一、bean管理(注解)

导入jar包（基本的jar包+aop的jar包）
创建类，创建方法
创建spring配置文件，引入约束（第一天做ioc基本功能，引入约束beans + 做注解还要引入新的约束context schema）

```
<?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:sch
emaLocation="
        http://www.springframework.org/schema/beans http://www.springfram
ework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springfr
amework.org/schema/context/spring-context.xsd">

    <!-- 开启注解扫描
        (1)到包里面扫描类、方法、属性上面是否有注解
    -->
    <context:component-scan base-package="cn.itcast"></context:component-
scan>

    <!--
        (2)只扫描属性上面的注解
     -->
    <!-- <context:annotation-config></context:annotation-config> -->
</beans>
```

二、注解创建对象

在创建对象的类上面使用注解实现

```java
package cn.itcast.anno;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;

@Service(value="user")    // 相当于<bean id="user" class=""/>
@Scope(value="prototype")    // 单实例还是多实例
public class User {

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

```java
package cn.itcast.anno;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContex
t;

public class TestAnno {

    @Test
    public void testUser() {
        ApplicationContext context =
                new ClassPathXmlApplicationContext("bean1.xml");
        User user = (User) context.getBean("user");
        System.out.println(user);
        user.add();
    }
}
```

创建对象有四个注解：\@Component \@Controller(web层) \@Service(业务层) \@Repository(持久层)，这四个注解功能是一样的，都创建对象，spring后续版本会对其增强

创建对象是单实例还是多实例`@Scope(value="prototype")`

三、注解注入属性

创建dao和service对象

```java
package cn.itcast.anno;
import org.springframework.stereotype.Component;

@Component(value="userDao")
public class UserDao {

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

在service类中定义dao类型属性

1. 注入属性第一个注解方式(自动注入\@Autowired)

```java
package cn.itcast.anno;
import javax.annotation.Resource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service(value="userService")
public class UserService {

    //得到dao对象
    //1 定义dao类型属性
    //在dao属性上面使用注解 完成对象注入
    @Autowired
    private UserDao userDao;

    public void add() {
        System.out.println("service..........");
        userDao.add();
    }
}
```

```
package cn.itcast.anno;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationCo
ntext;

public class TestAnno {

    @Test
    public void testService() {
        ApplicationContext context =
                new ClassPathXmlApplicationContext("bean1.xml");
        UserService userService = (UserService) context.getBean("use
rService");
        userService.add();
    }
}
```

2. 注入属性第二个注解方式(\@Resource)

```
package cn.itcast.anno;
import javax.annotation.Resource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service(value="userService")
public class UserService {

    @Resource(name="userDao")
    private UserDao userDao;

    public void add() {
        System.out.println("service..........");
        userDao.add();
    }
}
```

四、配置文件和注解混合使用

创建对象操作使用配置文件方式实现

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

    <!-- 开启注解扫描
        (1)到包里面扫描类、方法、属性上面是否有注解
    -->
    <context:component-scan base-package="cn.itcast"></context:component-scan>

    <!-- 配置对象 -->
    <bean id="bookService" class="cn.itcast.xmlanno.BookService"></bean>
    <bean id="bookDao" class="cn.itcast.xmlanno.BookDao"></bean>
    <bean id="ordersDao" class="cn.itcast.xmlanno.OrdersDao"></bean>
</beans>
```

注入属性的操作使用注解方式实现

```java
package cn.itcast.xmlanno;

public class BookDao {

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

```java
package cn.itcast.xmlanno;

public class OrdersDao {

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

```
package cn.itcast.xmlanno;

import javax.annotation.Resource;

public class BookService {

    //得到bookdao和ordersdao对象
    @Resource(name="bookDao")
    private BookDao bookDao;

    @Resource(name="ordersDao")
    private OrdersDao ordersDao;

    public void add() {
        System.out.println("service.........");
        bookDao.book();
        ordersDao.buy();
    }
}
```

junit测试

```
package cn.itcast.xmlanno;

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

public class TestAnno {

    @Test
    public void testService() {
        ApplicationContext context =
                new ClassPathXmlApplicationContext("bean2.xml");
        BookService bookService = (BookService) context.getBean("bookServ
ice");
        bookService.add();
    }
}
```

五、AOP概念

1 aop：面向切面（方面）编程，扩展功能不修改源代码实现

2 AOP采取横向抽取机制，取代了传统纵向继承体系重复性代码

3 aop底层使用动态代理实现
（1）第一种情况，有接口情况，使用动态代理创建接口实现类代理对象
（2）第二种情况，没有接口情况，使用动态代理创建类的子类代理对象

六、AOP操作

导入jar包
增加aop约束

