

Day26

java课程-李彦伯

Day26

匿名类(补充)

注解

注解的作用

常见注解

自定义注解(了解)

使用注解(掌握)

元注解

@Retention

@Target

通过反射的方式进行获取(了解)

Junit单元测试

使用junit步骤和要求

Junit的其他测试相关注解

@Test属性(了解)

断言判断(了解)

Spring

一站式框架

Spring环境搭建以及HelloWorld案例

配置schema约束步骤

schema讲解

IOC控制反转

bean元素的配置和创建

name属性和class属性

bean元素的创建

scope属性

生命周期属性

DI依赖注入

注入方式

set方法注入

构造方法注入

p名称空间注入

SPEL表达式注入

复杂类型注入

Spring 的分模块配置文件开发

CRM改为让Spring进行管理

安装STS

匿名类(补充)

```
public class TestDemo {

    public static void main(String[] args) {

        new People().setName("张三");

        MyInterface mi = new MyInterfaceImp
l();
        MyInterface mi2 = new MyInterface()
{
            @Override
            public void addUser() {
                System.out.println(1111);
            }
        };
        mi2.addUser();
        test(new MyInterface() {
            @Override
            public void addUser() {
                System.out.println("匿名接口
实现类");
            }
        });

        test( new MyInterface() {

            @Override
            public void addUser() {

            }

        });
    }
}
```

```

        Car ca = new Car() {

            @Override
            public void run() {

            }

        };
        test2(new Car() {

            @Override
            public void run() {
                // TODO Auto-generated method stub

            }

        });

    }

    public static void test(MyInterface m){
        System.out.println("调用");
        m.addUser();
    }

    public static void test2(Car cc){

    }

}

```

注解

我们可以认为注解Annotation是一种属于代码级别的说明,是一种具有功能的说明.是从jdk1.5版本以后添加的新特性,其格式为 `@注解名称`.对比注释而言,注释是写给人看的,是没有功能的文字.注解是写个JVM看的,具有功能的代码

注解的作用

在目前的主流应用在替代配置文件

- 对比servlet2.5,Servlet3.0新加了很多的新特性,其中就有提供了注解的支持,也就是不需要使用web.xml可以进行路径的配置
- 优点:开发效率高
- 确定:耦合性太强,不便于维护

常见注解

- `@Override` : 告知编译器此方法是覆盖父类的
- `@Deprecated` : 标注过时
- `@SuppressWarnings` : 压制警告

自定义注解(了解)

- 使用关键字@interface表示注解
- 注解中定义的属性格式为 修饰符 数据类型 属性名称()
default 具体的值;
 - 修饰符只能是public abstract,和接口相同会默认添加
 - 数据类型可以是: 基本类型, String, 枚举类型, 注解类型, Class类型, 以上类型的一维数组类型
 - default是默认值可以不写,如果不写在使用注解的时候就必须进行设置,如果有默认值就可以不用进行设置
- 注意: 如果属性的名字是value, 并且注解的属性设置的时候只需设置这一个 那么在使用注解时可以省略value

使用注解(掌握)

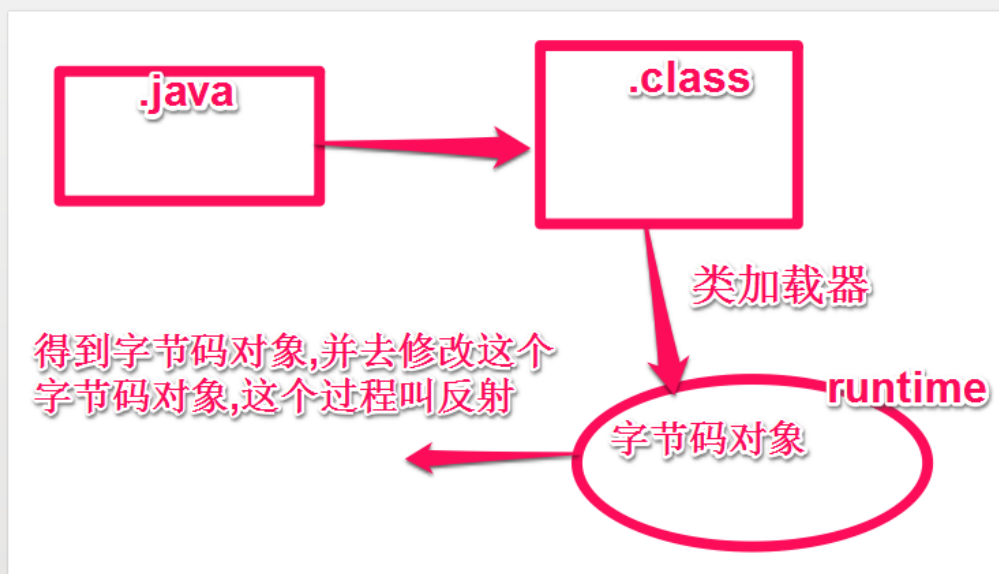
在类,方法,字段 上面是 @注解名称 (属性名=属性值, 属性名=属性值)

元注解

可以认为是注解的注解

@Retention

代表注解的可见范围



- SOURCE: 注解在源码级别可见(默认是source级别)
- CLASS : 注解在字节码文件级别和源码级别可见
- RUNTIME : 注解在整个运行阶段以及之前的两个级别都可见
- 注意:要想解析使用了注解的类 , 那么该注解的Retention 必须设置成Runtime

@Target

代表注解修饰的范围：类上使用，方法上使用，字段上使用

- FIELD:字段上可用此注解
- METHOD:方法上可以用此注解
- TYPE:类/接口上可以使用此注解

- PARAMETER:参数上可以使用
- CONSTRUCTOR:构造函数上可以使用
- LOCAL_VARIABLE:局部变量上可以使用

```
@Target({TYPE, FIELD, METHOD, PARAMETER, CON  
STRUCTOR, LOCAL_VARIABLE})  
@Retention(RetentionPolicy.SOURCE)  
public @interface SuppressWarnings {  
    String[] value();  
}
```

通过反射的方式进行获取(了解)

```
Class ac = AnnotationTest.class;  
Method[] methods = ac.getMethods();  
for(Method me : methods){  
    if(me.isAnnotationPresent(MyAnno.class))  
    {  
        MyAnno annotation = me.getAnnotatio  
n(MyAnno.class);  
        System.out.println(annotation.heigh  
t()+"--"+annotation.value());  
    }  
}
```

Junit单元测试

在我们编写代码的过程中,可能会写很多的方法,当写完某个方法的时候我们往往需要测试一下某个方法是正确的,这个时候我们就可以使用Junit单元测试帮助我们完成Junit是java语言的单元测试框架,属于第三方的一个工具,一般情况都需要导入相应的jar包才能使用,不过多数的java开发环境已经帮我们集成了这个工具,在我们的eclipse中同样集成了这个测试工具

使用junit步骤和要求

- 方法必须是 `public void` 修饰
- 方法必须是无参的方法
- 在方法上添加注解@Test导入并导入相应的jar包
- run as选中Junit进行测试
- **注意:**如果一个类中有多个@Test的测试,那么会一起执行,如果需要执行具体某一个,可以选中方法后run as

Junit的其他测试相关注解

- @Test : 把一个方法标记为测试方法
- @Before : 每一个测试方法执行前自动调用一次(需要配合@Test使用)
- @After : 每一个测试方法执行完自动调用一次(需要配合@Test使用)
- @BeforeClass : 所有测试方法执行前执行一次,在测试类还没有实例化就已经被加载,所以用static修饰(需要配合@Test使用)

- `@AfterClass` : 所有测试方法执行完执行一次，在测试类还没有实例化就已经被加载，所以用static修饰(需要配合@Test使用)
- `@Ignore` : 暂不执行该测试方法(需要配合@Test使用)

```
public class JunitTest {
    @Test
    public void test01(){
        System.out.println("test01");
    }
    @Test
    public void test02(){
        System.out.println("test02");
    }
    @BeforeClass
    public static void testBeforeClass(){
        System.out.println("testBeforeClass");
    }
    @AfterClass
    public static void AfterClass(){
        System.out.println("testAfterClass");
    }
    @Before
    public void testBefore(){
        System.out.println("testBefore");
    }
    @After
    public void testAfter(){
        System.out.println("testAfter");
    }
    @Ignore
    public void testIgnore(){
        System.out.println("testIgnore");
    }
}
```

```
}
```

@Test属性(了解)

- expect属性,用来测试异常相关,其格式为 `@Test(expect = xxxException.class)`,如果出现异常测试成功,如果未出现异常测试失败

```
@Test(expected=ArithmeticException.class)
public void test01(){
    System.out.println("test02");
    System.out.println(1/0);
}
```

