ooa(object oriented analysis)面向对象分析
ood(object oriented design)面向对象设计
Oop 面向对象编程
Spring javabean 的生命周期
配置　初始化 调用 销毁

Spring

1、spring 原理

2、IoC 概念：控制权由对象本身转向容器：由容器根据配置文件区创建实例并创建各个实例之间依赖关系。

spring 的最大作用 ioc/di,将类与类的依赖关系写在配置文件中，

程序在运行时根据配置文件动态加载依赖的类，降低的类与类之间

的藕合度。它的原理是在 applicationContext.xml 加入 bean 标记,

在 bean 标记中通过 class 属性说明具体类名、通过 property 标签说明

该类的属性名、通过 constructor-args 说明构造子的参数。其一切都是

返射，当通过 applicationContext.getBean("id 名称")得到一个类实例时，

就是以 bean 标签的类名、属性名、构造子的参数为准，通过反射实例对象，

唤起对象的 set 方法设置属性值、通过构造子的 newInstance 实例化得到对象。

正因为 spring 一切都是反射，反射比直接调用的处理速度慢，所以这也是 spring

的一个问题。

spring 第二大作用就是 aop，其机理来自于代理模式，代理模式

有三个角色分别是通用接口、代理、真实对象

代理、真实对象实现的是同一接口，将真实对象作为

代理的一个属性，向客户端公开的是代理，当客户端

调用代理的方法时，代理找到真实对象，调用真实对象

方法，在调用之前之后提供相关的服务，如事务、安全、

日志。其名词分别是代理、真实对象、装备、关切点、连接点。

2、动态代理:不用写代理类，虚拟机根据真实对象实现的接口产生一个类，通过

类实例化一个动态代理，在实例化动态代理时将真实对象

及装备注入到动态代理中，向客户端公开的是动态代理，

当客户端调用动态代理方法时，动态代理根据类的返射得

到真实对象的 Method,调用装备的 invoke 方法，将动态代理、

Method、方法参数传与装备的 invoke 方法，invoke 方法在唤

起 method 方法前或后做一些处理。

1、产生动态代理的类:

java.lang.refect.Proxy

2、装备必须实现 InvocationHandler 接口实现 invoke 方法

3、反射

什么是类的返射?

通过类说明可以得到类的父类、实现的接口、内部类、构造函数、方法、属性

并可以根据构造器实例化一个对象，唤起一个方法，取属性值，改属性值。

如何得到一个类说明?

Class cls=类.class;

Class cls=对象.getClass();

Class.forName(“类路径”);

如何得到一个方法并唤起它?

Class cls=类.class;

Constructor cons=cls.getConstructor(new Class[]{String.class});

Object obj=cons.newInstance(new Object[]{“aaa”});

Method method=cls.getMethod(“方法名”,new Class[]{String.class,Integer.class});

method.invoke(obj,new Object[]{“aa”,new Integer(1)});

4、spring 的三种注入方式是什么?

setter

interface

constructor

5、spring 的核心接口及核类配置文件是什么?

FactoryBean:工厂 bean 主要实现 ioc/di

ApplicationContext ac=new FileXmlApplicationContext(“applicationContext.xml”);

Object obj=ac.getBean(“id 值”);

applicationContext.xml

Struts2

一个请求在 Struts2框架中的处理大概分为以下几个步骤：

1 客户端初始化一个指向 Servlet 容器（例如 Tomcat）的请求；
2 这个请求经过一系列的过滤器（Filter）（这些过滤器中有一个叫做 ActionContextCleanUp 的可选过滤器，这个过滤器对于 Struts2和其他框架的集成很有帮助，例如：SiteMesh Plugin）
3 接着 FilterDispatcher 被调用，FilterDispatcher 询问 ActionMapper 来决定这个请是否需要调用某个 Action
4 如果 ActionMapper 决定需要调用某个 Action，FilterDispatcher 把请求的处理交给 ActionProxy
5 ActionProxy 通过 Configuration Manager 询问框架的配置文件，找到需要调用的 Action 类
6 ActionProxy 创建一个 ActionInvocation 的实例。
7 ActionInvocation 实例使用命名模式来调用，在调用 Action 的过程前后，涉及到相关拦截器（Intercepter）的调用。
8 一旦 Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果。返回结果通常是（但不总是，也可 能是另外的一个 Action 链）一个需要被表示的 JSP 或者 FreeMarker 的模版。在表示的过程中可以使用 Struts2 框架中继承的标签。在这个过程中需要涉及到 ActionMapper

