`ResourceBundleViewResolver`	Resolves views from a resource bundle (typically a properties file).

Understanding view resolution

View resolver	Description
`TilesViewResolver`	Resolves views as Apache Tile definitions, where the tile ID is the same as the view name. Note that there are two different `TilesViewResolver` implementations, one each for Tiles 2.0 and Tiles 3.0.
`UrlBasedViewResolver`	Resolves views directly from the view name, where the view name matches the name of a physical view definition.
`VelocityLayoutViewResolver`	Resolves views as Velocity layouts to compose pages from different Velocity templates.
`VelocityViewResolver`	Resolves views as Velocity templates.
`XmlViewResolver`	Resolves views as bean definitions from a specified XML file. Similar to `BeanNameViewResolver`.
`XsltViewResolver`	Resolves views to be rendered as the result of an XSLT transformation.

Creating JSP views

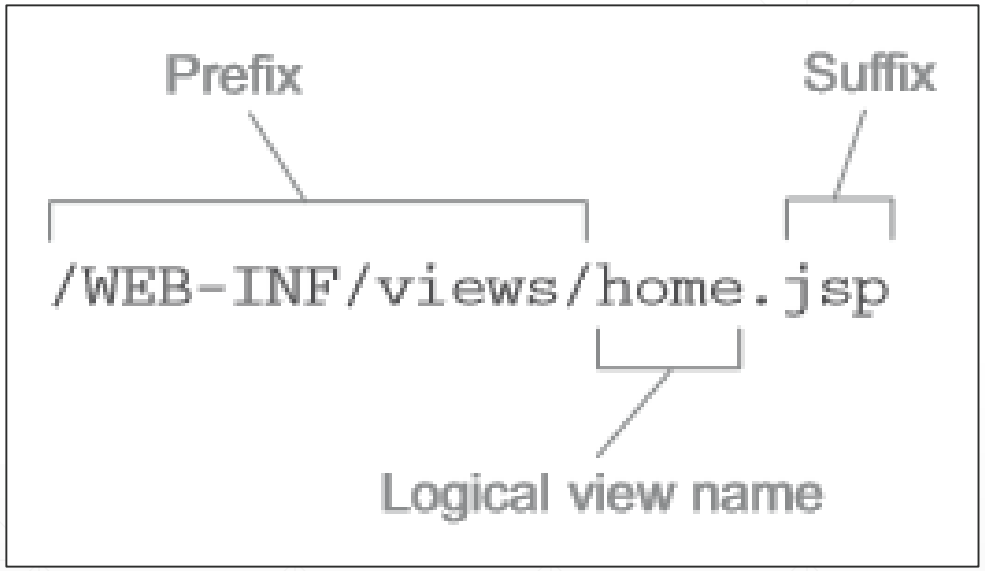
Spring supports JSP views in two ways:

- `InternalResourceViewResolver` can be used to resolve view names into JSP files. Moreover, if you're using JavaServer Pages Standard Tag Library (JSTL) tags in your JSP pages, `InternalResourceViewResolver` can resolve view names into JSP files fronted by `JstlView` to expose JSTL locale and resource bundle variables to JSTL's formatting and message tags.
- Spring provides two JSP tag libraries, one for form-to-model binding and one providing general utility features.

Configuring a JSP-ready view resolver

You can configure InternalResourceViewResolver to apply this convention when resolving views by configuring it with this @Bean-annotated method:

```
@Bean
public ViewResolver viewResolver() {
  InternalResourceViewResolver resolver =
      new InternalResourceViewResolver();
  resolver.setPrefix("/WEB-INF/views/");
  resolver.setSuffix(".jsp");
  return resolver;
}
```



Or in xml:

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.
                          InternalResourceViewResolver"
      p:prefix="/WEB-INF/views/"
      p:suffix=".jsp" />
```

Configuring a JSP-ready view resolver

With this configuration of `InternalResourceViewResolver` in place, you can expect it to resolve logical view names into JSP files such as this:

- `home` resolves to /WEB-INF/views/home.jsp
- `productList` resolves to /WEB-INF/views/productList.jsp
- `books/detail` resolves to /WEB-INF/views/books/detail.jsp

Resolving JSTL views

```
@Bean
public ViewResolver viewResolver() {
  InternalResourceViewResolver resolver =
      new InternalResourceViewResolver();
  resolver.setPrefix("/WEB-INF/views/");
  resolver.setSuffix(".jsp");
  resolver.setViewClass(
      org.springframework.web.servlet.view.JstlView.class);
  return resolver;
}
```

Or in xml:

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.
                        InternalResourceViewResolver"
      p:prefix="/WEB-INF/views/"
      p:suffix=".jsp"
      p:viewClass="org.springframework.web.servlet.view.JstlView" />
```

Using Spring´s JSP libraries

JSP tag	Description
`<sf:checkbox>`	Renders an HTML `<input>` tag with `type` set to `checkbox`.
`<sf:checkboxes>`	Renders multiple HTML `<input>` tags with `type` set to `checkbox`.
`<sf:errors>`	Renders field errors in an HTML `<span>` tag.
`<sf:form>`	Renders an HTML `<form>` tag and exposed binding path to inner tags for data-binding.
`<sf:hidden>`	Renders an HTML `<input>` tag with `type` set to `hidden`.
`<sf:input>`	Renders an HTML `<input>` tag with `type` set to `text`.
`<sf:label>`	Renders an HTML `<label>` tag.
`<sf:option>`	Renders an HTML `<option>` tag. The `selected` attribute is set according to the `bound` value.
`<sf:options>`	Renders a list of HTML `<option>` tags corresponding to the bound collection, array, or map.
`<sf:password>`	Renders an HTML `<input>` tag with `type` set to `password`.
`<sf:radiobutton>`	Renders an HTML `<input>` tag with `type` set to `radio`.
`<sf:radiobuttons>`	Renders multiple HTML `<input>` tags with `type` set to `radio`.
`<sf:select>`	Renders an HTML `<select>` tag.
`<sf:textarea>`	Renders an HTML `<textarea>` tag.

Using Spring´s JSP libraries

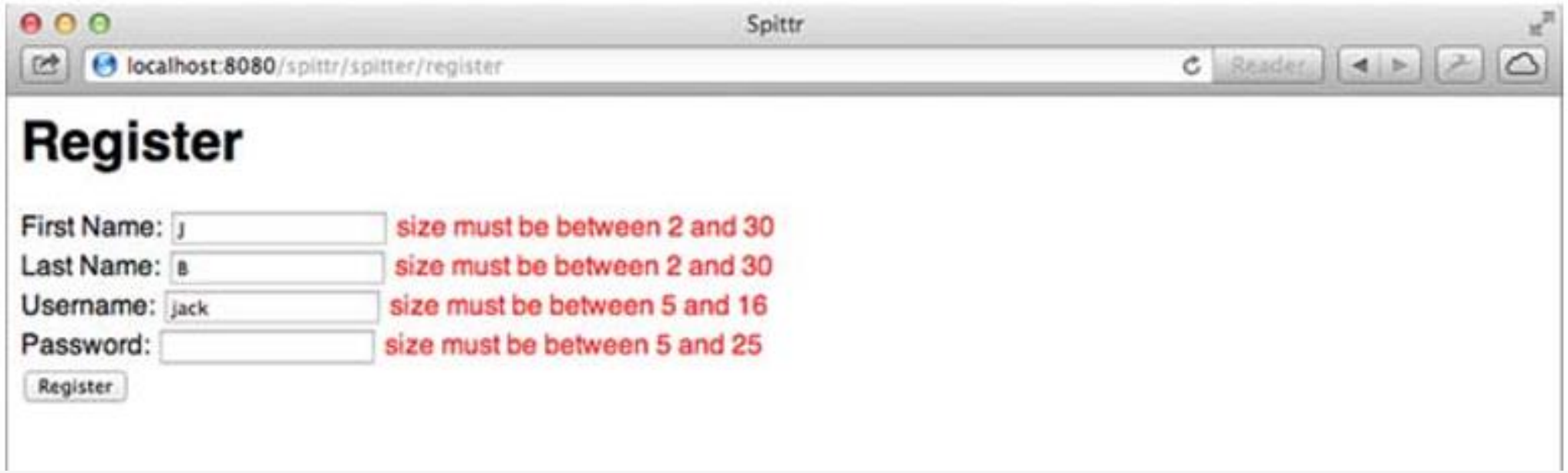
```
<form id="spitter" action="/spitter/spitter/register" method="POST">
  First Name:
      <input id="firstName"
             name="firstName" type="text" value="J"/><br/>
  Last Name:
      <input id="lastName"
             name="lastName" type="text" value="B"/><br/>
  Email:
      <input id="email"
             name="email" type="text" value="jack"/><br/>
  Username:
      <input id="username"
             name="username" type="text" value="jack"/><br/>
  Password:
      <input id="password"
             name="password" type="password" value=""/><br/>
  <input type="submit" value="Register" />
</form>
```

Using Spring´s JSP libraries

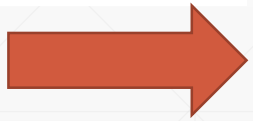
- Displaying errors:

```
<sf:form method="POST" commandName="spitter" >
  First Name: <sf:input path="firstName" />
    <sf:errors path="firstName" cssClass="error" /><br/>
...
</sf:form>
```

↑ Own style



JSP tag	Description
`<s:bind>`	Exports a bound property status to a page-scoped `status` property. Used along with `<s:path>` to obtain a bound property value.
`<s:escapeBody>`	HTML and/or JavaScript escapes the content in the body of the tag.
`<s:hasBindErrors>`	Conditionally renders content if a specified model object (in a request attribute) has bind errors.
`<s:htmlEscape>`	Sets the default HTML escape value for the current page.
`<s:message>`	Retrieves the message with the given code and either renders it (default) or assigns it to a page-, request-, session-, or application-scoped variable (when using the `var` and `scope` attributes).



JSP tag	Description
`<s:nestedPath>`	Sets a nested path to be used by `<s:bind>`.
`<s:theme>`	Retrieves a theme message with the given code and either renders it (default) or assigns it to a page-, request-, session-, or application-scoped variable (when using the `var` and `scope` attributes).
`<s:transform>`	Transforms properties not contained in a command object using a command object's property editors.
`<s:url>`	Creates context-relative URLs with support for URI template variables and HTML/XML/JavaScript escaping. Can either render the URL (default) or assign it to a page-, request-, session-, or application-scoped variable (when using the `var` and `scope` attributes).
`<s:eval>`	Evaluates Spring Expression Language (SpEL) expressions, rendering the result (default) or assigning it to a page-, request-, session-, or application-scoped variable (when using the `var` and `scope` attributes).