- timeout属性,是用来测试超时操作的单位是毫秒其格式为 `@Test(timeout=毫秒值)`,如果运行时间在设置之内,测试通过,如果超出测试失败

```
@Test(timeout=1)
public void test01(){
    System.out.println("test01");
    for(int i = 0 ; i < 10000; i++){
        System.out.println("sssss");
    }
}
```

断言判断(了解)

- 判断结果是否是预期的结果格式为 `assertEquals(期望结果, 实际结果);`
- 断言的包的导入 `import static org.junit.Assert.*;`

使用了JDK5中的静态导入，只要在import关键字后面加上static关键字，就可以把后面的类的static的变量和方法导入到这个类中，调用的时候和调用自己的方法没有任何区别。

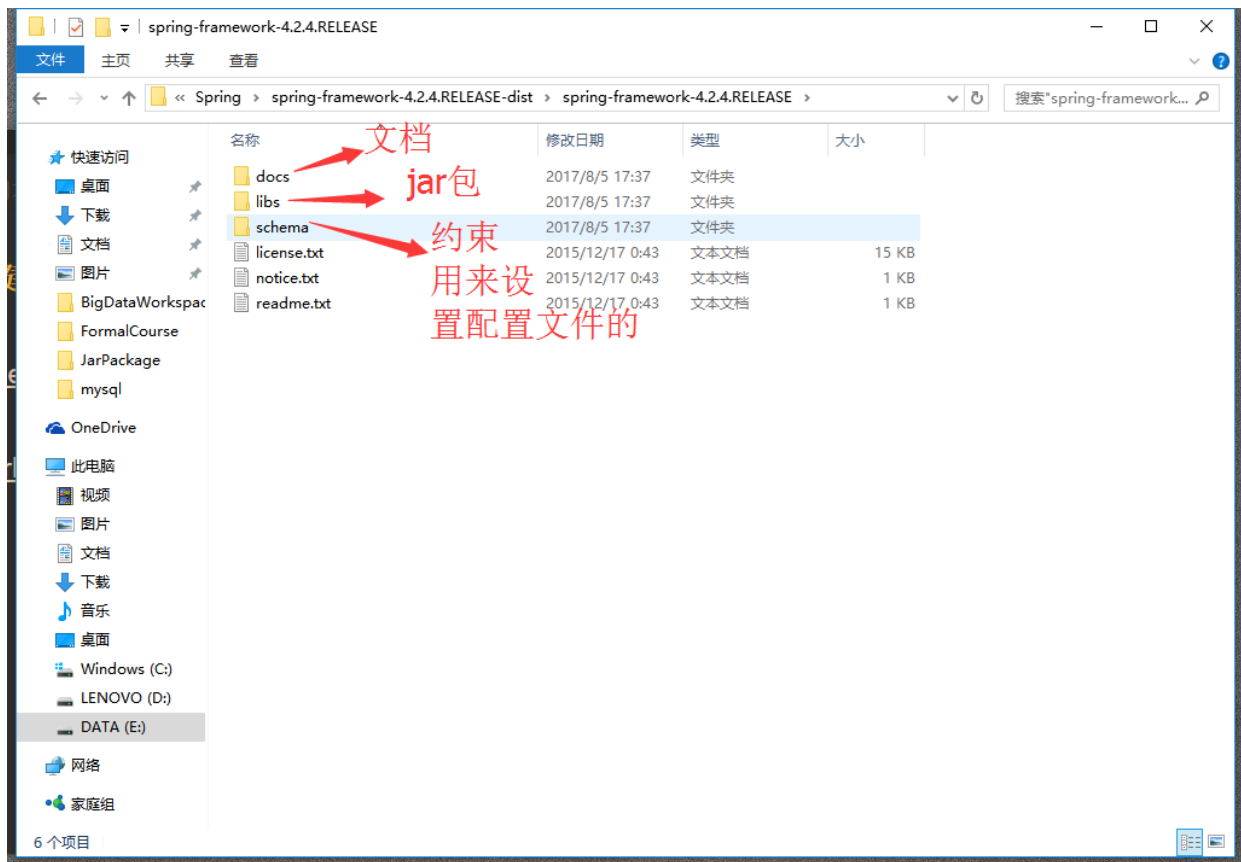
Spring

Spring是于 2003 年兴起的一个轻量级的 Java 开发框架.Spring 的核心是控制反转（IOC）和面向切面（AOP）。Spring 是一个分层的 JavaSE/EEfull-stack(一站式) 轻量级开源框架。

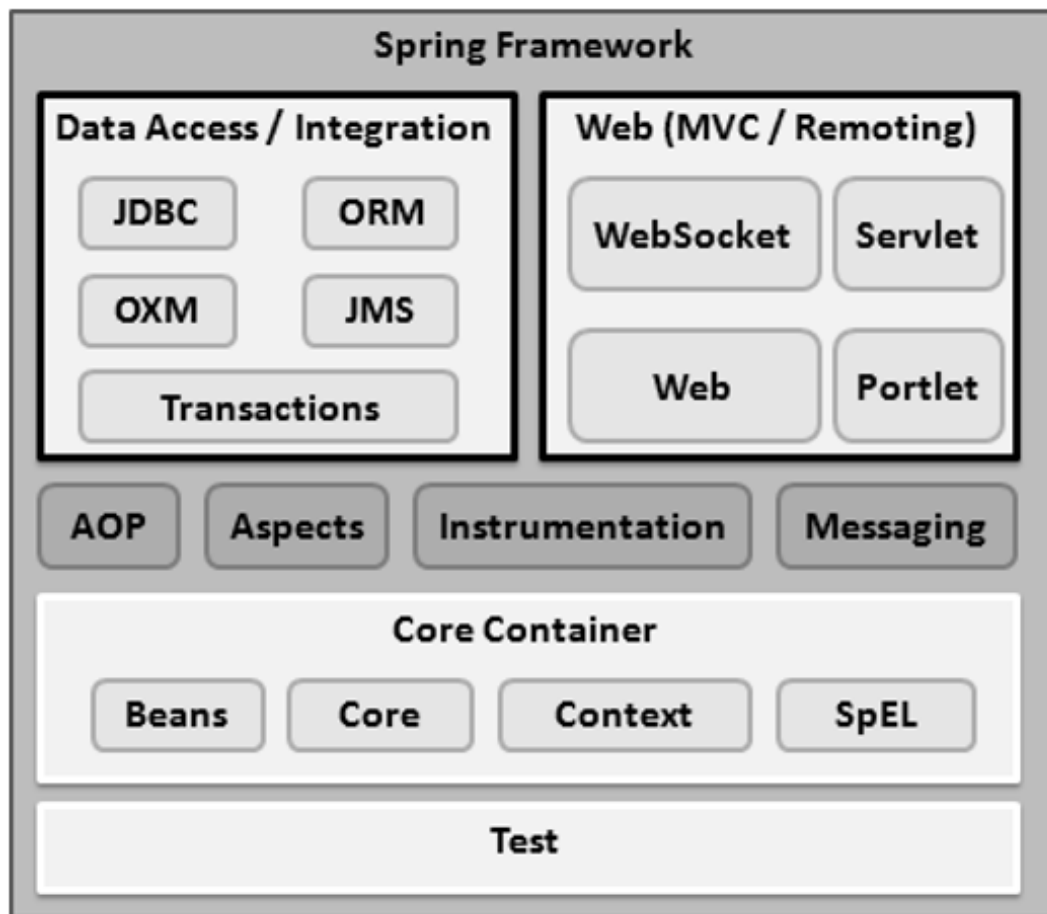
一站式框架

- web层:可以使用Spring提供的SpringMVC框架进行请求的处理
- service层:可以使用Spring的IOC进行对象的创建
- dao层:可以使用Spring提供的JDBC模板进行处理
- spring不但能够自身完成web应用的开发,并且能够完美兼容其他开源框架,因为spring是一个容器

- 下载Spring相关jar包<http://repo.springsource.org/libs-release-local/org/springframework/spring>



