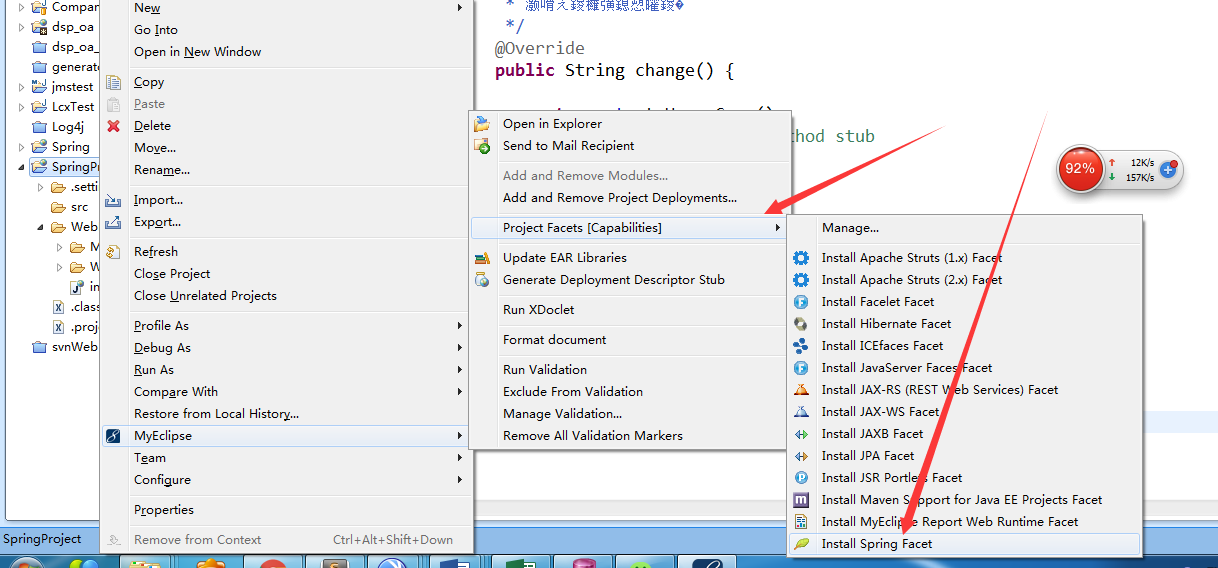
# 1、spring项目的建立

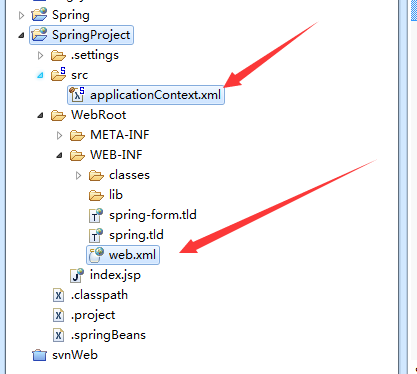
## 1、先建立一个简单的web项目

## 2、选中项目鼠标右键搭载spring环境



## 3、spring搭载完成主要信息如下

### 1、项目结构如下图



### 2、web.xml中

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"* version=*"3.0"*>  <display-name>SpringProject</display-name>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext.xml</param-value>  </context-param>  </web-app> |

### 2.1、如果想在这里就直接加载其他的spring配置文件可以直接设置参数

|  |
| --- |
| <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>  /WEB-INF/classes/application.xml,  /WEB-INF/classes/application-login.xml,  </param-value>  </context-param> |

### 3、spring配置文件

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans  xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd"*>  </beans> |

### 4、测试spring项目建立成功

#### 1、建立service

|  |
| --- |
| **package** com.hlj.springTestSuccess;  **public** **class** TestService {  **private** String name;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }    **public** **void** testSpringBean(){  System.*out*.println("bye"+name);  }  } |

#### 2、spring中注入service

|  |
| --- |
| <bean id=*"testService"* class=*"com.hlj.springTestSuccess.TestService"*>  <!-- 这里就体现出注入的概念. -->  <property name=*"name"* value=*"张宇晋"* />  </bean> |

