**纯手写SpringBoot框架**

# 什么是SpringBoot

在您第1次接触和学习Spring框架的时候，是否因为其繁杂的配置而退却了？在你第n次使用Spring框架的时候，是否觉得一堆反复黏贴的配置有一些厌烦？那么您就不妨来试试使用Spring Boot来让你更易上手，更简单快捷地构建Spring应用！

Spring Boot让我们的Spring应用变的更轻量化。比如：你可以仅仅依靠一个Java类来运行一个Spring引用。你也可以打包你的应用为jar并通过使用java -jar来运行你的Spring Web应用。

Spring Boot的主要优点：

为所有Spring开发者更快的入门

开箱即用，提供各种默认配置来简化项目配置

内嵌式容器简化Web项目

没有冗余代码生成和XML配置的要求

本章主要目标完成Spring Boot基础项目的构建，并且实现一个简单的Http请求处理，通过这个例子对Spring Boot有一个初步的了解，并体验其结构简单、开发快速的特性。

SpringBoot 是一个快速开发的框架,能够快速的整合第三方框架，简化XML配置，全部采用注解形式，内置Tomcat容器,帮助开发者能够实现快速开发，SpringBoot的Web组件 默认集成的是SpringMVC框架。

SpringMVC是控制层。

## SpringBoot核心原理

基于SpringMVC无配置文件（纯Java）完全注解化+内置tomcat-embed-core实现SpringBoot框架，Main函数启动。

SpringBoot核心快速整合第三方框架原理:Maven继承依赖关系

SpringBoot内嵌入tomcat-embed-core

SpringBoot采用SpringMVC注解版本实现无配置效果

注意:本节课说的不是SpringBoot流程还是SpringBoot核心原理

# 内置Tomcat容器

Java提供内置Tomcat容器框架，使用Java语言操作Tomcat容器。

案例： 使用Java语言创建一个Tomcat容器

## Maven依赖

|  |
| --- |
| <dependencies>  <!--Java语言操作tomcat -->  <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-core</artifactId>  <version>8.5.16</version>  </dependency>  <!-- spring-web -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>5.0.4.RELEASE</version>  <scope>compile</scope>  </dependency>  <!-- spring-mvc -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>5.0.4.RELEASE</version>  <scope>compile</scope>  </dependency>  <!-- tomcat对jsp支持 -->  <dependency>  <groupId>org.apache.tomcat</groupId>  <artifactId>tomcat-jasper</artifactId>  <version>8.5.16</version>  </dependency>  </dependencies> |

## 创建Servet类

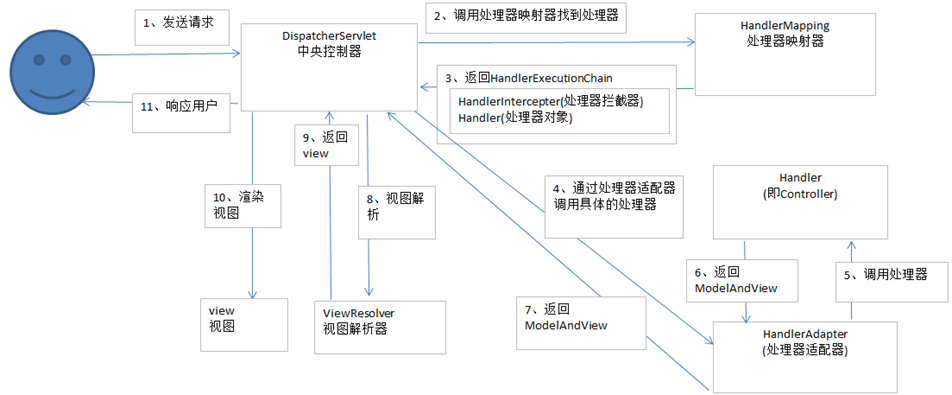
|  |
| --- |
| **public** **class** IndexServet **extends** HttpServlet {  @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  doPost(req, resp);  }  @Override  **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  resp.getWriter().print("springboot2.0");  }  } |