在上述过程中所有的对象（Action，Results，Interceptors，等）都是通过 ObjectFactory 来创建的。

Struts2的目标很简单–使 Web 开发变得更加容易。为了达成这一目标，Struts2中提供了很多新特性，比如智能的默认设置、 annotation 的使用以及"惯例重于配置"原则的应用，而这一切都大大减少了 XML 配置。Struts2中的 Action 都是 POJO，这一方面增 强了 Action 本身的可测试性，另一方面也减小了框架内部的耦合度，而 HTML 表单中的输入项都被转换成了恰当的类型以供 action 使用。开发人员还 可以通过拦截器（可以自定义拦截器或者使用 Struts2提供的拦截器）来对请求进行预处理和后处理，这样一来，处理请求就变得更加模块化，从而进一步减 小耦合度。模块化是一个通用的主题–可以通过插件机制来对框架进行扩展；开发人员可以使用自定义的实现来替换掉框架的关键类，从而获得框架本身所不具备 的功能；可以用标签来渲染多种主题（包括自定义的主题）；Action 执行完毕以后，可以有多种结果类型–包括渲染 JSP 页面，Velocity 和 Freemarker 模板，但并不仅限于这些。最后，依赖注入也成了 Struts2王国中的一等公民，这项功能是通过 Spring 框架的插件和 Plexus 共同提供的，与 PicoContainer 的结合工作还正在进行中

Struts 2设计的精巧之处就是使用了 Action 代理，Action 代理可以根据系统的配置，加载一系列的拦截器，由拦截器将 HttpServletRequest 参数解析出来，传入 Action。同样，Action 处理的结果也是通过拦截器传入 HttpServletResponse，然后由 HttpServletRequest 传给用户。
其实，该处理过程是典型的 AOP（面向切面编程）的方式，读者可以在后面详细了解到。Struts 2处理过程模型如图3.2所示。

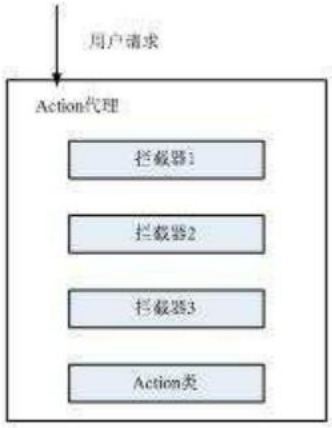

图3.2 Struts 2处理过程模型

★ 说明 ★

拦截器是 Struts 2框架的核心，通过拦截器，实现了 AOP（面向切面编程）。使用拦截器，可以简化 Web 开发中的某些应用，例如，权限拦截器可以简化 Web 应用中的权限检查。

业务控制器 Action 是由开发者自己编写实现的，Action 类可以是一个简单的 Java 类，与 Servlet API 完全分离。Action 一般都有一个 execute()方法，也可以定义其他业务控制方法，详细内容将在后面介绍。

Action 的 execute()返回一个 String 类型值，这与 Struts 1返回的 ActionForward 相比，简单易懂。Struts 2提供了一个 ActionSupport 工具类，该类实现了 Action 接口和 validate()方法，一般开发者编写 Action 可以直接继承 ActionSupport 类。编写 Action 类后，开发者还必须在配置文件中配置 Action。一个 Action 的配置应该包含下面几个元素：

— 该 Action 的 name，即用户请求所指向的 URL。

— Action 所对应的 class 元素，对应 Action 类的位置。

— 指定 result 逻辑名称和实际资源的定位。

Action 是业务控制器，笔者建议在编写 Action 的时候，尽量避免将业务逻辑放到其中，尽量减少 Action 与业务逻辑模块或者组件的耦合程度。

业务模型组件可以是实现业务逻辑的模块，可以是 EJB、POJO 或者 JavaBean，在实际开发中，对业务模型组件的区分和定义也是比较模糊的，实际上 也超出了 Struts 2框架的范围。不同的开发者或者团队，都有自己的方式来实现业务逻辑模块，Struts 2框架的目的就是使用 Action 来调用业务逻辑模块。例如一个银行存款的业务逻辑模块，如代码3.3所示。

