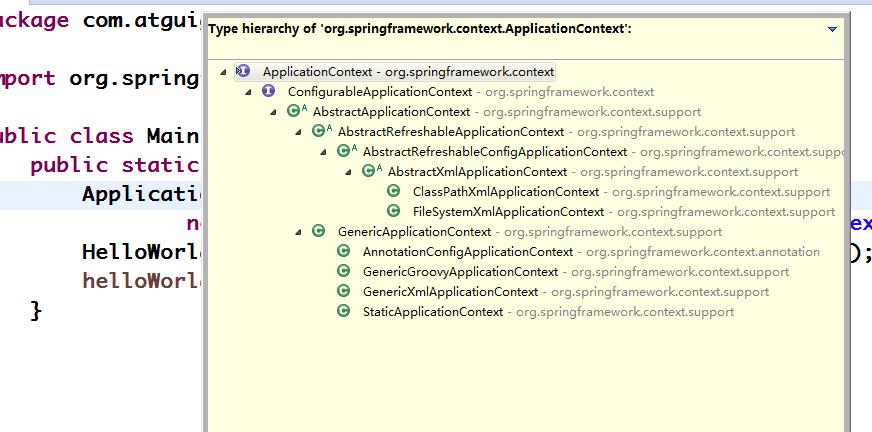
Video1-3：

1. spring的helloworld
2. 装eclipse插件
3. 导Jar包 （logging+4必须） -->build path
4. applicationContext.xml配置bean
5. ApplicationContext接口(它代表IOC容器)继承最底层IOC容器BeanFactory接口

继承图：ctrl+t





1. 属性注入 用set方法/constructor-arg注入

a). <bean id=*"person2"* class=*"com.atguigu.spring.helloworld.Person"*>

<constructor-arg value=*"Jerry"*></constructor-arg>

<constructor-arg value=*"25"*></constructor-arg>

<constructor-arg ref=*"car"*></constructor-arg>

<!-- 为级联属性car.maxSpeed赋值 -->

<property name=*"car.maxSpeed"* value=*"300000"*> </property>

</bean>

必须有ref=”car”不能直接给级联属性赋值，struts可以…(这点很少用)

b).

使用<list> <set>配置集合属性，如一人有多辆车。。

集合标签还可以定义内部bean

一人多车实例

<bean id=*"person3"* class=*"com.atguigu.spring.beans.collection.Person"*>

<property name=*"name"* value=*"Lilly"*></property>

<property name=*"age"* value=*"28"*></property>

<property name=*"cars"*>

<list>

<ref bean=*"car"*/>

<ref bean=*"car2"*/>

<bean class=*"com.atguigu.spring.helloworld.Car"*>

<constructor-arg value=*"Ford"*></constructor-arg>

<constructor-arg value=*"changan"*></constructor-arg>

<constructor-arg value=*"200000.0"* type=*"double"*></constructor-arg>

</bean>

</list>

</property>

</bean>

配置map：用<map>+entry子节点就好了。。。简单

<bean id=*"newperson"* class=*"com.atguigu.spring.beans.collection.NewPerson"*>

<property name=*"name"* value=*"Rose"*></property>

<property name=*"age"* value=*"29"*></property>

<property name=*"cars"*>

<map>

<entry key=*"AAAA"* value-ref=*"car"*></entry>

<entry key=*"BBBB"* value-ref=*"car2"*></entry>

</map>

</property>

</bean>

另外：使用<props>配置java.util.Properties（这东西是map的一个子类）

<bean id=*"dataSource"* class=*"com.atguigu.spring.beans.collection.DataSource"*>

