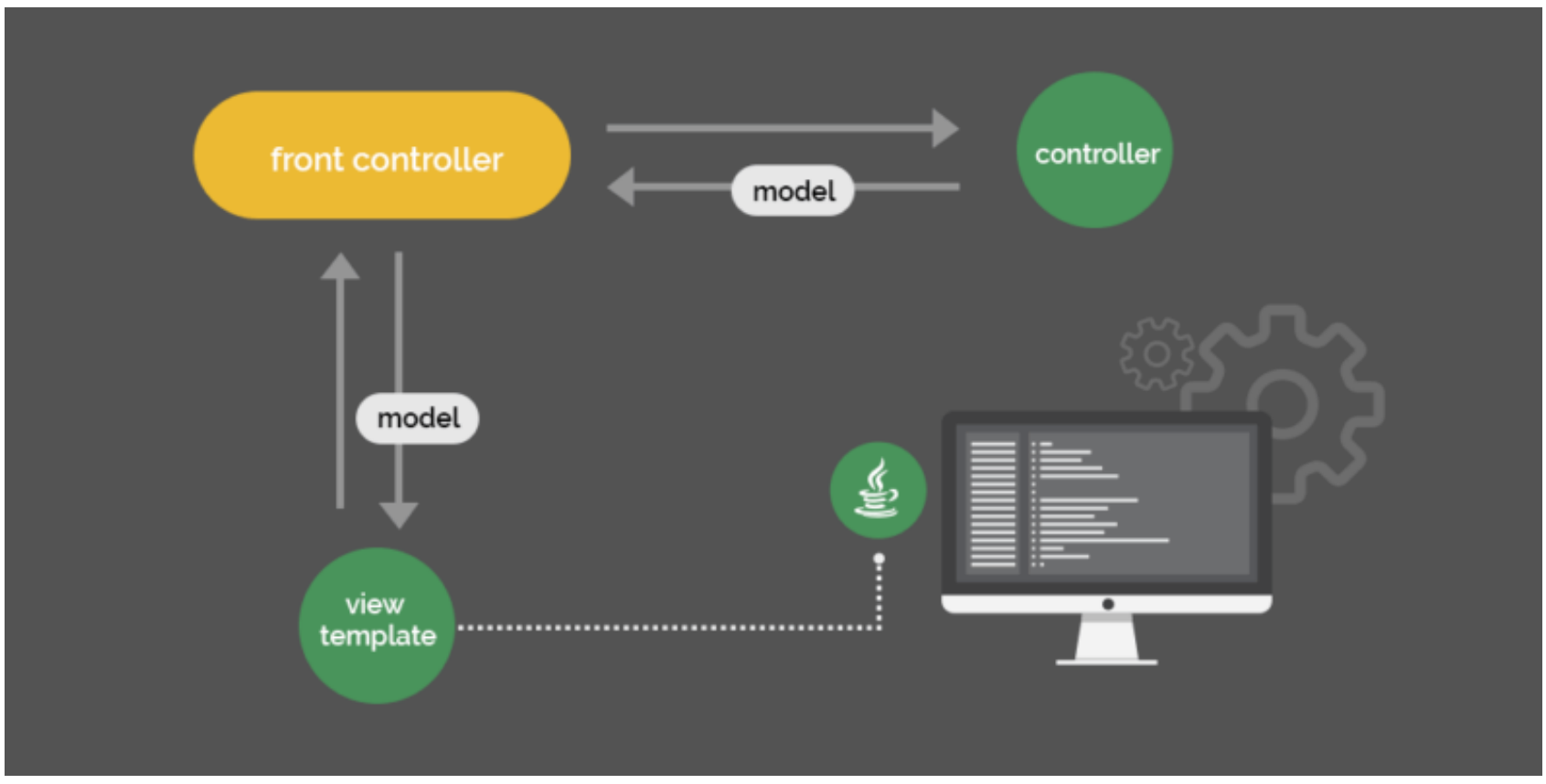
Spring MVC 到底是如何工作的?

2017-11-23　分类：JAVA开发、编程开发、首页精华0人评论

本文由码农网 – 小峰原创翻译，转载请看清文末的转载要求，欢迎参与我们的付费投稿计划!