Displaying internationalized messages

```
<h1><s:message code="spittr.welcome" /></h1>
```

message.source:

```
@Bean
public MessageSource messageSource() {
  ResourceBundleMessageSource messageSource =
          new ResourceBundleMessageSource();
  messageSource.setBasename("messages");
  return messageSource;
}
```

Creating URLs

`<s:url>`:

```
<a href="<s:url href="/spitter/register" />">Register</a>
```

If the application's servlet context is named `spittr`, then the following HTML will be rendered in the response:

```
<a href="/spittr/spitter/register">Register</a>
```

Creating URLs

- Enable JavaScript code
- Passing parameter

```
<s:url value="/spittles" var="spittlesJSUrl" javaScriptEscape="true">
  <s:param name="max" value="60" />
  <s:param name="count" value="20" />
</s:url>
```

Escaping content

```
<s:escapeBody htmlEscape="true">
<h1>Hello</h1>
</s:escapeBody>
```

This renders the following to the body of the response:

```
&lt;h1&gt;Hello&lt;/h1&gt;
```

Also JavaScript is supported:

```
<s:escapeBody javaScriptEscape="true">
<h1>Hello</h1>
</s:escapeBody>
```

Configuring a Tiles view resolver

Listing 6.1 Configuring `TilesConfigurer` to resolve tile definitions

```
@Bean
public TilesConfigurer tilesConfigurer() {
  TilesConfigurer tiles = new TilesConfigurer();
  tiles.setDefinitions(new String[] {
      "/WEB-INF/layout/tiles.xml"
  });
  tiles.setCheckRefresh(true);
  return tiles;
}
```

Specify tile definition locations

Enable refresh

Defining Tiles

Listing 6.2 Defining tiles for the Spittr application

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tiles-definitions PUBLIC
       "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"
       "http://tiles.apache.org/dtds/tiles-config_3_0.dtd">
<tiles-definitions>

  <definition name="base"                                    <-- Define a base tile
              template="/WEB-INF/layout/page.jsp">
    <put-attribute name="header"
                   value="/WEB-INF/layout/header.jsp" />
    <put-attribute name="footer"
                   value="/WEB-INF/layout/footer.jsp" />     <-- Set an attribute
  </definition>

  <definition name="home" extends="base">                    <-- Extend the base tile
    <put-attribute name="body"
                   value="/WEB-INF/views/home.jsp" />
  </definition>
```

```
  <definition name="registerForm" extends="base">
    <put-attribute name="body"
                   value="/WEB-INF/views/registerForm.jsp" />
  </definition>

  <definition name="profile" extends="base">
    <put-attribute name="body"
                   value="/WEB-INF/views/profile.jsp" />
  </definition>

  <definition name="spittles" extends="base">
    <put-attribute name="body"
                   value="/WEB-INF/views/spittles.jsp" />
  </definition>

  <definition name="spittle" extends="base">
    <put-attribute name="body"
                   value="/WEB-INF/views/spittle.jsp" />
  </definition>

</tiles-definitions>
```

Defining Tiles

Listing 6.3 Main layout template: references other templates to create a view

```
<%@ taglib uri="http://www.springframework.org/tags" prefix="s" %>
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="t" %>
<%@ page session="false" %>
<html>
  <head>
    <title>Spittr</title>
    <link rel="stylesheet"
          type="text/css"
          href="<s:url value="/resources/style.css" />" >
  </head>
  <body>
    <div id="header">
      <t:insertAttribute name="header" />          <-- Insert the header
    </div>
    <div id="content">
      <t:insertAttribute name="body" />            <-- Insert the body
    </div>
    <div id="footer">
      <t:insertAttribute name="footer" />          <-- Insert the footer
    </div>
  </body>
</html>
```

Defining Tiles

```
<definition name="home" template="/WEB-INF/layout/page.jsp">
  <put-attribute name="header" value="/WEB-INF/layout/header.jsp" />
  <put-attribute name="footer" value="/WEB-INF/layout/footer.jsp" />
  <put-attribute name="body" value="/WEB-INF/views/home.jsp" />
</definition>
```

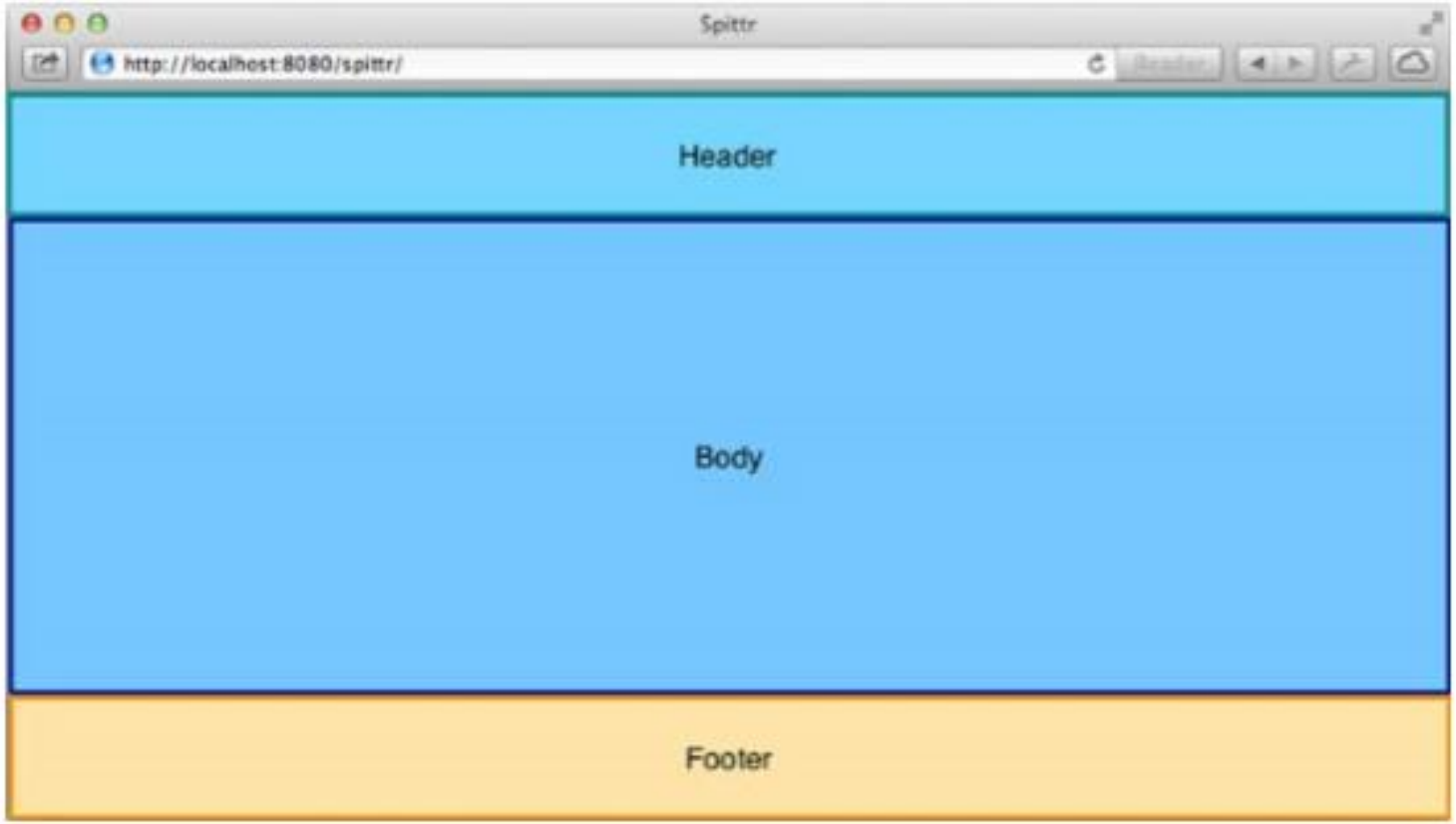
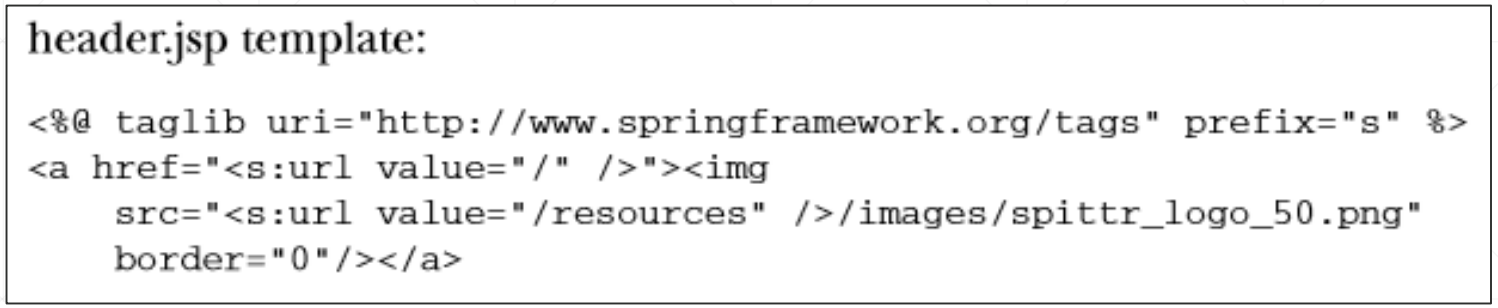


Figure 6.4 A general layout defining a header, a body, and a footer

Simple Example:



header.jsp template:

```
<%@ taglib uri="http://www.springframework.org/tags" prefix="s" %>
<a href="<s:url value="/" />"><img
    src="<s:url value="/resources" />/images/spittr_logo_50.png"
    border="0"/></a>
```

Simple Example:

The footer.jsp template is even simpler:

```
Copyright &copy; Craig Walls
```

Simple Example:

home.jsp:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<h1>Welcome to Spittr</h1>

<a href="<c:url value="/spittles" />">Spittles</a> |
<a href="<c:url value="/spitter/register" />">Register</a>
```

Working with Thymeleaf

In order to use Thymeleaf with Spring, you'll need to configure three beans that enable Thymeleaf-Spring integration:

- A `ThymeleafViewResolver` that resolves Thymeleaf template views from logical view names
- A `SpringTemplateEngine` to process the templates and render the results
- A `TemplateResolver` that loads Thymeleaf templates

Listing 6.4 Configuring Thymeleaf support for Spring in Java configuration

```
@Bean
public ViewResolver viewResolver(                          <-- Thymeleaf view resolver
      SpringTemplateEngine templateEngine) {
  ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
  viewResolver.setTemplateEngine(templateEngine);
  return viewResolver;
}

@Bean
public TemplateEngine templateEngine(                      <-- Template engine
      TemplateResolver templateResolver) {
  SpringTemplateEngine templateEngine = new SpringTemplateEngine();
  templateEngine.setTemplateResolver(templateResolver);
  return templateEngine;
}

@Bean
public TemplateResolver templateResolver() {               <-- Template resolver
  TemplateResolver templateResolver =
          new ServletContextTemplateResolver();
  templateResolver.setPrefix("/WEB-INF/templates/");
  templateResolver.setSuffix(".html");
  templateResolver.setTemplateMode("HTML5");
  return templateResolver;
}
```

Working with Thymeleaf

Declaring the beans in XML:

Listing 6.5 Configuring Thymeleaf support for Spring in XML

```
<bean id="viewResolver"                                    <-- Thymeleaf view resolver
      class="org.thymeleaf.spring3.view.ThymeleafViewResolver"
      p:templateEngine-ref="templateEngine" />

<bean id="templateEngine"                                  <-- Template engine
      class="org.thymeleaf.spring3.SpringTemplateEngine"
      p:templateResolver-ref="templateResolver" />

<bean id="templateResolver" class=                         <-- Template resolver
    "org.thymeleaf.templateresolver.ServletContextTemplateResolver"
      p:prefix="/WEB-INF/templates/"
      p:suffix=".html"
      p:templateMode="HTML5" />
```

Defining Thymeleaf beans

Listing 6.6 home.html: home page template using the Thymeleaf namespace

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">          <-- Declare Thymeleaf namespace
  <head>
    <title>Spittr</title>
    <link rel="stylesheet"
          type="text/css"
          th:href="@{/resources/style.css}"></link>  <-- th:href link to stylesheet
  </head>
  <body>
    <h1>Welcome to Spittr</h1>

    <a th:href="@{/spittles}">Spittles</a> |          <-- th:href links to pages
    <a th:href="@{/spitter/register}">Register</a>
  </body>
</html>
```

Registration page using Thymeleaf

Listing 6.7 Registration page, using Thymeleaf to bind a form to a command object

```
<form method="POST" th:object="${spitter}">
  <div class="errors" th:if="${#fields.hasErrors('*')}">
    <ul>
      <li th:each="err : ${#fields.errors('*')}"
          th:text="${err}">Input is incorrect</li>
    </ul>
  </div>
  <label th:class="${#fields.hasErrors('firstName')}? 'error'">
    First Name</label>:
  <input type="text" th:field="*{firstName}"
         th:class="${#fields.hasErrors('firstName')}? 'error'" /><br/>

  <label th:class="${#fields.hasErrors('lastName')}? 'error'">
    Last Name</label>:
  <input type="text" th:field="*{lastName}"
         th:class="${#fields.hasErrors('lastName')}? 'error'" /><br/>
```

Display errors

First name

Last name

Email →
```
<label th:class="${#fields.hasErrors('email')}? 'error'">
    Email</label>:
  <input type="text" th:field="*{email}"
         th:class="${#fields.hasErrors('email')}? 'error'" /><br/>
```

Username →
```
<label th:class="${#fields.hasErrors('username')}? 'error'">
    Username</label>:
  <input type="text" th:field="*{username}"
         th:class="${#fields.hasErrors('username')}? 'error'" /><br/>
```

Password →
```
<label th:class="${#fields.hasErrors('password')}? 'error'">
    Password</label>:
  <input type="password" th:field="*{password}"
         th:class="${#fields.hasErrors('password')}? 'error'" /><br/>

  <input type="submit" value="Register" />
</form>
```

Alternate Spring MVC configuration

- If you plan to use Servlet 3.0 support for multipart configuration, you need to enable DispatcherServlet's registration to enable multipart requests.

```
@Override
protected void customizeRegistration(Dynamic registration) {
registration.setMultipartConfig(
    new MultipartConfigElement("/tmp/spittr/uploads"));
}
```

Adding additional servlets and filters

Listing 7.1 Implementing `WebApplicationInitializer` to register a servlet

```
package com.myapp.config;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration.Dynamic;
import org.springframework.web.WebApplicationInitializer;
import com.myapp.MyServlet;

public class MyServletInitializer implements WebApplicationInitializer {

  @Override
  public void onStartup(ServletContext servletContext)
          throws ServletException {

    Dynamic myServlet =
        servletContext.addServlet("myServlet", MyServlet.class);

    myServlet.addMapping("/custom/**");
  }

}
```

Register the servlet

Map the servlet

Adding additional servlets and filters

Listing 7.2 A `WebApplicationInitializer` that can also register filters

```
@Override
public void onStartup(ServletContext servletContext)
        throws ServletException {

  javax.servlet.FilterRegistration.Dynamic filter =
      servletContext.addFilter("myFilter", MyFilter.class);

  filter.addMappingForUrlPatterns(null, false, "/custom/*");
}
```

Register filter

Add filter mapping

Declaring DispatcherServlet in web.xml

Listing 7.3 Setting up Spring MVC in web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/root-context.xml</param-value>
  </context-param>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

</web-app>
```

Set root context location

Register ContextLoaderListener

Register DispatcherServlet

Map DispatcherServlet to /

Declaring DispatcherServlet in web.xml

Listing 7.4 Configuring web.xml to use Java configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <context-param>
    <param-name>contextClass</param-name>
    <param-value>
      org.springframework.web.context.support.
                              ➥ AnnotationConfigWebApplicationContext
    </param-value>
  </context-param>

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>com.habuma.spitter.config.RootConfig</param-value>
  </context-param>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
```

Use Java configuration

Specify root configuration class

Declaring DispatcherServlet in web.xml

```
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>contextClass</param-name>
    <param-value>
      org.springframework.web.context.support.
                          ➥ AnnotationConfigWebApplicationContext
    </param-value>
  </init-param>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      com.habuma.spitter.config.WebConfigConfig
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>
```

Use Java configuration

Specify DispatcherServlet configuration class

Processing multipart form data

```
firstName=Charles&lastName=Xavier&email=professorx%40xmen.org
&username=professorx&password=letmein01
```

Not robust enough to carry binary data

```
charles@xmen.com
------WebKitFormBoundaryqgkaBn8IHJCuNmiW
Content-Disposition: form-data; name="username"

professorx
------WebKitFormBoundaryqgkaBn8IHJCuNmiW
Content-Disposition: form-data; name="password"

letmein01
------WebKitFormBoundaryqgkaBn8IHJCuNmiW
Content-Disposition: form-data; name="profilePicture"; filename="me.jpg"
Content-Type: image/jpeg

  [[ Binary image data goes here ]]
------WebKitFormBoundaryqgkaBn8IHJCuNmiW--
```

Configuring a multipart resolver

Since Spring 3.1, Spring comes with two out-of-the-box implementations of `MultipartResolver` to choose from:

- `CommonsMultipartResolver`—Resolves multipart requests using Jakarta Commons FileUpload
- `StandardServletMultipartResolver`—Relies on Servlet 3.0 support for multipart requests (since Spring 3.1)

Resolving multipart requests with Servlet 3.0

The Servlet 3.0-compatible `StandardServletMultipartResolver` has no constructor arguments or properties to be set. This makes it extremely simple to declare as a bean in your Spring configuration, as shown here:

```
@Bean
public MultipartResolver multipartResolver() throws IOException {
  return new StandardServletMultipartResolver();
}
```

Resolving multipart requests with Servlet 3.0

set the temporary location to /tmp/spittr/uploads:

```
DispatcherServlet ds = new DispatcherServlet();
Dynamic registration = context.addServlet("appServlet", ds);
registration.addMapping("/");
registration.setMultipartConfig(
    new MultipartConfigElement("/tmp/spittr/uploads"));
```

Resolving multipart requests with Servlet 3.0

In addition to the temporary location path, the other constructor accepts the following:

- The maximum size (in bytes) of any file uploaded. By default there is no limit.
- The maximum size (in bytes) of the entire multipart request, regardless of how many parts or how big any of the parts are. By default there is no limit.
- The maximum size (in bytes) of a file that can be uploaded without being written to the temporary location. The default is 0, meaning that all uploaded files will be written to disk.

```
@Override
protected void customizeRegistration(Dynamic registration) {
  registration.setMultipartConfig(
      new MultipartConfigElement("/tmp/spittr/uploads",
          2097152, 4194304, 0));
}
```

Resolving multipart requests with Servlet 3.0

- Or in XML:

```
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
  <multipart-config>
    <location>/tmp/spittr/uploads</location>
    <max-file-size>2097152</max-file-size>
    <max-request-size>4194304</max-request-size>
  </multipart-config>
</servlet>
```

Handling multipart requests

Update the registrationForm.html:

```
<form method="POST" th:object="${spitter}"
      enctype="multipart/form-data">

...

  <label>Profile Picture</label>:
    <input type="file"
           name="profilePicture"
           accept="image/jpeg,image/png,image/gif" /><br/>

...

</form>
```

Tells the browser to submit the form as multipart data

Handling multipart requests

Change the processRegistration method to accept the uploaded image:

```
@RequestMapping(value="/register", method=POST)
public String processRegistration(
    @RequestPart("profilePicture") byte[] profilePicture,
    @Valid Spitter spitter,
    Errors errors) {
  ...
}
```

Receiving a MultipartFile

Listing 7.5 Spring's `MultipartFile` interface for working with uploaded files

```
package org.springframework.web.multipart;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;

public interface MultipartFile {
  String getName();
  String getOriginalFilename();
  String getContentType();
  boolean isEmpty();
  long getSize();
  byte[] getBytes() throws IOException;
  InputStream getInputStream() throws IOException;
  void transferTo(File dest) throws IOException;
}
```

Handling exceptions

Handling exceptions

Spring offers a handful of ways to translate exceptions to responses:

- Certain Spring exceptions are automatically mapped to specific HTTP status codes.
- An exception can be annotated with `@ResponseStatus` to map it to an HTTP status code.
- A method can be annotated with `@ExceptionHandler` to handle the exception.

Handling exceptions

Spring exception	HTTP status code
BindException	400 - Bad Request
ConversionNotSupportedException	500 - Internal Server Error
HttpMediaTypeNotAcceptableException	406 - Not Acceptable
HttpMediaTypeNotSupportedException	415 - Unsupported Media Type
HttpMessageNotReadableException	400 - Bad Request
HttpMessageNotWritableException	500 - Internal Server Error
HttpRequestMethodNotSupportedException	405 - Method Not Allowed
MethodArgumentNotValidException	400 - Bad Request
MissingServletRequestParameterException	400 - Bad Request
MissingServletRequestPartException	400 - Bad Request
NoSuchRequestHandlingMethodException	404 - Not Found
TypeMismatchException	400 - Bad Request

Mapping exceptions to HTTP status codes

- Could result in an HTTP 404 status (but doesn´t):

```
@RequestMapping(value="/{spittleId}", method=RequestMethod.GET)
public String spittle(
    @PathVariable("spittleId") long spittleId,
    Model model) {
  Spittle spittle = spittleRepository.findOne(spittleId);
  if (spittle == null) {
    throw new SpittleNotFoundException();
  }
  model.addAttribute(spittle);
  return "spittle";
}
```

- If the SpittleNotFoundException will result with a 500 (Internal Server Error) status code:

```
package spittr.web;
public class SpittleNotFoundException extends RuntimeException {
}
```

Mapping exceptions to HTTP status codes

Listing 7.8 `@ResponseStatus` annotation: maps exceptions to a specified status code

```
package spittr.web;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value=HttpStatus.NOT_FOUND,
                reason="Spittle Not Found")
public class SpittleNotFoundException extends RuntimeException {
}
```

Map exception to HTTP Status 404

Writing exception-handling methods

Listing 7.9 Handling an exception directly in a request-handling method