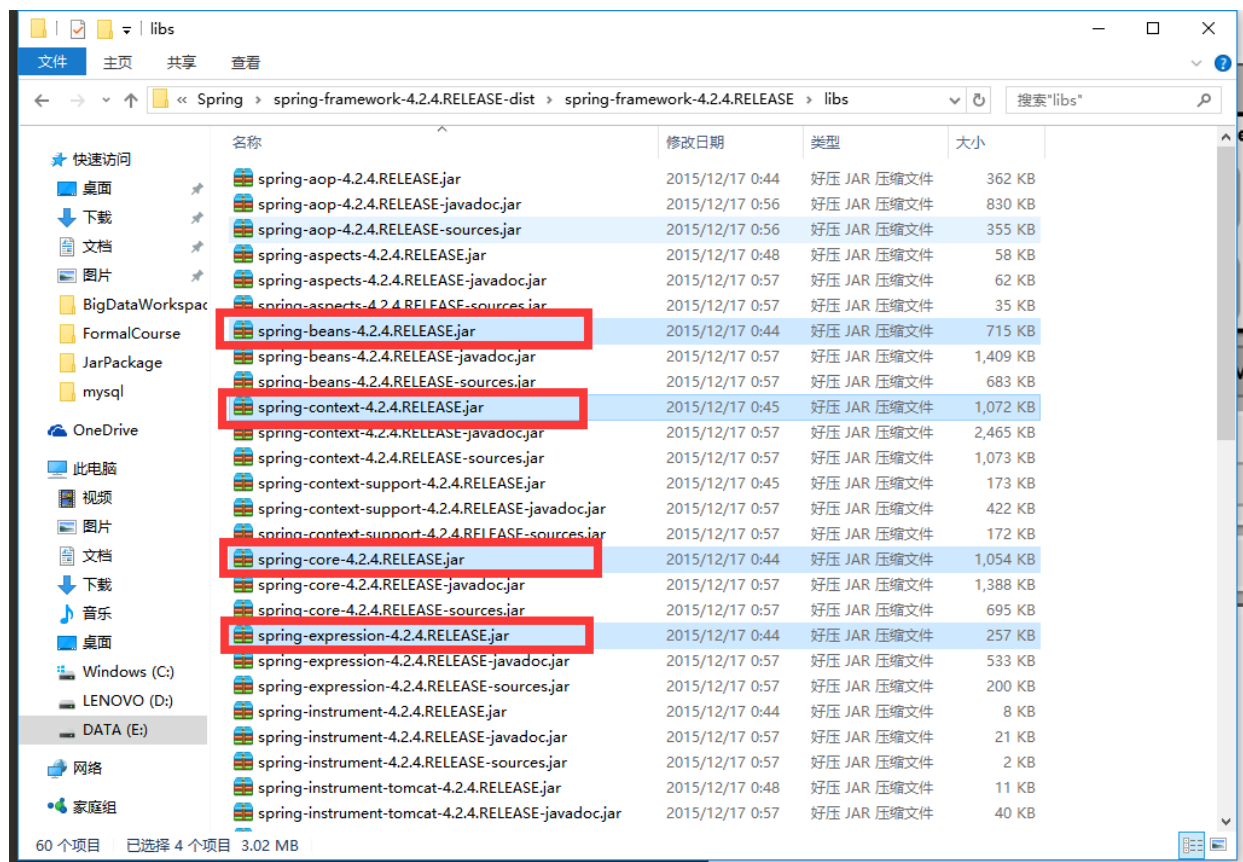
- 导包(4+2)



