一.Spring 框架简介及官方压缩包目录

介绍

1.主要发明者:Rod Johnson

2.轮子理论推崇者:

2.1 轮子理论:不用重复发明轮子.

2.2 IT 行业:直接使用写好的代码.

3.Spring 框架宗旨:不重新发明技术,让原有技术使用起来更加方便.

4.Spring 几大核心功能

4.1 IoC/DI 控制反转/依赖注入

4.2 AOP 面向切面编程

4.3 声明式事务.

5.Spring 框架 runtime

5.1 test: spring 提供测试功能

5.2 Core Container:核心容器.Spring 启动最基本的条件.

5.2.1 Beans : Spring 负责创建类对象并管理对象

5.2.2 Core: 核心类

5.2.3 Context: 上下文参数.获取外部资源或这管理注解等

5.2.4 SpEl: expression.jar

5.3 AOP: 实现 aop 功能需要依赖

5.4 Aspects: 切面 AOP 依赖的包

5.5 Data Access/Integration : spring 封装数据访问层相关内容

5.5.1 JDBC : Spring 对 JDBC 封装后的代码.

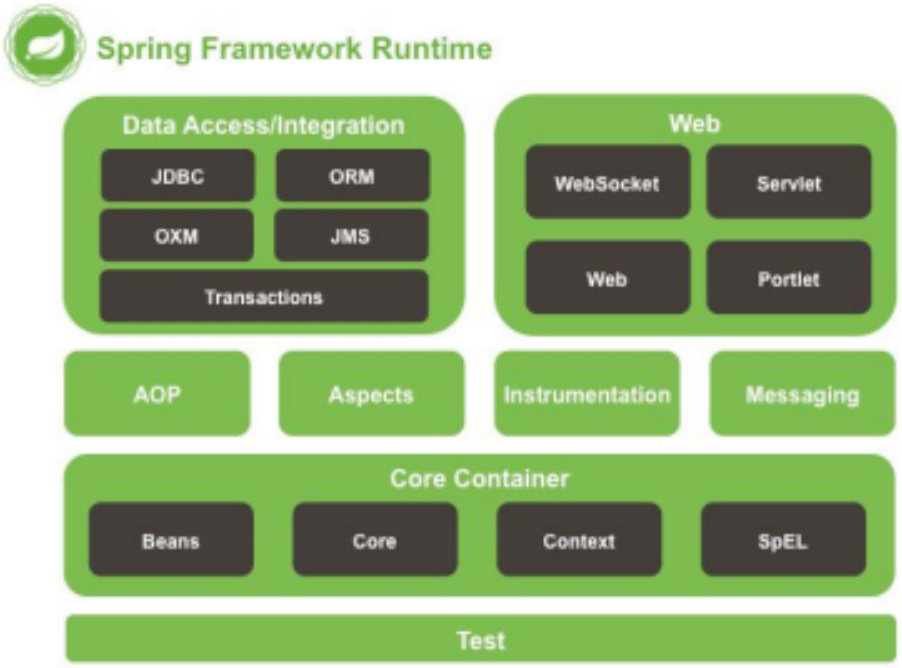
5.5.2 ORM: 封装了持久层框架的代码.例如 Hibernate

5.5.3 transactions:对应 spring-tx.jar,声明式事务使用.

5.6 WEB:需要 spring 完成 web 相关功能时需要.

5.6.1 例如:由 tomcat 加载 spring 配置文件时需要有 spring-web

包



6.Spring 框架中重要概念

6.1 容器(Container): Spring 当作一个大容器.

6.2 BeanFactory 接口.老版本.

6.2.1 新版本中 ApplicationContext 接口,是 BeanFactory 子接口.BeanFactory 的功能在 ApplicationContext 中都有.

7.从 Spring3 开始把 Spring 框架的功能拆分成多个 jar.

7.1 Spring2 及以前就一个 jar

二.IoC

1.中文名称:控制反转

2.英文名称:(Inversion of Control)

3.ioC是什么?

