Spring MVC

Agenda

- Overview of Spring MVC
- Getting started
- Writing controllers
- Resolving views
- Spring MVC's JSP tags
- RESTful HTTP Methods
- Validation

Spring-Loaded

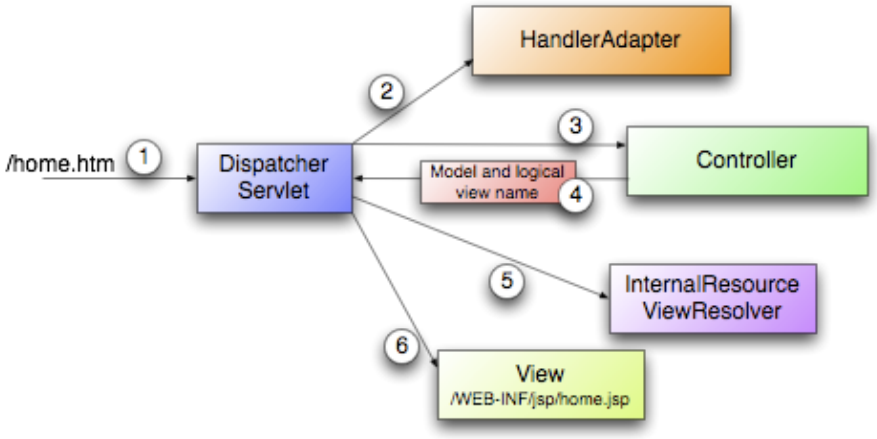
Spring MVC Overview

Key pieces of Spring MVC

- DispatcherServlet
  - ContextLoaderListener
- Handler adapters and mappings
- Controllers
  - ...and their dependencies
- View resolvers
- Views

Spring-Loaded

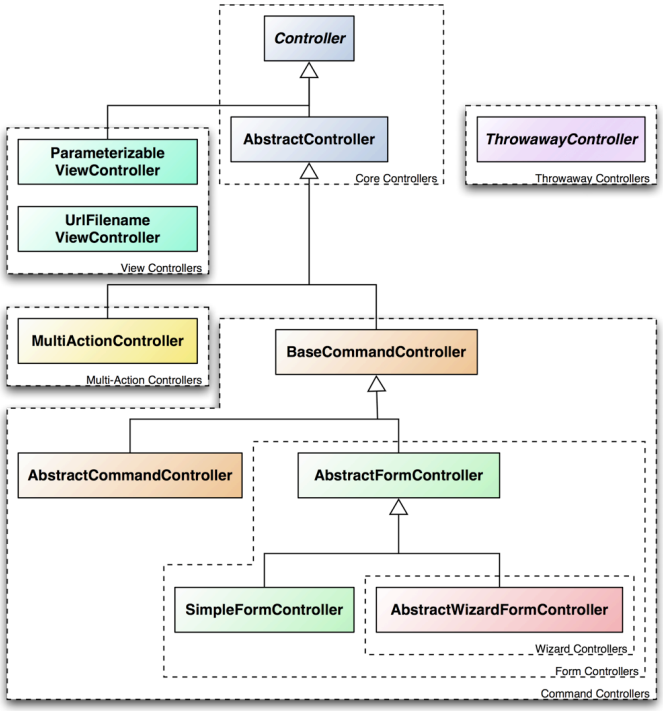
The life of a request



Spring MVC: A Look Back

- An original part of the Spring Framework
  - Including in Spring 1.0
- A rare “wheel reinvention” by Spring
  - “We can do better” than Struts and WebWork

Legacy controller hierarchy



Spring MVC 2.5

- Annotation-oriented controllers
- Controller hierarchy deprecated

Spring MVC 3.0

- Continued annotation-orientation
- REST support

Spring-Loaded

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

Getting Started with Spring MVC

DispatcherServlet

In web.xml

```
<servlet>
    <servlet-name>spitter</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spitter</servlet-name>
    <url-pattern>/app/*</url-pattern>
</servlet-mapping>
```

Loads Spring context from /WEB-INF/{servlet-name}-context.xml

SPRING-LOADED

ContextLoaderListener

In web.xml

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath:service-context.xml
        classpath:persistence-context.xml
        classpath:dataSource-context.xml
        classpath:setup-context.xml
    </param-value>
</context-param>

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

SPRING-LOADED

URL Rewriting

In web.xml

```
<filter>
    <filter-name>UrlRewriteFilter</filter-name>
    <filter-class>
        org.tuckey.web.filters.urlrewrite.UrlRewriteFilter
    </filter-class>
</filter>

<filter-mapping>
    <filter-name>UrlRewriteFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Optional, but helpful with RESTful URLs

SPRING-LOADED

URL Rewriting (2)

In WEB-INF/urlrewrite.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE urlrewrite
    PUBLIC "-//tuckey.org//DTD UrlRewrite 3.0//EN"
    "http://tuckey.org/res/dtds/urlrewrite3.0.dtd">
<urlrewrite default-match-type="wildcard">
  <rule>
    <from>/resources/**</from>
    <to>/resources/$1</to>
  </rule>
  <rule>
    <from>/**</from>
    <to>/app/$1</to>
  </rule>
  <outbound-rule>
    <from>/app/**</from>
    <to>/$1</to>
  </outbound-rule>
</urlrewrite>
```

