# 模拟面试卷二

## 数据库

### 什么是存储过程？它有什么优点？

答：存储过程是一组予编译的SQL语句，

它的优点有：

允许模块化程序设计，就是说只需要创建一次过程，以后在程序中就可以调用该过程任意次。

允许更快执行，如果某操作需要执行大量SQL语句或重复执行，存储过程比SQL语句执行的要快。

减少网络流量，例如一个需要数百行的SQL代码的操作有一条执行语句完成，不需要在网络中发送数百行代码。

更好的安全机制，对于没有权限执行存储过程的用户，也可授权他们执行存储过程。

### 存储过程和函数有什么区别？

Oracle中的函数与存储过程的**区别**:

A:函数必须有返回值,而过程没有.

B:函数可以单独执行.而过程必须通过execute执行.

C:函数可以嵌入到SQL语句中执行.而过程不行.

其实我们可以将比较复杂的查询写成函数.然后到存储过程中去调用这些函数.

Oracle中的函数与存储过程的**特点**:

A. 一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。

B. 对于存储过程来说可以返回参数，而函数只能返回值或者表对象。

C.存储过程一般是作为一个独立的部分来执行，而函数可以作为查询语句的一个部分来调用，由于函数可以返回一个表对象，因此它可以在查询语句中位于FROM关键字的后面。

### 什么是事务？

答：事务是指一个工作单元，它包含了一组数据操作命令，并且所有的命令作为一个整体一起向系统提交或撤消请求操作，即这组命令要么都执行，要么都不执行。

**原子性:**事务必须是原子工作单元；对于其数据修改，要么全都执行，要么全都不执行。

**一致性:**事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构（如 B 树索引或双向链表）都必须是正确的。

**隔离性:**由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。事务查看数据时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的数据。这称为可串行性，因为它能够重新装载起始数据，并且重播一系列事务，以使数据结束时的状态与原始事务执行的状态相同。

**持久性:**事务完成之后，它对于系统的影响是永久性的。该修改即使出现系统故障也将一直保持。

### 什么是索引？它有什么优点？

答：索引象书的目录类似，索引使数据库程序无需扫描整个表，就可以在其中找到所需要的数据，索引包含了一个表中包含值的列表，其中包含了各个值的行所存储的位置，索引可以是单个或一组列，索引提供的表中数据的逻辑位置，合理划分索引能够大大提高数据库性能。

### 什么是触发器，它有哪些优点？

答：触发器是一种特殊类型的存储过程，触发器主要通过事件触发而被执行的，

触发器的优点：

1. 强化约束，触发器能够提供比CHECK约束。
2. 跟踪变化，触发器可以跟踪数据库内的操作，从而不允许未经允许许可的更新和变化。

3.联级运算，比如某个表上的触发器中包含对另一个表的数据操作，而该操作又导致该表上的触发器被触发。

### 触发器分为事前触发和事后触发的区别。语句级触发和行级触发有何区别？

事前触发器运行于触发事件发生之前，通常可以获取事件之前和新的字段值

事后触发器运行于触发事件发生之后。

语句级触发器可以在语句执行前或后执行，

行级触发在触发器所影响的每一行触发一次

### 视图是什么？游标是什么？

答：视图是一种虚拟表，虚拟表具有和物理表相同的功能，可以对虚拟表进行增该查操作，视图通常是一个或多个表的行或列的子集，视图的结果更容易理解（修改视图对基表不影响），获取数据更容易（相比多表查询更方便），限制数据检索（比如需要隐藏某些行或列），维护更方便。