3.1ioC完成的事情就是原先由程序员主动通过new实例化对象事情转交给Spring负责

3.2控制反转中控制指的是:控制类的对象

3.3控制反转中反转指的是转交给Spring负责

3.4ioC最大的作用:解耦

3.4.1程序员不需要管理对象,解除了对象管理和程序员之间的耦合

三.环境搭建

1.导入jar包

1.1四个核心一个log

2.在src下新建applicationContext.xml

2.1文件名称和路径自定义

2.2记住Spring容器ApplicationContext,applicationContext.xml配置的信息最终都

存储到了ApplicationContext容器中

2.3spring配置文件是基于schema

2.3.1schema文件扩展名是.xsd

2.3.2把schema理解成DTD的升级版

2.3.2.1比DTD具备了更好的扩展性

2.3.3每次引入一个xsd文件时是namespace(xmlns)

2.4配置文件中只需要引入基本schema

xmlns　　xml NameSpace : xml文件命名空间

xsi　　xml schema instance : xml Schema 实例

xsi:schemaLocation　　指定命名空间 + Schema文件的位置

xmlns:alias　　xmlns:alias : 命名空间的别名

```
<?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.xsd">
</beans>
```

四.Spring创建对象的三种方式

1.通过无参方法创建

1.1无参构造创建:默认情况

1.2有参构造创建:需要明确配置

1.2.1需要在类中提供有参构造方法

1.2.2在applicationContext.xml中设置调用哪个构造方法创建对象

1.2.2.1如果设定的条件匹配多个构造方法执行最后的构造方法

1.2.2.2index:参数的索引,从0开始

1.2.2.3name:参数名

1.2.2.4type:类型(区分开关键字和封装类int 和 Integer)

```
<bean id="peo" class="com.Spring1.pojo.People">
  <!--ref引用另一个bean value基本数据类型或者String等-->
  <constructor-arg index="0" name="id" type="int" value="123"></constructor-arg>
  <constructor-arg index="1" name="name" type="java.lang.String" value="张三"></constructor-arg>
</bean>
```

2.实例工厂

2.1工厂设计模式:帮助创建类对象,一个工厂可以生产多个对象

2.2实例工厂:需要先创建工厂,才能生产对象

2.3实现步骤

2.3.1必须要有一个实例工厂

2.3.2在applicationContext.xml中配置工厂对象和需要创建的对象

```
<bean id="factory" class="com.Spring1.pojo.PeopleFactory"></bean>
<bean id="peo1" factory-bean="factory" factory-method="newInstance"></bean>
```

3.静态工厂

```
<bean id="peo2" class="com.Spring1.pojo.PeopleFactory" factory-method="newInstance"></bean>
```

五.如何给Bean的属性赋值(注入)

1.通过构造方法设置值

2.设置注入(通过set方法)

2,1如果属性是基本数据类型或String类型

```
<bean id="peo" class="com.Spring2.Pojo.People">
   <property name="id" value="222"></property>
   <property name="name" value="张三"></property>
</bean>
```

2.2如果属性是Set<?>类型或者List<?>类型或者数组或者Map

2.3如果属性是一个properties

六.DI

1.中文名称:依赖注入

2.英文名称:Dependency Injection

3.DI是什么?

3.1DI和IoC是一样的

3.2当一个类(A)中需要依赖另一个类(B)对象时,把B赋值给A的过程叫做依赖注入

七.使用Spring简化MyBatis

1.导入mybatis所有jar包和spring基本包,spring-jdbc,spring-tx,spring-aop,

spring整合mybatis的包

2.配置applicationContext.xml配置文件

3.编写代码

3.1正常编写pojo

3.2编写mapper包下时必须使用接口绑定方案,或注解方案(必须有接口)

3.3正常编写service接口和Service实现类

3.3.1需要在Service实现类声明Mapper接口对象,并生成get/set方法

3.3.2spring无法管理Servlet

七.AOP(OOP)