## 创建Tomcat运行

|  |
| --- |
| // 端口号  **private** **static** **int** *PORT* = 8080;  // 项目名称  **private** **static** String *CONTEXTPATH* = "/itmayiedu";  **public** **static** **void** main(String[] args) **throws** LifecycleException {  // 创建Tomcat服务器  Tomcat tomcatServer = **new** Tomcat();  // 设置Tomcat端口号  tomcatServer.setPort(*PORT*);  tomcatServer.getHost().setAutoDeploy(**false**);  // 创建Context上下文  StandardContext standardContext = **new** StandardContext();  standardContext.setPath(*CONTEXTPATH*);  standardContext.addLifecycleListener(**new** FixContextListener());  // tomcat容器添加standardContext  tomcatServer.getHost().addChild(standardContext);  // 创建servlet  tomcatServer.addServlet(*CONTEXTPATH*, "IndexServet", **new** IndexServet());  // 添加servleturl映射  standardContext.addServletMappingDecoded("/index", "IndexServet");  tomcatServer.start();  System.***out***.println("tomcat启动...");  tomcatServer.getServer().await();  } |

# SpringMVC（无配置启动）

## SpringMVC原理回顾



1. 用户发送请求至前端控制器DispatcherServlet

⑵ DispatcherServlet收到请求调用HandlerMapping处理器映射器。

⑶ 处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。

⑷ DispatcherServlet通过HandlerAdapter处理器适配器调用处理器

⑸ 执行处理器(Controller，也叫后端控制器)。

⑹ Controller执行完成返回ModelAndView

⑺ HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet

⑻ DispatcherServlet将ModelAndView传给ViewReslover视图解析器

⑼ ViewReslover解析后返回具体View

⑽ DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）。

⑾ DispatcherServlet响应用户。

## SpringMVC注解启动方式

DispatcherServlet是Spring MVC的核心，每当应用接受一个HTTP请求，由DispatcherServlet负责将请求分发给应用的其他组件。

在旧版本中，DispatcherServlet之类的servlet一般在web.xml文件中配置，该文件一般会打包进最后的war包种；但是Spring 3引入了注解，我们这节课讲解，如何基于注解配置Spring MVC。

## Maven依赖

|  |
| --- |
| <dependencies>  <!--Java语言操作tomcat -->  <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-core</artifactId>  <version>8.5.16</version>  </dependency>  <!-- spring-web -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>5.0.4.RELEASE</version>  <scope>compile</scope>  </dependency>  <!-- spring-mvc -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>5.0.4.RELEASE</version>  <scope>compile</scope>  </dependency>  <!-- tomcat对jsp支持 -->  <dependency>  <groupId>org.apache.tomcat</groupId>  <artifactId>tomcat-jasper</artifactId>  <version>8.5.16</version>  </dependency>  </dependencies> |

## 加载SpringMVCDispatcherServlet

*AbstractAnnotationConfigDispatcherServletInitializer*这个类负责配置*DispatcherServlet*、初始化Spring MVC容器和Spring容器。*getRootConfigClasses()*方法用于获取Spring应用容器的配置文件，这里我们给定预先定义的*RootConfig.class*；*getServletConfigClasses*负责获取Spring MVC应用容器，这里传入预先定义好的*WebConfig.class*；*getServletMappings()*方法负责指定需要由*DispatcherServlet*映射的路径，这里给定的是"/"，意思是由*DispatcherServlet*处理所有向该应用发起的请求。

|  |
| --- |
| /\*\*  \* 加载SpringMVCDispatcherServlet  \*  \* **@作者说明** 每特教育-余胜军 <br>  \* **@联系方式** qq644064779|www.itmayieducom-蚂蚁课堂<br>  \*/  **public** **class** SpittrWebAppInitializer **extends** AbstractAnnotationConfigDispatcherServletInitializer {  // 加载根容器  **protected** Class<?>[] getRootConfigClasses() {  // **TODO** Auto-generated method stub  **return** **new** Class[] { RootConfig.**class** };  }  // 加载SpringMVC容器  **protected** Class<?>[] getServletConfigClasses() {  **return** **new** Class[] { WebConfig.**class** };  }  // SpringMVCDispatcherServlet 拦截的请求 /  **protected** String[] getServletMappings() {  **return** **new** String[] { "/" };  }  } |