这篇文章将深入探讨Spring框架的一部分——Spring Web MVC的强大功能及其内部工作原理。

这篇文章的源代码可以在GitHub上找到。



项目安装

在本文中，我们将使用最新、最好的Spring Framework 5。我们将重点介绍Spring的经典Web堆栈，该堆栈从框架的第一个版本中就崭露头角，并且现在依然是用Spring构建Web应用程序的主要方式。

对于初学者来说，为了安装测试项目，最好使用Spring Boot和一些初学者依赖项；还需要定义parent：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.M5</version>
```

```
    <relativePath/>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
</dependencies>
```

请注意，为了使用Spring 5，我们还需要使用Spring Boot 2.x。截止到撰写本文之时，这依然是里程碑发布版，可在Spring Milestone Repository中找到。让我们把这个存储库添加到你的Maven项目中：

```
<repositories>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

你可以在Maven Central上查看Spring Boot的当前版本。

示例项目

为了理解Spring Web MVC是如何工作的，我们将通过一个登录页面实现一个简单的应用程序。为了显示登录页面，我们需要为上下文根创建带有GET映射的@Controller注解类InternalController。

hello()方法是无参数的。它返回一个由Spring MVC解释为视图名称的String（在示例中是login.html模板）：

```
import org.springframework.web.bind.annotation.GetMapping;
@GetMapping("/")
public String hello() {
```

```
    return "login";
}
```

为了处理用户登录，需要创建另一个用登录数据处理POST请求的方法。然后根据结果将用户重定向到成功或失败的页面。

请注意，login()方法接收域对象作为参数并返回ModelAndView对象：

```
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.servlet.ModelAndView;
@PostMapping("/login")
public ModelAndView login(LoginData loginData) {
    if (LOGIN.equals(loginData.getLogin())
      && PASSWORD.equals(loginData.getPassword())) {
        return new ModelAndView("success",
          Collections.singletonMap("login", loginData.getLogin()));
    } else {
        return new ModelAndView("failure",
          Collections.singletonMap("login", loginData.getLogin()));
    }
}
```

ModelAndView是两个不同对象的持有者：

- Model——渲染页面数据的键值映射
- View——填充模型数据的页面模板

连接这些是为了方便，这样控制器方法可以一次返回它们。

要渲染HTML页面，使用Thymeleaf作为视图模板引擎，该引擎具有可靠和开箱即用的与Spring的集成。

Servlet作为Java Web应用程序的基础

那么，当在浏览器中输入http:// localhost:8080/时，按Enter键，然后请求到达Web服务器，实际发生了什么？你如何从这个请求中看到浏览器中的Web表单？

鉴于该项目是一个简单的Spring Boot应用程序，因此可以通过Spring5Application运行它。

Spring Boot默认使用Apache Tomcat。因此，运行应用程序时，你可能会在日志中看到以下信息：

```
2017-10-16 20:36:11.626  INFO 57414 --- [main]
  o.s.b.w.embedded.tomcat.TomcatWebServer  :
  Tomcat initialized with port(s): 8080 (http)
2017-10-16 20:36:11.634  INFO 57414 --- [main]
  o.apache.catalina.core.StandardService   :
  Starting service [Tomcat]
2017-10-16 20:36:11.635  INFO 57414 --- [main]
  org.apache.catalina.core.StandardEngine  :
  Starting Servlet Engine: Apache Tomcat/8.5.23
```

由于Tomcat是一个Servlet容器，因此发送给Tomcat Web服务器的每个HTTP请求自然都由Java servlet处理。所以Spring Web应用程序入口点是一个servlet，这并不奇怪。

简单地说，servlet就是任何Java Web应用程序的核心组件；它是低层次的，不会像MVC那样在特定的编程模式中诸多要求。

一个HTTP servlet只能接收一个HTTP请求，以某种方式处理，然后发回一个响应。

而且，从Servlet 3.0 API开始，你现在可以超越XML配置，并开始利用Java配置（只有很小的限制条件）。

DispatcherServlet作为Spring MVC的核心

作为一个Web应用程序的开发人员，我们真正想要做的是抽象出以下繁琐和模板化的任务，并专注于有用的业务逻辑：

- 将HTTP请求映射到某个处理方法
- 将HTTP请求数据和标题解析成数据传输对象（DTO）或域对象
- 模型 – 视图 – 控制器集成
- 从DTO、域对象等生成响应