1.AOP:中文名称面向切面编程

2.英文名称(Aspect Oriented Programming)

3.正常程序执行流程都是纵向执行流程

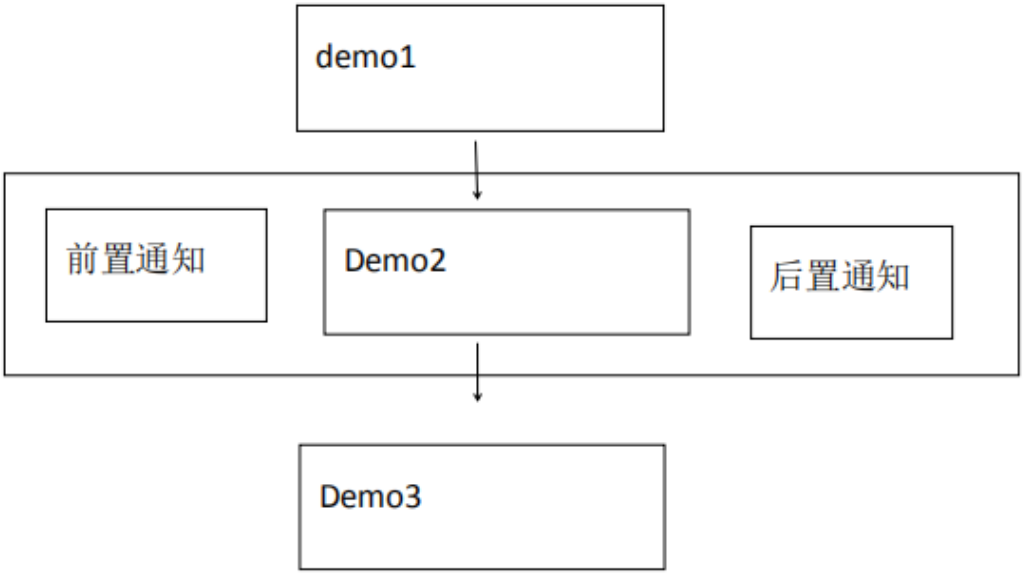
demo1--->demo2--->demo3

3.1又叫面向切面编程,在原有纵向执行流程中添加横切面

3.2不需要修改原有程序代码(体现出程序高扩展性)

3.2.1高扩展性

3.2.2原有功能相当于释放了部分逻辑.让职责更加明确



4.面向切面编程是什么?

4.1在程序原有纵向执行流程中,针对某一个或某一些方法添加通知,形成横切面

过程就叫做面向切面编程

5.常用概念

5.1原有功能:切点,pointcut

5.2前置通知:在切点之前执行的功能:before advice

5.3后置通知:在切点之后执行的功能:after advice

5.4如果切点执行过程中出现异常,会触发异常通知.throws advice

5.5所有功能总称叫做切面

5.6织入:把切面嵌入到原有功能的过程叫做织入

6.spring提供了2中AOP实现方式

6.1Schema-based

6.1.1每个通知都需要实现接口或类

6.1.2配置spring配置文件时在<aop:config>配置

6.2AspectJ

6.2.1每个通知不需要实现接口或类

6.2.2配置spring配置文件是在<aop:config>的子标签<aop:aspect>中配置

6.3Schema-based实现步骤

6.3.1导入jar包(aopalliance.jar aspectjweaver.jar)

6.3.2新建通知类

6.3.2.1新建前置通知类(MethodBeforeAdvice)

method:切点方法对象Method对象

objects:切点方法参数

o:切点方法所在的类的对象

```
public void before(Method method, Object[] objects, Object o) throws Throwable
```

6.3.2.2新建后置通知类(AfterReturningAdvice)

o:切点方法的返回值

method:切点方法对象

objects:切点方法的参数

o1:切点方法所在的类的对象

```
public void afterReturning(Object o, Method method, Object[] objects, Object o1) throws Throwable
```