SPRING-LOADED

Mapping URLs to controllers

- Several ways...including...
  - ControllerClassNameHandlerMapping
  - ControllerBeanNameHandlerMapping
  - SimpleUrlHandlerMapping
  - DefaultAnnotationHandlerMapping

I'm going to go with the annotation-based adapter

SimpleUrlHandlerMapping

In Spring configuration

```
<bean class=
     "org.springframework.web.servlet.handler.
                                        SimpleUrlHandlerMapping">
  <property name="mappings">
    <value>
       /home.htm=homeController
       /viewCart.htm=shoppingCartController
       /product.htm=productPageController
    </value>
  </property>
</bean>
```

Handling Annotated Controllers

In Spring configuration

```
<bean class=
   "org.springframework.web.servlet.mvc.annotation.
                        AnnotationMethodHandlerAdapter" />
```

...or...

```
<bean class=
   "org.springframework.web.servlet.mvc.annotation.
                        DefaultAnnotationHandlerMapping" />
```

Both are automatically registered by DispatcherServlet

SPRING-LOADED

Spring MVC namespace

New in Spring 3.0:

```
<mvc:annotated-controllers />
```

SPRING-LOADED

Writing Controllers

Controller design styles

- Use-case-oriented
  - One controller per item of functionality
  - Example: AddItemToCartController
- Resource-oriented
  - Encouraged by RESTful design
  - One controller per resource type
  - Example: ProductController

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

A bit on RESTful URLs

- URLs are resource locators
- Typically made up of nouns
- Rely on HTTP methods to decide what is to be done
  - GET, PUT, POST, DELETE

SPRING-LOADED

A simple controller

```
@Controller
public class HomeController {
  public static final int SPITTLE_COUNT = 25;

  @RequestMapping({"/","/home"})
  public String showHomePage(Map<String, Object> model) {
    model.put("spittles", spitterService.getRecentSpittles(SPITTLE_COUNT));
    return "home";
  }

  @Autowired
  SpitterService spitterService;
}
```

SPRING-LOADED

E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGINACTION.COM TWITTER: HABUMA

Testing controllers

```
public class HomeControllerTest {
  @Test
  public void shouldDisplayRecentSpittles() {
    SpitterService spitterService =
        createMock(SpitterService.class);
    spitterService.getRecentSpittles(SPITTLE_COUNT);
    List<Spittle> expectedSpittles =
        asList(new Spittle(), new Spittle(), new Spittle());
    expectLastCall().andReturn(expectedSpittles);
    replay(spitterService);

    HomeController homeController = new HomeController();
    homeController.spitterService = spitterService;

    HashMap<String, Object> model = new HashMap<String, Object>();
    homeController.showHomePage(model);
    assertSame(expectedSpittles, model.get("spittles"));
  }
}
```


A more interesting controller

```
@Controller
@RequestMapping("/spittle")
public class SpittleController {

  @RequestMapping(value="/{spittleId}", method=DELETE)
  public String deleteSpittle(@PathVariable String spittleId) {
    spitterService.deleteSpittle(spittleId);
    return "redirect:/home";
  }

  @RequestMapping(value="/form", method=GET)
  public void showSpittleForm(Map<String,Object> model) {
    model.put("spittle", new Spittle());
  }

  @RequestMapping(method=POST)
  public String addSpittle(Spittle spittle) {
    spitterService.addSpittle(spittle);
    return "redirect:/home";
  }

  @Autowired
  SpitterService spitterService;
}
```

Yet another controller

SPRING-LOADED

```
@Controller
@RequestMapping("/spitter")
public class SpitterController {
  @RequestMapping(method=POST)
  public String addSpitter(Spitter spitter) {
    spitterService.saveSpitter(spitter);
    return "redirect:/" + spitter.getUsername();
  }

  @RequestMapping(value="/form", method=GET)
  public String showSpitterForm(Map<String, Object> model) {
    model.put("spitter", new Spitter());
    return "spitterform";
  }

  @RequestMapping(value="/{username}", method=GET)
  public String spittlesForSpitter(@PathVariable String username,
          Map<String, Object> model) {
      Spitter spitter = spitterService.getSpitter(username);
      model.put("spitter", spitter);
      model.put("spittles", spitterService.getSpittlesForSpitter(spitter));
      return "spittles";
  }

  @Autowired
  SpitterService spitterService;

}
```