## 加载SpringMVC容器

*正如可以通过多种方式配置*DispatcherServlet*一样，也可以通过多种方式启动Spring MVC特性。原来我们一般在xml文件中使用*<mvc:annotation-driven>*元素启动注解驱动的Spring MVC特性。*

|  |
| --- |
| /\*\*  \* SpringMVC容器<br>  \*  \* **@EnableWebMvc** 开启SpringMVC功能<br>  \* **@Configuration** 配置<br>  \* **@作者说明** 每特教育-余胜军 <br>  \* **@联系方式** qq644064779|www.itmayieducom-蚂蚁课堂<br>  \*/  @Configuration  @EnableWebMvc  @ComponentScan("com.itmayiedu.controller")  **public** **class** WebConfig **extends** ~~WebMvcConfigurerAdapter~~ {  } |

## RootConfig容器

|  |
| --- |
| /\*\*  \* 根容器 <br>  \*  \* **@作者说明** 每特教育-余胜军 <br>  \* **@联系方式** qq644064779|www.itmayieducom-蚂蚁课堂<br>  \*/  @Configuration  @ComponentScan(basePackages = "com.itmayiedu")  **public** **class** RootConfig {  } |

## 运行代码

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** ServletException, LifecycleException {  *start*();  }  **public** **static** **void** start() **throws** ServletException, LifecycleException {  // 创建Tomcat容器  Tomcat tomcatServer = **new** Tomcat();  // 端口号设置  tomcatServer.setPort(9090);  // 读取项目路径  StandardContext ctx = (StandardContext) tomcatServer.addWebapp("/", **new** File("src/main").getAbsolutePath());  // 禁止重新载入  ctx.setReloadable(**false**);  // class文件读取地址  File additionWebInfClasses = **new** File("target/classes");  // 创建WebRoot  WebResourceRoot resources = **new** StandardRoot(ctx);  // tomcat内部读取Class执行  resources.addPreResources(  **new** DirResourceSet(resources, "/WEB-INF/classes", additionWebInfClasses.getAbsolutePath(), "/"));  tomcatServer.start();  // 异步等待请求执行  tomcatServer.getServer().await();  } |

## 项目添加业务逻辑层

### 创建Service层

|  |
| --- |
| @Service  **public** **class** UserService {  **public** String index() {  **return** "这是我的第一个UserService,哈哈哈";  }  } |

### 修改Controller

|  |
| --- |
| @Autowired  **private** UserService userService;  @RequestMapping(value = "/index", produces = "text/html;charset=UTF-8")  **public** String index() {  **return** userService.index();  } |

## 对JSP页面支持

### 修改WebConfig创建视图解析器

|  |
| --- |
| // 创建SpringMVC视图解析器  @Bean  **public** ViewResolver viewResolver() {  InternalResourceViewResolver viewResolver = **new** InternalResourceViewResolver();  viewResolver.setPrefix("/WEB-INF/views/");  viewResolver.setSuffix(".jsp");  //可以在JSP页面中通过${}访问beans  viewResolver.setExposeContextBeansAsAttributes(**true**);  **return** viewResolver;  } |

### 新增控制器层

|  |
| --- |
| @Controller  **public** **class** IndexController {  @RequestMapping("/pageIndex")  **public** String pageIndex() {  **return** "pageIndex";  }  } |