<property name=*"properties"*>

<props>

<prop key=*"user"*>root</prop>

<prop key=*"password"*></prop>

<prop key=*"jdbcUrl"*>jdbc:mysql:///hibernate</prop>

<prop key=*"driverClass"*>com.jdbc.mysql.Driver</prop>

</props>

</property>

</bean>

c）:配置单利的集合bean，以供大家公用。。。。

<!-- 配置单利的集合bean，以供大家使用。。。。需要导入util namespace -->

<util:list id=*"cars"*>

<ref bean=*"car"*/>

<ref bean=*"car2"*/>

</util:list>

<bean id=*"person4"* class=*"com.atguigu.spring.beans.collection.Person"*>

<property name=*"name"* value=*"Jack"*></property>

<property name=*"age"* value=*"88"*></property>

<property name=*"cars"* ref=*"cars"*></property>

</bean>

d). <!-- 使用p:方式给属性赋值， 需要导入p namespace,这样更简单了-->

<bean id=*"person5"* class=*"com.atguigu.spring.beans.collection.Person"* p:age=*"20"* p:name=*"Queen"* p:cars-ref=*"cars"*></bean>

Video5..自动装配

Spring根据byName或者byType自动装配，bean里的

要求：

byName：若有setAddress方法，则bean id必须要是address才能装配上

byType：根据bean的类型和当前bean的属性的类型自动匹配，如果有两个address、address2，则不能装配。。。

实际开发中，很少自动装配。。。整合第三方框架时用的多些。。

Video6.

1.继承 beans-relation.xml

就是parent、abstract属性的配置

若某个bean没有class属性，它直接就是个模板bean

<?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:p=*"http://www.springframework.org/schema/p"*

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

<bean id=*"address"* p:city=*"Beijing~~~"* p:street=*"Huolushan"* abstract=*"true"*></bean>

<bean id=*"address2"* class=*"com.atguigu.spring.beans.aotuwire.Address"*

parent=*"address"* p:city=*"wuhan"* p:street=*"Wudaok"*></bean>

</beans>

1. depends-on属性：依赖。。。不会赋值。。。。

<bean id=*"address"*

p:city=*"Beijing~~~"* p:street=*"Huolushan"* abstract=*"true"*></bean>

<bean id=*"address2"* class=*"com.atguigu.spring.beans.aotuwire.Address"*

parent=*"address"* p:city=*"wuhan"* p:street=*"Wudaokou"* ></bean>

<bean id=*"car"* class=*"com.atguigu.spring.beans.aotuwire.Car"*

p:brand=*"audi"* p:price=*"120000"*></bean>

<bean id=*"person"* class=*"com.atguigu.spring.beans.aotuwire.Person"*

p:name=*"Tom"* p:address-ref=*"address2"* depends-on=*"car"*/>

Video7。。bean的作用域

<bean id=*"car"*

class=*"com.atguigu.spring.beans.aotuwire.Car"*

scope=*"prototype"*

p:brand=*"audi"* p:price=*"10000"*>

</bean>

Video8。。加载外部属性文件（如mysql数据库连接配置信息）

PPT有。。。简单。

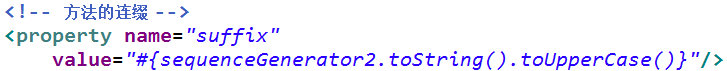
Video9。。SpEL

几个特别的：

引用其他对象的属性：



调用其他方法，还可以链式操作。。



等等 正则。。。，。。。。见PPT

Spel例子：

<bean id=*"address"* class=*"com.atguigu.spring.beans.spel.Address"*>

<property name=*"city"* value=*"#{'BeiJing'}"*></property>

<property name=*"street"* value=*"五道口"*></property>

</bean>

<bean id=*"car"* class=*"com.atguigu.spring.beans.spel.Car"*>

<property name=*"brand"* value=*"Audi"*></property>

<property name=*"price"* value=*"500000"*></property>

<!-- 用spel引用静态变量 -->

<property name=*"tyrePerimeter"* value=*"#{T(java.lang.Math).PI \* 80}"*></property>

