luckyStar总结

Spring IOC 与 AOP 配置与应用

注：第一个 spring 用完整的代码其他都在这上面改就把要该的代码写上了

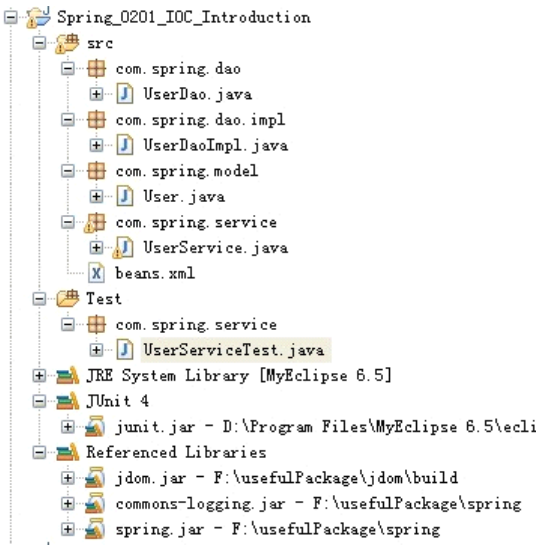
1.FAQ:不给提示：

a) window-preferences-myeclipse-xml-xml catalog
b) User Specified Entries-add
   i. Location:
      D:\share\0900_Spring\soft\spring-framework-2.5.6\dist\resource\spring-beans-2.5.xsd
   ii. URI:
      file:///D:/share/0900_Spring/soft/spring_framework-2.5.6/dist/resources/spring-beans-2.5.xsd
   iii. keyType: Schema Location
   iv . key:
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

2. 注入类型

a)setter（重要）
b)构造方法（可以忘记）
c)接口注入(可以忘记)

下面详细说一下 setter,和构造函数，剩下的方法不常用自己查 spring 文档
这是项目所建立的文件和用到得包



从上到下文件代码：

UserDao.java

```
package com.spring.dao;

import com.spring.model.User;

public interface UserDao {
    public void save(User u);
}
```

++

UserDaoImpl.java

```
package com.spring.dao.impl;

import com.spring.dao.UserDao;
import com.spring.model.User;

public class UserDaoImpl implements UserDao {

    public void save(User u) {
        System.out.println("save user!");

    }

}
```

++

User.java

```
package com.spring.model;

public class User {
    private String username;
    private String password;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
```

```
    }

}
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
UserService.java

package com.spring.service;

import com.spring.dao.UserDao;
import com.spring.dao.impl.UserDaoImpl;
import com.spring.model.User;

public class UserService {
    private UserDao userDao;


    public UserDao getUserDao() {
        return userDao;
    }


    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }


    public void add(User user){
        userDao.save(user);
    }
}
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Beans.xml //这个是 Spring 的配置文件
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="u" class="com.spring.dao.impl.UserDaoImpl"></bean>

    <bean id="userService" class="com.spring.service.UserService">
//这个实现依赖注入
        <property name="userDao" ref="u"></property>
```

```
    </bean>
</beans>
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
UserServiceTest.java   //这个是测试类
package com.spring.service;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContex
t;

import com.spring.model.User;


public class UserServiceTest {

    @Test
    public void testAdd() throws Exception {
        //new 一个 bean 工厂
        ApplicationContext factory = new
ClassPathXmlApplicationContext("beans.xml");
        UserService service =
(UserService)factory.getBean("userService");

        User u = new User();
        u.setUsername("testName");
        u.setPassword("testPassword");
        service.add(u);
    }

}
```

如果是构造函数的方式：

在 UserService.java 中加上构造函数：

```
    public UserService(UserDao userDao) {
        super();
        this.userDao = userDao;
    }
```

在 beans.xml 中这样写：

```
<?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-2.5.xsd">

    <bean id="u" class="com.spring.dao.impl.UserDaoImpl"></bean>

    <bean id="userService" class="com.spring.service.UserService">
        <!--
        <property name="userDao" ref="u"></property>
         -->
        <constructor-arg>
            <ref bean="u"/>
        </constructor-arg>
    </bean>
</beans>
```

其他不变。

++

3.id vs.name

a)可以互相替换 name 可以用特殊字符

```
<bean name="u" class="com.spring.dao.impl.UserDaoImpl"></bean>

<bean id="userService" class="com.spring.service.UserService">
```

