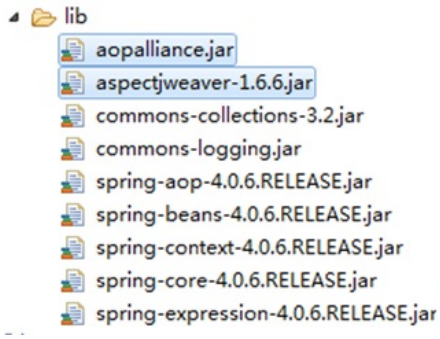
【Java框架型项目从入门到装逼】第二节 - Spring框架 AOP的丧心病狂解说，你喜欢露娜的月下无限连吗？

继续上一节的内容，多几个jar包：



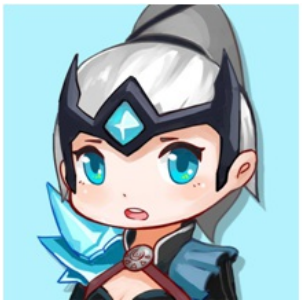
aop技术是面向切面编程思想，作为OOP的延续思想添加到企业开发中，用于弥补OOP开发过程中的缺陷而提出的编程思想。AOP底层也是面向对象；只不过面向的不是普通的Object对象，而是特殊的AOP对象。AOP的关注点是组成系统的非核心通用服务模块（比如登录检查等），相对于普通对象，aop不需要通过继承、方法调用的方式来提供功能，只需要在xml文件中以引用的方式，将非核心服务功能引用给需要改功能的核心业务逻辑对象或方法中。最终实现对象的解耦。spring 中ioc技术实现了核心业务逻辑对象之间的解耦（如LoginAction与DaoImpl）

AOP可以说是Spring中最难理解的一个知识点了，你可能网上找了很多资料，也买过很多本书，但都不是很理解到底什么是AOP？我曾经也是琢磨了好久才有了一定的了解。那么，到底怎么讲这个知识点呢。来不及解释了，快上车！听完这个例子，我相信你一定会对AOP有一个非常深刻的理解！
让我们新建一个英雄类：

```
package com.spring.bean;

public class Hero {

    private String heroName;
    private String type;
    private String description;

    public String getHeroName() {
        return heroName;
    }
    public void setHeroName(String heroName) {
        this.heroName = heroName;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    @Override
    public String toString() {
        return "Hero [heroName=" + heroName + ", type=" + type + ", description=" + description + "]";
    }

}
```

再来个露娜类，继承自英雄类：

啊啊啊，放错图片了，应该是这个：



```
package com.spring.bean;

public class Luna extends Hero{

    /**
     * 秀操作
     */
    public void operation(){
        System.out.println("看我月下无限连！");
    }

    /**
     * 跑路
     */
    public void run(){
        System.out.println("我操，大空了，赶紧跑！");
    }

    /**
     * 发信息
     * @param str
     */
    public void say(String str){
        System.out.println(str);
    }

}
```

可以看到，露娜类有三个方法，分别是秀操作，跑路和发信息。
再写一个团战类：

```
package com.spring.test;

import com.spring.bean.Luna;

/**
 * 团战类
 * @author Administrator
 *
 */
public class Battle {

    public void tuan(){
        Luna luna = new Luna();
        luna.say("上去开团！");
        luna.operation();

    }

}
```

