2002 Rod Johnon <Expoer One-on-one j2eedevelopment and Design>

Spring 2003 ,IOC Aop

Spring data,spring boot,spring cloud,spring framework ,spring social

**IOC :控制反转 （DI：依赖注入）**

1.搭建Spring环境

下载jars

http://maven.springframework.org/release/org/springframework/spring/

spring-framework-4.3.9.RELEASE-dist.zip

开发spring至少需要使用的jar(5个+1个):

spring-aop.jar 开发AOP特性时需要的JAR

spring-beans.jar 处理Bean的jar <bean>

spring-context.jar 处理spring上下文的jar <context>

spring-core.jar spring核心jar

spring-expression.jar spring表达式

三方提供的日志jar

commons-logging.jar 日志

2.编写配置文件

为了编写时有一些提示、自动生成一些配置信息：

方式一：增加sts插件

可以给eclipse增加 支持spring的插件：spring tool suite(https://spring.io/tools/sts/all)

下载springsource-tool-suite-3.9.4.RELEASE-e4.7.3a-updatesite.zip,然后在Eclipse中安装：Help-Install new SoftWare.. - Add

方式二：

直接下载sts工具（相当于一个集合了Spring tool suite的Eclipse）: https://spring.io/tools/sts/

新建：bean configuration .. - applicationContext.xml

3.开发Spring程序(IOC)

ApplicationContext conext = new ClassPathXmlApplicationContext("applicationContext.xml") ;

//执行从springIOC容器中获取一个 id为student的对象

Student student = (Student)conext.getBean("student") ;

可以发现，springioc容器 帮我们new了对象，并且给对象赋了值

SpringIOC发展史：

1.

Student student = new Student();

student.setXxx();

2.

简单工厂

3.ioc （超级工厂）

IOC（控制反转）也可以称之为DI（依赖注入）：

控制反转：将 创建对象、属性值 的方式 进行了翻转，从new、setXxx() 翻转为了 从springIOC容器getBean()

依赖注入：将属性值 注入给了属性，将属性 注入给了bean，将bean注入给了ioc容器；

总结：ioc/di ，无论要什么对象，都可以直接去springioc容器中获取，而不需要自己操作（new\setXxx()）

因此之后的ioc分为2步：1 先给springioc中存放对象并赋值 2 拿

DI:依赖注入 ，

Teacher

Course : cname teacher

IOC容器赋值：如果是简单类型（8个基本+String），value；

如果是对象类型，ref="需要引用的id值"，因此实现了 对象与对象之间的依赖关系

conext.getBean(需要获取的bean的id值)

依赖注入3种方式：

1.set注入：通过setXxx()赋值

赋值，默认使用的是 set方法();

依赖注入底层是通过反射实现的。

<property...>

2.构造器注入：通过构造方法赋值

<constructor-arg value="ls" type="String" index="0" name="name"></constructor-arg>

需要注意：如果 <constructor-arg>的顺序 与构造方法参数的顺序不一致，则需要通过type或者index或name指定。

3.p命名空间注入

引入p命名空间

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

<bean id="course" class="org.lanqiao.entity.Course" p:courseHour="300" p:courseName="hadoop" p:teacher-ref="teacher">

简单类型：

p:属性名="属性值"

引用类型（除了String外）：

p:属性名-ref="引用的id"

注意多个 p赋值的时候 要有空格。

注意：

无论是String还是Int/short/long，在赋值时都是 value="值" ，

因此建议 此种情况 需要配合 name\type进行区分

示例：

注入各种集合数据类型: List Set map properties

set、list、数组 各自都有自己的标签<set> <list> <array>，但是也可以混着用

给对象类型赋值null ：

<property name="name" >

<null/> -->注意 没有<value>

</property>

赋空值 ""

<property name="name" >

<value></value>

</property>

在ioc中定义bean的前提：该bean的类 必须提供了 无参构造

自动装配（只适用于 ref类型 ）：

约定优于配置

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。

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

**1.使用注解实现事务（声明式事务）**

目标：通过事务 使以下方法 要么全成功、要么全失败

public void addStudent()