Spring的模块划分

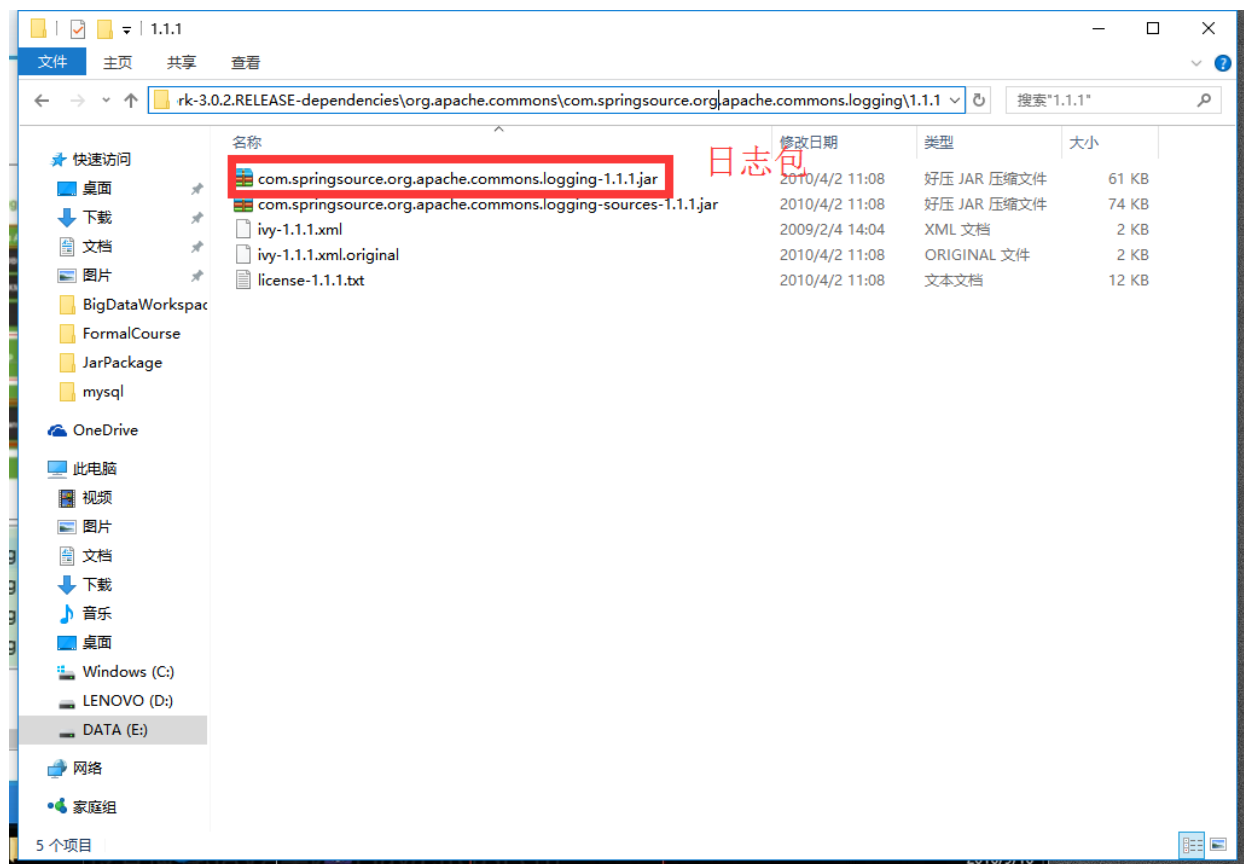
Spring的模块划分

-



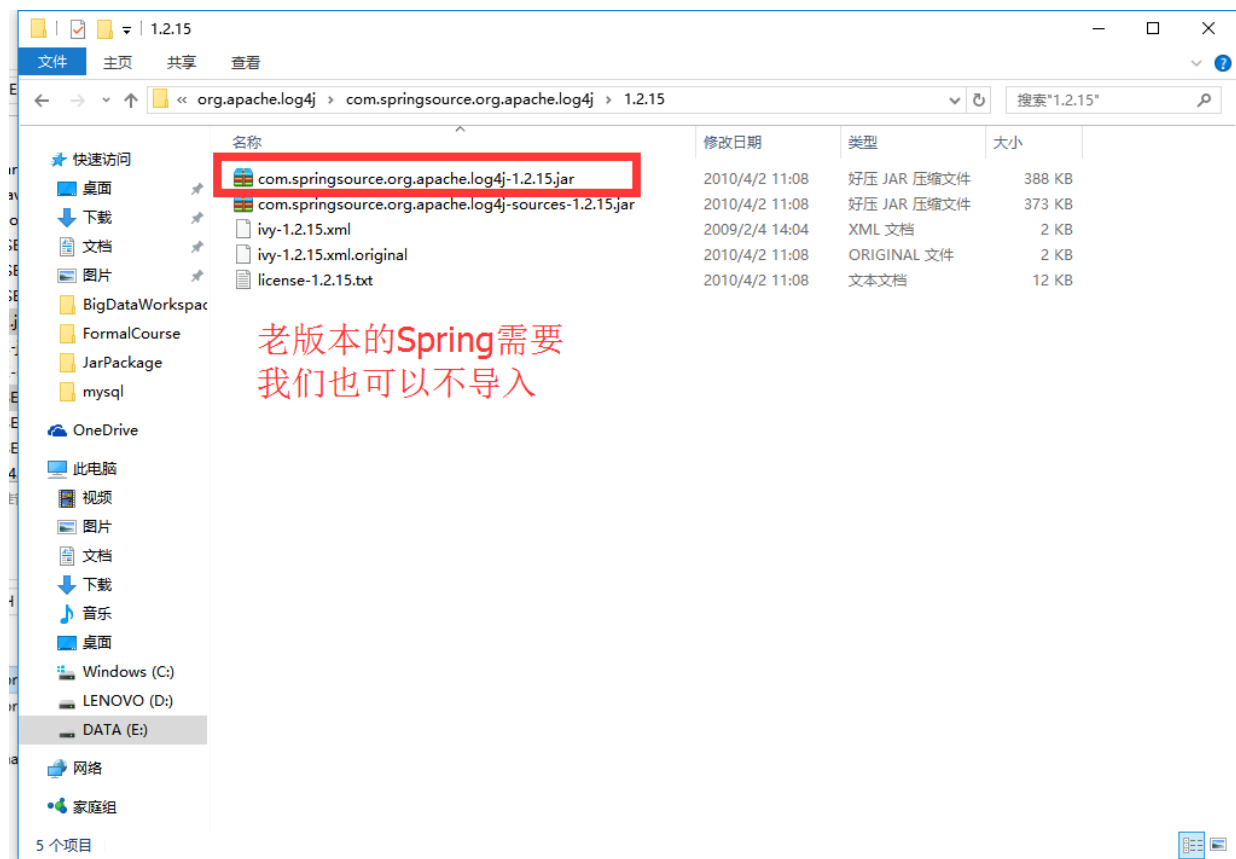
Spring lib文件夹中的四个jar包

Spring lib文件夹中的四个jar包



依赖包中的日志包

依赖包中的日志包



依赖包中的log4j

依赖包中的log4j

- 创建普通的类,让Spring帮我们创建

```

public class People {
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public void eat(){
        System.out.println("吃饭了");
    }
    @Override
    public String toString() {
        return "People [name=" + name + ", age=" + age + "]\n";
    }
}

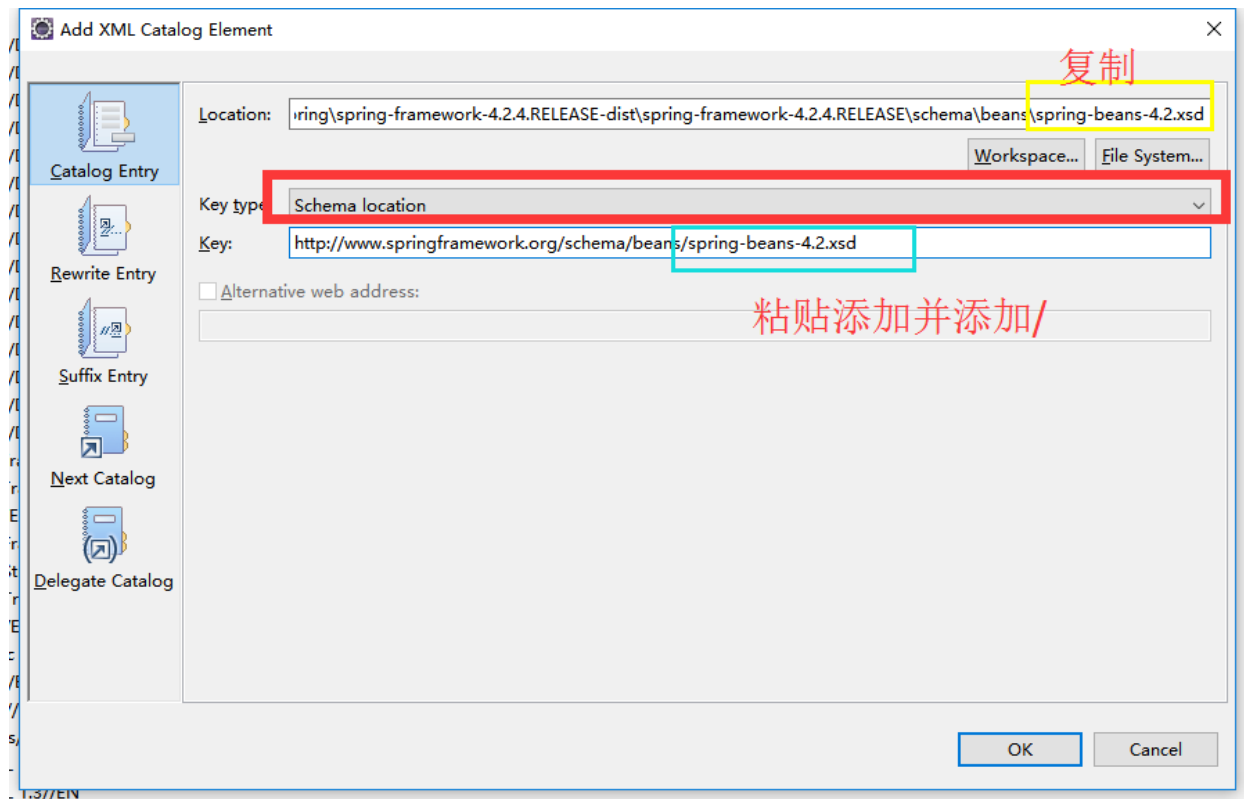
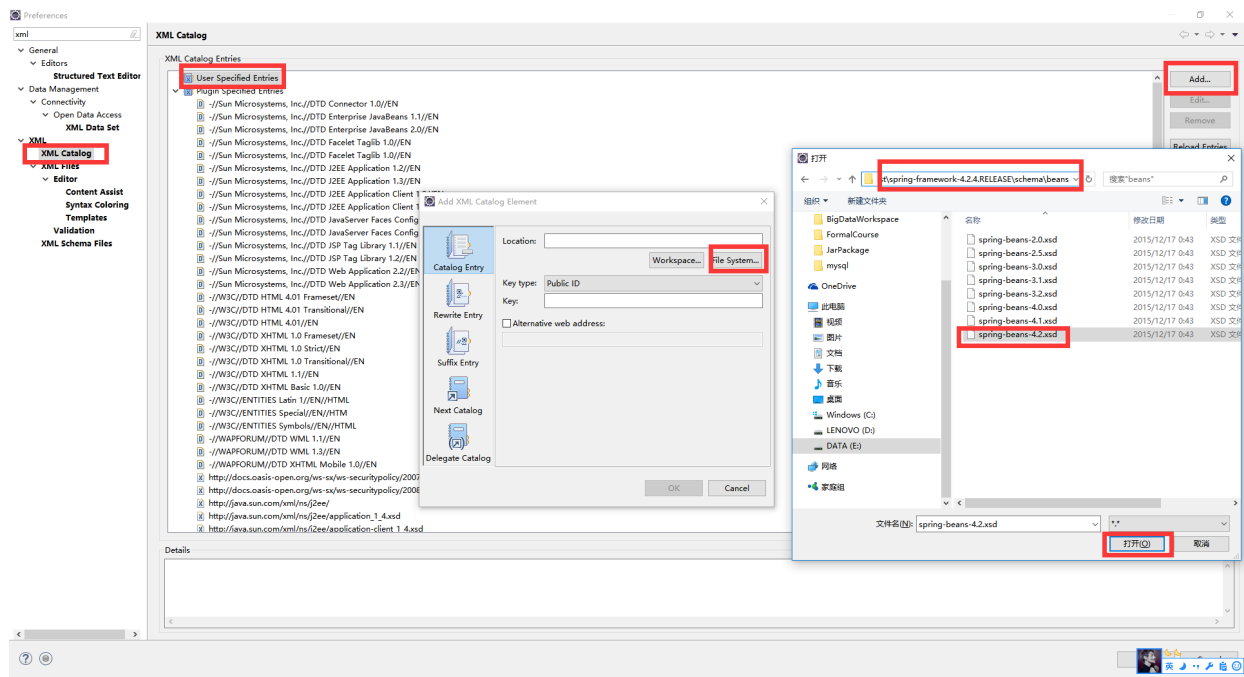
```