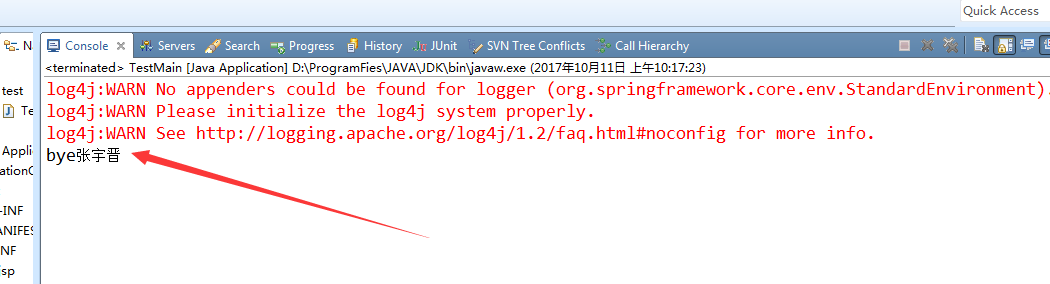
#### 3、建立单例模式获取spring配置文件

|  |
| --- |
| **final** **public** **class** ApplicaionContextUtil {  **private** **static** ApplicationContext *ac*=**null**;    **private** ApplicaionContextUtil(){    }      **privice** **static** **synchronized** **void** installAc(){  **if**(*ac*==**null**){  *ac* = **new** ClassPathXmlApplicationContext("applicationContext.xml");  }  }    **public** **static** ApplicationContext getApplicationContext(){  **if**(*ac*==**null**){  *installAc*();  }  **return** *ac*;  }    } |

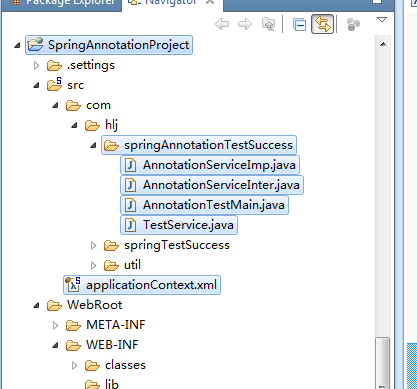
#### 4、开始测试

|  |
| --- |
| /\*\*  \* **@author** 作者 HealerJean  \* **@version** 创建时间：2017年10月11日 上午10:05:22  \* 类说明  \*/  **public** **class** TestMain {  **public** **static** **void** main(String[] args) {    ApplicationContext applicationContext = ApplicaionContextUtil.*getApplicationContext*();    TestService testService = (TestService) applicationContext.getBean("testService");  testService.testSpringBean();  }  } |

#### 5、控制台打印



# 2、spring项目使用注解开发SpringAnnotationProject



## 1、上面利用eclipse建立的spring 配置文件applicationContext 中缺少信息，所以不能使用注解扫描配置，需要替换为下面的更全面（中科软复制过来的）

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd"*>  <context:annotation-config /> |

## 2、建立接口AnnotationServiceInter

|  |
| --- |
| **public** **interface** AnnotationServiceInter {  **public** **void** testSpringAnnotationBean();  } |

## 3、实现接口类AnnotationService

|  |
| --- |
| **public** **class** *AnnotationServiceImp* **implements** AnnotationServiceInter {  **public** **void** testSpringAnnotationBean(){  System.*out*.println("AnnotationTestService");  }  } |

## 4、spring中注册这个实现接口实现类

|  |
| --- |
| <context:annotation-config />  <bean id=*"idannotationService"* class=*"com.hlj.springAnnotationTestSuccess.AnnotationServiceImp"*></bean> |

## 5、为了使用Resource，要再建立一个service，对象的名字是随意起的，就可以成功。

|  |
| --- |
| **public** **class** TestService {    @Resource  **public** AnnotationServiceInter nameService;    **public** **void** test (){  nameService.testSpringAnnotationBean();  }  } |

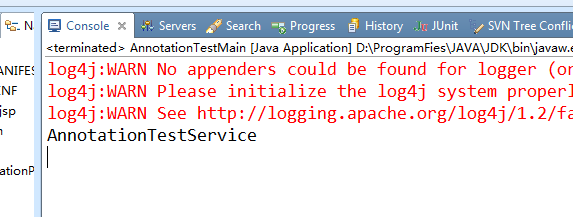
## 6、注册这个TestService

|  |
| --- |
| <context:annotation-config />  <bean id=*"idannotationService"* class=*"com.hlj.springAnnotationTestSuccess.AnnotationServiceImp"*></bean>  <bean id=*"idtestService"* class=*"com.hlj.springAnnotationTestSuccess.TestService"*></bean> |