游标对查询出来的结果集作为一个单元来有效的处理，游标可以定位在结果集的特定行、从结果集的当前位置检索一行或多行、可以对结果集中当前位置进行修改。

### 怎么优化数据库（在数据百万条记录的数据库中 Oracle）？

使用索引

建立分区，分区索引

使用存储过程

### 如何优化SQL语句

1. 对操作符的优化 尽量不采用不利用索引的操作符

如：in ，not in ， is nul， is not null，<>等

2. 对条件字段的一些优化

     采用函数处理的字段不能利用索引，

进行了显式或隐式的运算的字段不能进行索引

条件内包括了多个本表的字段运算时不能进行索引

3. 在业务密集的SQL当中WHERE后面的条件顺序影响

4. 应用ORACLE的HINT（提示）处理

5. 查询表顺序的影响

### 使用索引查询一定能提高查询的性能吗？为什么？

不能。如果返回的行数目较大，使用全表扫描的性能较好。

### 操作符优化

#### IN 操作符

用IN写出来的SQL的优点是比较容易写及清晰易懂，这比较适合现代软件开发的风格。但是用IN的SQL性能总是比较低的，从Oracle执行的步骤来分析用IN的SQL与不用IN的SQL有以下区别：

ORACLE试图将其转换成多个表的连接，如果转换不成功则先执行IN里面的子查询，再查询外层的表记录，如果转换成功则直接采用多个表的连接方式查询。由此可见用IN的SQL至少多了一个转换的过程。一般的SQL都可以转换成功，但对于含有分组统计等方面的SQL就不能转换了。

**推荐方案：**在业务密集的SQL当中尽量不采用IN操作符，用EXISTS 方案代替。

#### NOT IN操作符

此操作是强列不推荐使用的，因为它不能应用表的索引。

**推荐方案：**用NOT EXISTS 方案代替

#### IS NULL 或IS NOT NULL操作（判断字段是否为空）

判断字段是否为空一般是不会应用索引的，因为索引是不索引空值的。

**推荐方案**：用其它相同功能的操作运算代替，如：a is not null 改为 a>0 或a>’’等。不允许字段为空，而用一个缺省值代替空值，如申请中状态字段不允许为空，缺省为申请。

#### > 及 < 操作符（大于或小于操作符）

大于或小于操作符一般情况下是不用调整的，因为它有索引就会采用索引查找，但有的情况下可以对它进行优化，如一个表有100万记录，一个数值型字段A，30万记录的A=0，30万记录的A=1，39万记录的A=2，1万记录的A=3。那么执行A>2与A>=3的效果就有很大的区别了，因为A>2时ORACLE会先找出为2的记录索引再进行比较，而A>=3时ORACLE则直接找到=3的记录索引。

#### LIKE操作符

LIKE操作符可以应用通配符查询，里面的通配符组合可能达到几乎是任意的查询，但是如果用得不好则会产生性能上的问题，如LIKE ‘%5400%’ 这种查询不会引用索引，而LIKE ‘X5400%’则会引用范围索引。

一个实际例子：用YW\_YHJBQK表中营业编号后面的户标识号可来查询营业编号 YY\_BH LIKE ‘%5400%’ 这个条件会产生全表扫描，如果改成YY\_BH LIKE ’X5400%’ OR YY\_BH LIKE ’B5400%’ 则会利用YY\_BH的索引进行两个范围的查询，性能肯定大大提高。

#### UNION操作符

UNION在进行表链接后会筛选掉重复的记录，所以在表链接后会对所产生的结果集进行排序运算，删除重复的记录再返回结果。实际大部分应用中是不会产生重复的记录，最常见的是过程表与历史表UNION。如：  
 select \* from gc\_dfys  
 union  
 select \* from ls\_jg\_dfys  
这个SQL在运行时先取出两个表的结果，再用排序空间进行排序删除重复的记录，最后返回结果集，如果表数据量大的话可能会导致用磁盘进行排序。

**推荐方案：**采用UNION ALL操作符替代UNION，因为UNION ALL操作只是简单的将两个结果合并后就返回。

select \* from gc\_dfys  
 union all  
 select \* from ls\_jg\_dfys

### 说说你知道的一些关于查询优化的方案？

1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：?

select id from t where num is null?

可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：?

select id from t where num=0

3.应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。

4.应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：?

select id from t where num=10 or num=20?

可以这样查询：?

select id from t where num=10?

union all?

select id from t where num=20

5.in 和 not in 也要慎用，否则会导致全表扫描，如：?

select id from t where num in(1,2,3)?