Spring DispatcherServlet能够提供这些。它是Spring Web MVC框架的核心；此核心组件接收所有请求到应用程序。

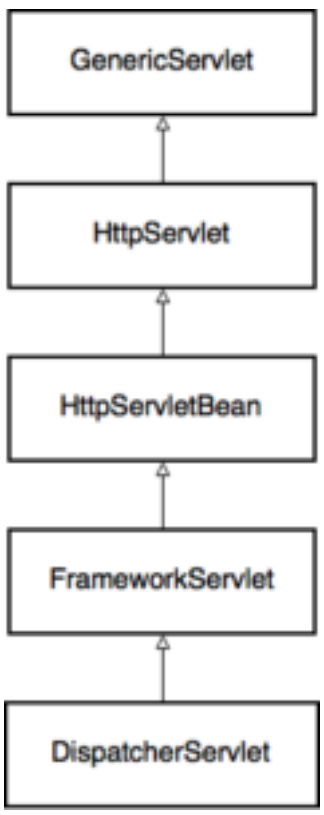
正如你所看到的，DispatcherServlet是非常可扩展的。例如，它允许你插入不同的现有或新的适配器进行大量的任务：

- 将请求映射到应该处理它的类或方法（HandlerMapping接口的实现）
- 使用特定模式处理请求，如常规servlet，更复杂的MVC工作流，或POJO bean中的方法（HandlerAdapter接口的实现）
- 按名称解析视图，允许你使用不同的模板引擎，XML，XSLT或任何其他视图技术（ViewResolver接口的实现）
- 通过使用默认的Apache Commons文件上传实现或编写你自己的MultipartResolver来解析多部分请求
- 使用任何LocaleResolver实现解决语言环境，包括cookie，会话，Accept HTTP头，或任何其他确定用户所期望的语言环境的方式

处理HTTP请求

首先，我们将简单的HTTP请求的处理追踪到在控制器层中的一个方法，然后返回到浏览器/客户端。

DispatcherServlet具有很长的继承层次结构；自上而下地逐个理解这些是有价值的。请求处理方法最让我们感兴趣。



理解HTTP请求，无论是在本地还是远程的标准开发中，都是理解MVC体系结构的关键部分。

GenericServlet

GenericServlet是Servlet规范的一部分，不直接关注HTTP。它定义了接收传入请求并产生响应的service()方法。

注意，ServletRequest和ServletResponse方法参数如何与HTTP协议无关：

```
public abstract void service(ServletRequest req, ServletResponse res)
  throws ServletException, IOException;
```

这是最终被任何请求调用到服务器上的方法，包括简单的GET请求。

HttpServlet

顾名思义，HttpServlet类就是规范中定义的基于HTTP的Servlet实现。

更实际的说，HttpServlet是一个抽象类，有一个service()方法实现，service()方法实现通过HTTP方法类型分割请求，大致如下所示：

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    String method = req.getMethod();
    if (method.equals(METHOD_GET)) {
        // ...
        doGet(req, resp);
    } else if (method.equals(METHOD_HEAD)) {
        // ...
        doHead(req, resp);
    } else if (method.equals(METHOD_POST)) {
        doPost(req, resp);
        // ...
    }
```

HttpServletBean

接下来，HttpServletBean是层次结构中第一个Spring-aware类。它使用从web.xml或WebApplicationInitializer接收到的servlet init-param值来注入bean的属性。

在请求应用程序的情况下，doGet()，doPost()等方法应特定的HTTP请求而调用。

FrameworkServlet

FrameworkServlet集成Servlet功能与Web应用程序上下文，实现了ApplicationContextAware接口。但它也能够自行创建Web应用程序上下文。

正如你已经看到的，HttpServletBean超类注入init-params为bean属性。所以，如果在servlet的contextClass init-param中提供了一个上下文类名，那么这个类的一个实例将被创建为应用程序上下文。否则，将使用默认的XmlWebApplicationContext类。

由于XML配置现在已经过时，Spring Boot默认使用AnnotationConfigWebApplicationContext配置DispatcherServlet。但是你可以轻松更改。

例如，如果你需要使用基于Groovy的应用程序上下文来配置Spring Web MVC应用程序，则可以在web.xml文件中使用以下DispatcherServlet配置：

```
dispatcherServlet
        org.springframework.web.servlet.DispatcherServlet
        contextClass
        org.springframework.web.context.support.GroovyWebApplicationContext
```

使用WebApplicationInitializer类，可以用更现代的基于Java的方式来完成相同的配置。

DispatcherServlet：统一请求处理

HttpServlet.service()实现，会根据HTTP动词的类型来路由请求，这在低级servlet的上下文中是非常有意义的。然而，在Spring MVC的抽象级别，方法类型只是可以用来映射请求到其处理程序的参数之一。

因此，FrameworkServlet类的另一个主要功能是将处理逻辑重新加入到单个processRequest()方法中，processRequest()方法反过来又调用doService()方法：

```
@Override
protected final void doGet(HttpServletRequest request,
  HttpServletResponse response) throws ServletException, IOException {
    processRequest(request, response);
}
```

