安装的jar包

下载jar
http://maven.springframework.org/release/org/springframework/spring/
spring-framework-4.3.9.RELEASE-dist.zip

> spring-aop-4.3.9.RELEASE
>
> spring-beans-4.3.9.RELEASE
>
> spring-context-4.3.9.RELEASE
>
> spring-core-4.3.9.RELEASE
>
> spring-expression-4.3.9.RELEASE
>
> commons-logging-1.1.1

准备

1.安装sts

2.链接对象

```
package nuc.wcy.entiy;

import nuc.wcy.newinstance.HtmlCourse;
import nuc.wcy.newinstance.ICourse;
import nuc.wcy.newinstance.JavaCourse;

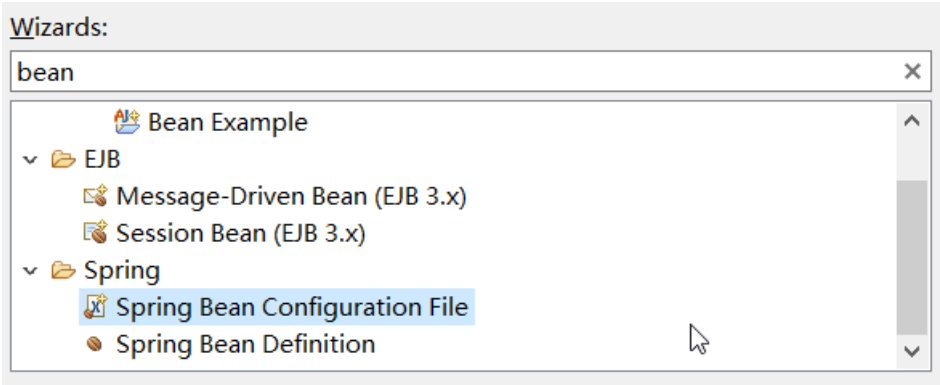
public class Student {
    private int stuno;
    private String stuname;
    private int stuage;

    public Student() {
        super();
    }
    public Student(int stuno, String stuname, int stuage) {
        super();
        this.stuno = stuno;
```

```
        this.stuname = stuname;
        this.stuage = stuage;
    }
    public int getStuno() {
        return stuno;
    }
    public void setStuno(int stuno) {
        this.stuno = stuno;
    }
    public String getStuname() {
        return stuname;
    }
    public void setStuname(String stuname) {
        this.stuname = stuname;
    }
    public int getStuage() {
        return stuage;
    }
    public void setStuage(int stuage) {
        this.stuage = stuage;
    }
    @Override
    public String toString() {
        return "Student [stuno=" + stuno + ", stuname=" +
stuname + ", stuage=" + stuage + "]";
    }
}
```

3.配置xml

File -> new file-> other




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

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

    <bean id="student" class="nuc.wcy.entiy.Student">
        <!--保证name与class对应-->
        <property name="stuno" value="2"></property>
        <property name="stuname" value="ls"></property>
        <property name="stuage" value="24"></property>
    </bean>

</beans>
```

3.测试

```
package nuc.wcy.entiy;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import nuc.wcy.newinstance.ICourse;

public class Test {
    public static void main(String[] args) {
        hh();
        learnCourseWithFactory();
    }

    public static void hh() {
        ApplicationContext content = new
ClassPathXmlApplicationContext("application.xml");
        Student student = (Student)content.getBean("student");
        System.out.println(student);
    }
}
```

开始!

赋值必须有无参构造!!!

IOC(超级工厂)

XML

1.IOC容器（工厂）

```
<bean id="javaCourse" class="nuc.wcy.newinstance.JavaCourse">
</bean>
    <bean id="htmlCourse"
class="nuc.wcy.newinstance.HtmlCourse"></bean>
```

2.引用工厂内容

```
public void learnJava() {
//      ICourse course = new HtmlCourse();
//      course根据naem获取相应的课程


        //直接在ioc容器获取
        //2.直接在ioc中取
        ApplicationContext content = new
ClassPathXmlApplicationContext("application.xml");
        ICourse course =
(ICourse)content.getBean("javaCourse");
        course.learn();
    }
```

引用类:

> teacher:

```
private String name;
private int age;
```

course：

```
private String courseName;
    private int courseHour;
    private Teacher teacher;
```

```
<bean id="teacher" class="nuc.wcy.entiy.Teacher">
        <property name="name" value = "zs"></property>
        <property name="age" value="18"></property>
    </bean>
    <bean id="course" class="nuc.wcy.entiy.Course">
        <property name="courseName" value="java"></property>
        <property name="courseHour" value="19"></property>
        <property name="teacher" ref="teacher"></property>
    </bean>
```

自动匹配

`autowirte="buName"` 使用引用类目之间自动匹配（根据id查找）