对于连续的数值，能用 between 就不要用 in 了：?

select id from t where num between 1 and 3

6.下面的查询也将导致全表扫描：?

select id from t where name like '%abc%'?

若要提高效率，可以考虑全文检索。

7.如果在 where 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：?

select id from t where num=@num?

可以改为强制查询使用索引：?

select id from t with(index(索引名)) where num=@num

8.应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：?

select id from t where num/2=100?

应改为:?

select id from t where num=100\*2

9.应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：?

select id from t where substring(name,1,3)='abc'--name以abc开头的id?

select id from t where datediff(day,createdate,'2005-11-30')=0--‘2005-11-30’生成的id?

应改为:?

select id from t where name like 'abc%'?

select id from t where createdate>='2005-11-30' and createdate<'2005-12-1'

10.不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

11.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

12.不要写一些没有意义的查询，如需要生成一个空表结构：?

select col1,col2 into #t from t where 1=0?

这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：?

create table #t(...)

13.很多时候用 exists 代替 in 是一个好的选择：?

select num from a where num in(select num from b)?

用下面的语句替换：?

select num from a where exists(select 1 from b where num=a.num)

14.并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。

15.索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。

16.应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。

17.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

18.尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

19.任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。

20.尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

21.避免频繁创建和删除临时表，以减少系统表资源的消耗。

22.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

23.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

24.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

25.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

26.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

27.与临时表一样，游标并不是不可使用。对小型数据集使用 FAST\_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

28.在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向客户端发送 DONE\_IN\_PROC 消息。

29.尽量避免大事务操作，提高系统并发能力。

30.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

### 谈谈你对SQL海量数据查询优化性能的一些看法？

数据库系统是管理信息系统的核心，基于数据库的联机事务处理（OLTP）以及联机分析处理(OLAP)是银行、企业、政府等部门最为重要的计算机应用之一。从大多数系统的应用实例来看，查询操作在各种数据库操作中所占据的比重最大，而查询操作所基于的SELECT语句在SQL语句中又是代价最大的语句。举例来说，如果数据的量积累到一定的程度，比如一个银行的账户数据库表信息积累到上百万甚至上千万条记录，全表扫描一次往往需要数十分钟，甚至数小时。如果采用比全表扫描更好的查询策略，往往可以使查询时间降为几分钟，由此可见查询优化技术的重要性。

**百万数据查询优化技巧三十则(见问题12)**

### 数据库内联与外联的区别？

内部联接（inner join）一个联接，返回两表的公共列都匹配的行

外部联接（outer join） 一个联接，该联接还包括那些和联接表中记录不相关的记录。您可以创建一个外部联接的三种变形来指定所包括的不匹配行：

**左外部联接、右外部联接和完全外部联接**。

* 左外部联接（left outer join） 左边的表是主表，列所有；右表无取null
* 右外部联接（right outer join） 右边的表是主表，列所有；左边表只列匹配的行，没有值置null
* 完全外部联接 列所有，没有值置null