4.简单属性注入

a)<property value=...>

UserDaoImpl.java 中这样写：加入两个属性 daoId; daoStatus;

```
package com.spring.dao.impl;

import com.spring.dao.UserDao;
import com.spring.model.User;

public class UserDaoImpl implements UserDao {

    private int daoId;
    private String daoStatus;


    public int getDaoId() {
        return daoId;
```

```
    }

    public void setDaoId(int daoId) {
        this.daoId = daoId;
    }

    public String getDaoStatus() {
        return daoStatus;
    }

    public void setDaoStatus(String daoStatus) {
        this.daoStatus = daoStatus;
    }

    public void save(User u) {
        System.out.println("save user!");

    }

}
```

++

在 bean.xml 中可以对属性简单赋值 ：

```
<?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-2.5.xsd">
<!—对属性简单赋值-->
    <bean name="u" class="com.spring.dao.impl.UserDaoImpl">
        <property name="daoId" value="8"></property>
        <property name="daoStatus" value="good"></property>
    </bean>

    <bean id="userService" class="com.spring.service.UserService">
        <!--
        <property name="userDao" ref="u"></property>
         -->
        <constructor-arg>
            <ref bean="u"/>
```

```
        </constructor-arg>
    </bean>
</beans>
```

5.<bean 中的 scope 属性

a)singleton 单例

b)prototype 每次创建新的对象　　struts 的 action 中经常用

```
<!--
<bean id="userService" class="com.spring.service.UserService" scope="singleton">
 -->
<bean id="userService" class="com.spring.service.UserService" scope="prototype">
```

6.集合注入

a)很少用，不重要！参考程序

UserDaoImpl.java 中这样写：加入三个集合 Set<String> sets；List<String> lists; Map<String,String> maps;

```
package com.spring.dao.impl;


import java.util.List;
import java.util.Map;
import java.util.Set;

import com.spring.dao.UserDao;
import com.spring.model.User;

public class UserDaoImpl implements UserDao {

    private Set<String> sets;
    private List<String> lists;
    private Map<String,String>  maps;


    public Set<String> getSets() {
        return sets;
    }


    public void setSets(Set<String> sets) {
        this.sets = sets;
```

```
    }

    public List<String> getLists() {
        return lists;
    }

    public void setLists(List<String> lists) {
        this.lists = lists;
    }

    public Map<String, String> getMaps() {
        return maps;
    }

    public void setMaps(Map<String, String> maps) {
        this.maps = maps;
    }

    public void save(User u) {
        System.out.println("save user!");

    }
    @Override
    public String toString() {

        return "sets size"+sets.size()+"| lists size"+lists.size()+"|
maps size"+maps.size();
    }

}
```

++

Bean.xml 中对集合进行注入

```
<?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-2.5.xsd">

    <bean name="u" class="com.spring.dao.impl.UserDaoImpl">
```

```
        <property name="sets">
            <set>
                <value>1</value>
                <value>2</value>
            </set>
        </property>
        <property name="lists">
            <list>
                <value>1</value>
                <value>2</value>
                <value>3</value>
            </list>
        </property>
        <property name="maps">
            <map>
                <entry key="1" value="1"></entry>
                <entry key="2" value="2"></entry>
                <entry key="3" value="3"></entry>
                <entry key="4" value="4"></entry>
                <entry key="5" value="5"></entry>
            </map>
        </property>
    </bean>
    <!--
        <bean id="userService" class="com.spring.service.UserService"
scope="singleton">
    -->
    <bean id="userService" class="com.spring.service.UserService"
        scope="prototype">
        <!--
            <property name="userDao" ref="u"></property>
        -->
        <constructor-arg>
            <ref bean="u" />
        </constructor-arg>
    </bean>

</beans>
```

7.自动装配

a)byName

b)byType

c)如果所有的 bean 都用同一种，可以使用 beans 的属性:default-autowire

bean.xml中<bean id="userService" class="com.spring.service.UserService"

```
scope="prototype" autowire="byType">
```

autowire 设置 byName 还是 byType 实现自动装配：

```
<?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-2.5.xsd">
    <!--
    <bean name="userDao1" class="com.spring.dao.impl.UserDaoImpl">
        <property name="daoId" value="1"></property>
    </bean>
     -->
    <bean name="userDao2" class="com.spring.dao.impl.UserDaoImpl">
        <property name="daoId" value="2"></property>
    </bean>
    <!--
    <bean id="userService" class="com.spring.service.UserService"
scope="singleton">
     -->
     <!-- autowire="byType" autowire="byName"他们的区别-->
    <bean id="userService" class="com.spring.service.UserService"
scope="prototype" autowire="byType">
    </bean>

</beans>
```

