链滴

注解型 SpringMvc 为基础的框架设计（一）

作者：xixiaoming

原文链接：https://ld246.com/article/1501493471977

来源网站：链滴

关于现下流行的框架，自己做个整合，如果有想自己写个框架的，可以参考下，包含内容

1）MVC框架（SpringMVC）

2）数据池（druid）

3）映射框架（hibernate） ---- 用hibernate做的映射，操作数据库采用Hibernate+SpringJdbc的式

4）CSS框架（bootstrap）

5）日志管理（slf4j+log4j）

6）缓存管理（oacache）

7）异常管理

8）AOP

9）注解声明式事务管理

10）伪静态页面（thymeleaf）

11）文件系统（Webservice）

一：注解型SpringMVC

我们先来说说MVC的概念：浏览器发送请求，核心控制器参考配置文件将各个请求分配给不同的actio去处理，在控制器把请求交给action去处理之前，会将请求参数封装成一个参数对象，参数对象获得调用校验方法进行校验，通过后将对象传递给指定的方法，action执行完后返回结果视图，通过路径析关联到某个页面。

1 核心控制器

web程序入口在web.xml，配置DispatcherServlet，相当于struts2的FilterDispatcher，也是作为Spr ng的入口，所有*.action的请求都会进入这个调度器分发执行，他是负责流程控制（职责调度）的一 Servlet，他同时会把default.xml下的bean实例化到容器中。

```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/default.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>*.action</url-pattern>
</servlet-mapping>
```

Spring4下程序可继承WebApplicationInitializer随容器一起启动

```
@Configuration
public class ViewConfig extends WebMvcConfigurerAdapter
```

```
ServletRegistration.Dynamic registration =
  servletContext.addServlet("dispatcher", new DispatcherServlet(webContext));
registration.setLoadOnStartup(1);
registration.addMapping("/");
```

因为maven打包web项目会去找web.xml文件，此时去掉web.xml后需要配置maven打包插件，添加failOnMissingWebXml属性

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>2.3</version>
    <configuration>
        <warName>${war-name}</warName>
        <failOnMissingWebXml>false</failOnMissingWebXml>
    </configuration>
</plugin>
```

注：之后所有的配置均采用零配置文件的方式，不会把xml配置文件和java配置文件都写出来

初始化DispatcherServlet会把Bean实例化到容器中，我们采用注解的方式注明哪些Bean需要实例化

```
@Configuration
@ComponentScan(basePackages = {
    "com.es.*"
})
public class WebConfig extends WebMvcConfigurationSupport
```

2 参数对象+结果视图

bean初始化之后，请求根据转向路径进入不同的action，在控制器把请求交给action去处理之前，会请求参数封装成一个参数对象，在spring里面就是一个ModelAndView对象（spring参数对象颇多，处先用ModelAndView做例，下面会详细介绍），用来替代struts2繁琐的属性封装，ModelAndVie=model+view，model就是他作为容器和页面共享数据的意思（其实他们都相当于一个参数对象的mp而已），view是指Action执行完后要返回的结果视图，当然这个视图和struts2一样也可以是一个字串，再通过路径解析找到页面

对转向页面的路径解析（结果视图解析）：

```
@Bean
public ViewResolver getViewResolver(ContentNegotiationManager manager){
    logger.info("ViewResolver");
    ContentNegotiatingViewResolver bean = new ContentNegotiatingViewResolver();
    bean.setContentNegotiationManager(manager);
    List<ViewResolver> viewResolvers = new ArrayList<ViewResolver>();
    viewResolvers.add(new BeanNameViewResolver());
    //viewResolvers.add(new XmlViewResolver());
    viewResolvers.add(new JsonViewResolver());

    InternalResourceViewResolver jspViewResolver = new InternalResourceViewResolver();
    jspViewResolver.setPrefix("/jsp/");
    jspViewResolver.setSuffix(".jsp");
    viewResolvers.add(jspViewResolver);
```

```
    bean.setViewResolvers(viewResolvers);
    return bean;
}

  @Controller
  public class MainAction {
      @RequestMapping("/mainPage")
      //如果你要返回一个string作为结果视图，那么你需要把一个modelMap作为参数，把对象共享
容器中
      public String mainPage(HttpServletRequest request,ModelMap modelMap) {
          String message = request.getParameter("message");
          modelMap.addAttribute("message",message);
          return "main/main";
      }

      @RequestMapping("/mainPage1")
      //如果你返回一个ModelAndView那么把共享的对象放在ModelAndView里面（此时没有路径解
)
      public ModelAndView mainPage1(HttpServletRequest request) {
          String message = request.getParameter("message");
          ModelMap modelMap = new ModelMap();
          modelMap.addAttribute("message",message);
          return new ModelAndView("forward:jsp/main/main.jsp",modelMap);
      }
  }
```

在页面上接收message直接采用${message}即可。

由此为止，一个请求到处理结束的转发均已完成，也就是一个MVC的完成。

之后写数据池、hibernate映射和操作数据库。