## 7、开始测试

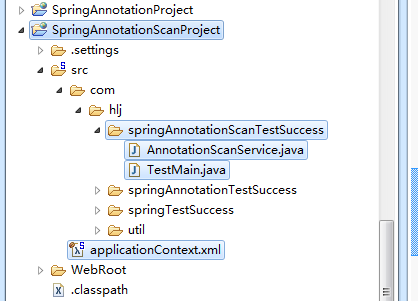
|  |
| --- |
| **public** **class** AnnotationTestMain {    **public** **static** **void** main(String[] args) {    ApplicationContext applicationContext = ApplicaionContextUtil.*getApplicationContext*();    TestService testService = (TestService) applicationContext.getBean("idtestService");  //下面这个是错误的，因为new的话，不能够将它里面的resource 注入进去  // TestService testService = new TestService();  testService.test();  }    } |

## 8、控制台打印



# 3、spring 使用扫描包 <context:component-scan

## 这种方式了具有<context:annotation-config>的功能之外，<context:component-scan>还可以在指定的package下扫描以及注册javabean），也就是说有了这个可以不使用 <context:annotation-config />也可以对@service生效



## 1、建立service使用注解@Service

|  |
| --- |
| @Service  **public** **class** AnnotationScanService {  **public** **void** testSpringAnnotationScanBean(){  System.*out*.println("AnnotationScanService");  }    } |

## 2、applicatonContext.xml添加自动扫描包路径，只写com也能生效，这样就是扫描com下所有的类

|  |
| --- |
| <context:component-scan base-package=*"com"* />  <context:component-scan base-package=*"com.hlj.springAnnotationTestSuccess"* /> |

## 3、测试，不在applicationContext.xml中注册bean也可以获取

|  |
| --- |
| **public** **class** TestMain {  **public** **static** **void** main(String[] args) {    ApplicationContext applicationContext = ApplicaionContextUtil.*getApplicationContext*();    AnnotationScanService annotationScanService = (AnnotationScanService) applicationContext.getBean(AnnotationScanService.**class**);    annotationScanService.testSpringAnnotationScanBean();  }  } |

# 4、SpringmavenWeb

# 5、spring mvc

## 1设置web.xml中的内容为下面的

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>

<display-name>StudySpringMvcOne</display-name>

<!-- springmvc前端控制器 -->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- contextConfigLocation配置springmvc加载的配置文件（配置处理器映射器、适配器等等）

如果不配置contextConfigLocation，默认加载的是/WEB-INF/servlet名称-serlvet.xml（springmvc-servlet.xml）

-->

<init-param>

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

<param-value>classpath:springmvc-servlet.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<!--

第一种：\*.action，访问以.action结尾 由DispatcherServlet进行解析,决定了请求URL必须是一个带后缀的URL

第二种：/，所有访问的地址都由DispatcherServlet进行解析，对于静态文件的解析需要配置不让DispatcherServlet进行解析，例如<mvc:default-servlet-handler />

使用此种方式可以实现 RESTful风格的url

第三种：/\*，这样配置不对，使用这种配置，最终要转发到一个jsp页面时，

仍然会由DispatcherServlet解析jsp地址，不能根据jsp页面找到handler，会报错。

-->

<url-pattern>/</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

</web-app>

## 2.设置springmvc-servlet.xml（src目录下创建）

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd"*>

<!--

如果不想在xml文件中配置bean，我们可以给我们的类加上spring组件注解，

只需再配置下spring的扫描器就可以实现bean的自动载入。

-->

<context:component-scan base-package=*"com.hlj.control"*/>

<!-- 针对于web中的 /

在springMVC-servlet.xml中配置<mvc:default-servlet-handler />后，

会在Spring MVC上下文中定义一个org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler，

它会像一个检查员，对进入DispatcherServlet的URL进行筛查，如果发现是静态资源的请求，

就将该请求转由Web应用服务器默认的Servlet处理，如果不是静态资源的请求，才由DispatcherServlet继续处理。

一般Web应用服务器默认的Servlet名称是"default"，因此DefaultServletHttpRequestHandler可以找到它。

如果你所有的Web应用服务器的默认Servlet名称不是"default"，则需要通过default-servlet-name属性显示指定：

<mvc:default-servlet-handler default-servlet-name="所使用的Web服务器默认使用的Servlet名称" />

-->

<mvc:default-servlet-handler/>

<!-- 开启注解支持

使用了<mvc:annotation-driven />， 它会自动注册DefaultAnnotationHandlerMapping

与AnnotationMethodHandlerAdapter 这两个bean,所以就没有机会再给它注入interceptors属性，就无法指定拦截器。

当然我们可以通过人工配置上面的两个Bean，不使用 <mvc:annotation-driven />，就可以 给interceptors属性 注入拦截器了。

-->

<mvc:annotation-driven/>

<!-- InternalResourceViewResolver视图解析器 -->

<bean id=*"viewResolver"* class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 前缀 -->