{

//增加班级

//增加学生

//crdu

}

a. jars

spring-tx-4.3.9.RELEASE

ojdbc.jar

commons-dbcp.jar 连接池使用到数据源

commons-pool.jar 连接池

spring-jdbc-4.3.9.RELEASE.jar

aopalliance.jar

b.配置

jdbc\mybatis\spring

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver"></property>

<property name="url" value="jdbc:mysql://localhost:3306/testjdbc"></property>

<property name="username" value="root"></property>

<property name="password" value="9520"></property>

<property name="maxActive" value="10"></property>

<property name="maxIdle" value="6"></property>

</bean>

<!-- 配置事务管理器txManager -->

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

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

</bean>

<!-- 增加对事务的支持 -->

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

c.使用

将需要 成为事务的方法 前增加注解：

@Transactional(readOnly=false,propagation=Propagation.REQUIRED)

**2.AOP：面向方面编程**

一个普通的类 -> 有特定功能的类

a.继承类 b.实现接口 c.注解 d.配置

public class MyFilter exntends/implements Xx

{

}

类 -> “通知” ：实现接口

前置通知实现步骤：

a.jar

aopaliance.jar

aspectjweaver.jar

b.配置

c.编写

aop：每当之前add()之前 自动执行一个方法log();

addStudent(); 业务方法（IStudentService.java中的 addStudent()）

before(); 自动执行的通知，即aop前置通知

public class Xxx

{

@Test

a(){}

}

如果出现异常：类似java.lang.NoClassDefFoundError: org/apache/commons/pool/impl/GenericObjectPool

则说明缺少jar

后置通知：

a.通知类 ，普通实现接口

b.业务类、业务方法

StudentServiceImpl中的addStudent()

c.配置：

将业务类、通知 纳入springIOC容器

定义切入点（一端）、定义通知类（另一端），通过pointcut-ref将两端连接起来

异常通知：

根据异常通知接口的定义可以发现，异常通知的实现类 必须编写以下方法：

public void afterThrowing([Method, args, target], ThrowableSubclass)：

a.public void afterThrowing(Method, args, target, ThrowableSubclass)

b.public void afterThrowing( ThrowableSubclass)

环绕通知： 在目标方法的前后、异常发生时、最终等各个地方都可以 进行的通知，最强大的一个通知；

可以获取目标方法的 全部控制权（目标方法是否执行、执行之前、执行之后、参数、返回值等）

在使用环绕通知时，目标方法的一切信息 都可以通过invocation参数获取到

环绕通知 底层是通过拦截器实现的。

二、实现注解实现 通知 ,aop

a.jar

与 实现接口 的方式相同

b.配置

将业务类、通知 纳入springIOC容器

开启注解对AOP的支持<aop:aspectj-autoproxy></aop:aspectj-autoproxy>

业务类 addStudent - 通知

c.编写

通知：

@Aspect //声明该类 是一个 通知

public class LogBeforeAnnotation {

}

注意：通过注解形式 将对象增加到 ioc容器时，需要设置 扫描器

<context:component-scan base-package="org.lanqiao.aop"></context:component-scan>

扫描器 会将 指定的包 中的 @Componet @Service @Respository @Controller修饰的类产生的对象 增加到IOC容器中

@Aspect不需要 加入扫描器，只需要开启即可：<aop:aspectj-autoproxy></aop:aspectj-autoproxy>

通过注解形式 实现的aop，如果想获取 目标对象的一些参数，则需要使用一个对象：JointPoint

注解形式的返回值：

a.声明返回值 的参数名：

@AfterReturning( pointcut= "execution(public \* addStudent(..))" ,returning="returningValue" )

public void myAfter(JoinPoint jp,Object returningValue) {//returningValue是返回值，但需要告诉spring

System.out.println("返回值："+returningValue );

注解形式实现aop时，通知的方法的参数不能多、少

实现接口形式、注解形式 只捕获声明的特定类型的异常，而其他类型异常不捕获。

cath()

三、通过 配置将 类->通知

基于Schema配置

类似 与 实现接口的方式

接口方式通知：public class LogAfter implements AfterReturningAdvice

Schema方式通知：

a.编写一个普通类 public class LogAfter {}

b.将该类 通过配置，转为一个“通知”

如果要获取目标对象信息：

注解、schema：JoinPoint

接口：Method method, Object[] args, Object target

schema形式 和注解形式相似，不同之处： 注解形式 使用了注册@After， schmema形式进行了多余的配置