</bean>

<bean id=*"person"* class=*"com.atguigu.spring.beans.spel.Person"*>

<property name=*"name"* value=*"Tom"*></property>

<property name=*"car"* value=*"#{car}"*></property>

<property name=*"city"* value=*"#{address.city}"*></property>

<property name=*"info"* value=*"#{car.price > 30 ?'金领':'金领'}"*></property>

</bean>

Video10：配置bean的生命周期：

<bean id=*"car"* class=*"com.atguigu.spring.beans.cycle.Car"*

init-method=*"init"* destroy-method=*"destroy"*>

<property name=*"brand"* value=*"Audi.."*></property>

</bean>

<bean class=*"com.atguigu.spring.beans.cycle.MyBeanPostProcessor"*></bean>

**public** **class** Main {

**public** **static** **void** main(String[] args) {

ClassPathXmlApplicationContext ctx = **new** ClassPathXmlApplicationContext("beans-cycle.xml");

Car car = (Car) ctx.getBean("car");

System.***out***.println(car);

ctx.close();

}

}

Car`s constructor..

set Brand......

postProcessBeforeInitialization

Car`s constructor..

set Brand......

init...

postProcessAfterInitialization.....

Car [brand=QQ..]

三月 05, 2015 6:56:09 下午 org.springframework.context.support.AbstractApplicationContext doClose

信息: Closing org.springframework.context.support.ClassPathXmlApplicationContext@4ebaecb8: startup date [Thu Mar 05 18:56:08 CST 2015]; root of context hierarchy

destroy...

Video11:整合第三方框架的时候需要用到配置bean的其他方式

1. 使用工厂方法：

直接调用某个类的静态工厂方法返回bean实例（可以使用static代码块初始化）

**public** **class** StaticCarFactory {

**public** **static** Map<String, Car> *cars* = **new** HashMap<String, Car>();

**static**{

*cars*.put("Audi", **new** Car("Audi", 300000.0, 20.0));

*cars*.put("Dazhong", **new** Car("Dazhong", 20000.0, 30));

}

**public** **static** Car getCar(String name){

**return** *cars*.get(name);

}

}

<bean id=*"car1"*

class=*"com.atguigu.spring.beans.factory.StaticCarFactory"*

factory-method=*"getCar"*

>

<constructor-arg value=*"Audi"*></constructor-arg>

</bean>

2.用实例（这里表示非静态）工厂方法

<!-- 配置工厂实例 -->

<bean id=*"carFactory"* class=*"com.atguigu.spring.beans.factory.InstanceCarFactory"*></bean>

<!-- 通过实例工厂方法配置bean -->

<bean id=*"car2"* factory-bean=*"carFactory"* factory-method=*"getCar"*>

<constructor-arg value=*"xiandai"*></constructor-arg>

</bean>

**public** **class** InstanceCarFactory {

**public** InstanceCarFactory() {

cars = **new** HashMap<String, Car>();

cars.put("audi", **new** Car("audi...", 2000, 20));

cars.put("xiandai", **new** Car("Hed...", 3000, 30));

}

**private** Map<String , Car> cars = **null**;

**public** Car getCar(String brand){

**return** cars.get(brand);

}

}

3.通过Spring框架自身提供的FactoryBean接口来配置bean

Video13：基于注解配置bean(1)

1. @Component:基本注解
2. @Repository:持久层
3. @Service：业务层
4. @Controller:表现层

组件命名规则：使用非限定性类名，第一个字母小写。

也可以在注解中通过value属性值标识名称

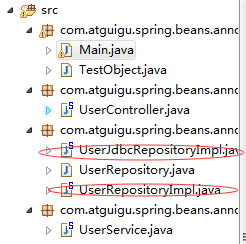
具体操作见eclipseSpring里的spring-1工程下的com.atguigu.spring.beans.annotation下的main方法的操作



Video14.通过注解配置bean(2)