Resolving Views

VIEW RESOLVERS

- Several to choose from...
  - InternalResourceViewResolver
  - ContentNegotiatingViewResolver
  - BeanNameViewResolver
  - FreeMarkerViewResolver
  - JasperReportsViewResolver
  - ResourceBundleViewResolver
  - TilesViewResolver
  - VelocityViewResolver
  - XmlViewResolver
  - XsltViewResolver

SPRING-LOADED

INTERNALRESOURCEVIEWRESOLVER

```
<bean class=
       "org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/jsp/" />
  <property name="suffix" value=".jsp" />
</bean>
```

SPRING-LOADED

E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGINACTION.COM TWITTER: HABUMA

ContentNegotiatingViewResolver

- Delegates to other view resolvers
  - Selects based on representation requested
- Two strategies
  - By file extension:
    - http://www.habuma.com/spitter/habuma.htm
    - http://www.habuma.com/spitter/habuma.json
  - By HTTP `Accept` header in request:
    - Same URL, regardless of representation: http://.../spitter/habuma
    - Can't specify through browser

SPRING-LOADED

ContentNegotiatingViewResolver

```
<bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
  <property name="mediaTypes">
    <map>
      <entry key="json" value="application/json"/>
      <entry key="html" value="text/html"/>
    </map>
  </property>
  <property name="viewResolvers">
    <list>
      <bean class="org.springframework.web.servlet.view.BeanNameViewResolver" />
      <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
      </bean>
    </list>
  </property>
</bean>

<bean id="spittles"
      class="org.springframework.web.servlet.view.json.MappingJacksonJsonView" />
```

SPRING-LOADED

Spring MVC's JSP Tags

Two tag libraries

- Original
  - URI: http://www.springframework.org/tags
- Form-binding
  - URI: http://www.springframework.org/tags/form

Spring-Loaded

Using the tags

```
<%@ page contentType="text/html"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

<form:form modelAttribute="spittle" method="POST">
  <p><spring:message code="label.spittle" text="Enter spittle:"/></p>
  <form:textarea path="text" rows="5" cols="40" />
  <br/><br/>
  <input type="submit" value="Submit"/>
</form:form>
```


E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma

REST Methods

HTTP Methods

- HTTP supports 4 methods:
  - GET
  - POST
  - PUT
  - DELETE
- Browsers only support 2
  - GET
  - POST

SPRING-LOADED

Hidden HTTP Method Filter

```
<filter>
    <filter-name>httpMethodFilter</filter-name>
    <filter-class>
          org.springframework.web.filter.HiddenHttpMethodFilter
    </filter-class>
</filter>

<filter-mapping>
    <filter-name>httpMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

SPRING-LOADED

Handling REST methods

```
@RequestMapping(value="/{spittleId}", method=DELETE)
public String deleteSpittle(@PathVariable String spittleId) {
  return "redirect:/spittle/form";
}
```


E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGINACTION.COM TWITTER: HABUMA

REST in JSP forms

```
<spring:url value="/spittle" var="spittle_url" />
<spring:message code="button.delete.spittle"
                text="Delete It!"
                var="deleteLabel" />
<form:form method="delete" action="${spittle_url}${spittleId}">
      <input type="submit" value="${deleteLabel}"/>
</form:form>
```


E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGINACTION.COM TWITTER: HABUMA

What really happens

The JSP tags render:

```
<form id="command" action="/web/spittle/1234" method="post">
    <input type="hidden" name="_method" value="delete"/>
    <p class="submit"><input type="submit" value="Delete It!"/></p>
</form>
```

A POST request is sent

The filter translates the request to DELETE before DispatcherServlet gets it

E-MAIL: CRAIG@HABUMA.COM BLOG: HTTP://WWW.SPRINGINACTION.COM TWITTER: HABUMA

Validating Input

Enabling validation

```
<bean class=
   "org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
 <property name="webBindingInitializer">
     <bean class=
           "org.springframework.web.bind.support.ConfigurableWebBindingInitializer">
         <property name="validator" ref="validator" />
     </bean>
 </property>
</bean>

<bean id="validator"
      class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean"
      />
```


Annotating validation

In domain class:

```
@NotNull
@Size(min=10, max=140, message="Spittles must be between 10 and 140 characters")
public String getText() {
  return this.text;
}
```

In controller:

```
@RequestMapping(method=POST)
public String addSpittle(@Valid Spittle spittle) {
  spitterService.addSpittle(spittle);
  return "redirect:/spittle/form";
}
```

Summary

Summary

- Spring provides a flexible web MVC framework
- Annotation-based as of Spring 2.5
- Supports REST as of Spring 3.0
- Supports JSR-303 validation as of Spring 3.0

E-mail: craig@habuma.com Blog: http://www.springinaction.com Twitter: habuma