2小时入门
Spring Boot

课程介绍

by万夜

目录

- 课程目标
- 授课方式
- 课程内容
- 关于评论

课程目标

- 2小时迅速上手
- 可以独立开发、搭建完整的HTTP应用

授课方式

- 现场撸码！！！
- MAC + IntelliJ IDEA + Maven

课程内容

- 包含
  - **整合模板（FreeMarker/Jsp)**
  - **配置Servlet,Filter,Listener,Interceptor**
  - **整合日志组件、静态资源处理及启动加载**
  - **数据库（JDBC/Mybatis/事务原理）**
  - **部署到Tomcat**
- 不包含
  - **整合缓存redis等**
  - **监控、定时任务quarz等**
  - **微服务相关（服务发现、服务治理）**

关于评论

- 深度不够
- 音质不好

感谢观看

- Netty快速入门
- 案例：Socks Proxy

2小时入门
Spring Boot

Hello World

by万夜

目录

- 创建工程
- pom文件配置
- 编写rest接口 - hello world
- 注解含义
- 整合jsp和freemarker
- 总结

准备

- 环境
  - Jdk8
  - Ide intelliJ IDEA
  - Maven 3

Pom依赖

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.1.RELEASE</version>
</parent>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

创建启动类

```java
@SpringBootApplication
public class Start {

    public static void main(String[] args) {
        SpringApplication.run(Start.class, args);
    }
}
```

创建Controller

```
@Controller
public class HelloController {

    @RequestMapping("/hello")
    @ResponseBody
    public Map<String, String> hello(){
        Map<String, String> hello = new HashMap<String, String>();
        hello.put("data", "hello 小红");
        hello.put("status", "SUCCESS");
        return hello;
    }
}
```

注解含义

@SpringBootApplication

@Configuration + @ComponentScan + @EnableAutoConfiguration

@RequestMapping(“/hello")

@RestController

@Controller

@ResponseBody

关于Controller

- 模版引擎包括：FreeMarker 、Groovy 、Thymeleaf （Spring 官网使用这个） 、Velocity 、JSP

- 接收参数可以使用@RequestBody、@RequestParam、@ModelAttribute、JSONObject、HttpEntity 等

通过JSP模板引擎渲染

- Pom增加

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>
```

- 添加文件
  - 增加目录“src/main/webapp/WEB-INF/jsp/”，将jsp文件放入这个目录中
  - 在目录“resources”中，增加application.properties配置文件

```
# 页面默认前缀目录
spring.mvc.view.prefix=/WEB-INF/jsp/
# 响应页面默认后缀
spring.mvc.view.suffix=.jsp
```

- 启动方式

必须用sping-boot:run启动

通过FreeMarker模板引擎渲染

- Pom

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-freemarker</artifactId>
</dependency>
```

- 添加文件
  - 在resources下创建templates文件夹，将.flt文件放入
  - application.properties文件中无需配置（删除刚刚jsp配置内容）
- 启动方式
  - 主函数启动或者spring-boot:run

总结

- pom依赖配置（web,jsp,freemarker)
- 注解
    - @SpringBootApplication
    - @Controller @ResponseBody @RestController
    - @RequestMapping
- 整合jsp/freemarker
- 启动方法（main/spring-boot:run)

Spring Boot

- Servlet / Filter / Listener / 拦截器

by万夜

目录

- 注册servlet的两种方式
- 实现servlet/filter/listener/拦截器
- 注解含义
- 总结

注册Servlet方式

- **通过代码注册**
  - 代码注册通过ServletRegistrationBean、FilterRegistrationBean 和 ServletListenerRegistrationBean 获得控制。
- **注解自动注册**
  - 在 SpringBootApplication 上使用@ServletComponentScan 注解后，Servlet、Filter、Listener 可以直接通过 @WebServlet、@WebFilter、@WebListener 注解自动注册，无需其他代码。

通过代码注册Servlet

- 1. 创建servlet

```
public class HelloServlet extends HttpServlet{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        System.out.println(">>doGet<<");
        doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        System.out.println(">>doPost<<");
        resp.setContentType("text/html;charset=utf-8");
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<head><title>Hello 小红</title></head>");
        out.println("<body>");
        out.println("Hello 小红");
        out.println("</body>");
        out.println("</html>");
    }
}
```

- 2. 注册servlet到spring

```
@Bean
public ServletRegistrationBean servletRegistrationBean() {
    return new ServletRegistrationBean(new HelloServlet(),"/xiaohong");
}
```