@Autowired 注解自动装配**具有兼容类型**的单个 Bean属性

* + **构造器, 普通字段(即使是非 public), 一切具有参数的方法都可以应用@Autowired 注解**
  + **默认情况下, 所有使用 @Autowired 注解的属性都需要被设置. 当 Spring 找不到匹配的 Bean 装配属性时, 会抛出异常, 若某一属性允许不被设置, 可以设置 @Autowired 注解的 required 属性为 false**
  + 默认情况下, 当 IOC 容器里存在多个类型兼容的 Bean 时, 通过类型的自动装配将无法工作. 此时可以在 **@Qualifier** 注解里提供 Bean 的名称. **Spring 允许对方法的入参标注 @Qualifiter 已指定注入 Bean 的名称**



**实例代码：**

* **private** UserRepository userRepository;
* @Autowired
* @Qualifier("userJdbcRepositoryImpl")//括号里是bean的名字，小写大头
* **public** **void** setUserRepository(UserRepository userRepository) {
* **this**.userRepository = userRepository;
* }

Video15.泛型依赖注入，具体见com.atguigu.spring.beans.generic.di下的所有代码

1. @Autowired

**protected** BaseRepository<T> baseRepository;

这两句表示可以被子类继承

**public** **class** BaseService<T> **extends** BaseRepository<T>{

@Autowired

**protected** BaseRepository<T> baseRepository;

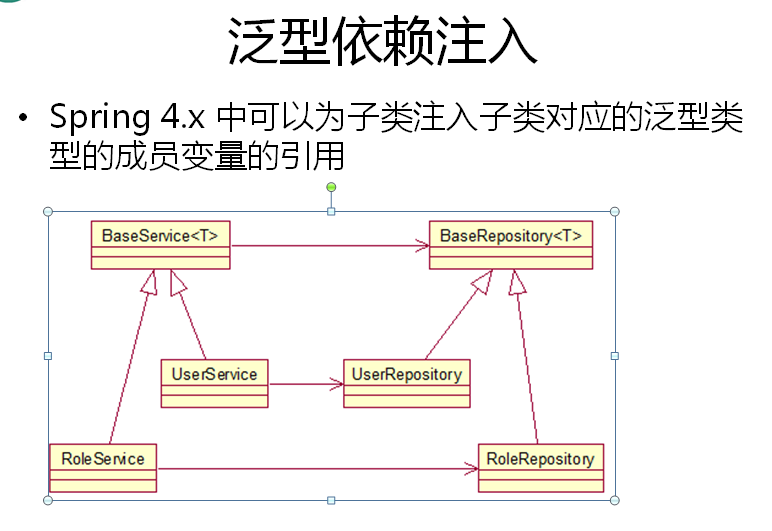
**public** **void** add(){

System.***out***.println("BaseService的add方法....");

System.***out***.println(baseRepository);

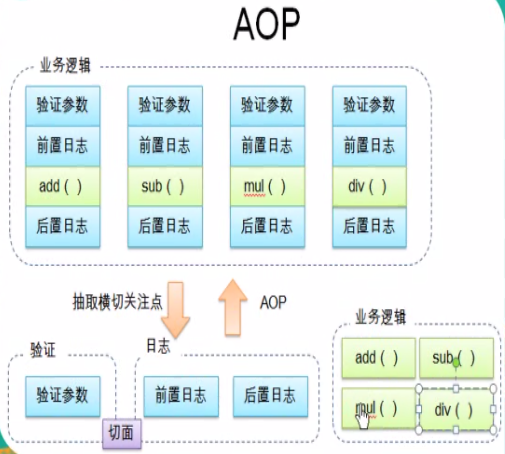
}

}



Video-16 AOP

1. 动态代理（很重要，已弄懂），实现代理的代码见spring-2下的包com.atguigu.weibiao.dynamicProxy
2. AOP的意思 很简单 面向关注的类型去编程