```
<bean id="course" class="nuc.wcy.entiy.Course"
autowirte="buName" >
        <property name="courseName" value="java"></property>
        <property name="courseHour" value="19"></property>
    </bean>
```

注入（赋值）

set方式的依赖注入：

- 赋值默认使用set方法；依赖注入底层的方式实现

- ```
  <property name="name" value = "zs"></property>
          <property name="age" value="18"></property>
  ```

  2.构造方法

  ```
  <constructor-arg value="24" index="1"></constructor-arg>
          <constructor-arg value="ls" index="0">
  </constructor-arg>
  ```

  ```
  <constructor-arg value="24" type="int" name="age">
  </constructor-arg>
          <constructor-arg value="ls" type="String"
  name="name"></constructor-arg>
  ```

  3.p方法

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

  ```
  <bean id="teacher" class="nuc.wcy.entiy.Teacher"
  p:name="ls" p:age="12"></bean>
  ```

数组的注入

```
<bean id="all" class="nuc.wcy.entiy.AllColectionType">
        <property name="list">
        <list>
            <value>足球</value>
            <value>蓝球</value>
            <value>乒乓球</value>
        </list>
        </property>
        <property name="array">
        <array>
            <value>足球</value>
            <value>蓝球</value>
            <value>乒乓球</value>
        </array>
        </property>
        <property name="set">
        <set>
```
```

```
            <value>足球1</value>
            <value>蓝球1</value>
            <value>乒乓球1</value>
        </set>
        </property>
        <property name="map">
        <map>
            <entry>
                <key>
                    <value>foot</value>
                </key>
                <value>足球2</value>
            </entry>
            <entry>
                <key>
                    <value>bask</value>
                </key>
                <value>蓝球2</value>
            </entry>
            <entry>
                <key>
                    <value>ping</value>
                </key>
                <value>乒乓球2</value>
            </entry>
        </map>
        </property>
    </bean>
```

> 同理平时的属性配置也可以这么做

```
    <bean id="teacher2" class="nuc.wcy.entiy.Teacher">
        <property name="name">
            <value type="java.lang.String">sdlk<!
[CDATA[<>$]]>afj</value>
        </property>
        <property name="age">
            <value type="int">18</value>
        </property>
        <!--当空时用null，“”等方法-->
    </bean>
```

> 转意用$]]>或转移码

自动装配

<bean ... class="org.lanqiao.entity.Course"
autowire="byName|byType|constructor|no" > byName本质是byId
byName: 自动寻找：其他bean的id值=该Course类的属性名
byType: 其他bean的类型(class) 是否与 该Course类的ref属性类型一致
（注意，此种方式 必须满足：当前Ioc容器中 只能有一个Bean满足条件 ）
constructor： 其他bean的类型(class) 是否与 该Course类的构造方法参数 的类型一致；此种方式的本质就是byType

可以在头文件中 一次性将该ioc容器的所有bean 统一设置成自动装配：
<beans xmlns="http://www.springframework.org/schema/beans"
...
default-autowire="byName">

自动装配虽然可以减少代码量，但是会降低程序的可读性，使用时需要谨慎。

注解（在java中自动导入）

使用注解定义bean：通过注解的形式 将bean以及相应的属性值 放入ioc容器

<context:component-scan base-package="org.lanqiao.dao">
</context:component-scan>Spring在启动的时候，会根据base-package在 该包中扫描所有类，查找这些类是否有注解 `@Component("studentDao")`，如果有，则将该类 加入spring Ioc容器。

@Component细化：

dao层注解： `@Repository`
service层注解： `@Service`
控制器层注解： `@Controller`

`@Autowired` 自动匹配数值

properties

```
#bean.properties
accountService = com.wcy.service.impl.AccountServiceImpl
accountDao = com.wcy.dao.impl.AccountDaoImpl
```