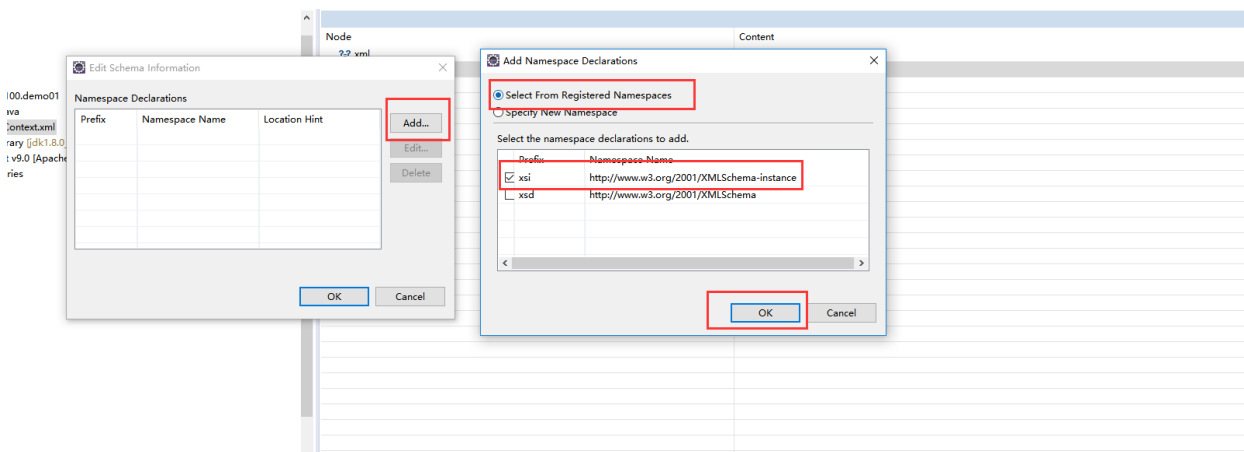
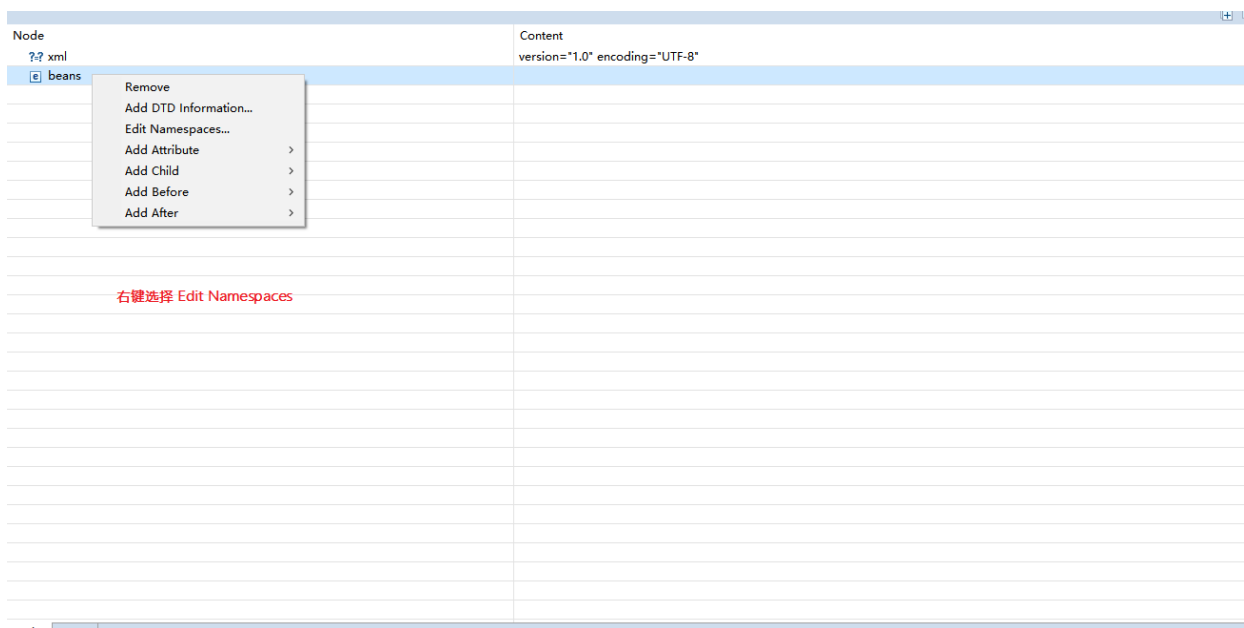
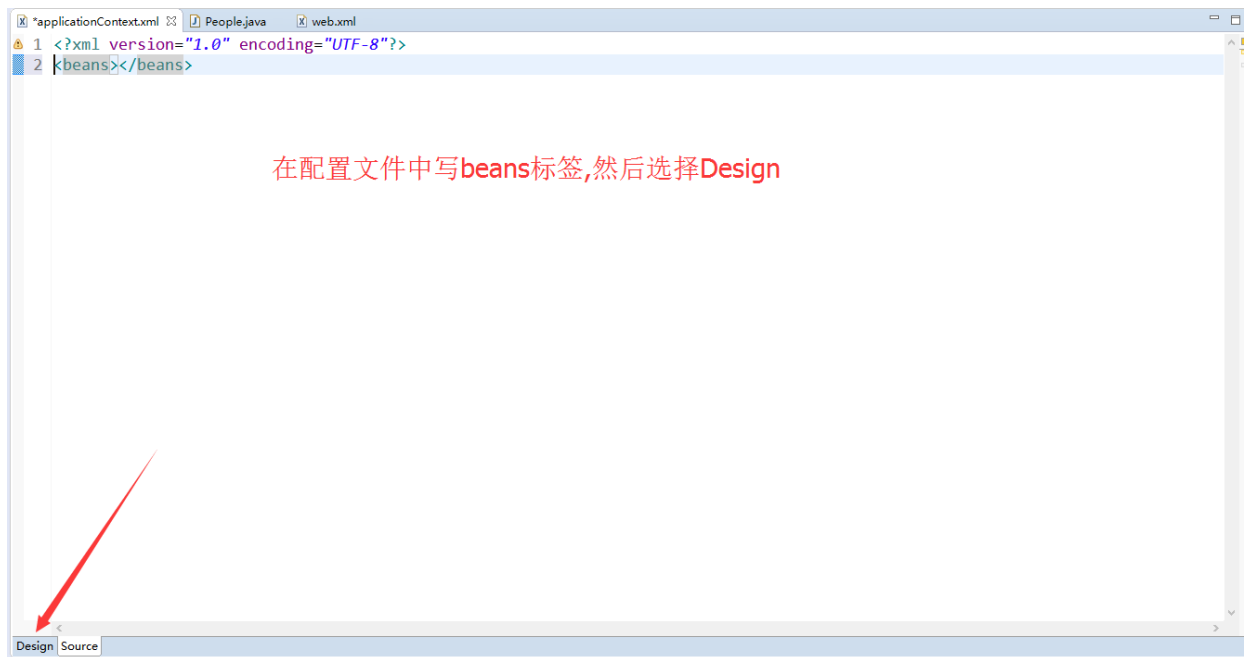
- 创建xml的配置文件,将我们写的类配置到对应的文件中,建议放在src下,建议命名为applicationContext.xml,可以认为是spring的规范,如果不按照此规范也不会出错
- 设置schema约束,是需要按照对应的命名空间来决定标签中的顺序的,也可以认为配置后书写标签能够有提示

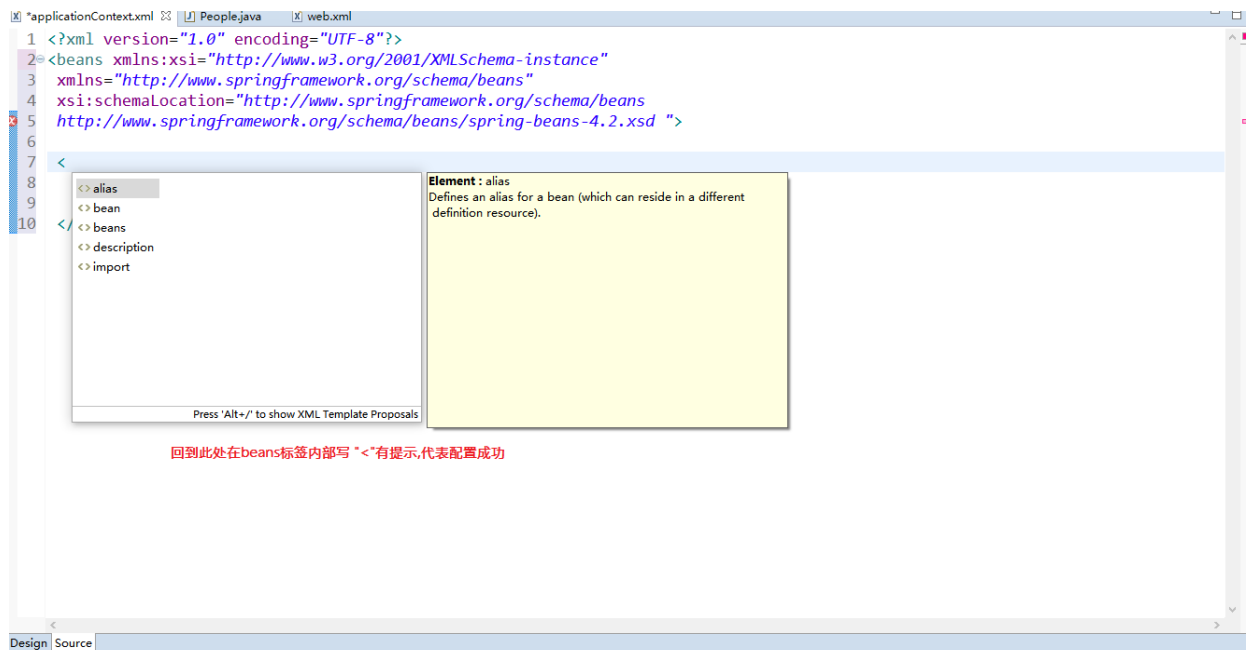
- 在配置文件写 `<bean name="people" class="com.zhiyou100.demo01.People"></bean>`
- 创建一个测试类,写一个测试方法

```
public class TestPeople {  
    @Test  
    public void test01(){  
        //创建spring容器对象,加载src下的配置文件  
        ApplicationContext context = new Cla  
ssPathXmlApplicationContext("applicationCont  
ext.xml");  
        //从容器中通过name获取对象  
        People person = (People) context.get  
Bean("people");  
        //调用对象的方法  
        person.eat();  
    }  
}
```