```
@RequestMapping(method=RequestMethod.POST)
public String saveSpittle(SpittleForm form, Model model) {
  try {
    spittleRepository.save(
        new Spittle(null, form.getMessage(), new Date(),
            form.getLongitude(), form.getLatitude()));
    return "redirect:/spittles";
  } catch (DuplicateSpittleException e) {
     return "error/duplicate";
  }
}
```

Catch the exception

Carrying data across redirect requests

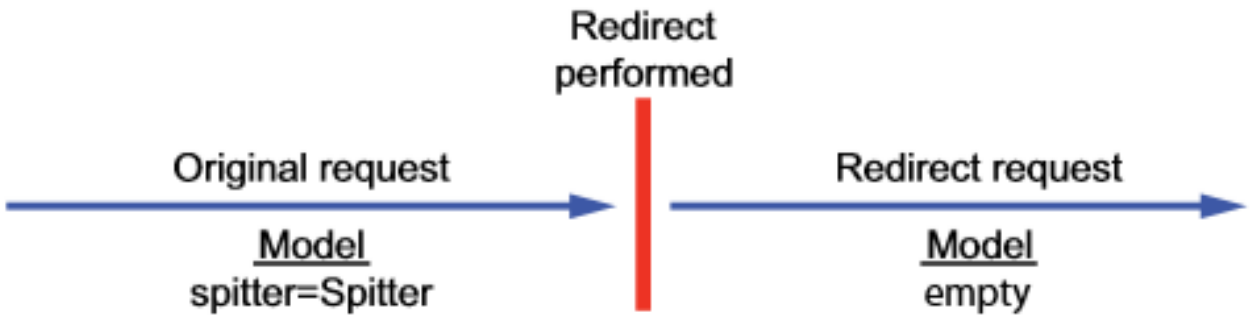


Figure 7.1 Model attributes are carried in a request as request attributes and don't survive a redirect.

But there are a couple of options to get the data from the redirecting method to the redirect-handling method:

- Passing data as path variables and/or query parameters using URL templates
- Sending data in flash attributes

Redirecting with URL templates

```
@RequestMapping(value="/register", method=POST)
public String processRegistration(
    Spitter spitter, Model model) {
  spitterRepository.save(spitter);
  model.addAttribute("username", spitter.getUsername());
  model.addAttribute("spitterId", spitter.getId());
  return "redirect:/spitter/{username}";
}
```

Working with flash attributes

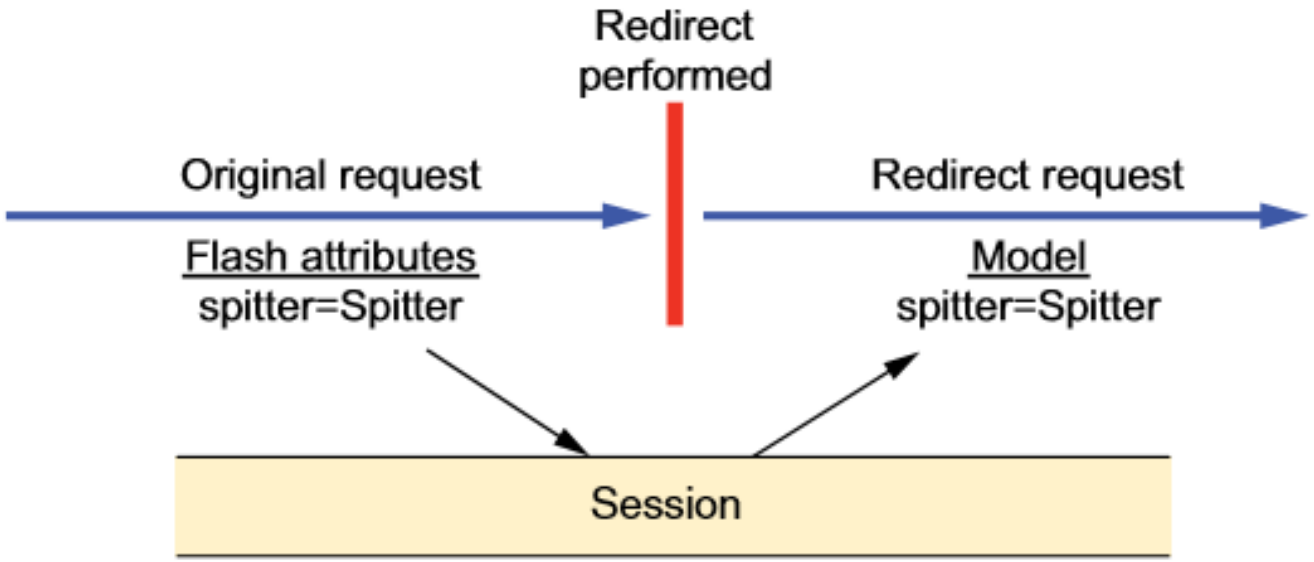


Figure 7.2 Flash attributes are stored in the session and then retrieved into the model, surviving a redirect.

Working with flash attributes

```
@RequestMapping(value="/register", method=POST)
public String processRegistration(
    Spitter spitter, RedirectAttributes model) {
  spitterRepository.save(spitter);
  model.addAttribute("username", spitter.getUsername());
  model.addFlashAttribute("spitter", spitter);
  return "redirect:/spitter/{username}";
}
```

Working with Spring Web Flow

Configuring Web Flow in Spring

Only XML support:

```
<?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:flow="http://www.springframework.org/schema/webflow-config"
 xsi:schemaLocation=
  "http://www.springframework.org/schema/webflow-config
   http://www.springframework.org/schema/webflow-config/[CA]
   spring-webflow-config-2.3.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
```

Wiring a flow executor

The `<flow:flow-executor>` element creates a flow executor in Spring:

```
<flow:flow-executor id="flowExecutor" />
```

Although the flow executor is responsible for creating and executing flows, it's not responsible for loading flow definitions. That responsibility falls to a flow registry, which you'll create next.

Configuring a flow registry


```
<flow:flow-registry id="flowRegistry">
   <flow:flow-location id="pizza"
         path="/WEB-INF/flows/springpizza.xml" />
</flow:flow-registry>
```

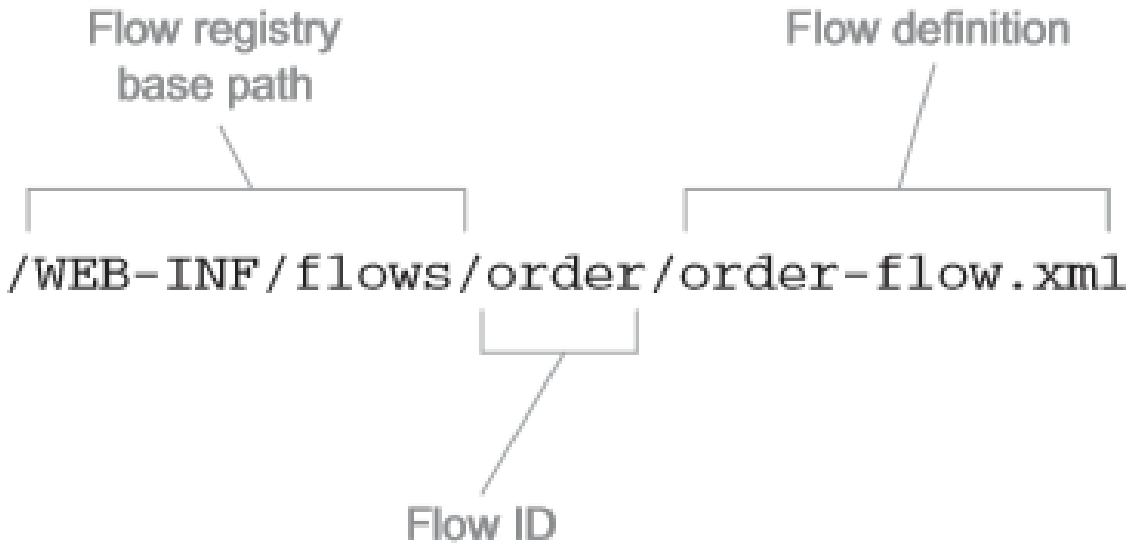


Figure 8.1 When using a flow location pattern, the path to the flow definition file relative to the base path is used as the flow's ID.

Handling flow requests

- FlowHandlerMapping:

```
<bean class=
        "org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
  <property name="flowRegistry" ref="flowRegistry" />
</bean>
```

- FlowHandlerAdapter

```
<bean class=
        "org.springframework.webflow.mvc.servlet.FlowHandlerAdapter">
  <property name="flowExecutor" ref="flowExecutor" />
</bean>
```

States

State type	What it's for
Action	Action states are where the logic of a flow takes place.
Decision	Decision states branch the flow in two directions, routing the flow based on the outcome of evaluating flow data.
End	The end state is the last stop for a flow. Once a flow has reached its end state, the flow is terminated.
Subflow	A subflow state starts a new flow in the context of a flow that is already underway.
View	A view state pauses the flow and invites the user to participate in the flow.

Action state

In the flow definition XML, action states are expressed with the <action-state> element. Here's an example:

```
<action-state id="saveOrder">
  <evaluate expression="pizzaFlowActions.saveOrder(order)" />
  <transition to="thankYou" />
</action-state>
```

Decision state

```
<decision-state id="checkDeliveryArea">
  <if test="pizzaFlowActions.checkDeliveryArea(customer.zipCode)"
      then="addCustomer"
      else="deliveryWarning" />
</decision-state>
```

Flow data

- Daclaring variables:

```
<var name="customer" class="com.springinaction.pizza.domain.Customer"/>

  <evaluate result="viewScope.toppingsList"
     expression="T(com.springinaction.pizza.domain.Topping).asList()" />