通过注解注册Servlet

```
1. 增加注解，开启servlet扫描
@SpringBootApplication
@ServletComponentScan //
public class Start {
    public static void main(String[] args) {
        SpringApplication.run(Start.class, args);
    }
}
```

```
2. 增加注解，标识该类是servlet，并声明urlPath
@WebServlet("/xiaohong1") //
public class HelloServlet1 extends HttpServlet{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        System.out.println(">>doGet<<");
        doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        System.out.println(">>doPost<<");
        resp.setContentType("text/html;charset=utf-8");
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<head><title>Hello 小红</title></head>");
        out.println("<body>");
        out.println("Hello 小红");
        out.println("</body>");
        out.println("</html>");
    }
}
```

通过注解创建Filter

实现功能：过滤所有请求，判断请求参数中是否包含”key”，同时”key”==“xiaohong”，如不包含，则认为是非法请求，返回“param error”，如合法则继续访问。

- 1. 增加注解@ServletComponentScan，开启servlet扫描
- 2. 增加注解@WebFilter，标识该类是Filter

```
@WebFilter
public class HelloFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println(">>filter init<<");
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException,
ServletException {
        System.out.println(">>filter done<<");
        String key = servletRequest.getParameter("key");
        if (null != key && "xiaohong".equals(key)) {
            System.out.println(">>filter match<<");
            filterChain.doFilter(servletRequest, servletResponse);
        } else {
            System.out.println("filter param error");
            PrintWriter out = servletResponse.getWriter();
            out.print("param error");
            out.close();
        }
    }

    @Override
    public void destroy() {
        System.out.println(">>filter destroy<<");
    }
}
```

通过注解创建Listener

实现功能：系统启动时，加载请求参数配置”key”==”xiaoming”，并在过滤器中进行合法参数验证。

1. 增加注解@ServletComponentScan，开启servlet扫描
2. 增加注解@WebListener，标识该类是Listener

```
@WebListener
public class HelloServletListener implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent servletContextEvent) {
        // 配置key==xiaoming
        servletContextEvent.getServletContext().setAttribute("key","xiaoming");
        System.out.println(">>context listener init<<");
    }

    @Override
    public void contextDestroyed(ServletContextEvent servletContextEvent) {
        System.out.println(">>context listener destroyed<<");
    }
}
```

```
修改HelloFilter，动态加载key
String _key = (String) servletRequest.getServletContext().getAttribute("key");
if (null != key && _key.equals(key)) {
```

创建http拦截器

- 1. 创建拦截器类并实现 HandlerInterceptor接口

```
public class HelloInterceptor implements HandlerInterceptor {
   @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        System.out.println(">>interceptor preHandle<<");
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView
modelAndView) throws Exception {
        System.out.println(">>interceptor postHandle<<");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)
throws Exception {
        System.out.println(">>interceptor afterCompletion<<");
    }
}
```

- 2. 创建一个Java类继承WebMvcConfigurerAdapter，并重写 addInterceptors 方法，@Configuration

- 3. 实例化我们自定义的拦截器，然后将对像手动添加到拦截器链中（在addInterceptors方法中添加）

```
@Configuration
public class HelloConfig extends WebMvcConfigurerAdapter {
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new HelloInterceptor()).addPathPatterns("/**");
    }
}
```

注解含义

```
@Bean // @Bean明确地指示了产生一个bean的方法，并且交给Spring容器管理

@ServletComponentScan // 当使用@ServletComponentScan扫描Servlet组件时，Servlet、过滤器和监听器可以是通过@WebServlet、@WebFilter和@WebListener自动注册

@WebServlet(“/hello")

@WebFilter

@WebListener
```

总结

- 注册Servlet/Filter/Listener的两种方式
- 注解
  - @Bean
  - @ServletComponenScan
  - @WebServlet/@WebFilter/@WebListener
- 通过注解注册servlet/filter/listener
- 两个功能：
  - 过滤器验证所有请求参数
  - 监听器，初始化验证规则
- 创建拦截器