代码3.3 模拟一个银行业务的实现模块

```
package ch3;
public class Bank {
//定义银行账户
private String accounts;
//定义操作金额
private double money;
//属性的 getter 和 setter 方法
public String getAccounts() {
return accounts;
}
public void setAccounts(String accounts) {
```

```
this.accounts = accounts;
}
public double getMoney() {
return money;
}
public void setMoney(double money) {
this.money = money;
}
//模拟银行存款方法
public boolean saving(String accounts, double money) {
//调用 DAO 等模块读写数据库
return dosomeing();
}

    }
```

上面实例在实际开发中没有任何意义，这里只是作为业务逻辑模块来说明，在执行 saving(String accounts,double money)方法时，可以调用相应的数据库访问其他组件，来实现存款操作。使用 Action 调用该业务逻辑组件可以在 execute()方法中实现，如代 码3.4所示。

代码3.4 业务控制器 Bank_Saving_Action

```
package ch3;
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

    public class Bank_Saving_Action extends ActionSupport {
//定义银行账户
private String accounts;
//定义操作金额
private double money;

    public String execute() throws Exception {
//创建 Bank 实例
Bank bk=new Bank();
//调用存款方法
if (bk.saving(accounts, money)){
return SUCCESS;
}else{
return ERROR;
}
}
//属性的 getter 和 setter 方法
public String getAccounts() {
return accounts;
}

    public void setAccounts(String accounts) {
```

```
this.accounts = accounts;
}

    public double getMoney() {
return money;
}

    public void setMoney(double money) {
this.money = money;
}
```

Bank_Saving_Action 演示了对银行存款业务逻辑组件的调用，这里是通过在 Action 中创建业务逻辑组件实例的方式实现的。在实际开发 中，可以使用静态工厂获得业务逻辑组件的实例或者使用 IoC 容器来管理。Action 中不实现任何业务逻辑，只是负责组织调度业务逻辑组件。调用关系如图 3.3所示。

图3.3 调用业务逻辑组件

★ 说明 ★

业务控制器 Action 一般情况下不是直接创建业务逻辑组件实例，而是使用工厂模式或者是从 Spring 容器中获得业务逻辑组件实例，这样可以提高系统的性能。

Struts 1只能支持 JSP 作为视图资源，而 Struts 2的进步之处就是可以使用其他视图技术，如 FreeMarker、Velocity 等。通过前面的学习和示例，读者会知道 Action 的返回结果只是一个 简单的字符串，也就是一个逻辑上的视图名称，要与实际视图资源对应，必须通过配置文件来实现。

在 struts.xml 配置文件中，每一个 Aciton 定义都有 name 和 class 属性，同时还要指定 result 元素。result 元素指定了逻辑视 图名称和实际视图的对应关系。每个 result 都有一个 type 属性，前面介绍的 struts.xml 中并没有显式指定 type 值，即使用了默认的 type 类型：dispatcher，该结果类型支持 JSP 所谓视图资源。

Hibernate 和 ibatis 的区别总结

Hibernate 简介

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了非常轻量级的对象封装，使得 Java 程序员可以随心所欲的使用对象编 程思维来操纵数据库。Hibernate 可以应用在任何使用 JDBC 的场合，既可以在 Java 的客户端程序使用，也可以在 Servlet/JSP 的 Web 应用中使用，最具革命意义的是，Hibernate 可以在应用 EJB 的 J2EE 架构中取代 CMP，完成数据持久化的重任。

一、基本功能

Hibernate 作为数据持久化的中间件，足以让数据库在业务逻辑层开发中去冬眠。它通过可扩展标记语言（XML）实现了类和数据表之间的映射，使程序员在业务逻辑的开发中面向数据库而改为面向对象开发。使整个项目开发分工更加明确，提高了程序开发的效率。

configuration 对象：

Configuration 类负责管理 Hibernate 的配置信息。Hibernate 运行时需要

获取一些底层实现的基本信息，其中几个关键属性包括：

1． 数据库 URL

2． 数据库用户

3． 数据库用户密码

4． 数据库 JDBC 驱动类

5．数据库 dialect，用于对特定数据库提供支持，其中包含了针对特定数据库特性的实现，如 Hibernate 数据类型到特定数据库数据类型的映射等。

以上信息一般情况下由 hibernate.cfg.xml 或者 hibernate.properties 文件来配置，实现与不同数据库的连接。

Session 对象：

Session 是持久层操作的基础，相当于 JDBC 中的 Connection:

实例通过 SessionFactory 实例构建:

```
Configuration config = new Configuration().configure();
SessionFactory sessionFactory = config.buildSessionFactory();
Session session = sessionFactory.openSession();
```