8.生命周期：

a)lazy-init(忘了吧)//当 bean 非常多启动慢的情况下在某个 bean 下写上 lazy-init=true 当 context 初始化的时候，这个 bean 不进行初始化，什么时候用到了什么时候初始化

b)init-method destroy-methd 不要和 prototype 一起用(了解)

UserService.java

```
import com.spring.dao.UserDao;
import com.spring.dao.impl.UserDaoImpl;
import com.spring.model.User;

public class UserService {
    private UserDao userDao;

    public void init(){
        System.out.println("init");
    }
```

```
    public UserDao getUserDao() {
        return userDao;
    }


    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }


    public void add(User user){
        userDao.save(user);
    }


    public UserService(UserDao userDao) {
        super();
        this.userDao = userDao;
    }

    public void destroy(){
        System.out.println("destroy");
    }

}
```

++

Beans.xml

```
<?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-2.5.xsd">

    <bean id="u" class="com.spring.dao.impl.UserDaoImpl"></bean>

    <bean id="userService" class="com.spring.service.UserService"
init-method="init" destroy-method="destroy">
        <!--
        <property name="userDao" ref="u"></property>
         -->
        <constructor-arg>
            <ref bean="u"/>
```

```
        </constructor-arg>
    </bean>
</beans>
```

9.Annotation 第一步：

a)修改 xml 文件，参考文档<context:annotation-config/>
beans.xml 头文件变化和需要加入的<context:annotation-config/>标签

```
<?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-2.5.xsd
        http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd"
>
        <context:annotation-config/>
    </beans>
```

10.@Autowired//自动装配

a)默认按类型 by type
b)如果想用 byName，使用@Qulifier//使用某个名字的 bean
c)写在 private field(第三种注入形式)(不建议，破坏封装)
d)如果写在 set 上，@qualifier 需要写在参数上

UserService.java

```
package com.spring.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

import com.spring.dao.UserDao;
import com.spring.model.User;

public class UserService {
    private UserDao userDao;

    public void init(){
        System.out.println("init");
```

```
    }
    public UserDao getUserDao() {
        return userDao;
    }


    //Qualifier 可以指定使用哪个名字的那个 bean
    @Autowired
    public void setUserDao(@Qualifier("u2")UserDao userDao) {
        this.userDao = userDao;
    }


    public void add(User user){
        userDao.save(user);
    }



    public void destroy(){
        System.out.println("destroy");
    }

}
```

++

Beans.xml

```
<?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-2.5.xsd
        http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd"
>

        <context:annotation-config/>

    <bean id="u" class="com.spring.dao.impl.UserDaoImpl"></bean>
    <bean id="u2" class="com.spring.dao.impl.UserDaoImpl"></bean>
```

```
    <bean id="userService" class="com.spring.service.UserService" >

    </bean>
</beans>
```

11.@Resource(重要)

a)加入:j2ee/common-annotations.jar
b)默认按名称，名称找不到，按类型
c)可以指定特定名称
d)推荐使用
e)不足：如果没有源码，就无法使用 annotation,只能使用 xml
在下面 12 中有代码

12.@Component 组件||@Repository 数据仓库与数据库相关||@Service 服务层||@Controller mvc 中的控制层;在目前版本中这四个没有区别

a)初始化的名字默认为类名首字母小写
b)可以指定初始化 bean 的名字

```
UserDaoImpl.java
package com.spring.dao.impl;

import org.springframework.stereotype.Component;

import com.spring.dao.UserDao;
import com.spring.model.User;

@Component("u")  //Component 是一个组件；初始化的名字默认为类名首字母小写；
一般加上名字
public class UserDaoImpl implements UserDao {

    public void save(User u) {
        System.out.println("save user!");

    }

}
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
UserService.java

package com.spring.service;
```

```
import javax.annotation.Resource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

import com.spring.dao.UserDao;
import com.spring.model.User;

@Component("userService")  // 与 UserServiceTest 中的 getBean 中的
userService 名字相同
public class UserService {
   private UserDao userDao;

   public void init(){
      System.out.println("init");
   }
   public UserDao getUserDao() {
      return userDao;
   }


//用 Resource 指定注入的 userDao
   @Resource(name="u")
   public void setUserDao(UserDao userDao) {
      this.userDao = userDao;
   }


   public void add(User user){
      userDao.save(user);
   }



   public void destroy(){
      System.out.println("destroy");
   }

}
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Bean.xml
```

```
<?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-2.5.xsd
        http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd"
>

        <context:annotation-config/>
        <context:component-scan base-package="com.spring"/>
</beans>
```

