

**Spring 原理：实现了ioc,autowire,SPringmvc部分功能**

1. **配置阶段**

1.1：准备一个maven项目，pom文件引入Servlet .jar包

1.2：准备一个application.properties文件。--scanPackage=tang.demo

表示要扫描的包

1.3:对web.xml进行配置—为我们后面拦截用户请求进入我们手写TangDispatchServlet做准备。其实我们应该是配置一个拦截器，我们写的额这个类当作拦截类处理用户请求。我们为了方便，把这个servlet路径配置/\*，拦截所有请求。当作拦截器。这样用户的请求就会到我们手写的方法中---配置如下

<servlet>

<servlet-name>tangmvc</servlet-name>

<servlet-class>tang.mvcframework.servlet.TangDispatchServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>application.properties</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

<!-- 大于等于1代表容器启动就会初始化这个servlet -->

</servlet>

<servlet-mapping>

<servlet-name>tangmvc</servlet-name>

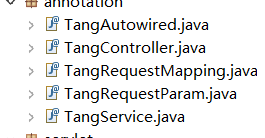
<url-pattern>/\*</url-pattern>

</servlet-mapping>

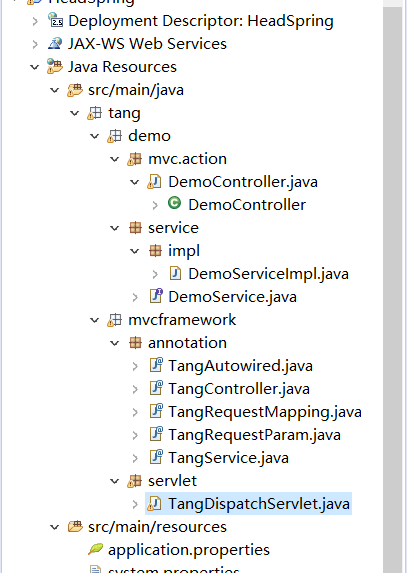
1.4：<init-param>项目启动时候加载application.xml文件的，这样我们后面需要根据这个配置文件的配置信息，根据配置的包，进行ioc操作

配置后，在servlet中，使用ServltContext.getInitParameter("contextConfigLocation")，即可得到web。Xml文件配置的application.properties，然后使用properties解析文件。Jdbc中很常用。

1.5:创建注解类：比如以下几个



1.6：完整目录结构



1. **初始化阶段:-在servlet方法里面init方法进行初始化-主要有如下几步：**

System.***out***.println("初始化开始");

//1.加载配置文件loadConfig(config.getInitParameter("contextConfigLocation"))

// 2.扫描相关类

scannerClass(*props*.getProperty("scanPackage"));

// 3.把扫描出来的相关类实例化，并且存入到ioc容器

instanceClass();

// 4.完成自动赋值，依赖注入

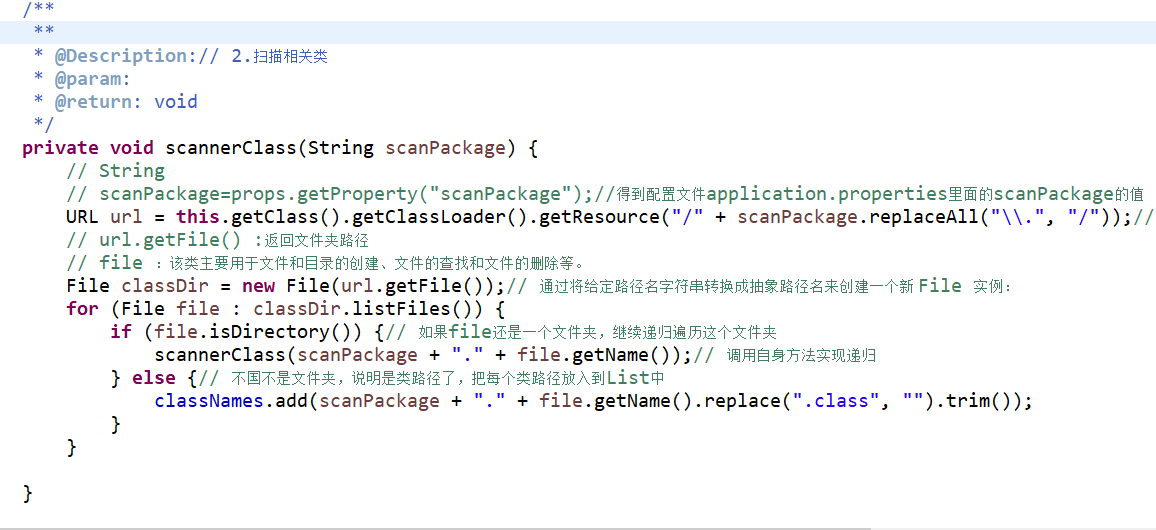
autowiredDI();