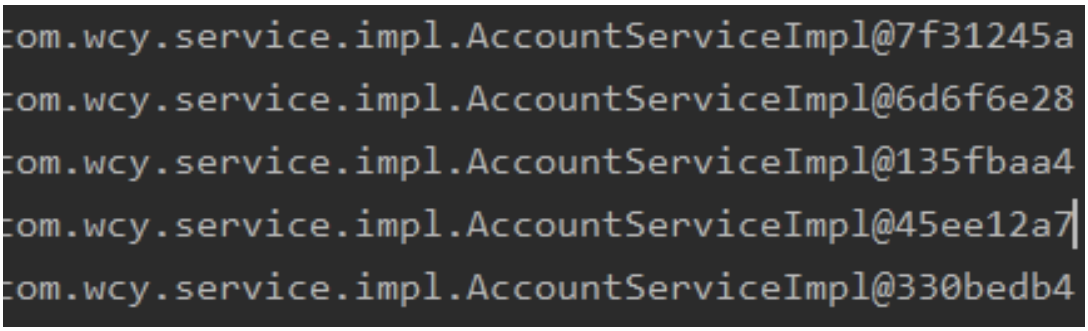
```
public class BeanFactory {
    //定义一个properties的对象
    static Properties props;

    static {
        try {
            //实例化对象
            props =  new Properties();
            //获取文件的流对象
            InputStream in =
BeanFactory.class.getClassLoader().getResourceAsStream("bean.p
roperties");
            props.load(in);
        } catch (IOException e) {
            throw new ExceptionInInitializerError("初始化
properties对象失败");
        }
    }

    /**
     * 根据Bean名称定义Bean对象
     * @param beanName
     * @return
     */
    public static Object getBean(String beanName){
        Object bean=null;
        //获取真正的类
        String beamPath = props.getProperty(beanName);
//        System.out.println(beamPath);
        try {
            bean = Class.forName(beamPath).newInstance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
```

```
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return bean;
    }
}
```

多例



单例:

```
public class BeanFactory {
    //定义一个properties的对象
    static Properties props;

    //定义一个MAp,用于存放我们要创建的对象，我们把它称之为容器
    private  static Map<String,Object> beans;
    static {
        try {
            //实例化对象
            props =  new Properties();
            //获取文件的六对象
            InputStream in =
BeanFactory.class.getClassLoader().getResourceAsStream("bean.p
roperties");
            props.load(in);
            //实例化容器
            beans = new HashMap<String, Object>();
            //去获取配置文件中所有的类
            Enumeration keys =  props.keys();
            //遍历枚举
            while (keys.hasMoreElements()){
                String key = keys.nextElement().toString();
                //根据key获取value
```

```
                String beanPath = props.getProperty(key);
                //反射创键对象
                Object value =
Class.forName(beanPath).newInstance();
                //把key和value存入容器中
                beans.put(key,value);
            }
        }  catch (Exception e) {
            throw new ExceptionInInitializerError("初始化
properties对象失败");
        }
    }

    public static Object getBean(String beanName){
        return beans.get(beanName);
    }
```

Bean

Bean的三种创建方式

```
        ApplicationContext ac = new
ClassPathXmlApplicationContext("bean.xml");
        IAccountService as1 =
(IAccountService)ac.getBean("accountService");
        System.out.println(as1);
```

要有对应的无参构造

1.默认方式创建

```
<!--    把对象的创建交给spring来管理-->
    <!-- 第一种方式：使用默认方式配置 -->
<!--    <bean id="accountService"
class="com.wcy.service.impl.AccountServiceImpl"></bean>-->
```

2.使用普通工厂创建

```
public class InstanceFactory {
    public IAccountService getAccountService(){
        return new AccountServiceImpl();
    }
}
```

```
<!--    2.使用普通工厂创建-->
<!--    <bean id="instanceFactory"
class="com.wcy.factory.InstanceFactory"></bean>
    <bean id="accountService" factory-bean="instanceFactory"
factory-method="getAccountService"></bean>-->
```

3.使用静态工厂创建

```
public class StaticFactory {
    public static IAccountService getAccounService(){
        return new AccountServiceImpl();
    }
}
```

```
<!--    3.使用静态工厂创建-->
    <bean id="accountService"
class="com.wcy.factory.StaticFactory" factory-
method="getAccounService"></bean>
```

构造函数的注入

要有对应的有参构造

使用的标签：`constructor-arg`
标签出现的位置：bean内部
标签的属性
`type`:根据类型赋值
`index`:根据索引复制
`name`:根据名称复制
以上用于指定给构造函数中哪个参数数值

`value`：用于进本类型的赋值
`ref`：用于指定其他的ban对象

```
bean id="accountService"
class="com.wcy.service.impl.AccountServiceImpl">
        <constructor-arg name="name" value="test">
</constructor-arg>
        <constructor-arg name="age" value="18"></constructor-
arg>
        <constructor-arg name="brithday" ref="now">
</constructor-arg>
    </bean>
    <bean id="now" class="java.util.Date"></bean>
```