测试代码如下：

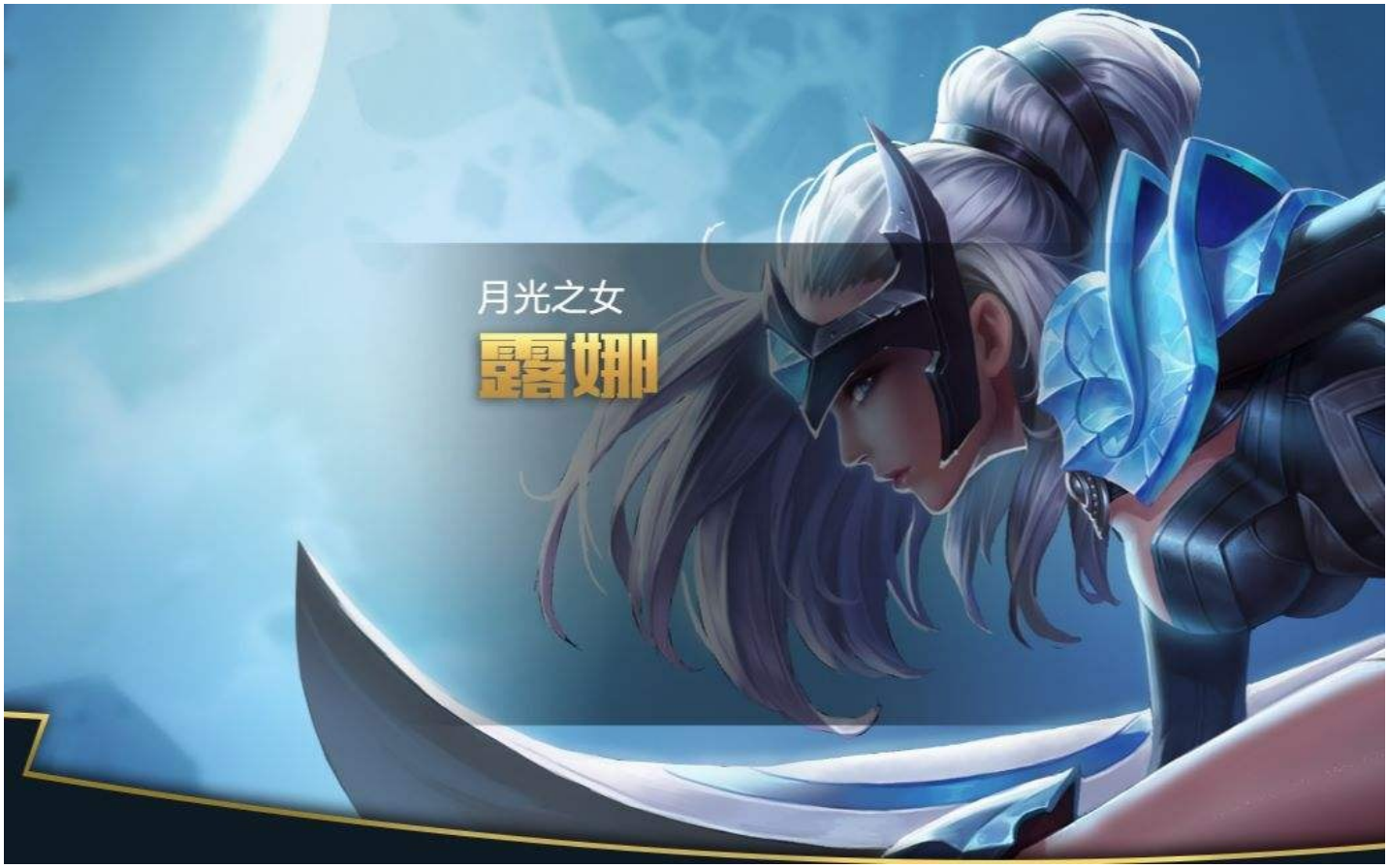
```
ApplicationContext context = new ClassPathXmlApplicationContext("spring-aop.xml");
Battle battle = (Battle)context.getBean("Battle");
battle.tuan();
```

我们用spring把Battle类配上去。

spring-aop.xml

```
<bean id = 'Battle' class="com.spring.test.Battle"></bean>
```

运行测试代码：
上去开团！
看我月下无限连！



在团战方法里面，我们新建一个露娜的对象，然后发出信息“上去开团”，接着又秀了一把操作。这是一个比较普通的流程。而事实上，露娜可能需要在团战前就提醒我方队友“等我集合打团”，不要人都没齐，队友就无脑往前冲。OK，我们如何通过代码来实现这个过程呢？很显然，这个过程需要在团战方法执行之前就被执行。这就需要AOP面向切面的技术了。

我们需要写一个类，实现MethodBeforeAdvice接口。

```
/**
 * Notice 定义一个通知打团的信号 - 团战之前
 * @author Administrator
 *
 */
public class BeforeTuanZhan implements MethodBeforeAdvice{

    @Override
    public void before(Method arg0, Object[] arg1, Object arg2) throws Throwable {
        System.out.print(this.getClass().getName() + " -- ");
        Luna luna = new Luna();
        luna.setHeroName("露娜");
        luna.setType("战士/法师");
        luna.setDescription("月光女神");
        luna.say("等我集合打团！");

    }

}
```

我们希望这个方法在团战之前就被执行，怎么做呢？没错，就是在XML文件中做如下配置：

```
<bean id = 'BeforeTuanZhan' class="com.spring.service.BeforeTuanZhan"></bean>

<aop:config>

        <!-- 定义所有可供露娜切入的点（方法） -->
        <!-- 原则上只要时机正确，任何团战露娜都可以切进去！ -->
        <aop:pointcut expression="execution(* com.spring.test.Battle.*(..))" id="pointcut"/>

        <aop:advisor advice-ref="BeforeTuanZhan" pointcut-ref="pointcut"/>


    </aop:config>
```

execution(* com.spring.test.Battle.*(..))
这句话的含义就是，返回值为任意，com.spring.test包里面的Battle类，这个类里面所有的方法都需要切入。所谓的切入，就是在方法执行前，执行

中，发生异常的时候执行某个其他的方法。执行中用的不多，一般就用另外三种情况。

现在，我们重新执行测试代码：

```
ApplicationContext context = new ClassPathXmlApplicationContext("spring-aop.xml");
Battle battle = (Battle)context.getBean("Battle");
battle.tuan();
```

```
com.spring.service.BeforeTuanZhan -- 等我集合打团!
上去开团!
看我月下无限连!
```