之后我们就可以调用 Session 所提供的 save、find、flush 等方法完成持久层操作。因此 Session 对象也封装了所有对数据库的操作来实现 Hibernate 对数据库的操纵功能，如：

Save()方法实现增加和保存；

Delete()方法实现数据的删除；

Update()方法实现数据更新和修改；

Find()方法实现数据的检索；

Hibernate 会根据不同的操作自动生成相应的 SQL 语句，从而实现了程序员对 PO 对象的操作转化为对数据库关系表的操作。

二、使用步骤

1．编写 Hibernate 配置文件

Hibernate 配置文件有两种，分别是 hibernate.cfg.xml 文件和 hibernate.properties,推荐使用 hibernate.cfg.xml。

2．PO 和映射文件

使用 middlegen 和 hibernate-extensions 从数据库导出 PO 的映射文件，并在 hibernate.cfg.xml 当中声明。

3．编写 DAO

对每一张关系表编写一个 DAO，提供一组增、删、改、查方法供业务逻辑对数据库操作使用。

更多的细节请大家参阅 hibernate 的网站获取详细的信息。并在各自的实践和开发中加深体会。

Ibatis 简介

相对 Hibernate 和 Apache OJB 等”一站式”ORM 解决方案而言，ibatis 是一种”半自动化”的 ORM 实现。

所谓"半自动"，可能理解上有点生涩。纵观目前主流的ORM，无论Hibernate 还是Apache OJB，都对数据库结构提供了较为完整的封装，提供了从POJO 到数据库表的全套映射机制。程序员往往只需定义好了POJO 到数据库表的映射关系，即可通过Hibernate或者OJB 提供的方法完成持久层操作。程序员甚至不需要对SQL 的熟练掌握，Hibernate/OJB 会根据制定的存储逻辑，自动生成对应的SQL并调用JDBC 接口加以执行。

Ibatis 最直接的好处就是不但为程序员提供了对象与关系数据库之间的映射，同时提供操作方法与SQL 间的直接影射，设计者可以直接为一个方法指定一条SQL 语句，从而取得更加准确的数据，同时为优化查询、连接查询提供了方便。

一、基本功能

作为又一个轻量级的ORM 中间件，ibatis 除了提供了对数据库基本的增、删、改、查外还提供了连接管理，缓存支持，线程支持，（分布式）事物管理等一套教为完整的数据库管理功能。

SqlMapClient 对象是ibatis 持久层操作的基础，相当于hibernate 中的session，提供对SQL 映射的方法。

insert()方法实现对插入SQL 语句的映射；

delete()方法实现对删除SQL 语句的映射；

update()方法实现对更新SQL 语句的影射；

queryForList()、queryForMap()、queryForObject()、queryForPaginatedList()等方法提供了一组查询SQL语句的影射；

二、使用步骤

1．ibatis SQL Map 配置文件

文件中对所用数据库的连接做了基本配置，包括数据库驱动类型、用户名、密码，以及连接池的相关管理数据。

2．PO 和映射文件

和hibernate 一样，PO 作为数据库关系表的影射，也需要响应的映射配置文件，可以手写，也可以借助hibernate 的相关工具生成PO，不会影 响PO 在ibatis 中的使用。与hibernate 不同的是，ibatis 的映射文件中没有对PO 中每个属性做响应的描述，而是指定了一系列与PO 有关 的SQL 相关操作，也体现了ibatis 良好的灵活性与可扩展性。

3．编写DAO

在DAO 中，可以使用SqlMapClient 提供的方法来对应的指定对PO 操作的SQL 语句，从而使业务逻辑层的开发仍然是面向对象的操作。

选择Hibernate 还是iBATIS 都有它的道理：

Hibernate 的特点：

Hibernate 功能强大，数据库无关性好，O/R 映射能力强，如果你对Hibernate 相当精通，而且对Hibernate进行了适当的封 装，那么你的项目整个持久层代码会相当简单，需要写的代码很少，开发速度很快，非常爽。以数据库字段一一对应映射得到的PO 和Hibernte 这种对象化 映射得到的PO 是截然不同的，本质区别在于这种PO 是扁平化的，不像Hibernate 映射的PO 是可以表达立体的对象继承，聚合等等关系的，这将会直接 影响到你的整个软件系统的设计思路。Hibernate 对数据库结构提供了较为完整的封装，Hibernate 的O/R Mapping 实现了POJO 和数据库表之间的映射，以及SQL 的自动生成和执行。程序员往往只需定义好了POJO 到数据库表的映射关系，即可通过Hibernate 提供的方法完成持久层操作。程序员甚至不需要对SQL 的熟练掌握， Hibernate/OJB 会根据制定的存储逻辑，自动生成对应的SQL 并调用JDBC 接口加以执行。Hibernate 的缺点就是学习门槛不低，要精通门槛更高，而且怎么设计O/R 映射，在性能和对象模型之间如何权衡取得平衡，以及怎样用 好Hibernate 方面需要你的经验和能力都很强才行，但是Hibernate 现在已经是主流O/R Mapping 框架，从文档的丰富性，产品的完善性，版本的开发速度都要强于iBATIS。