```
@Override
protected final void doPost(HttpServletRequest request,
  HttpServletResponse response) throws ServletException, IOException {
    processRequest(request, response);
}
// …
```

DispatcherServlet：丰富请求

最后，DispatcherServlet实现doService()方法。在这里，它增加了一些可能会派上用场的有用对象到请求：Web应用程序上下文，区域解析器，主题解析器，主题源等：

```
request.setAttribute(WEB_APPLICATION_CONTEXT_ATTRIBUTE,
  getWebApplicationContext());
request.setAttribute(LOCALE_RESOLVER_ATTRIBUTE, this.localeResolver);
request.setAttribute(THEME_RESOLVER_ATTRIBUTE, this.themeResolver);
request.setAttribute(THEME_SOURCE_ATTRIBUTE, getThemeSource());
```

另外，doService()方法准备输入和输出Flash映射。Flash映射基本上是一种模式，该模式将参数从一个请求传递到另一个紧跟的请求。这在重定向期间可能非常有用（例如在重定向之后向用户显示一次性信息消息）：

```
FlashMap inputFlashMap = this.flashMapManager
  .retrieveAndUpdate(request, response);
if (inputFlashMap != null) {
    request.setAttribute(INPUT_FLASH_MAP_ATTRIBUTE,
      Collections.unmodifiableMap(inputFlashMap));
}
request.setAttribute(OUTPUT_FLASH_MAP_ATTRIBUTE, new FlashMap());
```

然后，doService()方法调用负责请求调度的doDispatch()方法。

DispatcherServlet：调度请求

dispatch()方法的主要目的是为请求找到合适的处理程序，并为其提供请求/响应参数。处理程序基本上是任何类型的object，不限于特定的接口。这也意味着Spring需要为此处理程序找到适配器，该处理程序知道如何与处理程序“交谈”。

为了找到匹配请求的处理程序，Spring检查HandlerMapping接口的注册实现。有很多不同的实现可以满足你的需求。

SimpleUrlHandlerMapping允许通过URL将请求映射到某个处理bean。例如，可以通过使用java.util.Properties实例注入其mappings属性来配置，就像这样：

```
/welcome.html=ticketController
/show.html=ticketController
```

可能处理程序映射最广泛使用的类是RequestMappingHandlerMapping，它将请求映射到@Controller类的@ RequestMapping注释方法。这正是使用控制器的hello()和login()方法连接调度程序的映射。

请注意，Spring-aware方法使用@GetMapping和@PostMapping进行注释。这些注释依次用@RequestMapping元注释标记。

dispatch()方法还负责其他一些HTTP特定任务：

- 在资源未被修改的情况下，GET请求的短路处理
- 针对相应的请求应用多部分解析器
- 如果处理程序选择异步处理该请求，则会短路处理该请求

处理请求

现在Spring已经确定了请求的处理程序和处理程序的适配器，是时候来处理请求了。下面是HandlerAdapter.handle()方法的签名。请注意，处理程序可以选择如何处理请求：

- 自主地编写数据到响应对象，并返回null
- 返回由DispatcherServlet呈现的ModelAndView对象

```
@Nullable
ModelAndView handle(HttpServletRequest request,
                    HttpServletResponse response,
                    Object handler) throws Exception;
```

有几种提供的处理程序类型。以下是SimpleControllerHandlerAdapter如何处

理Spring MVC控制器实例（不要将其与@ Controller注释POJO混淆）。

注意控制器处理程序如何返回ModelAndView对象，并且不自行呈现视图：

```
public ModelAndView handle(HttpServletRequest request,
  HttpServletResponse response, Object handler) throws Exception {
    return ((Controller) handler).handleRequest(request, response);
}
```

第二个是SimpleServletHandlerAdapter，它将常规的Servlet作为请求处理器。

Servlet不知道任何有关ModelAndView的内容，只是简单地自行处理请求，并将结果呈现给响应对象。所以这个适配器只是返回null而不是ModelAndView：

```
public ModelAndView handle(HttpServletRequest request,
  HttpServletResponse response, Object handler) throws Exception {
    ((Servlet) handler).service(request, response);
    return null;
}
```

我们碰到的情况是，控制器是有若干@RequestMapping注释的POJO，所以任何处理程序基本上是包装在HandlerMethod实例中的这个类的方法。为了适应这个处理器类型，Spring使用RequestMappingHandlerAdapter类。

处理参数和返回处理程序方法的值

注意，控制器方法通常不会使用HttpServletRequest和HttpServletResponse，而是接收和返回许多不同类型的数据，例如域对象，路径参数等。

此外，要注意，我们不需要从控制器方法返回ModelAndView实例。可能会返回视图名称，或ResponseEntity，或将被转换为JSON响应等的POJO。

RequestMappingHandlerAdapter确保方法的参数从HttpServletRequest中解析出来。另外，它从方法的返回值中创建ModelAndView对象。

在RequestMappingHandlerAdapter中有一段重要的代码，可确保所有这些转换魔法的发生：