6.3.3配置spring配置文件

6.3.2.1引入aop命名空间以及schemalocation

6.3.2.2配置通知类对象以及配置切面

6.3.2.3*是通配符匹配任意方法名,任意类名,任意一级包名

6.3.2.3如果希望匹配任意方法参数 (..)

```
execution(* com.Spring3.*.service.impl.*.*(..) )
```

```
<?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:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd">
<!--配置通知类对象,在切面引入-->
<bean id="beforeAdvice" class="com.Spring3.MyBeforeAdvice"></bean>
<bean id="afterAdvice" class="com.Spring3.MyAfterAdvice"></bean>
<bean id="demo" class="com.Spring3.Demo"></bean>
<!--配置切面-->
<aop:config>
   <!--配置切点-->
   <aop:pointcut id="mypoint" expression="execution(*
com.Spring3.Demo.demo02() )"/>
   <!--配置通知-->
   <aop:advisor advice-ref="beforeAdvice" pointcut-ref="mypoint">
</aop:advisor>
   <aop:advisor advice-ref="afterAdvice" pointcut-ref="mypoint"></aop:advisor>
</aop:config>
</beans>
```

6.4配置异常通知的步骤(AspectJ方式)

6.4.1只有当切点报异常才能触发异常通知

6.4.2在spring中只有AspectJ方式提供了异常通知的方法

6.4.3实现步骤

6.4.3.1新建类,在类中写任意名称的方法

```
public class MyThrowAdvice  {
   public void myexception(Exception e1){
      System.out.println("执行异常通知,异常message:"+e1.getMessage());
   }
}
```

6.4.3.2在applicationContext.xml中配置

<aop:aspect>的 ref 属性表示:方法在哪个类中.

<aop: xxxx/> 表示什么通知

method: 当触发这个通知时,调用哪个方法

throwing: 异常对象名,必须和通知中方法参数名相同(可以不在通知中声明异常对象)

```
<aop:config>
    <aop:aspect ref="mythrow">
        <aop:pointcut id="mypoint" expression="execution(*
com.Spring3.Demo.demo01())"></aop:pointcut>
        <aop:after-throwing method="myexception" pointcut-ref="mypoint"
throwing="e1" ></aop:after-throwing>
    </aop:aspect>
</aop:config>
```

6.5配置异常通知(Schema-base方式)

6.5.1新建一个类实现ThrowsAdvice接口

6.5.1.1必须自己写方法,且必须叫afterThrowing

6.5.1.2有两种参数方式(必须是1个或4个)

6.5.1.3异常类型必须要与切点报的异常一致

```
public void afterThrowing(Exception ex) throws Throwable{
    System.out.println("执行异常通知-schema-base方式");
}

public void afterThrowing(Method m, Object[] args, Object target, Exception ex) {
    System.out.println("执行异常通知");
}
```

6.6.环绕通知(Schema-based方式)

6.6.1把前置通知和后置通知都写到一个通知中,组成环绕通知

6.6.2实现步骤

6.6.2.1新建一个类实现MethodInterceptor(拦截器)

```
public Object invoke(MethodInvocation methodInvocation) throws Throwable {
    System.out.println("环绕-前置");
    Object result=methodInvocation.proceed();  //放行,调用切点方式
    System.out.println("环绕-后置");
    return result;
}
```

6.5使用注解(基于Aspect)

6.5.2引入context名称空间

6.5.1spring不会自动去寻找注解,必须告诉spring哪些包下类的中可能有注

解

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

6.5.2@Component

6.5.2.1相当于<bean/>

6.5.2.2如果没有参数,把类名首字母变小写,相当于<bean id=""/>

6.5.2.3@Component("自定义名称")

6.5.3实现步骤

6.5.3.1在spring配置文件中设置注解在哪些包中

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

6.5.3.2在Demo类中添加@Component

6.5.3.3在方法上添加@Poiuntcut("")定义切点

```
@Pointcut(value = "execution(public void
com.aop.annotation.Demo.demo1(String,int) ) && args(name,id)")
```