```
<?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" xsi:schemaLocat
ion="
        http://www.springframework.org/schema/beans http://www.springfram
ework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop http://www.springframew
ork.org/schema/aop/spring-aop.xsd">
</beans>
```

1. 基于aspectj的xml配置

使用表达式配置接入点

```
1 切入点：实际增强的方法

2 常用的表达式
execution(<访问修饰符>?<返回类型><方法名>(<参数>)<异常>)
（1）execution(* cn.itcast.aop.Book.add(..))          //要增强的方法的全路径.
只有Book类中的add()方法增强
（2）execution(* cn.itcast.aop.Book.*(..))            //Book类中的所有方法都
增强
（3）execution(* *.*(..))                             //所有类的所有方法
（4）匹配所有save开头的方法 execution(* save*(..))    //满足一定规则的方法都增强.
save开头的方法都增强
```

```
    <!-- 1  配置对象 -->
    <bean id="book" class="cn.itcast.aop.Book"></bean>
    <bean id="myBook" class="cn.itcast.aop.MyBook"></bean>

    <!-- 2 配置aop操作 -->
    <aop:config>
        <!-- 2.1 配置切入点
            id=""即给切入点起个名字
        -->
        <aop:pointcut expression="execution(* cn.itcast.aop.Book.*(..))"
id="pointcut1"/>

        <!-- 2.2 配置切面
            把增强用到方法上面
        -->
        <aop:aspect ref="myBook">
            <!-- 配置增强类型
                method:  增强类里面使用哪个方法作为前置
            -->
            <aop:before method="before1" pointcut-ref="pointcut1"/>

            <aop:after-returning method="after1" pointcut-
ref="pointcut1"/>

            <aop:around method="around1" pointcut-ref="pointcut1"/>
        </aop:aspect>
    </aop:config>
```

```
package cn.itcast.aop;

public class Book {

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

```
package cn.itcast.aop;

import org.aspectj.lang.ProceedingJoinPoint;

public class MyBook {

    public void before1() {
        System.out.println("前置增强......");
    }

    public void after1() {
        System.out.println("后置增强......");
    }

    //环绕通知
    public void around1(ProceedingJoinPoint proceedingJoinPoint) throws T
hrowable {
        //方法之前
        System.out.println("方法之前.....");

        //执行被增强的方法
        proceedingJoinPoint.proceed();

        //方法之后
        System.out.println("方法之后.....");
    }
}
```

```
package cn.itcast.aop;

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

public class TestAnno {

    @Test
    public void testService() {
        ApplicationContext context =
                new ClassPathXmlApplicationContext("bean3.xml");
        Book book = (Book) context.getBean("book");
        book.add();
    }
}
```

2. 基于aspectj的注解方式

使用配置文件创建对象

```
<?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" xsi:schemaLocat
ion="
        http://www.springframework.org/schema/beans http://www.springfram
ework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop http://www.springframew
ork.org/schema/aop/spring-aop.xsd">

    <!-- 开启aop操作 -->
    <aop:aspectj-autoproxy></aop:aspectj-autoproxy>

    <!-- 1  配置对象 -->
    <bean id="book" class="cn.itcast.aop.Book"></bean>
    <bean id="myBook" class="cn.itcast.aop.MyBook"></bean>

</beans>
```

在spring核心配置文件中，开启aop操作

`<aop:aspectj-autoproxy></aop:aspectj-autoproxy>`

在增强类上面使用注解完成aop操作

```
package cn.itcast.aop;

public class Book {

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

```java
package cn.itcast.aop;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class MyBook {

    //在方法上面使用注解完成增强配置
    @Before(value="execution(* cn.itcast.aop.Book.*(..))")
    public void before1() {
        System.out.println("before.............");
    }
}
```

```java
package cn.itcast.aop;

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

public class TestAop {

    @Test
    public void testDemo() {
        ApplicationContext context =
                new ClassPathXmlApplicationContext("bean3.xml");
        Book book = (Book) context.getBean("book");
        book.add();
    }
}
```

七、log4j

1 通过log4j可以看到程序运行过程中更详细的信息
（1）经常使用log4j查看日志

2 使用
（1）导入log4j的jar包
（2）复制log4j的配置文件log4j.properties，复制到src下面

3 设置日志级别
（1）info：看到基本信息
（2）debug：看到更详细信息