     <set name="flowScope.pizza"
        value="new com.springinaction.pizza.domain.Pizza()" />
```

Scoping flow data

Scope	Lifespan and visibility
Conversation	Created when a top-level flow starts, and destroyed when the top-level flow ends. Shared by a top-level flow and all of its subflows.
Flow	Created when a flow starts, and destroyed when the flow ends. Only visible in the flow it was created by.
Request	Created when a request is made into a flow, and destroyed when the flow returns.
Flash	Created when a flow starts, and destroyed when the flow ends. It's also cleared out after a view state renders.
View	Created when a view state is entered, and destroyed when the state exits. Visible only in the view state.

Putting it all together: the pizza flow (example)

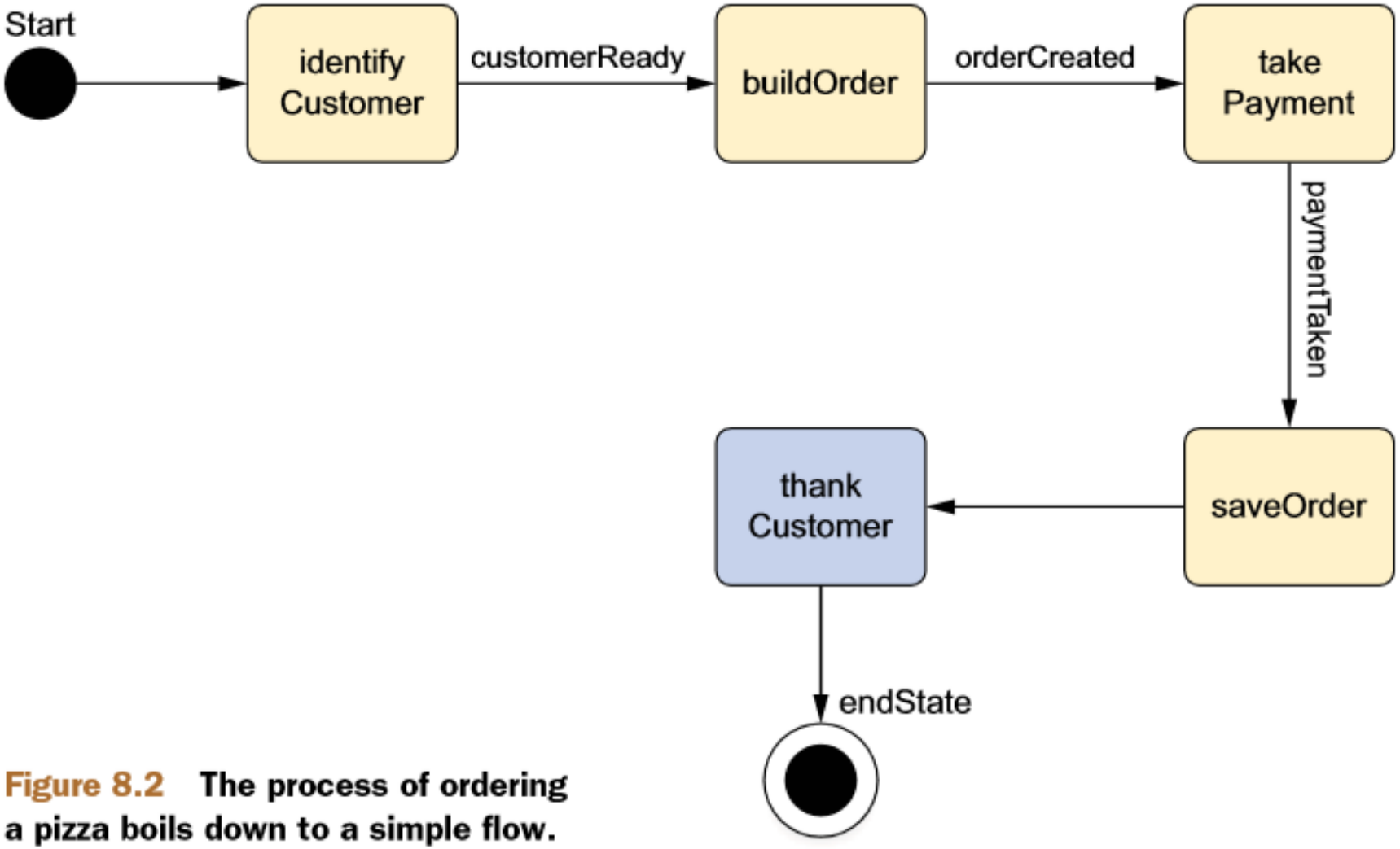


Figure 8.2 The process of ordering a pizza boils down to a simple flow.

Listing 8.1 Pizza order flow, defined as a Spring Web Flow

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/webflow
  http://www.springframework.org/schema/webflow/spring-webflow-2.3.xsd">
  <var name="order"
       class="com.springinaction.pizza.domain.Order"/>
  <subflow-state id="identifyCustomer" subflow="pizza/customer">
    <output name="customer" value="order.customer"/>
    <transition on="customerReady" to="buildOrder" />
  </subflow-state>
  <subflow-state id="buildOrder" subflow="pizza/order">
    <input name="order" value="order"/>
    <transition on="orderCreated" to="takePayment" />
  </subflow-state>
  <subflow-state id="takePayment" subflow="pizza/payment">
    <input name="order" value="order"/>
    <transition on="paymentTaken" to="saveOrder"/>
  </subflow-state>
```

Call customer subflow

Call order subflow

Call payment subflow

```
  <action-state id="saveOrder">                          <-- Save order
    <evaluate expression="pizzaFlowActions.saveOrder(order)" />
    <transition to="thankCustomer" />
  </action-state>
  <view-state id="thankCustomer">                        <-- Thank customer
    <transition to="endState" />
  </view-state>
  <end-state id="endState" />
  <global-transitions>
    <transition on="cancel" to="endState" />             <-- Global cancel transition
  </global-transitions>
</flow>
```

Listing 8.2 `Order`: carries all the details pertaining to a pizza order

```
package com.springinaction.pizza.domain;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
public class Order implements Serializable {
   private static final long serialVersionUID = 1L;
   private Customer customer;
   private List<Pizza> pizzas;
   private Payment payment;

   public Order() {
      pizzas = new ArrayList<Pizza>();
      customer = new Customer();
   }
   public Customer getCustomer() {
      return customer;
   }
   public void setCustomer(Customer customer) {
      this.customer = customer;
   }
```

```
    public List<Pizza> getPizzas() {
        return pizzas;
    }
    public void setPizzas(List<Pizza> pizzas) {
        this.pizzas = pizzas;
    }
    public void addPizza(Pizza pizza) {
        pizzas.add(pizza);
    }
    public float getTotal() {
        return 0.0f;
    }
    public Payment getPayment() {
        return payment;
    }
    public void setPayment(Payment payment) {
        this.payment = payment;
    }
}
```

Defining the base flow

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/webflow
  http://www.springframework.org/schema/webflow/spring-webflow-2.3.xsd"
  start-state="identifyCustomer">
...
</flow>
```

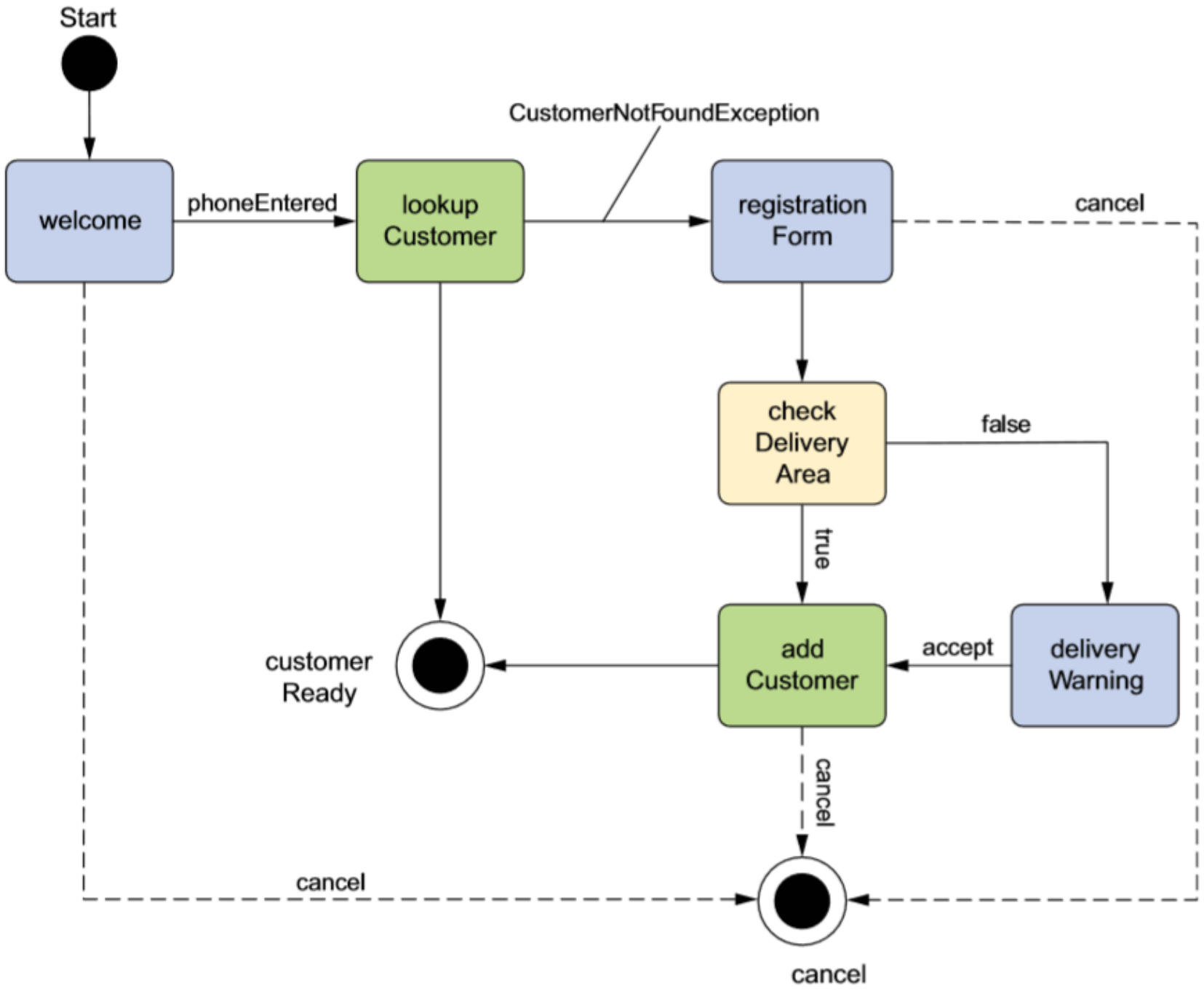
Defining the base flow

Listing 8.3 JSP view that thanks the customer for their order

```
<html xmlns:jsp="http://java.sun.com/JSP/Page">
  <jsp:output omit-xml-declaration="yes"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8" />
  <head><title>Spizza</title></head>
  <body>
    <h2>Thank you for your order!</h2>
    <![CDATA[
    <a href='${flowExecutionUrl}&_eventId=finished'>Finish</a>
    ]]>
    </body>
</html>
```

Fire finished event

Collecting customer information



Collecting customer information

Listing 8.4 Identifying the hungry pizza customer with a web flow

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/webflow
http://www.springframework.org/schema/webflow/spring-webflow-2.3.xsd">
<var name="customer"
     class="com.springinaction.pizza.domain.Customer"/>
<view-state id="welcome">                                    <-- Welcome customer
   <transition on="phoneEntered" to="lookupCustomer"/>
</view-state>
<action-
   state id="lookupCustomer">                                <-- Look up customer
   <evaluate result="customer" expression=
 "pizzaFlowActions.lookupCustomer(requestParameters.phoneNumber)" />
  <transition to="registrationForm" on-exception=
      "com.springinaction.pizza.service.CustomerNotFoundException" />
  <transition to="customerReady" />
</action-state>
```

```
  <view-state id="registrationForm" model="customer">
     <on-entry>
      <evaluate expression=
          "customer.phoneNumber = requestParameters.phoneNumber" />
    </on-entry>
    <transition on="submit" to="checkDeliveryArea" />
  </view-state>
  <decision-state id="checkDeliveryArea">
     <if test="pizzaFlowActions.checkDeliveryArea(customer.zipCode)"
        then="addCustomer"
        else="deliveryWarning"/>
  </decision-state>
  <view-state id="deliveryWarning">
     <transition on="accept" to="addCustomer" />
  </view-state>
  <action-state id="addCustomer">
     <evaluate expression="pizzaFlowActions.addCustomer(customer)" />
    <transition to="customerReady" />
  </action-state>
  <end-state id="cancel" />
  <end-state id="customerReady">
    <output name="customer" />
  </end-state>
  <global-transitions>
    <transition on="cancel" to="cancel" />
  </global-transitions>
</flow>
```