iBATIS 的特点：

iBATIS 入门简单，即学即用，提供了数据库查询的自动对象绑定功能，而且延续了很好的 SQL 使用经验，对于没有那么高的对象模型要求的项目来 说，相当完美。iBATIS 的缺点就是框架还是比较简陋，功能尚有缺失，虽然简化了数据绑定代码，但是整个底层数据库查询实际还是要自己写的，工作量也比 较大，而且不太容易适应快速数据库修改。当系统属于二次开发,无法对数据库结构做到控制和修改,那 iBATIS 的灵活性将比 Hibernate 更适合。系 统数据处理量巨大，性能要求极为苛刻，这往往意味着我们必须通过经过高度优化的 SQL 语句（或存储过程）才能达到系统性能设计指标。在这种情况下 iBATIS 会有更好的可控性和表现。

对于实际的开发进行的比较：

1． iBATIS 需要手写 sql 语句，也可以生成一部分，Hibernate 则基本上可以自动生成，偶尔会写一些 Hql。同样的需求,iBATIS 的工作量比 Hibernate 要大很多。类似的，如果涉及到数据库字段的修改，Hibernate 修改的地方很少，而 iBATIS 要把那些 sql mapping 的地方一一修改。

2. iBatis 可以进行细粒度的优化

比如说我有一个表，这个表有几个或者几十个字段，我需要更新其中的一个字段，iBatis 很简单，执行一个 sql UPDATE TABLE_A SET column_1=#column_1# WHERE id=#id# 但是用 Hibernate 的话就比较麻烦了,缺省的情况下 hibernate 会更新所有字段。当然我记得 hibernate 有一个选项可以控制只保存修改过的字段，但是我不太确定这个功能的负面效果。

例如：我需要列出一个表的部分内容，用 iBatis 的时候，这里面的好处是可以少从数据库读很多数据，节省流量 SELECT ID, NAME FROM TABLE_WITH_A_LOT_OF_COLUMN WHERE …一般情况下 Hibernate 会把所有的字段都选出来。比如说有一个上面表有8个字段，其中有一两个比较大的字段，varchar(255)/text。上面的场景中我为什么要把他们 也选出来呢？用 hibernate 的话，你又不能把这两个不需要的字段设置为 lazy load，因为还有很多地方需要一次把整个 domain object 加载出来。这个时候就能显现出 ibatis 的好处了。如果我需要更新一条记录（一个对象），如果使用 hibernate，需要现把对象 select 出来，然后再做 update。这对数据库来说就是两条 sql。而 iBatis 只需要一条 update 的 sql 就可以了。减少一次与数据库的交互，对于性能的提升是非常重 要。

3. 开发方面：

开发效率上，我觉得两者应该差不多。可维护性方面，我觉得 iBatis 更好一些。因为 iBatis 的 sql 都保存到单独的文件中。而 Hibernate 在有些情况下可能会在 java 代码中保 sql/hql。相对 Hibernate“O/R”而言，iBATIS 是一种“Sql Mapping”的 ORM 实现。而 iBATIS 的着力点，则在于 POJO 与 SQL 之间的映射关系。也就是说，iBATIS 并不会为程序员在运行期自动生成 SQL 执行。具体的 SQL 需要程序员编写，然后通过映射配置文件，将 SQL 所需的参数，以及返回的结果字段映射到指定 POJO。使用 iBATIS 提供的 ORM 机制，对业务逻辑实现人员而言，面对的是纯粹的 Java 对象，这一层与通过 Hibernate 实现 ORM 而言基本一致，而对于具体的数据操作，Hibernate 会自动生成 SQL 语句，而 iBATIS 则要求开发者编写具体的 SQL 语句。相对 Hibernate 而言，iBATIS 以 SQL 开发的工作量和数据库移植性上的让步，为系统设计提供了更大的自由空间。