6.5.3.3在通知类中配置

@Component类被spring管理

@Aspect相当于<aop:aspect/>表示通知方法在当前类中

八.自动注入

8.1在spring配置文件中对象名和ref="id"id名相同使用自动注入,可以不配置property

8.2两种配置方法

8.2.1在<bean>中通过autowire=""配置,只对这个<bean>生效

8.2.2在<beans>中通过default-autowire=""配置,表当当前文件中所有<bean>都是

全局配置内容

8.2.3autowire=""可取值

a. default: 默认值,根据全局 default-autowire=”” 值.默认全局和局

部都没有配置情况下,相当于 no

b. no: 不自动注入3.3 byName: 通过名称自动注入.在 Spring 容器中找类id

c. byType: 根据类型注入. byName:根据名称注入

d.spring 容器中不可以出现两个相同类型的<bean>

e.constructor: 根据构造方法注入.

f.提供对应参数的构造方法(构造方法参数中包含注入对戏那个)

g.底层使用 byName, 构造方法参数名和其他<bean>的 id相同.

九.Spring中加载properties文件

9.1在src下新建xxx.properties文件

9.2在spring配置文件中先引入xmlns:context,在下面添加

9.2.1如果配需要配置多个配置文件,逗号分隔

```
<context:property-placeholder location="classpath:com/login/db.properties"/>
```

9.3添加了属性文件记载,并且在<beans>中开启自动注入注意的地方

9.3.1用sqlSessionFactoryBeanName属性

9.4在被Spring管理的类中通过@Value("${key}")取出properties中内容

9.4.1添加注解扫描

```
<context:component-scan base-package="com.login.service">
</context:component-scan>
```

9.4.2在类中添加

a.key和变量名不必相同

b.变量名可以任意,只要保证key对应的value能转换成这个类型就可以

```
@Value("${my.demo}")
private String test;
```

十.scope属性

1.<bean>的属性

2.作用:控制对象有效范围(单例,多例)

3.<bean/>标签对应的对象默认是单例的

3.1无论获取多少次,都是同一个对象

4.scope可取值

4.1singleton 默认值，单例

4.2prototype 多例,每次请求重新实例化

4.3request 每次请求重新实例化

4.4session 每次会话对象时,对象是单例的

4.5application 在application对象内是单例的

4.6global session spring推出的一个对象,依赖于spring-webmvc-portlet,类

似于session

十一.声明式事务

1.编程式事务

1.1由程序员编程事务控制代码

1.2OpenSessionInView编程式事务

2.声明式事务

2.1事务控制代码已经由spring写好.程序员只需要声明出那些方法需要进行事务

控制和如何进行事务控制

3.声明式事务都是针对于ServiceImpl类下方法的

4.事务管理器基于通知(advice)的

5.在spring配置文件中配置声明式事务

```
<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
   <property name="dataSource" ref="datasource"></property>
</bean>
<!--配置声明事务-->
<tx:advice id="txAdvice"  transaction-manager="txManager" >
   <tx:attributes>
```

```
        <!--哪些方法需要有事务管理-->
        <!--方法以ins开头事务管理-->
        <tx:method name="ins*" />
        <tx:method name="del*"/>
        <tx:method name="upd*"/>
        <tx:method name="*" read-only="true"/>
    </tx:attributes>
</tx:advice>
<aop:config>
    <aop:pointcut id="mypoint" expression="execution(*
com.Transaction.Service.impl.*.*(..))"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="mypoint"></aop:advisor>
</aop:config>
```

6. 属性解释

6.1name=""哪些方法需要有事务管控

a. 支持通配符

6.2readonly="boolean" 是否是只读事务

a. 如果为true,告诉数据库此事务为只读事务.数据化优化,会对性能有一定

提升,所以只要是查询的方法,建议使用此数据

b. 如果为false(默认),事务需要提交的事务,建议新增,删除,修改

6.3propagation控制事务传播行为(数据库层面)

a. 当一个具有事务控制的方法,方法被另一个有事务控制的方法调用后,需要

如何管理事务(新建事务?在事务中执行?把事务挂起?报异常?)