配置schema约束步骤







配置约束是为了在xml文件中进行配置的时候,能够按照约束进行标签的校验

- 配置一个xml文件最多只能有一个匿名的命名空间,所以我们将bean这个命名空间配置为匿名,如果将其配置为xx,那么当使用bean中的标签的时候需要加 `<xx:bean>` `</xx:bean>`,在配置spring的时候,我们习惯于将bean标签配置为匿名的,可以认为是一种规范
- 因为要使用schemaLocation来进行配置键值对匹配,所以我们先配置了别名为xsi的

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" 命名空间
```

- 右配置了一个匿名的命名空间

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

- 通过xsi的schemaLocation,将键值对进行对应

```
xsi:schemaLocation="http://www.springframework.org/schema/beans  
http://www.springframework.org/schema/beans/spring-beans-4.2.xsd "
```

- 将来我们引用其他的命名空间的时候也是使用相同的方法进行配置

IOC控制反转

IOC控制反转(Inversion of Control)将对象的创建权交给了Spring,我们通过Spring,就不用每次去创建对象,只需要通过Spring容器调用getBean(“name属性或id属性”)获取对应的对象

- ApplicationContext实现类会在创建的时候,将内部配置的所有对象加载到容器中
- ClassPathXmlApplicationContext是ApplicationContext的接口实现类,是从classpath的目录下加载配置文件

bean元素的配置和创建

凡是交给spring容器管理的对象,都是使用bean元素进行配置

name属性和class属性

- name属性给管理的对象起名字,根据该名称能获取对象
- class属性是获取该对象的完整类名
- id属性跟name属性的功能和用法一模一样,和name属性相比,id属性唯一不重复,并且不能含有特殊字符,如果只设置name属性,那么name属性在容器中就必须是唯一的

bean元素的创建

默认调用空参构造方法进行创建对象,如果没有空参构造方法就会报错误

- 空参构造方法创建

```
<!--空参构造方法-->  
<bean name="people1" class="com.zhiyou100.de  
mo01.People"></bean>
```

- 静态工厂方式创建

```
<!--静态工厂-->  
<bean name="people2" class="com.zhiyou100.cr  
eate.PeopleFactory" factory-method="createPe  
ople"></bean>
```

- 实例工厂方式创建

```
<!--实例工厂-->  
<bean name="factoryBean" class="com.zhiyou10  
0.create.PeopleFactory"></bean>  
<bean name="people3" factory-bean="factoryBe  
an" factory-method="CreatePeople2"></bean>
```

scope属性

设置bean元素的创建对象的方式

- singleton单例模式创建,当前对象在容器中只会创建一个对象

```
<bean name="people1" class="com.zhiyou100.de  
mo01.People" scope="singleton"></bean>
```

- prototype多例模式创建,当前对象在容器中每次调用getBean返回的是新的对象

```
<bean name="people1" class="com.zhiyou100.de  
mo01.People" scope="prototype"></bean>
```

以后绝大多数的对象都使用默认值就行,在Struts2中的action要配置成prototype

生命周期属性

- init-method初始化方法
- destroy-method销毁方法
- 不能和prototype一起使用,如果是多例的话,spring容器将对象创建完成后就不再负责管理这个对象了,只有是singleton对象容器才会负责去管理它

```
<bean name="people1" class="com.zhiyou100.de  
mo01.People" scope="singleton" init-metho  
d="init" destroy-method="destory"></bean>
```

DI依赖注入

依赖注入(Dependency Injection),需要有 IOC 的环境,Spring 创建这个类的过程中,Spring 将类的依赖的属性设置进去.

注入方式

如果是基本类型使用value,如果是引用类型就需要使用ref

set方法注入

类中必须要有set方法,否则属性无法正常注入

```
<bean name="user" class="com.zhiyou100.di.User">
  <!--基本类型使用value-->
  <property name="name" value="张三"></property>
  <property name="age" value="18"></property>
  <!--引用类型使用ref-->
  <property name="car" ref="car"></property>
</bean>

<bean name="car" class="com.zhiyou100.di.Car">
  <property name="name" value="拖拉机"></property>
  <property name="price" value="8000"></property>
</bean>
```

构造方法注入

通过构造方法创建对象,并传递相应的参数给具体的属性

```
<!--
```

构造方法注入属性

`name` 表示参数名称

`value` 给参数赋的具体值

`index` 表示参数的索引

`type` 表示参数的类型

以上所有的属性并不是全部都要设置

```
-->
```

```
<bean name="user2" class="com.zhiyou100.d
i.User">
  <constructor-arg name="name" value="韩梅梅"
index="1" type="java.lang.String"></construc
tor-arg>
  <constructor-arg name="car" ref="car"></con
structor-arg>
</bean>
```

p名称空间注入

```
xmlns:p="http://www.springframework.org/sche
ma/p"
<bean name="user3" class="com.zhiyou100.di.U
ser" p:name="李雷" p:age="18" p:car-ref="ca
r"></bean>
```

SPEL表达式注入

spring expression language(表达式语言注入),必须在前面基础上完成,比如解决类类似配置一个bean的属性是另一个bean的属性值的时候,需要使用spel注入

```
<bean name="user4" class="com.zhiyou100.di.U
ser">
<property name="age" value="#{car.price > 30
0000 ? 35 : 20}"></property>
<property name="name" value="#{'zhangSan'.to
UpperCase()}"></property>
<property name="car" ref="car"></property>
</bean>
```

复杂类型注入

- 数组类型
- list类型
- map类型

```
public class ComplexTypeObject {
    Object[] arr;
    List<Object> list;
    Map<Object, Object> map;
    public Object[] getArr() {
        return arr;
    }
    public void setArr(Object[] arr) {
        this.arr = arr;
    }
    public List<Object> getList() {
        return list;
    }
    public void setList(List<Object> list) {
        this.list = list;
    }
    public Map<Object, Object> getMap() {
        return map;
    }
    public void setMap(Map<Object, Object> map) {
        this.map = map;
    }
    @Override
    public String toString() {
        return "ComplexTypeObject [arr=" + Arrays.toString(arr) + ", list=" + list + ", map=" + map + "]\n";
    }
}
```

```
<bean name="ct" class="com.zhiyou100.di.ComplexTypeObject">
  <property name="arr">
    <array>
      <value>休息</value>
      <value>开会</value>
      <ref bean="user4"/>
    </array>
  </property>
  <property name="list">
    <list>
      <value>北京</value>
      <value>上海</value>
      <ref bean="car"/>
    </list>
  </property>
  <property name="map">
    <map>
      <entry key="driverClass" value="com.mysql.jdbc.Driver"/>
      <entry key="car" value-ref="car"/>
      <entry key-ref="user4" value-ref="car"/>
    </map>
  </property>
</bean>
```

Spring 的分模块配置文件的开发

有时为了简介和便于阅读,会将配置文件按照不同的功能模块分成多个xml文件,在进行引入的时候可以使用标签 `<import resource="相对路径">` 进行引入,注意是相对路径,相对于当前引入到的xml文件的路径,不建议使用"/"开头

```
<beans>
  <import resource="services.xml"/>
  <import resource="resources/messageSource.xml"/>
  <import resource="/resources/themeSource.xml"/>