SET方法注入

涉及的的标签：property

出现位置：bean标签内部

标签的属性
`type`:根据类型赋值
`index`:根据索引复制
`name`:根据名称复制
以上用于指定给构造函数中哪个参数数值
`value`：用于进本类型的赋值
`ref`：用于指定其他的ban对象

```
    <bean id="accountService2"
class="com.wcy.service.impl.AccountServiceImpl">
        <property name="name" value="TEST"></property>
        <property name="age" value="18"></property>
        <property name="brithday" ref="now"></property>
    </bean>
```

数组的注入

```
private String[] myStrs;
    private List<String> myList;
    private Set<String> mySet;
    private Map<String,String> myMap;
    private Properties myProps;
```

复杂类型的注入

- set,array,list
- map,property

可以混用

```
    <bean id="accountService3"
class="com.wcy.service.impl.AccountServiceImpl3">
        <property name="myStrs">
            <array>
                <value>AAA</value>
                <value>BBB</value>
                <value>CCC</value>
            </array>
        </property>
        <property name="myList">
            <list>
                <value>AAA</value>
                <value>BBB</value>
                <value>CCC</value>
            </list>
        </property>
        <property name="mySet">
            <set>
                <value>AAA</value>
                <value>BBB</value>
                <value>CCC</value>
            </set>
        </property>
        <property name="myMap">
            <map>
                <entry key="testA" value="AAA"></entry>
                <entry key="testB">
                    <value>BB</value>
                </entry>
```

```
            </map>
        </property>
        <property name="myProps">
        <props>
            <prop key="testC">ccc</prop>
            <prop key="testD">ddd</prop>
        </props>
        </property>
    </bean>
```

注解（在java中自动导入）

使用注解定义bean：通过注解的形式 将bean以及相应的属性值 放入ioc容器

```
<?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"

xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
    <!--  告知spring在创建容器时要扫描的包-->
    <context:component-scan base-package="com.wcy">
</context:component-scan>
</beans>
```

</context:component-scan>Spring在启动的时候，会根据base-package在 该包中扫描所有类，查找这些类是否有注解`@Component("studentDao")`，如果有，则将该类 加入spring Ioc容器。

@Component细化：

dao层注解：`@Repository`
service层注解：`@Service`
控制器层注解：`@Controller`

`@Autowired`自动匹配数值

> 账户的业务层实现类
>
> 曾经XML的配置：
>
> <property name="" value="" | ref="">
>
> 用于创建对象的
> 他们的作用就和在XML配置文件中编写一个标签实现的功能是一样的
> Component:
> 作用：用于把当前类对象存入spring容器中
> 属性：
> value：用于指定bean的id。当我们不写时，它的默认值是当前类名，且首字母改小写。
> Controller：一般用在表现层
> Service：一般用在业务层
> Repository：一般用在持久层
> 以上三个注解他们的作用和属性与Component是一模一样。
> 他们三个是spring框架为我们提供明确的三层使用的注解，使我们的三层对象更加清晰
>
> 用于注入数据的
> 他们的作用就和在xml配置文件中的bean标签中写一个标签的作用是一样的
> Autowired:
> 作用：自动按照类型注入。只要容器中有唯一的一个bean对象类型和要注入的变量类型匹配，就可以注入成功
> 如果ioc容器中没有任何bean的类型和要注入的变量类型匹配，则报错。
> 如果Ioc容器中有多个类型匹配时：
> 出现位置：
> 可以是变量上，也可以是方法上
> 细节：
> 在使用注解注入时，set方法就不是必须的了。

Qualifier:
作用：在按照类中注入的基础之上再按照名称注入。它在给类成员注入时不能单独使用。但是在给方法参数注入时可以（稍后我们讲）
属性：
value：用于指定注入bean的id。
Resource
作用：直接按照bean的id注入。它可以独立使用
属性：
name：用于指定bean的id。
以上三个注入都只能注入其他bean类型的数据，而基本类型和String类型无法使用上述注解实现。
另外，集合类型的注入只能通过XML来实现。

Value
作用：用于注入基本类型和String类型的数据
属性：
value：用于指定数据的值。它可以使用spring中SpEL(也就是spring的el表达式)
SpEL的写法：${表达式}

用于改变作用范围的
他们的作用就和在bean标签中使用scope属性实现的功能是一样的
Scope
作用：用于指定bean的作用范围
属性：
value：指定范围的取值。常用取值：singleton prototype

和生命周期相关 了解
他们的作用就和在bean标签中使用init-method和destroy-methode的作用是一样的
PreDestroy
作用：用于指定销毁方法
PostConstruct
作用：用于指定初始化方法