13.@Scope

14.@PostConstruct=init-method;@PreDestroy=destroy-method

```
UserService.java
package com.spring.service;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.annotation.Resource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

import com.spring.dao.UserDao;
import com.spring.model.User;
@Scope("singleton")
@Component("userService")  // 与 UserServiceTest 中的 getBean 中的
userService 名字相同
public class UserService {
    private UserDao userDao;

    @PostConstruct //构造完成之后
    public void init(){
        System.out.println("init");
```

```
    }
    public UserDao getUserDao() {
        return userDao;
    }


//用 Resource 指定注入的 userDao
    @Resource(name="u")
    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }


    public void add(User user){
        userDao.save(user);
    }


    @PreDestroy //容器销毁之前
    public void destroy(){
        System.out.println("destroy");
    }

}
```

****

什么是 Spring AOP

1.面向切面的编程 Aspect-Oriented-Programming

a)是对面向对象的思维方式的有力补充

/*JDK_Proxy_InvocationHandler 动态代理*/

新建一个包 com.spring.aop;在包中 LogInterceptor.java

```
package com.spring.aop;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

public class LogInterceptor implements InvocationHandler{
    private Object target;

    public Object getTarget() {
        return target;
    }
```

```
    public void setTarget(Object target) {
        this.target = target;
    }
    public void beforeMenthod(Method m){
        System.out.println(m.getName()+" start");
    }
    public Object invoke(Object proxy, Method m, Object[] args)
            throws Throwable {
        beforeMenthod(m);
        m.invoke(target, args);
        return null;
    }
}
```

++

UserServiceTest.java

```
package com.spring.service;

import java.lang.reflect.Proxy;

import org.junit.Test;

import com.spring.aop.LogInterceptor;
import com.spring.dao.UserDao;
import com.spring.dao.impl.UserDaoImpl;
import com.spring.model.User;
import com.spring.spr.BeanFactory;
import com.spring.spr.ClassPathXmlApplicationContext;

public class UserServiceTest {

    @Test
    public void testAdd() throws Exception {
        //new 一个 bean 工厂
        BeanFactory factory = new ClassPathXmlApplicationContext();
        UserService                    service                    =
(UserService)factory.getBean("userService");

//      UserDao userDao = (UserDao) factory.getBean("u"); // 相 当 于
UserDao userDao=new UserDaoImpl();只是用读取 xml 文件的形式实现
//      service.setUserdao(userDao);
        User u = new User();
        service.add(u);
    }
```

```
    @Test
    public void proxy(){
        UserDao userDao=new UserDaoImpl();
        LogInterceptor li=new LogInterceptor();
        li.setTarget(userDao);//把被代理对象设置成userDao
        //通过Proxy的静态方法产生代理对象new Class[]{UserDao.class}
        UserDao
userDaoProxy=(UserDao)Proxy.newProxyInstance(userDao.getClass().g
etClassLoader(),userDao.getClass().getInterfaces(), li);

    //System.out.println(userDao.getClass().getInterfaces().getCla
ss());
    //System.out.println(userDao.getClass().getInterfaces()[0]);
        System.out.println(userDaoProxy.getClass());
        userDaoProxy.delete();
        userDaoProxy.save(new User());
    }
    /**
     * class $Proxy4 implements UserDao
     * save (User u){
     *  Method m=userDao.getclass().getMethod
     * li.invoke(this,m,u)
     * }
     *
     * */

}
```

2.Spring_AOP_Annotation

LogInterceptor.java

```
package com.spring.aop;


import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;

@Aspect
@Component
```

```
public class LogInterceptor {
    //@Before("execution(public                                void
com.spring.dao.impl.UserDaoImpl.save(com.spring.model.User))")

    //@Pointcut("execution(public * com.spring.dao..*.*(..))")//有接
口，用jdk自带的动态代理实现

    @Pointcut("execution(public                                    *
com.spring.service..*.add(..))")//无接口，用直接操纵二进制码的类库cjlib
来帮你产生代理的代码
    public void myMethod(){}

    //@Before("execution(public  *  com.spring.dao..*.*(..))")//不加
Pointcut的时候
    @Before("myMethod()")
    public void before(){
        System.out.println("Method before");
    }
    //@AfterReturning("execution(public                            *
com.spring.dao..*.*(..))")
    @AfterReturning("myMethod()")
    //@AfterThrowing("myMethod()") //抛异常之后
    public void after(){
        System.out.println("Method after return");
    }

    @Around("myMethod()")
    public void around(ProceedingJoinPoint pjp) throws Throwable{
        System.out.println("Method around before");
        pjp.proceed();
        System.out.println("Method around after");
    }
}
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
UserDaoImpl.java
package com.spring.dao.impl;


import org.springframework.stereotype.Component;

import com.spring.dao.UserDao;
import com.spring.model.User;

@Component("u")  //Component是一个组件；初始化的名字默认为类名首字母小写；
```