b. REQUIRED(默认值):如果当前有事务,就在事务中执行,如果当前没有事务

新建一个

c. SUPPORTS:如果当前有事务就在事务中执行,如果当前没有事务,就在非事

务状态下执行

d. MANDATORY:如果当前有事务就在事务中执行,如果当前没有事务,就报错

e. REQUIRED_NEW:必须在事务中执行,如果当前没有事务,新建事务,如果当

前有事务,把当前事务挂起.

f.NOT_SUPPORTED:必须在非事务下执行,如果当前没有事务,正常执行,如果

当前有事务,把当前事务挂起

g.NEVER:必须在非事务下执行,如果当前没有事务,正常执行,如果当前有事务

,报错

h.NESTED:必须在事务状态下执行,如果没有事务,新建事务,如果当前有

事务,创建一个嵌套事务

6.4isolation 事务隔离级别

6.4.1在多线程或并发访问下如何保证访问到数据具有完整性

6.4.2脏读:(读取未提交数据)

一个事务(A)读取到另一个事务(B)中未提交的数据,另一个事务中数据

可能进行了改变,此时A事务读取的数据可能和数据库数据不一致的,此

时认为数据时脏数据,读取脏数据过程叫做脏读

6.4.3不可重复读(前后多次读取,数据内容不一致)

a.主要针对的是某行数据(或行中某列)

b.主要针对的操作时修改操作

c.两次读取在同一次事务内

d.当事务A第一次读取事务后,事务B对事务A读取的数据进行修改,事务A

中再次读取的数据和之前读取的数据不一致,过程不可重复读

6.4.4幻读(前后多次读取,数据总量不一致)

a.主要针对的操作是新增或删除

b.两次事务的结果

c.事务 A 按照特定条件查询出结果,事务 B 新增了一条符合条件的数据.

事务 A 中查询的数据和数据库中的数据不一致的,事务 A 好像出现了幻

觉,这种情况称为幻读.

6.4.5DEFAULT:默认值,由底层数据库自动判断应该使用什么隔离级别

6.4.5READ_UNCOMMITTED:可以读取未提交数据,可能出现脏读,不可重复

读,幻读,效率最高

6.4.6 READ_COMMITTED:只能读取其他事务已提交数据.可以防止

脏读,可能出现不可重复读和幻读.

6.4.7REPEATABLE_READ: 读取的数据被添加锁,防止其他事务修改此数据,可以防止不可重复读.脏读,可能出现幻读.

6.4.8 SERIALIZABLE: 排队操作,对整个表添加锁.一个事务在操作数

据时,另一个事务等待事务操作完成后才能操作这个表.

6.5.rollback-for="异常类型全限定路径"

```
<tx:method name="upd*" propagation="NESTED" rollback-for="java.lang.Exception"/>
```

6.5.1.当出现什么异常时需要进行回滚

6.5.2.建议:给定该属性值

a.手动抛出异常一定要给定该属性值

6.6no-rollback-for=""

6.6.1当出现什么异常时不回滚事务

十一.Spring常用注解

1.Component创建类对象,相当于配置<bean/>

2.Service与@Component功能相同

2.1写在ServiceImpl类上

3.Repository与@Component功能相同

2.1写在数据访问层类上

4. Controller与@Component

4.1写在控制器类上

5. Resource(不需要写set,get方法)

5.1java中的注解

5.2默认按照byName注入,如果没有名称,按照byType注入

a.建议对象名和spring容器中对象名相同

6. AutoWired(不需要写set/get)

6.1spring的注解

6.2默认按照byType注入

7. value()获取properties文件中的内容

8. PointCut()定义切点

9. Aspect()定义切面类

10. Before前置通知

11. After后置通知

12. AfterReturning后置通知,必须切点正确执行

13. AfterThrowing异常通知

14. Arround:环绕通知