```
ServletInvocableHandlerMethod invocableMethod
  = createInvocableHandlerMethod(handlerMethod);
if (this.argumentResolvers != null) {
    invocableMethod.setHandlerMethodArgumentResolvers(
      this.argumentResolvers);
}
if (this.returnValueHandlers != null) {
    invocableMethod.setHandlerMethodReturnValueHandlers(
      this.returnValueHandlers);
}
```

argumentResolvers对象是不同的HandlerMethodArgumentResolver实例的组合。

有超过30个不同的参数解析器实现。它们允许从请求中提取任何类型的信息，并将其作为方法参数提供。这包括URL路径变量，请求主体参数，请求标头，cookies，会话数据等。

returnValueHandlers对象是HandlerMethodReturnValueHandler对象的组合。还有很多不同的值处理程序可以处理方法的结果来创建适配器所期望的ModelAndViewobject。

例如，当你从hello()方法返回字符串时，ViewNameMethodReturnValueHandler处理这个值。但是，当你从login()方法返回一个准备好的ModelAndView时，Spring会使用ModelAndViewMethodReturnValueHandler。

渲染视图

到目前为止，Spring已经处理了HTTP请求并接收了ModelAndView对象，所以它必须呈现用户将在浏览器中看到的HTML页面。它基于模型和封装在ModelAndView对象中的选定视图来完成。

另外请注意，我们可以呈现JSON对象，或XML，或任何可通过HTTP协议传输的其他数据格式。我们将在即将到来的REST-focused部分接触更多。

让我们回到DispatcherServlet。render()方法首先使用提供的LocaleResolver实例设置响应语言环境。假设现代浏览器正确设置了Accept头，并且默认使用AcceptHeaderLocaleResolver。

在渲染过程中，ModelAndView对象可能已经包含对所选视图的引用，或者只是一个视图名称，或者如果控制器依赖于默认视图，则什么都没有。

由于hello()和login()方法两者都指定所需的视图为String名称，因此必须用该名称查找。所以，这是viewResolvers列表开始起作用的地方：

```
for (ViewResolver viewResolver : this.viewResolvers) {
    View view = viewResolver.resolveViewName(viewName, locale);
    if (view != null) {
        return view;
    }
}
```

这是一个ViewResolver实例列表，包括由thymeleaf-spring5集成库提供的ThymeleafViewResolver。该解析器知道在哪里搜索视图，并提供相应的视图实例。

在调用视图的render()方法后，Spring最终通过发送HTML页面到用户的浏览器来完成请求处理。

REST支持

除了典型的MVC场景之外，我们还可以使用框架来创建REST Web服务。

简而言之，我们可以接受Resource作为输入，指定POJO作为方法参数，并使用@RequestBody对其进行注释。也可以使用@ResponseBody注释方法本身，以指定其结果必须直接转换为HTTP响应：

```
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.ResponseBody;
@ResponseBody
@PostMapping("/message")
public MyOutputResource sendMessage(
  @RequestBody MyInputResource inputResource) {
    return new MyOutputResource("Received: "
      + inputResource.getRequestMessage());
}
```

归功于Spring MVC的可扩展性，这也是可行的。

为了将内部DTO编组为REST表示，框架使用HttpMessageConverter基础结构。例如，其中一个实现是MappingJackson2HttpMessageConverter，它可以使用Jackson库将模型对象转换为JSON或从JSON转换。

为了进一步简化REST API的创建，Spring引入了@RestController注解。默认情况下，这很方便地假定了@ResponseBody语义，并避免在每个REST控制器上的明确设置：

```
import org.springframework.web.bind.annotation.RestController;
@RestController
public class RestfulWebServiceController {
    @GetMapping("/message")
    public MyOutputResource getMessage() {
        return new MyOutputResource("Hello!");
    }
}
```

结论

在这篇文章中，我们详细了介绍在Spring MVC框架中请求的处理过程。了解框架的不同扩展是如何协同工作来提供所有魔法的，可以让你能够事倍功半地处理HTTP协议难题。

来说两句吧...