生命周期

```
<!-- bean的作用范围
        bean标签的scope属性：
            作用：用于指定bean的作用范围
            取值：
            singleton：单例模式
            prototype：多例
            request：作用于web应用的请求范围
            session：作用于web应用的会化范围
            global-session：作用于集群环境的绘画范围（全局绘画范围），当不是集群环境时
        bean标签的生命周期
            初始化
            销毁
     -->
    <bean id="accountService" class="com.wcy.service.impl.AccountServiceImpl" scope="singleton" init-method="init" destroy-method="destroy"></bean>
```

MS合体

1.配置tx

jar：

> commons-dbcp.jar
>
> ojdbc.jar
>
> commons-pool.jar
>
> mybatis-spring.jar spring-tx.jar spring-jdbc.jar spring-expression.jar spring-context-support.jar spring-core.jar spring-context.jar spring-beans.jar spring-aop.jar spring-web.jar commons-logging.jar commons-dbcp.jar ojdbc.jar mybatis.jar log4j.jar commons-pool.jar

```
    <!-- 配置数据库相关-事务 -->
```

```
    <!-- 配置数据库相关 -->
    <bean id="dataSource"
class="org.apache.commons.dbcp2.BasicDataSource">
        <property name="driverClassName"
value="com.mysql.jdbc.Driver"></property>
        <property name="url" value="127.0.0.1:3306/test">
</property>
        <property name="username" value="root"></property>
        <property name="password" value="654321"></property>
    </bean>
    <!-- 配置事务管理器txManager -->
    <bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransacti
onManager">
        <property name="dataSource" ref="dataSource">
</property>
    </bean>
    <!-- 增加对事物的支持 -->
    <tx:annotation-driven transaction-manager="txManager"/>
```

AOP

通过**预编译**方式和运行期动态代理实现程序功能的统一维护的一种技术

add()

log()

//xml加入aop

6-10集重看

WEB合体

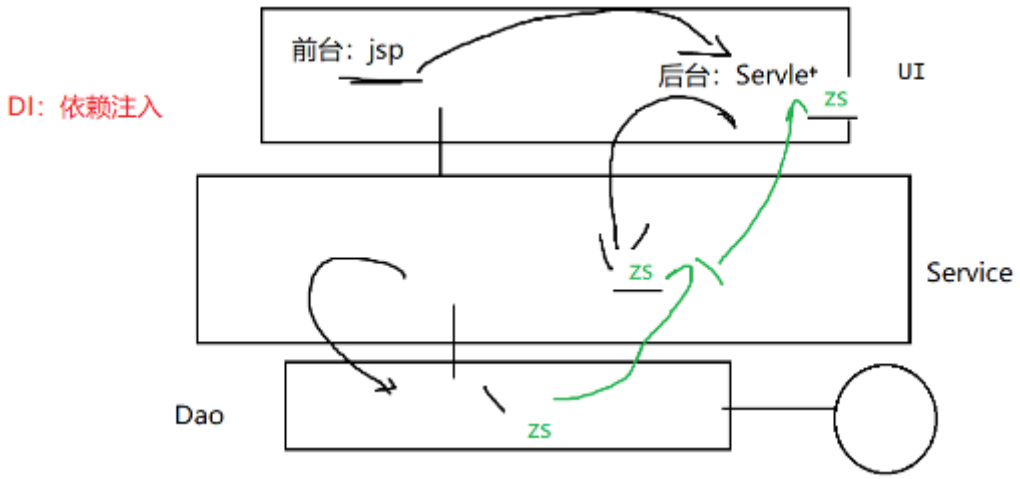
配置文件jar和capplicationContext.xml要放在 ***/WebContent/WEB-INF/lib*** 中

web项目启动时，会自动加载web.xml，因此需要在web.xml中加载 监听器（ioc容器初始化）。

Web项目启动时，启动实例化Ioc容器：

```
<!-- 指定 Ioc容器（applicationContext.xml）的位置-->
  <context-param>
        <!--  监听器的父类ContextLoader中有一个属性
contextConfigLocation，该属性值 保存着 容器配置文件
applicationContext.xml的位置 -->
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-
value>
  </context-param>
  <listener>
    <!-- 配置spring-web.jar提供的监听器，此监听器 可以在服务器启动
时 初始化Ioc容器。
        初始化Ioc容器（applicationContext.xml） ，
            1.告诉监听器 此容器的位置：context-param
            2.默认约定的位置    :WEB-INF/applicationContext.xml
     -->
    <listener-
class>org.springframework.web.context.ContextLoaderListener</l
istener-class>
  </listener>
```

在web中的对象与SpringIoc不相同



解决方法：

在"init()"中给Servlet的值赋值