Register new customer

Check delivery area

Show delivery warning

Add customer

Asking for the telephone number

Listing 8.5 Welcoming the customer and asking for their phone number

```
<html xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:form="http://www.springframework.org/tags/form">
  <jsp:output omit-xml-declaration="yes"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8" />
  <head><title>Spizza</title></head>
  <body>
    <h2>Welcome to Spizza!!!</h2>
    <form:form>
      <input type="hidden" name="_flowExecutionKey"
             value="${flowExecutionKey}"/>
       <input type="text" name="phoneNumber"/><br/>
      <input type="submit" name="_eventId_phoneEntered"
             value="Lookup Customer" />
    </form:form>
  </body>
</html>
```

← Flow execution key

← Fire phoneEntered event

Registering a new customer

Listing 8.6 Registering a new customer

```
<html xmlns:c="http://java.sun.com/jsp/jstl/core"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:spring="http://www.springframework.org/tags"
      xmlns:form="http://www.springframework.org/tags/form">
  <jsp:output omit-xml-declaration="yes"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8" />
  <head><title>Spizza</title></head>
  <body>
    <h2>Customer Registration</h2>
    <form:form commandName="customer">
      <input type="hidden" name="_flowExecutionKey"
             value="${flowExecutionKey}"/>
      <b>Phone number: </b><form:input path="phoneNumber"/><br/>
      <b>Name: </b><form:input path="name"/><br/>
      <b>Address: </b><form:input path="address"/><br/>
      <b>City: </b><form:input path="city"/><br/>
      <b>State: </b><form:input path="state"/><br/>
      <b>Zip Code: </b><form:input path="zipCode"/><br/>
      <input type="submit" name="_eventId_submit"
             value="Submit" />
      <input type="submit" name="_eventId_cancel"
             value="Cancel" />
    </form:form>
    </body>
</html>
```

Checking the delivery area

Listing 8.7 Warning a customer that pizza can't be delivered to their address

```
<html xmlns:jsp="http://java.sun.com/JSP/Page">
  <jsp:output omit-xml-declaration="yes"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8" />
  <head><title>Spizza</title></head>
  <body>
        <h2>Delivery Unavailable</h2>
        <p>The address is outside of our delivery area. You may
        still place the order, but you will need to pick it up
        yourself.</p>
        <![CDATA[
        <a href="${flowExecutionUrl}&_eventId=accept">
                                Continue, I'll pick up the order</a> |
        <a href="${flowExecutionUrl}&_eventId=cancel">Never mind</a>
        ]]>
  </body>
</html>
```

Building an order

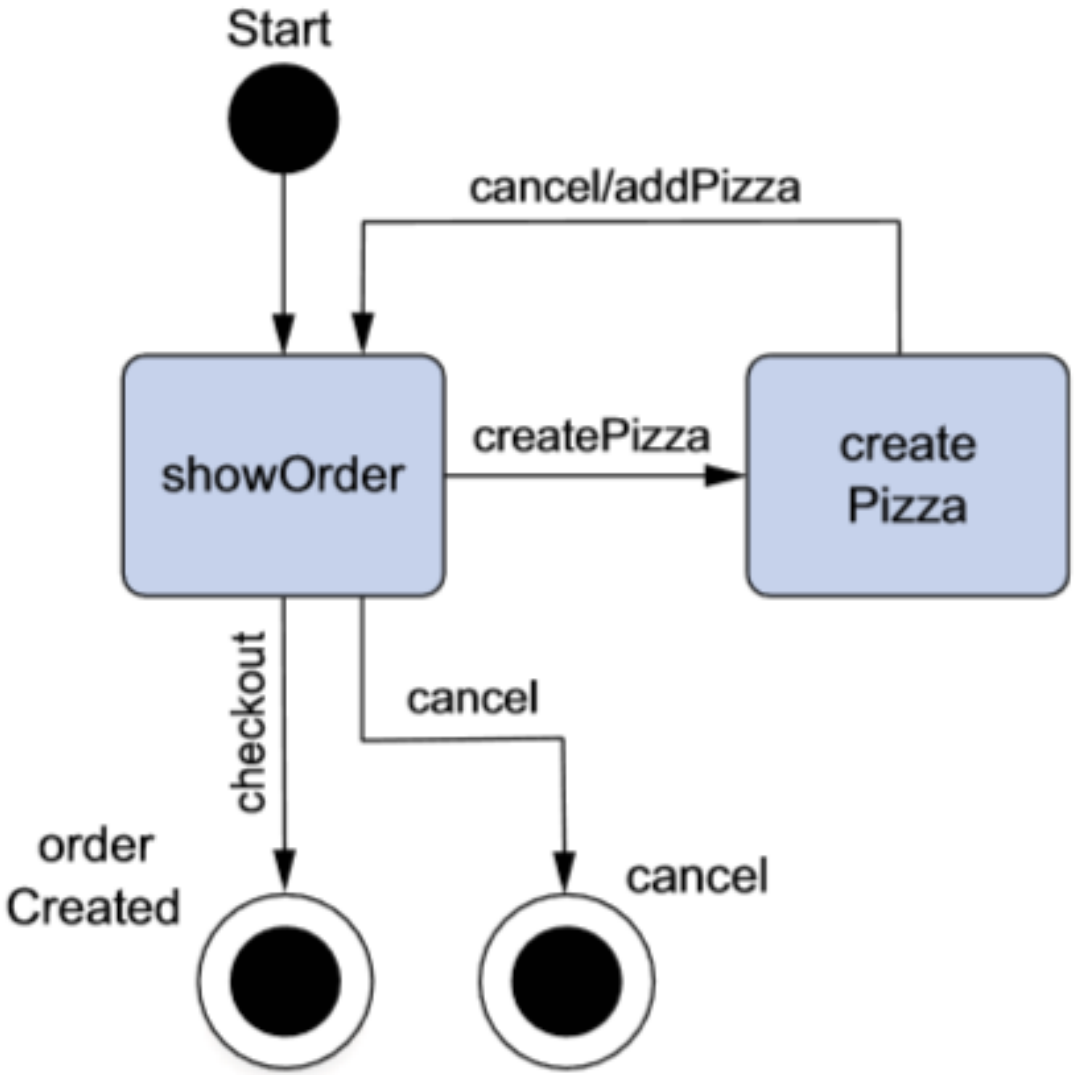


Figure 8.4 Pizzas are added via the order subflow.

Listing 8.8 Order subflow view shows states to display the order and create a pizza

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/webflow
  http://www.springframework.org/schema/webflow/spring-webflow-2.3.xsd">

  <input name="order" required="true" />          <-- Accept order as input

  <view-state id="showOrder">                     <-- Order display state
     <transition on="createPizza" to="createPizza" />
    <transition on="checkout" to="orderCreated" />
    <transition on="cancel" to="cancel" />
  </view-state>
```

```
<view-
   state id="createPizza" model="flowScope.pizza">
   <on-entry>
    <set name="flowScope.pizza"
        value="new com.springinaction.pizza.domain.Pizza()" />
    <evaluate result="viewScope.toppingsList"  expression=
         "T(com.springinaction.pizza.domain.Topping).asList()" />
  </on-entry>
  <transition on="addPizza" to="showOrder">
    <evaluate expression="order.addPizza(flowScope.pizza)" />
  </transition>
  <transition on="cancel" to="showOrder" />
</view-state>

<end-state id="cancel" /

<end-state id="orderCreated" />

</flow>
```

Pizza creation state

Cancel end state

Create order end state

Listing 8.9 Adding pizzas with an HTML form bound to a flow-scoped object

```
<div xmlns:form="http://www.springframework.org/tags/form"
     xmlns:jsp="http://java.sun.com/JSP/Page">
  <jsp:output omit-xml-declaration="yes"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8" />
    <h2>Create Pizza</h2>
    <form:form commandName="pizza">
      <input type="hidden" name="_flowExecutionKey"
          value="${flowExecutionKey}"/>
       <b>Size: </b><br/>
   <form:radiobutton path="size"
                     label="Small (12-inch)" value="SMALL"/><br/>
   <form:radiobutton path="size"
                     label="Medium (14-inch)" value="MEDIUM"/><br/>
   <form:radiobutton path="size"
                     label="Large (16-inch)" value="LARGE"/><br/>
   <form:radiobutton path="size"
                     label="Ginormous (20-inch)" value="GINORMOUS"/>
       <br/>
       <br/>
       <b>Toppings: </b><br/>
       <form:checkboxes path="toppings" items="${toppingsList}"
                        delimiter="&lt;br/&gt;"/><br/><br/>
       <input type="submit" class="button"
           name="_eventId_addPizza" value="Continue"/>
       <input type="submit" class="button"
           name="_eventId_cancel" value="Cancel"/>
    </form:form>
</div>
```

Taking payment

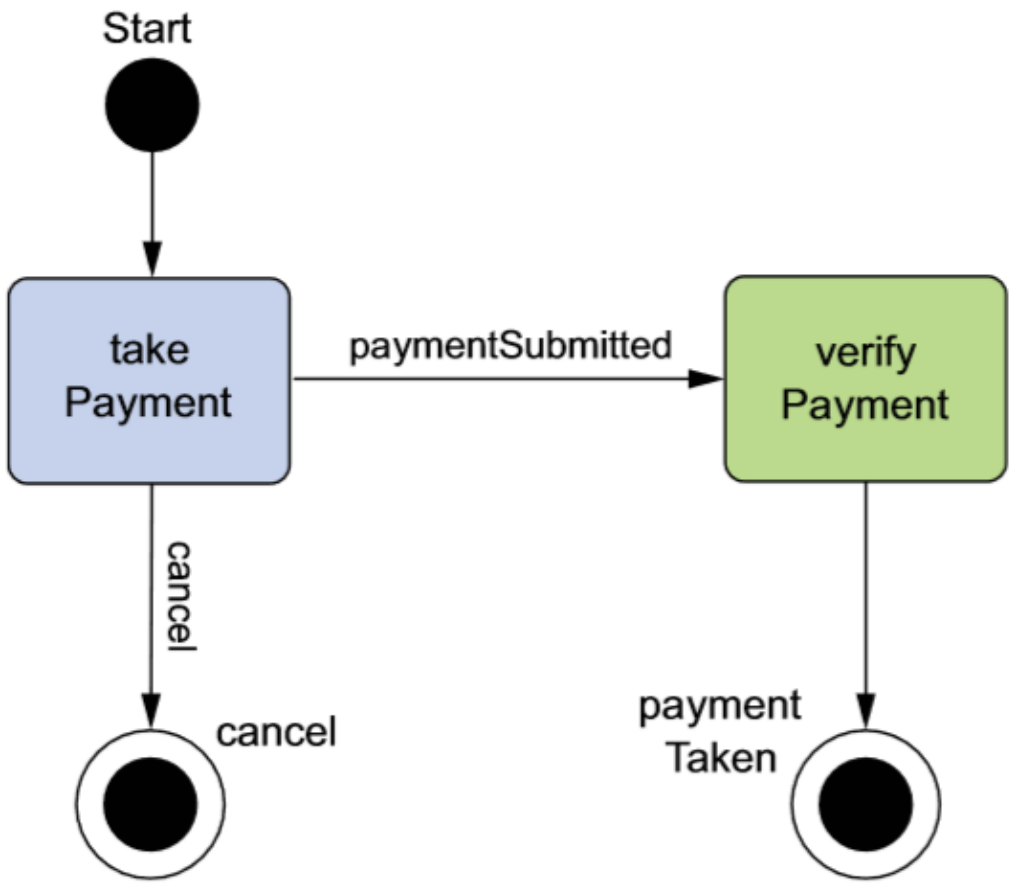


Figure 8.5 The final step in placing a pizza order is to take payment from the customer through the payment subflow.

Taking payment

Listing 8.10 Payment subflow, with one view state and one action state

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/webflow
  http://www.springframework.org/schema/webflow/spring-webflow-2.3.xsd">
  <input name="order" required="true"/>
  <view-state id="takePayment" model="flowScope.paymentDetails">
    <on-entry>
      <set name="flowScope.paymentDetails"
 value="new com.springinaction.pizza.domain.PaymentDetails()" />
      <evaluate result="viewScope.paymentTypeList" expression=
"T(com.springinaction.pizza.domain.PaymentType).asList()" />
    </on-entry>
    <transition on="paymentSubmitted" to="verifyPayment" />
    <transition on="cancel" to="cancel" />
  </view-state>
  <action-state id="verifyPayment">
    <evaluate result="order.payment" expression=
        "pizzaFlowActions.verifyPayment(flowScope.paymentDetails)" />
    <transition to="paymentTaken" />
  </action-state>

  <end-state id="cancel" />
  <end-state id="paymentTaken" />
</flow>
```