// 5.初始化HanderMapping:处理器映射器：处理请求

initHandlerMapping();

具体代码：





1. **运行阶段：根据用户请求的url(url需要修改),去map集合查找这个url,与对应的方法通过反射执行**
2. **private** **void** doDispatch(HttpServletRequest req, HttpServletResponse resp) **throws** Exception {
3. **if** (handMapping.isEmpty()) {
4. **return**;//如国这个集合是空的，下面就没有必要执行了，映射不上。
5. }
6. String requestURI = req.getRequestURI();//得到请求路径---绝对路径--包含项目路径
7. System.***out***.println("请求路径"+requestURI);
8. String contextPath=req.getContextPath();//这是项目路径
9. System.***out***.println("项目路径路径"+contextPath);
10. requestURI=requestURI.replace(contextPath, "").replaceAll("/+", "/");//把绝对路径里面的相对路径替换成空，就是对应集合里面封装的handmapping 集合里面路径了
11. System.***out***.println("修改后的请求路径："+requestURI);
12. //开始根据请求路径掉用方法了
13. **if**(!handMapping.containsKey(requestURI)) {
14. resp.getWriter().write("404");
15. **return**;
16. }
17. //使用一个集合，request封装请求带来的参数与值
18. Map<String,String[]> requestMapParms=req.getParameterMap();

21. //根据url得到方法
22. Method method = **this**.handMapping.get(requestURI);
24. //通过反射运行方法需要： method.invoke(oject,parms)设置方法的实例对象，参数
25. String beanName = method.getDeclaringClass().getSimpleName();//得到方法所在的类名，我们一会从容器里面取
26. //但是容器里面是小写，所以我们把类名首字符转成小写
27. beanName=lowerFisterCase(beanName);
29. //它们以声明顺序表示由此Method对象表示的方法的形式参数类型。如果底层方法没有参数，则返回长度为0的数组。
30. Class<?>[] methodTypes = method.getParameterTypes();
31. //用一个数组保存实参，大小与形参对应
32. Object[] objects=**new** Object[methodTypes.length];
33. //遍历形参列表
34. **for** (**int** i = 0; i < methodTypes.length; i++) {//为什么用传递遍历，因为要保证设置值的时候与参数对应
35. Class<?> parameterType = methodTypes[i];//得到方法参数的类型
36. **if** (parameterType==HttpServletRequest.**class**) {//如果是request
37. objects[i]=req;//给实参设置值保存在object数组中
38. **continue**;//这个if下面不在判断执行了
39. }**else** **if** (parameterType==HttpServletResponse.**class**) {
40. objects[i]=resp;
41. **continue**;
42. }**else** **if**(parameterType==String.**class**) {
43. **for** (Map.Entry<String, String[]> param : requestMapParms.entrySet()) {
44. //将数组转字符串，去掉数组里面的括号之类的
45. String value=Arrays.*toString*(param.getValue()).replaceAll("\\[|\\]", "").replace(",\\s", ",");
46. objects[i]=value;
48. }
49. //你要想封装其他参数，比如String 类型的其他类型的（就是RequestParm注解功能，你自己在下面加及格else if判断）
50. //为了方便，我们只封装request response.通过request也能调取到传递过来的参数
51. //虽然我们写了但是我们controller没用TangRequestParm注解，这个else if只是参考，指导怎么写
52. }
54. }
56. //method.invoke(this.ioc.get(beanName), objects);//可变参数，放object数组也行
57. method.invoke(**this**.ioc.get(beanName), req,resp);
59. }