Spring Boot

- 静态资源处理、启动加载、日志处理

by万夜

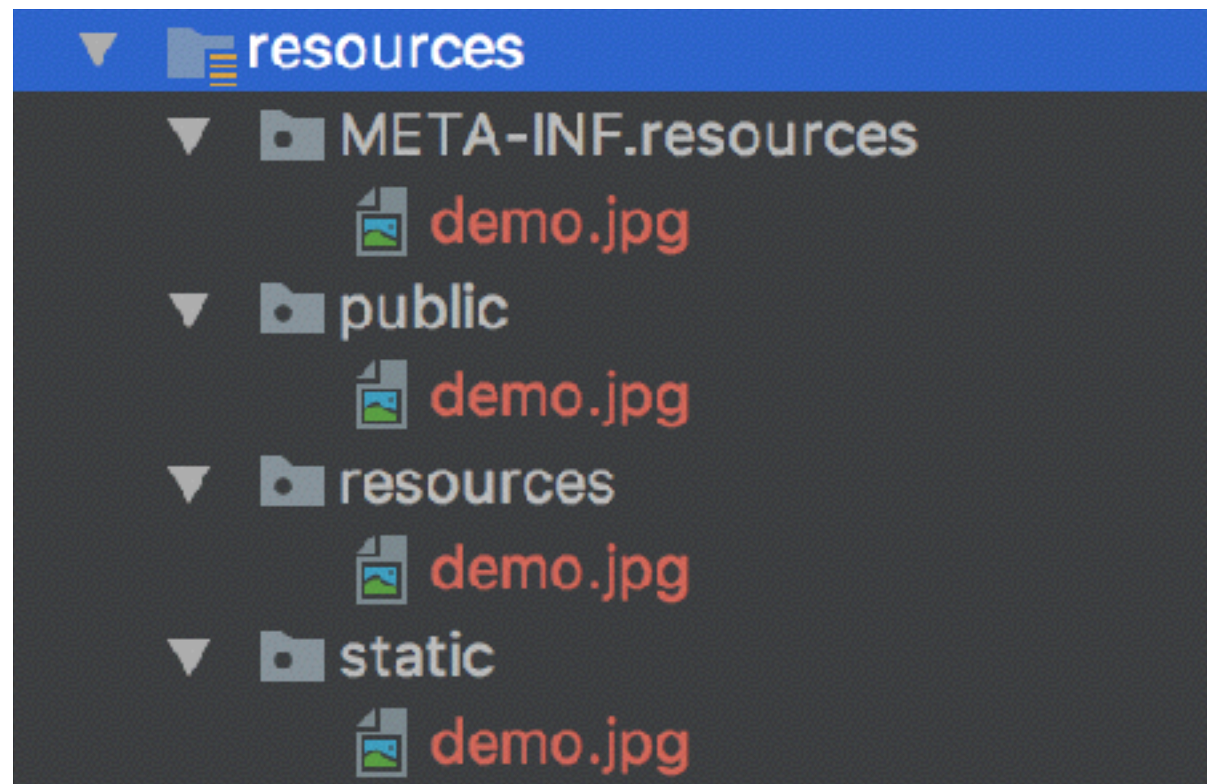
目录

- 静态资源处理
  - 默认资源映射
  - 自定义资源映射
- 启动加载数据
  - CommandLineRunner
- 日志处理
  - logback
- 总结

静态资源处理

—默认资源映射

- 默认配置的 /** 映射到 /static （或/public、/resources、/META-INF/resources）



- 优先级顺序为：META-INF/resources > resources > static > public

静态资源处理

—自定义资源映射

- 增加 /2017imgs/* 映射到 classpath:/2017imgs/* 为例的代码处理为： 实现类继承 WebMvcConfigurerAdapter 并重写方法 addResourceHandlers

- 在resources目录下，增加2017imgs目录

```
@Configuration
public class Config extends WebMvcConfigurerAdapter{

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/2017imgs/**").addResourceLocations("classpath:/2017imgs/");
    }
}
```

启动加载数据

- 创建类，并实现接口CommandLineRunner
- 当有多个启动加载的类，可以通过@Order来指定加载顺序,按value值从小到大顺序来执行
- args 等于 main方法的args

```
@Component
public class CacheInit implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
        System.out.println(">>cache init<<");
    }
}
```

日志处理

—logback

- 在resources下增加logback.xml配置
  - 控制台输出ConsoleAppender
  - 文件输出RollingFileAppender

日志处理

—logback.xml

```xml
<configuration debug="false" scan="true" scanPeriod="30 seconds">

    <property name="FILE_PATTERN" value="%d [%t] %5p %c - %m%n"/>

    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d [%t] %5p %logger - %m%n</pattern>
        </encoder>
    </appender>

    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <File>hello.log</File>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- daily rollover -->
            <FileNamePattern>hello.%d{yyyy-MM-dd}.log</FileNamePattern>
            <!-- keep 30 days' worth of history -->
            <maxHistory>30</maxHistory>
        </rollingPolicy>
        <encoder>
            <Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{35} - %msg %n</Pattern>
        </encoder>
    </appender>


    <root level="INFO">
        <appender-ref ref="STDOUT"/>
        <appender-ref ref="FILE"/>
    </root>
</configuration>
```

```java
private static final Logger logger = LoggerFactory.getLogger(CacheInit.class);
```

总结

- 静态资源处理
  - 默认资源映射(/**　- >　/resources/static,优先级)
  - 自定义资源映射　(继承WebMvcConfigurerAdapter 并重写方法addResourceHandlers)
- 启动加载数据
  - CommandLineRunner (实现接口，优先级）
- 日志处理
  - logback（配置，控制台输出，文件输出）