  <bean id="bean1" class="..." />
  <bean id="bean2" class="..." />
</beans>
```

In the preceding example, external bean definitions are loaded from three files: `services.xml`, `messageSource.xml`, and `themeSource.xml`. All location paths are relative to the definition file doing the importing, so `services.xml` must be in the same directory or classpath location as the file doing the importing, while `messageSource.xml` and `themeSource.xml` must be in a `resources` location below the location of the importing file. As you can see, a leading slash is ignored, but given that these paths are relative, it is better form not to use the slash at all. The contents of the files being imported, including the top level `<beans/>` element, must be valid XML bean definitions according to the Spring Schema.

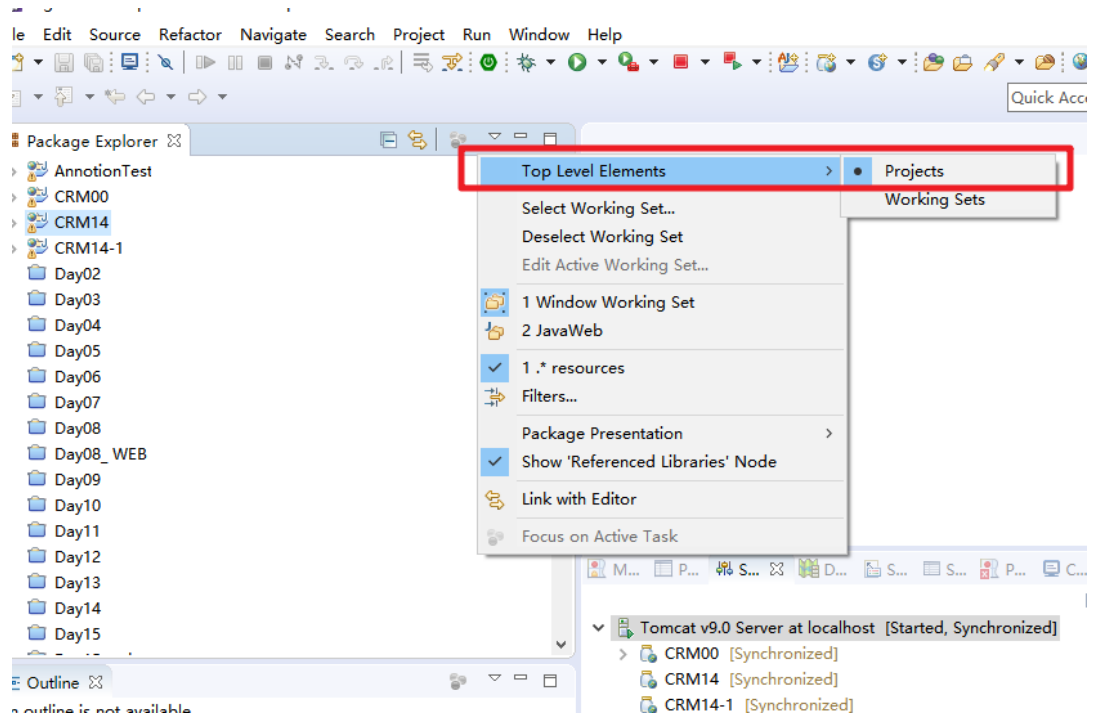
- 注意是相对路径,如果a.xml中引用b.xml,那么路径应该从a.xml文件所在的文件夹为起始路径

CRM改为让Spring进行管理

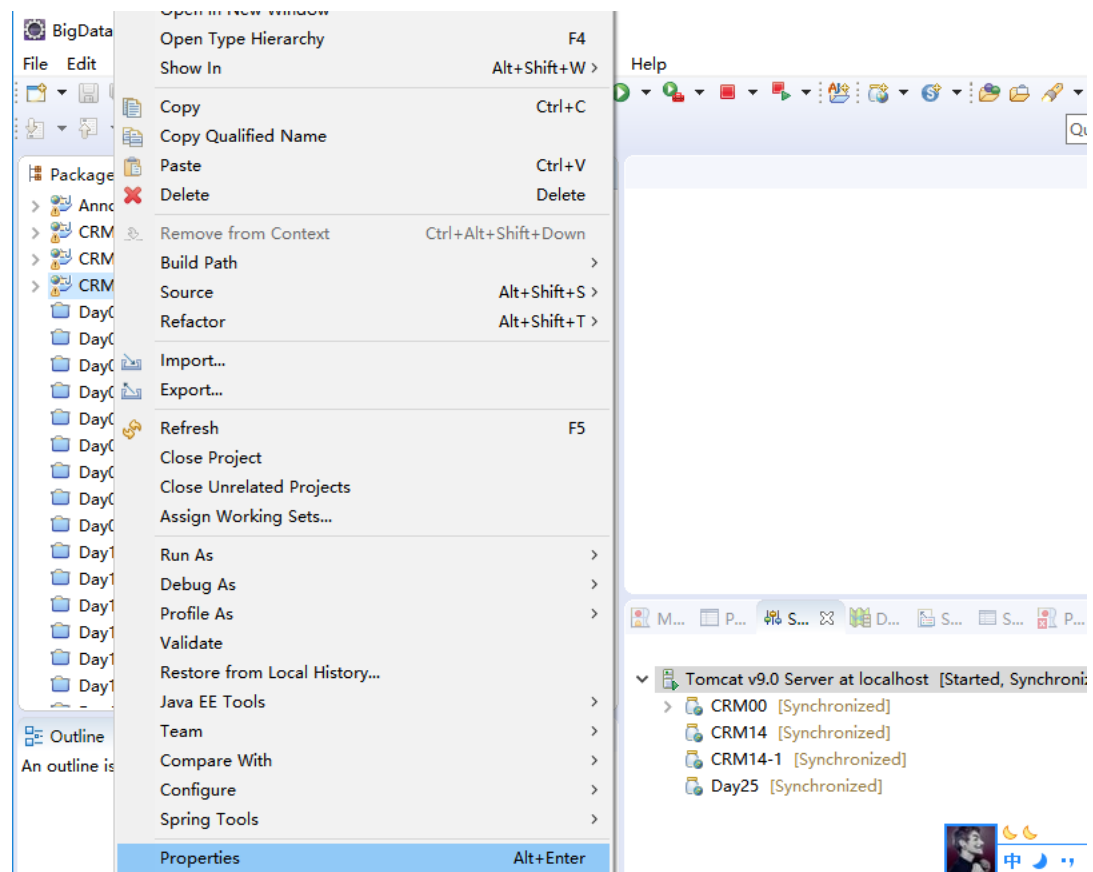
练习IOC和DI

- eclipse中复制项目练习IOC和DI

1.



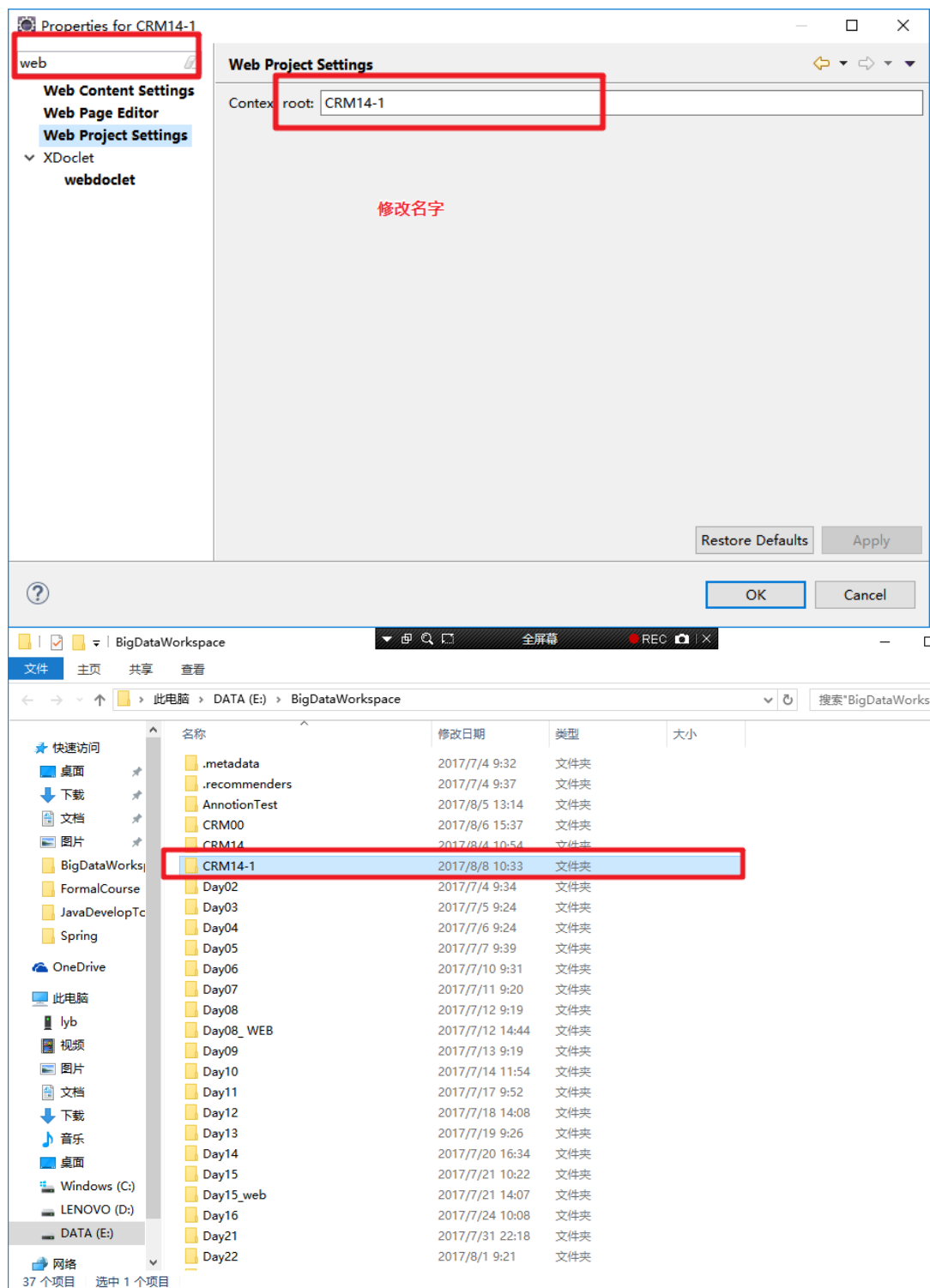
2. 对原来项目复制(ctrl+c)粘贴(ctrl+v)改名字



3.

新项目右键Properties

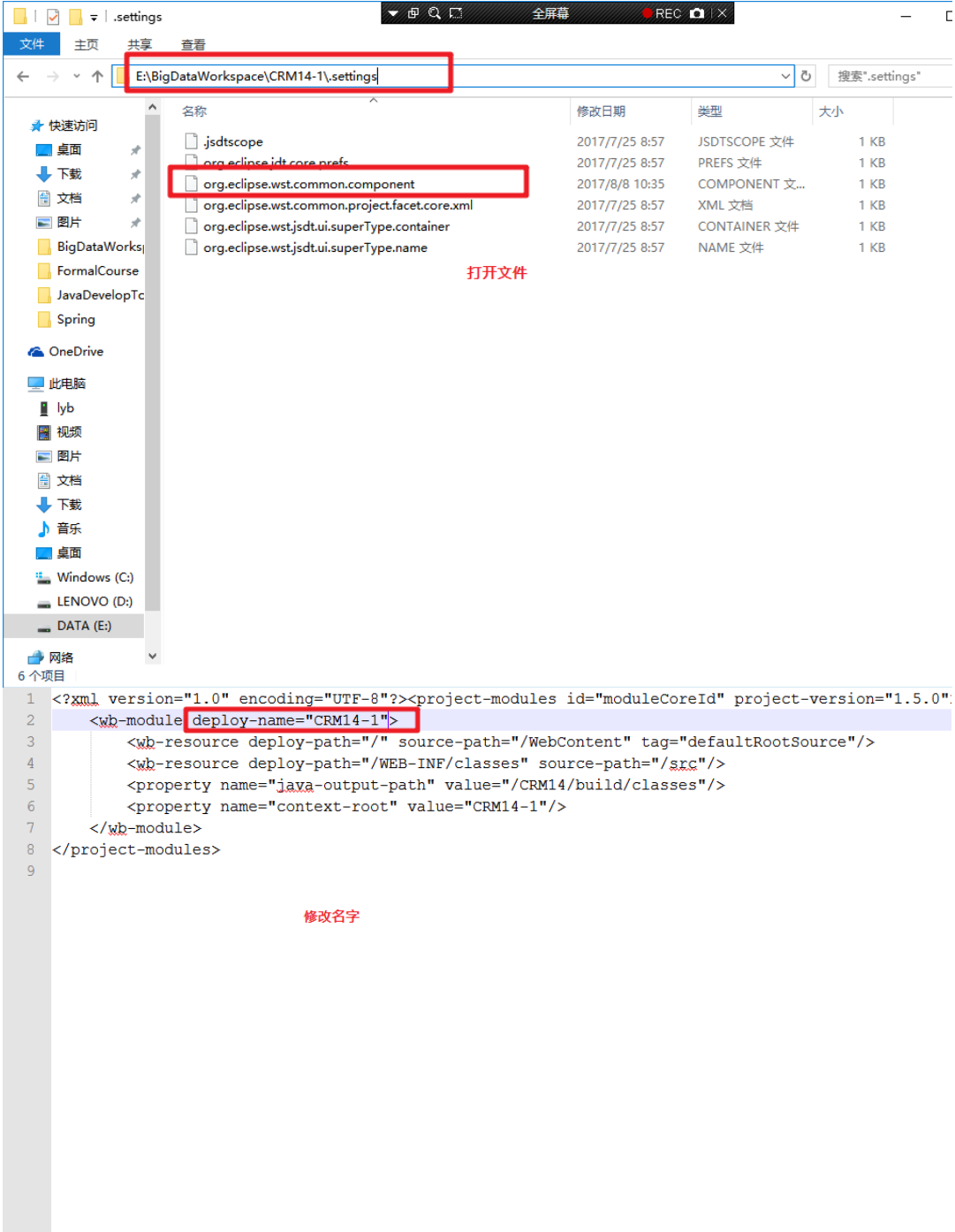
4.



5.

打开工作空间中的新项目

6.



The screenshot shows a Windows File Explorer window with the address bar set to `E:\BigDataWorkspace\CRM14-1\.settings`. The file list contains several files, with `org.eclipse.wst.common.component` highlighted. Below the file list, an XML snippet is displayed, showing the `deploy-name="CRM14-1"` attribute highlighted within a `<wb-module>` tag.

打开文件