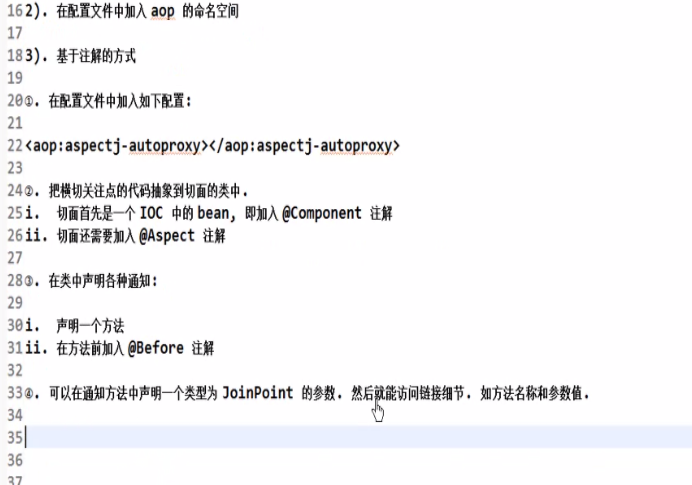


Video17. AspectJ

* **AspectJ**：Java 社区里最完整最流行的 AOP 框架.
* 在 Spring2.0 以上版本中, 可以使用基于 AspectJ 注解或基于 XML 配置的 AOP

**AOP步骤：**

**1).加入jar包**



Video18 后置通知

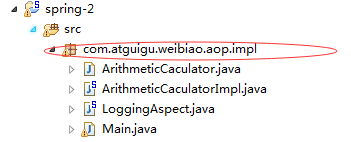
@After("execution(\* com.atguigu.weibiao.aop.impl.\*.\*(int , int ))")

Notice：无论是否发生异常，都会执行。

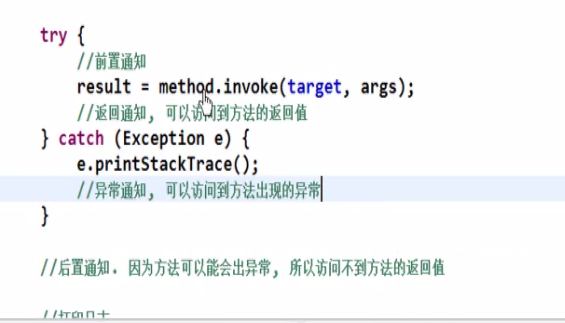
Video 19

尚硅谷\_佟刚\_Spring\_返回通知&异常通知&环绕通知

具体见代码。。包如下。。



1. 返回通知和后置通知的区别，唯一在异常。。
2. 异常通知、。。



Video20.切面优先级 用@Order

@Order(1)

括号里值越小越高优先级

Video21.重用表达式

/\*

\*

\* 定义一个方法，用于声明切入点表达式，一般的，该方法不需要加入其他的代码

\*/

@Pointcut("execution(public int com.atguigu.weibiao.aop.impl.ArithmeticCaculator.\*(..))")

**public** **void** declareJointPointExpression(){ }

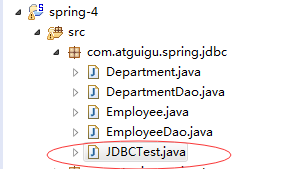
Video22.用xml文件配置AOP

Video 23. Spring对JDBC的支持

* + @Component: 基本注解, 标识了一个受 Spring 管理的组件
  + @Respository: 标识持久层组件
  + @Service: 标识服务层(业务层)组件

注意：

一共看两个即可：JDBCTemplate和NamedParameterJdbcTemplate,

具体用法见里的方法

Video25

1. mysql不支持check(stock > 0)

Video26.

针对V25里的问题

：@Override

**public** **void** purchase(String username, String isbn) {

// 1.查找

**int** price = bookShopDao.findBookPriceByIsbn(isbn);

// 2.更新库存

bookShopDao.updateBookStock(isbn);

// 3.更新余额

bookShopDao.updateUserAccount(username, price);

}

改进：

加入事务即可。

步骤：

1.在applicationContext.xml里加入

<!-- 配置事务管理器 -->

<bean id=*"transactionManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

<!-- 启动事务注解 -->

<tx:annotation-driven transaction-manager=*"transactionManager"*/>

1. //在Impl类里加入事务注解
2. Junit测试即可成功，具体参看代码

Video27.

当另外一个方法（或事务）调用本事务时，如何处理