现在各位想一下，如果团战打赢了怎么办，是不是马上就该去推塔或者打龙啊，这个时候，如果队友团战打赢了就发呆，那就很坑了。所以呢，你这个时候就得提醒队友下一步该做什么，这个提醒的步骤是在团战方法执行结束后才发生的。

我们需要新建一个AfterTuanZhan类，实现AfterReturningAdvice接口。

```
/**
 * Notice 定义一个团战结束后的类 - 团战之后
 * @author Administrator
 */
public class AfterTuanZhan implements AfterReturningAdvice{

    @Override
    public void afterReturning(Object arg0, Method arg1, Object[] arg2, Object arg3) throws Throwable {
        System.out.print(this.getClass().getName() + " -- ");
        Luna luna = new Luna();
        luna.setHeroName("露娜");
        luna.setType("战士/法师");
        luna.setDescription("月光女神");
        luna.say("进攻敌方防御塔！");
    }

}
```

配置到spring-aop.xml中：

```
<bean id = 'AfterTuanZhan' class="com.spring.service.AfterTuanZhan"></bean>
<!-- 定义一个切面 -->
    <aop:config>

        <!-- 定义所有可供露娜切入的点（方法） -->
        <!-- 原则上只要时机正确，任何团战露娜都可以切进去！ -->
        <aop:pointcut expression="execution(* com.spring.test.Battle.*(..))" id="pointcut"/>

        <aop:advisor advice-ref="BeforeTuanZhan" pointcut-ref="pointcut"/>
        <aop:advisor advice-ref="AfterTuanZhan" pointcut-ref="pointcut"/>

    </aop:config>
```

现在，我们重新执行测试代码：

```
ApplicationContext context = new ClassPathXmlApplicationContext("spring-aop.xml");
Battle battle = (Battle)context.getBean("Battle");
battle.tuan();
```

```
com.spring.service.BeforeTuanZhan -- 等我集合打团!
上去开团!
看我月下无限连!
com.spring.service.AfterTuanZhan -- 进攻敌方防御塔!
```

再来说说万一团战失利的情况，比如露娜断大了咋办，没错，这个时候就是团战发生了异常，我们在Battle类中手动设置一个异常：

```
/**
 * 团战类
 * @author Administrator
 *
 */
public class Battle {

    public void tuan(){
        Luna luna = new Luna();
        luna.say("上去开团！");
        luna.operation();

        int i = 1 / 0 ;

    }

}
```

然后，编写TuanZhanException类，实现ThrowsAdvice接口：

```
/**
 * 定义一个团战异常类，万一出现情况就进入这个类
 * @author Administrator
 *
 */
public class TuanZhanException implements ThrowsAdvice {
```

```
    //该方法会在露娜团战出现异常后自动执行
    public void afterThrowing(Method method, Object[] args,
            Object target, Exception ex){
        System.out.print(this.getClass().getName() + " -- ");
        Luna luna = new Luna();
        luna.run();
    }
}
```

配置到spring-aop.xml:

```
<bean id = 'TuanZhanException' class="com.spring.service.TuanZhanException"></bean>
<!-- 定义一个切面 -->
    <aop:config>

        <!-- 定义所有可供露娜切入的点（方法） -->
        <!-- 原则上只要时机正确，任何团战露娜都可以切进去！ -->
        <aop:pointcut expression="execution(* com.spring.test.Battle.*(..))" id="pointcut"/>

        <aop:advisor advice-ref="BeforeTuanZhan" pointcut-ref="pointcut"/>
        <aop:advisor advice-ref="AfterTuanZhan" pointcut-ref="pointcut"/>
        <aop:advisor advice-ref="TuanZhanException" pointcut-ref="pointcut"/>

    </aop:config>
```

现在，我们重新执行测试代码：

```
ApplicationContext context = new ClassPathXmlApplicationContext("spring-aop.xml");
Battle battle = (Battle)context.getBean("Battle");
battle.tuan();
```

```
com.spring.service.BeforeTuanZhan -- 等我集合打团!
上去开团!
看我月下无限连!
com.spring.service.TuanZhanException -- 我操，大空了，赶紧跑!
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at com.spring.test.Battle.tuan(Battle.java:17)
```

总结：

1. aop面向切面，切的是什么，没错，切的是方法！

2. 怎么切，你记好了，就是你先自己规定哪些方法需要切，然后设置切入的方式：方法执行之前做什么，执行之后做什么，如果方法出现异常，又要做什么？另外还有一种方法执行的过程中做什么，只是用的比较少，反正我还没有见过在哪里用了。用的最多的就是发生异常后做什么，比如事务管理。

据说露娜要重做，玩个屁，劳资退游吃鸡去了，谢谢。

如果喜欢我的文章，还请点一波关注，谢谢老铁！

源码链接：http://pan.baidu.com/s/1o8zHMqe 密码：om9i