```

1 <?xml version="1.0" encoding="UTF-8"?><project-modules id="moduleCoreId" project-version="1.5.0">
2   <wb-module deploy-name="CRM14-1">
3     <wb-resource deploy-path="/" source-path="/WebContent" tag="defaultRootSource"/>
4     <wb-resource deploy-path="/WEB-INF/classes" source-path="/src"/>
5     <property name="java-output-path" value="/CRM14/build/classes"/>
6     <property name="context-root" value="CRM14-1"/>
7   </wb-module>
8 </project-modules>
9

```

修改名字

7.

- 导包4+2+1(spring包中的web.jar)
- 配置applicationContext
- 在web.xml中配置Spring,让其随着项目的启动而启动
 - 配置ContextLoadListener
 - 配置全局初始化参数,参数的名称是 contextConfigLocation,参数的值为 classpath:application.xml,必须要添加classpath

```
<!--
```

spring中配置了一个ServletContextListener的实现类,会在服务器启动的时候创建,用来创建容器,当容器创建的时候

会读取全局的初始化参数,会通过一个名字找到对应的路径,我们要配置name和value,name就是他通过什么名字去找applicationContext.xml

value就是applicationContext.xml文件路径

一旦服务器启动,容器对象就会被创建,创建完成以后,spring把容器对象放在ServletContext中,我们需要从域中获取,并且spring给我们

提供了相应的方法

```
-->
```

```
<context-param>
```

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

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

```
</context-param>
```

```
<listener>
```

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

```
</listener>
```

- 在需要spring容器的时候书写代码

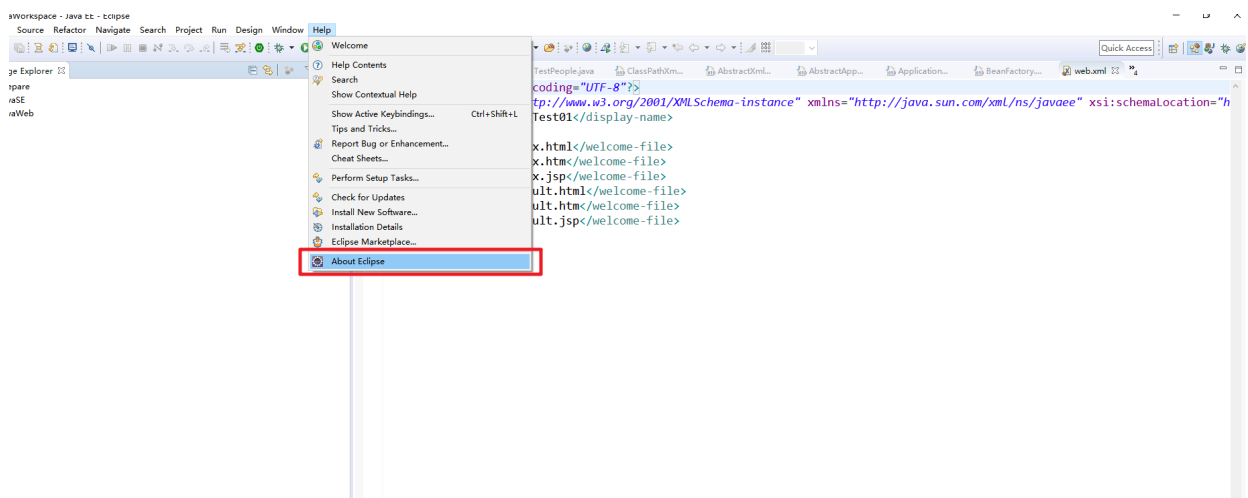
因为在服务器启动的时候, spring容器被创建出来, 并且放在application域中, 所以方法的参数是servletContext的对象, 通过此对象获取到spring容器

```
// 获取容器
WebApplicationContext wa = WebApplicationContext
    ntextUtils.getWebApplicationContext(this.get
        ServletContext());
// 获取UserService
UserService us = (UserService)wa.getBean("us
    erService");
```

安装STS

spring tool suite是spring开源的基于eclipse的插件, 能够方便我们的spring开发

- 查看eclipse的版本





- 下载地址<https://spring.io/tools/sts/all>,找到对应版本的sts插件,进行安装