Taking payment

Listing 8.11 `PaymentType` enumeration: defines customer choices for payment

```
package com.springinaction.pizza.domain;
import static org.apache.commons.lang.WordUtils.*;
import java.util.Arrays;
import java.util.List;
public enum PaymentType {
  CASH, CHECK, CREDIT_CARD;
  public static List<PaymentType> asList() {
    PaymentType[] all = PaymentType.values();
    return Arrays.asList(all);
  }
  @Override
  public String toString() {
    return capitalizeFully(name().replace('_', ' '));
  }
}
```

Securing web applications

Spring security modules

Module	Description
ACL	Provides support for domain object security through access control lists (ACLs).
Aspects	A small module providing support for AspectJ-based aspects instead of standard Spring AOP when using Spring Security annotations.
CAS Client	Support for single sign-on authentication using Jasig's Central Authentication Service (CAS).
Configuration	Contains support for configuring Spring Security with XML and Java. (Java configuration support introduced in Spring Security 3.2.)
Core	Provides the essential Spring Security library.
Cryptography	Provides support for encryption and password encoding.
LDAP	Provides support for LDAP-based authentication.
OpenID	Contains support for centralized authentication with OpenID.
Remoting	Provides integration with Spring Remoting.
Tag Library	Spring Security's JSP tag library.
Web	Provides Spring Security's filter-based web security support.

Filtering web requests

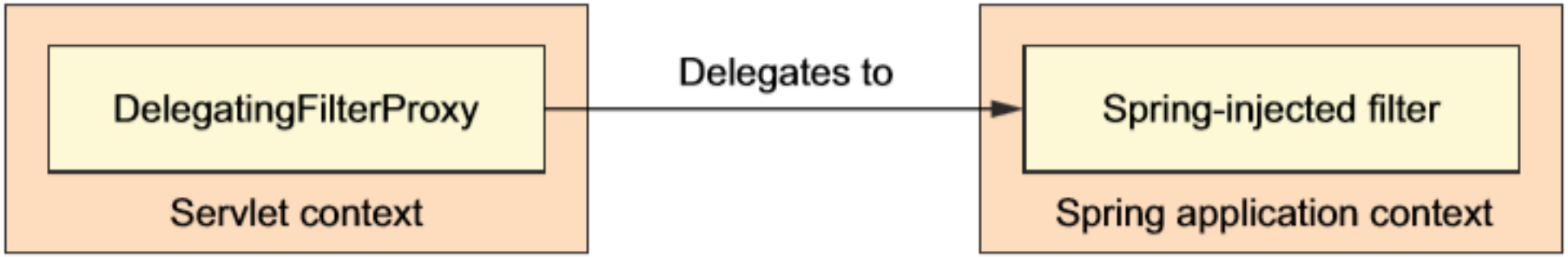


Figure 9.1 `DelegatingFilterProxy` proxies filter handling to a delegate filter bean in the Spring application context.

If you like configuring servlets and filters in the traditional web.xml file, you can do that with the `<filter>` element, like this:

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>
        org.springframework.web.filter.DelegatingFilterProxy
    </filter-class>
</filter>
```

Writing a simple security configuration

Listing 9.1 The simplest configuration class to enable web security for Spring MVC

```
package spitter.config;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.
                                                configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.
                                        configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
}
```

Enable web security

Writing a simple security configuration

`@EnableWebSecurity` is generally useful for enabling security in any web application. But if you happen to be developing a Spring MVC application, you should consider using `@EnableWebMvcSecurity` instead, as shown in the following listing.

Listing 9.2 The simplest configuration class to enable web security for Spring MVC

```
package spitter.config;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.
                                configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.annotation.web.servlet.
                                        configuration.EnableWebMvcSecurity;

@Configuration
@EnableWebMvcSecurity
 public class SecurityConfig extends WebSecurityConfigurerAdapter {
}
```

Enable Spring MVC security

Writing a simple security configuration

Table 9.2 Overriding `WebSecurityConfigurerAdapter`'s configure() methods

Method	Description
`configure(WebSecurity)`	Override to configure Spring Security's filter chain.
`configure(HttpSecurity)`	Override to configure how requests are secured by interceptors.
`configure(AuthenticationManagerBuilder)`	Override to configure user-details services.

Selecting user details services

Working with an in-memory user store

Listing 9.3 Configuring Spring Security to use an in-memory user store

```
package spitter.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.
                    authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.
                                configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.annotation.web.servlet.
                                        configuration.EnableWebMvcSecurity;

@Configuration
@EnableWebMvcSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(AuthenticationManagerBuilder auth)
                                                        throws Exception {

    auth
    .inMemoryAuthentication()
       .withUser("user").password("password").roles("USER").and()
      .withUser("admin").password("password").roles("USER", "ADMIN");
  }
```

Enable an in-memory user store.

Table 9.3 Methods for configuring user details

Module	Description
`accountExpired(boolean)`	Defines if the account is expired or not
`accountLocked(boolean)`	Defines if the account is locked or not
`and()`	Used for chaining configuration
`authorities(GrantedAuthority…)`	Specifies one or more authorities to grant to the user
`authorities(List<? extends GrantedAuthority>)`	Specifies one or more authorities to grant to the user
`authorities(String…)`	Specifies one or more authorities to grant to the user
`credentialsExpired(boolean)`	Defines if the credentials are expired or not
`disabled(boolean)`	Defines if the account is disabled or not
`password(String)`	Specifies the user's password
`roles(String…)`	Specifies one or more roles to assign to the user

Authenticating against database tables

```
@Autowired
DataSource dataSource;

@Override
protected void configure(AuthenticationManagerBuilder auth)
                                                    throws Exception {
  auth
    .jdbcAuthentication()
      .dataSource(dataSource);
}
```

The only thing you must configure is a `DataSource` so that it's able to access the relational database. The `DataSource` is provided here via the magic of autowiring.

Springs assumptions about your database schema:

```
public static final String DEF_USERS_BY_USERNAME_QUERY =
        "select username,password,enabled " +
        "from users " +
        "where username = ?";
public static final String DEF_AUTHORITIES_BY_USERNAME_QUERY =
        "select username,authority " +
        "from authorities " +
        "where username = ?";
public static final String DEF_GROUP_AUTHORITIES_BY_USERNAME_QUERY =
        "select g.id, g.group_name, ga.authority " +
        "from groups g, group_members gm, group_authorities ga " +
        "where gm.username = ? " +
        "and g.id = ga.group_id " +
        "and g.id = gm.group_id";
```

Configuring an embedded LDAP server

```
@Override
protected void configure(AuthenticationManagerBuilder auth)
        throws Exception {
  auth
  .ldapAuthentication()
    .userSearchBase("ou=people")
    .userSearchFilter("(uid={0})")
    .groupSearchBase("ou=groups")
    .groupSearchFilter("member={0}")
    .contextSource()
      .root("dc=habuma,dc=com");
}
```

Applying LDAP-backed authentication

```
@Override
protected void configure(AuthenticationManagerBuilder auth)
                                                    throws Exception {
  auth
    .ldapAuthentication()
      .userSearchFilter("(uid={0})")
      .groupSearchFilter("member={0}");
}
```

Configuring password comparison

```
@Override
protected void configure(AuthenticationManagerBuilder auth)
                                                    throws Exception {
  auth
    .ldapAuthentication()
      .userSearchBase("ou=people")
      .userSearchFilter("(uid={0})")
      .groupSearchBase("ou=groups")
      .groupSearchFilter("member={0}")
      .passwordCompare()
      .passwordEncoder(new Md5PasswordEncoder())
      .passwordAttribute("passcode");
}
```

Working with encoded passwords

```
@Override
protected void configure(AuthenticationManagerBuilder auth)
                                                    throws Exception {
  auth
    .jdbcAuthentication()
      .dataSource(dataSource)
      .usersByUsernameQuery(
        "select username, password, true " +
        "from Spitter where username=?")
      .authoritiesByUsernameQuery(
        "select username, 'ROLE_USER' from Spitter where username=?")
      .passwordEncoder(new StandardPasswordEncoder("53cr3t"));
}
```

Referring to a remote LDAP server

```
@Override
protected void configure(AuthenticationManagerBuilder auth)
        throws Exception {
  auth
  .ldapAuthentication()
    .userSearchBase("ou=people")
    .userSearchFilter("(uid={0})")
    .groupSearchBase("ou=groups")
    .groupSearchFilter("member={0}")
    .contextSource()
      .url("ldap://habuma.com:389/dc=habuma,dc=com");
}
```

Configuring a custom user service

Listing 9.4 Retrieve a UserDetails object from a SpitterRepository