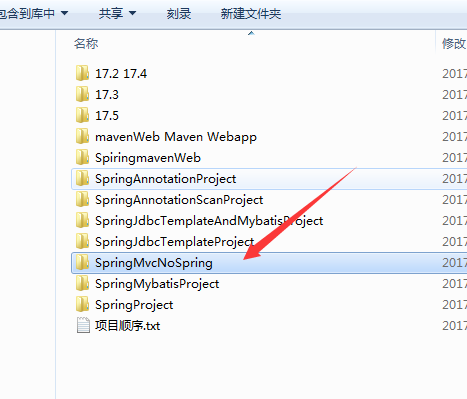
<property name=*"prefix"* value=*"/WEB-INF/jsp/"*></property>

<!-- 后缀 -->

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

## 3、代码位置



# 6、spring mvc 和spring整合

## 1.Spring配置文件 Spring接管(注解让spring mvc提供 exclude)

在src目录下创建SpringIOC的配置文件applicationContext.xml

<context:component-scan base-package=*"com.hlj.control.integrate"*>

<context:exclude-filter type=*"annotation"*

expression=*"org.springframework.stereotype.Controller"*/>

<context:exclude-filter type=*"annotation"*

expression=*"org.springframework.web.bind.annotation.ControllerAdvice"*/>

</context:component-scan>

## 2.在Web.xml中添加配置,配置spring

<!-- 15、整合SpringIOC和SpringMVC -->

<!-- configure the springIOC -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

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

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

## 3、在SpringMVC中进行一些配置（include 提供注解），防止SpringMVC和SpringIOC对同一个对象的管理重合，这个时候就可以注释掉之前的context:component-scan

在SpringMVC中进行一些配置，防止SpringMVC和SpringIOC对同一个对象的管理重合

-->

<!-- scan the package and the sub package -->

<context:component-scan base-package="com.hlj.control.integrate">

<context:include-filter type=*"annotation"*

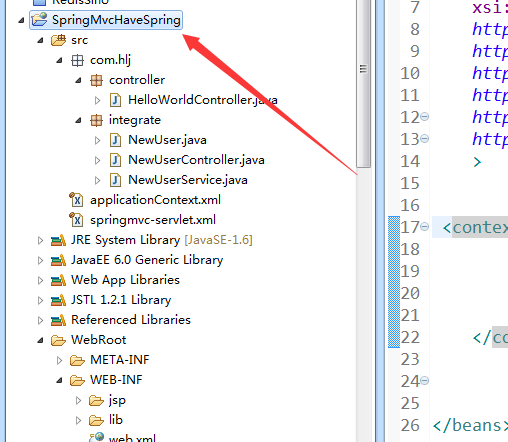
expression=*"org.springframework.stereotype.Controller"*/>

<context:include-filter type=*"annotation"*

expression=*"org.springframework.web.bind.annotation.ControllerAdvice"*/>

</context:component-scan>

## 4、代码位置



## 5、正常情况下 spring mvc 提供controller支持，spring 提供service

### **1 在主容器中（applicationContext.xml），将Controller的注解打消掉**

**[html]** [view plain](http://blog.csdn.net/liuwenbo0920/article/details/7260013) [copy](http://blog.csdn.net/liuwenbo0920/article/details/7260013)

1. **<context:component-scan** base-package="com"**>**
2. **<context:exclude-filter** type="annotation" expression="org.springframework.stereotype.Controller" **/>**
3. **</context:component-scan>**

### **2 而在springMVC配置文件中将Service注解给去掉**

**[html]** [view plain](http://blog.csdn.net/liuwenbo0920/article/details/7260013) [copy](http://blog.csdn.net/liuwenbo0920/article/details/7260013)

1. **<context:component-scan** base-package="com"**>**
2. **<context:include-filter** type="annotation" expression="org.springframework.stereotype.Controller" **/>**
3. **<context:exclude-filter** type="annotation" expression="org.springframework.stereotype.Service" **/>**
4. **</context:component-scan>**

**因为spring的context是父子容器，所以会产生冲突，Controller会进步前辈行扫描装配，而此时的Service还没有进行事务的加强处理惩罚，获得的将是原样的Service（没有经过事务加强处理惩罚，故而没有事务处理惩罚才能） ，最后才是applicationContext.xml中的扫描设备进行事务处理惩罚**

# 7、