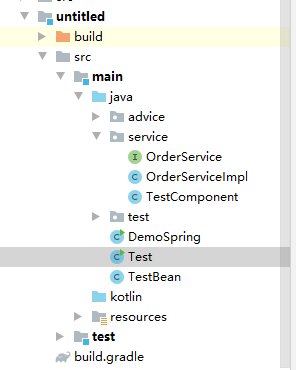
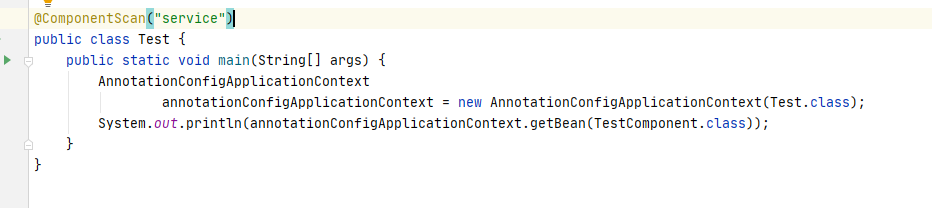
# Spring采用注解开发

1. 正常设置各种bean，如controller，service等
2. 在启动类或者可以定义一个@Configuration的注解的配置类再加上@ComponentScan 扫描bean的包即可

目录：





# 配置文件的加载处理

## 系统启动时，spring会将配置文件字符串保存至一个字符串数组(configLocations)，它会对字符串数组(configLocations)进行遍历

1. **protected** **void** loadBeanDefinitions(XmlBeanDefinitionReader reader) **throws** IOException {
2. String[] configLocations = getConfigLocations();
3. **if** (configLocations != **null**) {
4. **for** (String configLocation : configLocations) {
5. reader.loadBeanDefinitions(configLocation);
6. }
7. }
8. }

## 首页有个ResourceLoader对远程文件，classpath路径下和tomcat的war:路径的配置文件不同处理，最终都会将配置文件封装到Resource对象里面，这时候并没有加载配置文件

1. **public** **int** loadBeanDefinitions(String location, @Nullable Set<Resource> actualResources) **throws** BeanDefinitionStoreException {
2. // 获取默认的资源加载器
3. ResourceLoader resourceLoader = getResourceLoader();
4. **if** (resourceLoader == **null**) {
5. **throw** **new** BeanDefinitionStoreException(
6. "Cannot load bean definitions from location [" + location + "]: no ResourceLoader available");
7. }
9. **if** (resourceLoader **instanceof** ResourcePatternResolver) {
10. // Resource pattern matching available.
11. **try** {
12. // 把一个个文件用Resource封装
13. Resource[] resources = ((ResourcePatternResolver) resourceLoader).getResources(location);
14. //加载解析Resource
15. **int** count = loadBeanDefinitions(resources);
16. **if** (actualResources != **null**) {
17. Collections.addAll(actualResources, resources);
18. }
19. **if** (logger.isTraceEnabled()) {
20. logger.trace("Loaded " + count + " bean definitions from location pattern [" + location + "]");
21. }
22. **return** count;
23. }
24. **catch** (IOException ex) {
25. **throw** **new** BeanDefinitionStoreException(
26. "Could not resolve bean definition resource pattern [" + location + "]", ex);
27. }
28. }
29. }
30. **public** Resource[] getResources(String locationPattern) **throws** IOException {
31. Assert.notNull(locationPattern, "Location pattern must not be null");
32. // 看看配置文件路径开头有没有 “classpath\*:”
33. **if** (locationPattern.startsWith(CLASSPATH\_ALL\_URL\_PREFIX)) {
34. // a class path resource (multiple resources for same name possible)
35. // locationPattern.substring(CLASSPATH\_ALL\_URL\_PREFIX.length()) 得出的是 去掉“classpath\*:”部分，剩余的字符串
36. **if** (getPathMatcher().isPattern(locationPattern.substring(CLASSPATH\_ALL\_URL\_PREFIX.length()))) {
37. // a class path resource pattern
38. // 可以找到有指定性的配置文件，如 spring-\*.xml，spring-?.xml，{spring-config.xml}等等
39. **return** findPathMatchingResources(locationPattern);
40. }
41. **else** {
42. // all class path resources with the given name
43. // 可以找到classpath下的所有资源文件
44. **return** findAllClassPathResources(locationPattern.substring(CLASSPATH\_ALL\_URL\_PREFIX.length()));
45. }
46. }
47. **else** {
48. // Generally only look for a pattern after a prefix here,
49. // and on Tomcat only after the "\*/" separator for its "war:" protocol.
50. // 针对 tomcat 的处理
51. **int** prefixEnd = (locationPattern.startsWith("war:") ? locationPattern.indexOf("\*/") + 1 :
52. locationPattern.indexOf(':') + 1);
53. **if** (getPathMatcher().isPattern(locationPattern.substring(prefixEnd))) {
54. // a file pattern
55. **return** findPathMatchingResources(locationPattern);
56. }
57. **else** {
58. // a single resource with the given name
59. // 上面都不符合的话，
60. // 可能配置文件不在本地，但是还是有可能在本地，
61. // 所里里面还是有处理一边本地的，如果在服务器上则可以当成一个URL进行加载
62. **return** **new** Resource[] {getResourceLoader().getResource(locationPattern)};
63. }
64. }
65. }

## 在将字符串数组(configLocations)封装成Resource数组后就可以将每一个Resource里面的配置文件加载进来。

1. **public** **int** loadBeanDefinitions(Resource... resources) **throws** BeanDefinitionStoreException {
2. Assert.notNull(resources, "Resource array must not be null");
3. **int** count = 0;//
4. **for** (Resource resource : resources) {
5. count += loadBeanDefinitions(resource);
6. }
7. **return** count;
8. }

## 这里有个EncodedResource对象是对Resource的封装

1. **public** **int** loadBeanDefinitions(Resource resource) **throws** BeanDefinitionStoreException {
2. // EncodedResource 针对 Resource 的封装
3. **return** loadBeanDefinitions(**new** EncodedResource(resource));
4. }

## 最终这下面的方法会将配置文件加载成输入流

1. **public** **int** loadBeanDefinitions(EncodedResource encodedResource) **throws** BeanDefinitionStoreException {
2. Set<EncodedResource> currentResources = **this**.resourcesCurrentlyBeingLoaded.get();
3. **if** (currentResources == **null**) {
4. currentResources = **new** HashSet<>(4);
5. **this**.resourcesCurrentlyBeingLoaded.set(currentResources);
6. }// currentResources是空的就初始化容量为4
7. **if** (!currentResources.add(encodedResource)) {
8. **throw** **new** BeanDefinitionStoreException(
9. "Detected cyclic loading of " + encodedResource + " - check your import definitions!");
10. }
11. **try** {
12. // 获取文件的输入流
13. InputStream inputStream = encodedResource.getResource().getInputStream();
14. **try** {
15. // 对输入流的封装
16. InputSource inputSource = **new** InputSource(inputStream);
17. **if** (encodedResource.getEncoding() != **null**) {
18. inputSource.setEncoding(encodedResource.getEncoding());
19. }
20. // 将xml bean 变成 beandefinition
21. **return** doLoadBeanDefinitions(inputSource, encodedResource.getResource());
22. }
23. **finally** {
24. inputStream.close();
25. }
26. }
27. }

# bean实例化

* 1. 从上下文(context)获取beanDefinitions列表名进行遍历
  2. 根据当前bean名字找到父类bean，当前bean继承父类bean的属性
  3. 实例化bean对象
  4. 调用bean初始化方法
  5. bean完成初始化