一般加上名字

```
public class UserDaoImpl implements UserDao {

   public void save(User u) {
      System.out.println("save user!");
      //throw new RuntimeException("exception");//声明式抛异常管理

   }


}
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
UserService.java
package com.spring.service;

import javax.annotation.Resource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

import com.spring.dao.UserDao;
import com.spring.model.User;

@Component("userService")  // 与 UserServiceTest 中 的 getBean 中 的
userService 名字相同
public class UserService {
   private UserDao userDao;

   public void init(){
      System.out.println("init");
   }
   public UserDao getUserDao() {
      return userDao;
   }


//用 Resource 指定注入的 userDao
   @Resource(name="u")
   public void setUserDao(UserDao userDao) {
      this.userDao = userDao;
   }
```

```
    public void add(User user){
        userDao.save(user);
    }



    public void destroy(){
        System.out.println("destroy");
    }

}
```

++

Bean.xml

```
<?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:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans

    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
        http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd"
>

        <context:annotation-config/>
        <context:component-scan base-package="com.spring"/>
        <aop:aspectj-autoproxy/>
</beans>
```

++

UserServiceTest.java

```
package com.spring.service;

import org.junit.Test;
import
org.springframework.context.support.ClassPathXmlApplicationContex
t;

import com.spring.model.User;
```

```
public class UserServiceTest {

    @Test
    public void testAdd() throws Exception {
        //new 一个 bean 工厂
        ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");
        UserService service = (UserService)ctx.getBean("userService");
        System.out.println(service.getClass());//确认拿到的是一个代理对象
        service.add(new User());
        ctx.destroy();


    }

}
```

3.好处：可以动态的添加和删除在切面上的逻辑而不影响原来的执行代码

a)Filter
b)Struts2 的 interceptor

4.概念

a)JoinPoint //连接点，在切面的切点就是 JoinPoint
b)PointCut // 切 入 点 ， @PointCut("execution(* com.xyz.someapp.service.*.*(..))"), 是 JoinPoint 的集合
c)Aspect(切面) //类加上的逻辑就是切面
d)Advice //加入点上的建议和切面概念有点重合，指的是@before
e)Target //被代理对象
f)Weave //织入，一个面一个面的穿进去

5.AOP 的 xml 形式配置（重要）

```
LogInterceptor.java
package com.spring.aop;


import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
```

```
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;



public class LogInterceptor {



    public void myMethod(){}


    public void before(){
        System.out.println("Method before");
    }

    public void after(){
        System.out.println("Method after return");
    }

    public void around(ProceedingJoinPoint pjp) throws Throwable{
        System.out.println("Method around before");
        pjp.proceed();
        System.out.println("Method around after");
    }
}
```

++

UserService.java

```
package com.spring.service;

import javax.annotation.Resource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

import com.spring.dao.UserDao;
import com.spring.model.User;

@Component("userService") //与 UserServiceTest 中的 getBean 中的 userService 名字相同
public class UserService {
```

```
    private UserDao userDao;

    public void init(){
        System.out.println("init");
    }
    public UserDao getUserDao() {
        return userDao;
    }


//用 Resource 指定注入的 userDao
    @Resource(name="u")
    public void setUserDao(UserDao userDao) {
        this.userDao = userDao;
    }


    public void add(User user){
        userDao.save(user);
    }



    public void destroy(){
        System.out.println("destroy");
    }

}
```

++

Beans.xml

```
<?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:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans

    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
        http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd"
```

```
>

       <context:annotation-config/>
       <context:component-scan base-package="com.spring"/>
       <bean id="logInterceptor"
class="com.spring.aop.LogInterceptor"></bean>

       <aop:config>
        <aop:pointcut expression="execution(public *
com.spring.service..*.add(..))" id="servicePointcut"/>
        <aop:aspect id="logAspect" ref="logInterceptor">
           <aop:before method="before"
pointcut-ref="servicePointcut"/>
           <aop:after-returning method="after"
pointcut-ref="servicePointcut"/>
        </aop:aspect>
       </aop:config>
</beans>
```