```
package spittr.security;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.
                                                SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.
                                                    UserDetailsService;
import org.springframework.security.core.userdetails.
                                              UsernameNotFoundException;
import spittr.Spitter;
import spittr.data.SpitterRepository;

public class SpitterUserService implements UserDetailsService {

  private final SpitterRepository spitterRepository;

  public SpitterUserService(SpitterRepository spitterRepository) {
     this.spitterRepository = spitterRepository;
  }
```

Inject SpitterRepository

Configuring a custom user service

```
@Override
public UserDetails loadUserByUsername(String username)
    throws UsernameNotFoundException {
  Spitter spitter = spitterRepository.findByUsername(username);
   if (spitter != null) {
    List<GrantedAuthority> authorities =
        new ArrayList<GrantedAuthority>();
    authorities.add(new SimpleGrantedAuthority("ROLE_SPITTER"));

    return new User(
         spitter.getUsername(),
        spitter.getPassword(),
        authorities);
  }

    throw new UsernameNotFoundException(
        "User '" + username + "' not found.");
  }

}
```

Look up Spitter

Create authorities list

Return a User

Intercepting requests

Intercepting requests

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
    .authorizeRequests()
      .antMatchers("/spitters/me").authenticated()
      .antMatchers(HttpMethod.POST, "/spittles").authenticated()
      .anyRequest().permitAll();
}
```

Configuration methods to define how a path is to be secured

Method	What It does
`access(String)`	Allows access if the given SpEL expression evaluates to `true`
`anonymous()`	Allows access to anonymous users
`authenticated()`	Allows access to authenticated users
`denyAll()`	Denies access unconditionally
`fullyAuthenticated()`	Allows access if the user is fully authenticated (not remembered)

Configuration methods to define how a path is to be secured

Method	What It does
`hasAuthority(String)`	Allows access if the user has the given authority
`hasIpAddress(String)`	Allows access if the request comes from the given IP address
`hasRole(String)`	Allows access if the user has the given role
`not()`	Negates the effect of any of the other access methods
`permitAll()`	Allows access unconditionally
`rememberMe()`	Allows access for users who are authenticated via remember-me

Intercepting requests

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
    .authorizeRequests()
      .antMatchers("/spitters/me").hasAuthority("ROLE_SPITTER")
      .antMatchers(HttpMethod.POST, "/spittles")
                                   .hasAuthority("ROLE_SPITTER")
      .anyRequest().permitAll();
}
```

Securing with Spring Expressions

Table 9.5 Spring Security extends the Spring Expression Language with several security-specific expressions

Security expression	What it evaluates to
`authentication`	The user's authentication object
`denyAll`	Always evaluates to false
`hasAnyRole(list of roles)`	True if the user has any of the given roles
`hasRole(role)`	True if the user has the given role
`hasIpAddress(IP address)`	True if the request comes from the given IP address
`isAnonymous()`	True if the user is anonymous
`isAuthenticated()`	True if the user is authenticated

Securing with Spring Expressions

Security expression	What it evaluates to
`isFullyAuthenticated()`	True if the user is fully authenticated (not authenticated with remember-me)
`isRememberMe()`	True if the user was authenticated via remember-me
`permitAll`	Always evaluates to true
`principal`	The user's principal object

Securing with Spring Expressions

- Example:

```
.antMatchers("/spitter/me")
  .access("hasRole('ROLE_SPITTER') and hasIpAddress('192.168.1.2')")
```

Enforcing channel security

Listing 9.5 The `requiresChannel()` method enforces HTTPS for select URLs

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
    .authorizeRequests()
      .antMatchers("/spitter/me").hasRole("SPITTER")
      .antMatchers(HttpMethod.POST, "/spittles").hasRole("SPITTER")
      .anyRequest().permitAll();
    .and()
    .requiresChannel()
      .antMatchers("/spitter/form").requiresSecure();      <-- Require HTTPS
 }
```

Authenticating users

Listing 9.7 The `formLogin()` method enables a basic login page

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
    .formLogin()                                    <-- Enable default login page
     .and()
    .authorizeRequests()
      .antMatchers("/spitter/me").hasRole("SPITTER")
      .antMatchers(HttpMethod.POST, "/spittles").hasRole("SPITTER")
      .anyRequest().permitAll();
    .and()
    .requiresChannel()
      .antMatchers("/spitter/form").requiresSecure();
}
```

Default login page

```
<html>
<head><title>Login Page</title></head>
<body onload='document.f.username.focus();'>
<h3>Login with Username and Password</h3>
<form name='f' action='/spittr/login' method='POST'>
 <table>
    <tr><td>User:</td><td>
        <input type='text' name='username' value=''></td></tr>
    <tr><td>Password:</td>
        <td><input type='password' name='password'/></td></tr>
    <tr><td colspan='2'>
        <input name="submit" type="submit" value="Login"/></td></tr>
    <input name="_csrf" type="hidden"
           value="6829b1ae-0a14-4920-aac4-5abbd7eeb9ee" />
  </table>
</form>
</body>
</html>
```

Adding a custom login page

Listing 9.8 A custom login page for the Spittr application (as a Thymeleaf template)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Spitter</title>
    <link rel="stylesheet"
          type="text/css"
          th:href="@{/resources/style.css}"></link>
  </head>
  <body onload='document.f.username.focus();'>
    <div id="header" th:include="page :: header"></div>

  <div id="content">
  <form name='f' th:action='@{/login}' method='POST'>      <-- Submit to /login
    <table>
    <tr><td>User:</td><td>
        <input type='text' name='username' value='' /></td></tr>
    <tr><td>Password:</td>
        <td><input type='password' name='password'/></td></tr>
    <tr><td colspan='2'>
        <input name="submit" type="submit" value="Login"/></td></tr>
     </table>
   </form>
   </div>
   <div id="footer" th:include="page :: copy"></div>
   </body>
</html>
```

Enabling HTTP Basic authentication

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
    .formLogin()
      .loginPage("/login")
    .and()
    .httpBasic()
      .realmName("Spittr")
    .and()
  ...
}
```

Enabling remember-me functionality

```
@Override
  protected void configure(HttpSecurity http) throws Exception {
    http
      .formLogin()
        .loginPage("/login")
      .and()
      .rememberMe()
        .tokenValiditySeconds(2419200)
        .key("spittrKey")
  ...
}
```

Enabling remember-me functionality

- Checkbox for the user to activate remember-me function:

```
<input id="remember_me" name="remember-me" type="checkbox"/>
<label for="remember_me" class="inline">Remember me</label>
```

Logging out

- Simple adding the following link:

```
<a th:href="@{/logout}">Logout</a>
```

Logging out

- Overriding the default redirect link after the logout

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
    .formLogin()
      .loginPage("/login")
    .and()
    .logout()
      .logoutSuccessUrl("/")
  ...
}
```

Securing the view

Using Spring Security´s JSP tag library

Table 9.6 Spring Security supports security in the view layer with a JSP tag library

JSP tag	What it does
`<security:accesscontrollist>`	Conditionally renders its body content if the user is granted authorities by an access control list
`<security:authentication>`	Renders details about the current authentication
`<security:authorize>`	Conditionally renders its body content if the user is granted certain authorities or if a SpEL expression evaluates to `true`

Using Spring Security´s JSP tag library

To use the JSP tag library, we'll need to declare it in any JSP file where it will be used:

```
<%@ taglib prefix="security"
           uri="http://www.springframework.org/security/tags" %>
```

Accessing authentication details

Here's an example:

```
Hello <security:authentication property="principal.username" />!
```

Table 9.7 You can access several of the user's authentication details using the `<security:authentication>` JSP tag

Authentication property	Description
`authorities`	A collection of `GrantedAuthority` objects that represent the privileges granted to the user

Accessing authentication details

Authentication property	Description
`credentials`	The credentials that were used to verify the principal (commonly, this is the user's password)
`details`	Additional information about the authentication (IP address, certificate serial number, session ID, and so on)
`principal`	The user's principal

Conditional rendering

Listing 9.9 `<sec:authorize>` conditionally renders content based on SpEL

```
<sec:authorize access="hasRole('ROLE_SPITTER')">
 <s:url value="/spittles" var="spittle_url" />
  <sf:form modelAttribute="spittle"
           action="${spittle_url}">
  <sf:label path="text"><s:message code="label.spittle"
                  text="Enter spittle:"/></sf:label>
  <sf:textarea path="text" rows="2" cols="40" />
      <sf:errors path="text" />

  <br/>
  <div class="spitItSubmitIt">
    <input type="submit" value="Spit it!"
         class="status-btn round-btn disabled" />
  </div>
 </sf:form>
</sec:authorize>
```

Only with ROLE_SPITTER authority

Conditional rendering

```
<security:authorize
    access="isAuthenticated() and principal.username=='habuma'">
  <a href="/admin">Administration</a>
</security:authorize>
```

Could be manually entered into the browser!

```
.antMatchers("/admin")
  .access("isAuthenticated() and principal.username=='habuma'");
```

Working with Thymeleaf´s Spring Security dialect

Attribute	What it does
`sec:authentication`	Renders properties of the authentication object. Similar to Spring Security's `<sec:authentication/>` JSP tag.
`sec:authorize`	Conditionally renders content based on evaluation of an expression. Similar to Spring Security's `<sec:authorize/>` JSP tag.
`sec:authorize-acl`	Conditionally renders content based on evaluation of an expression. Similar to Spring Security's `<sec:accesscontrollist/>` JSP tag.
`sec:authorize-expr`	An alias for the `sec:authorize` attribute.
`sec:authorize-url`	Conditionally renders content based on evaluation of security rules associated with a given URL path. Similar to Spring Security's `<sec:authorize/>` JSP tag when using the `url` attribute.

Working with Thymeleaf´s Spring Security dialect

Listing 9.10 Registering Thymeleaf's Spring Security dialect

```
@Bean
public SpringTemplateEngine templateEngine(
                                    TemplateResolver templateResolver) {
  SpringTemplateEngine templateEngine = new SpringTemplateEngine();
  templateEngine.setTemplateResolver(templateResolver);
  templateEngine.addDialect(new SpringSecurityDialect());
   return templateEngine;
}
```

Register the security dialect

Working with Thymeleaf´s Spring Security dialect

With the security dialect, you're almost ready to start using its attributes in your Thymeleaf templates. First, declare the security namespace in the templates where you'll be using those attributes:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org"
      xmlns:sec=
          "http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
  ...
</html>
```

Working with Thymeleaf´s Spring Security dialect

Example:

```
<div sec:authorize="isAuthenticated()">
  Hello <span sec:authentication="name">someone</span>
</div>
```

Working with Thymeleaf´s Spring Security dialect

Example:

```
<span sec:authorize-url="/admin">
  <br/><a th:href="@{/admin}">Admin</a>
</span>
```

Spring MVC