## Struts2

### JavaEE软件三层结构和MVC的区别？

JavaEE软件三层机构是由sun公司提供JavaEE开发规范的：Web层(表现层)、业务逻辑层、数据持久层。【其中WEB层会使用前端控制器模式】

MVC是一种思想，是一种模式，将软件分为 Model模型、View视图、Controller控制器。【JavaEE开发更强调三层结构，web层开发更注重MVC】

Struts2 就是web层开发框架，符合MVC模式；struts1 、webwork 、jsf 、SpringMVC 都是MVC

### 简要说说Struts2的处理流程？

Struts2框架的大致处理流程如下：

1、加载类（FilterDispatcher）

2、读取配置（struts配置文件中的Action）

3、派发请求（客户端发送请求）

4、调用Action（FilterDispatcher从struts配置文件中读取与之相对应的Action ）

5、启用拦截器（WebWork拦截器链自动对请求应用通用功能，如验证）

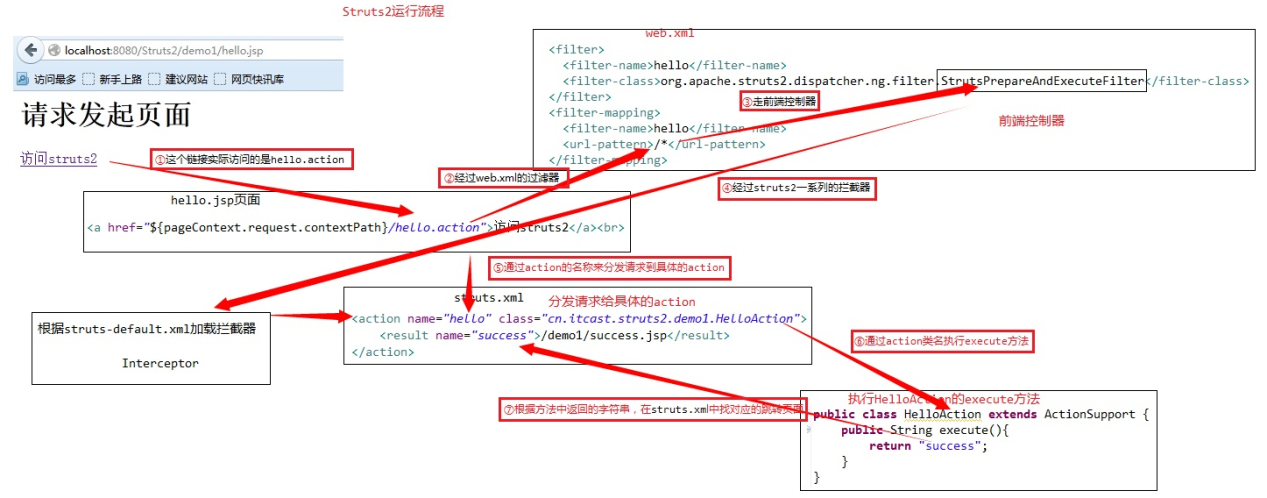
6、处理业务（回调Action的execute()方法）

7、返回响应（通过execute方法将信息返回到FilterDispatcher）

8、查找响应（FilterDispatcher根据配置查找响应的是什么信息如：SUCCESS、ERROER，将跳转到哪个jsp页面）

9、响应用户（jsp--->客户浏览器端显示）

10、struts2标签库（相比struts1的标签库，struts2是大大加强了，对数据的操作功能很强大）



请求(.action)---->经过StrutsPrepareAndExecuteFilter 核心控制器---->进入到Struts2的拦截器Interceptor(实现代码功能)----->通过action的名称找对应的Action类----->执行Action类的execute方法----->通过execute方法中返回的字符串，在Struts.xml中找对应的结果页面(result)【在action执行之前，执行了*defaultStack*拦截器栈】

\* 拦截器 在 struts-default.xml定义 【它位于sruts2-core-xxx.jar目录下】

\* 执行拦截器 是 defaultStack 中引用拦截器

### Struts和struts2的区别有哪些？

**Action类**   
 Struts 1要求Action类要扩展自一个抽象基类。Struts 1的一个共有的问题是面向抽象类编程而不是面向接口编程。   
 Struts 2的Action类实现了一个Action接口，连同其他接口一起实现可选择和自定义的服务。Struts 2提供一个名叫ActionSupport的基类实现一般使用的接口。虽然，Action接口不是必须的。任何使用execute方法的POJO对象可以 被当作Struts 2的Action对象使用。   
**程模型**   
 Struts 1 Action类是单例类，因只有一个示例控制所有的请求。单例类策略造成了一定的限制且给开发带来了额外的烦恼。Action资源必须是程安全或者同步 的。   
 Struts 2 Action对象每一个请求都实例化对象，所以没有程安全的问题。（实践中，servlet容器生许多丢的对象对于每一个请求，多于一个的对象并不影响垃 圾收集）   
**Servlet 依赖**   
 Struts 1的Action类依赖于servlet API以HttpServletRequest和HttpServletResponse作参数传给execute方法当Action被调用时。   
 Struts 2的Action不和容器有关。Servlet上下文被表现简单的Maps，允许Action被独立的测试。Struts 2的Action可以访问最初的请求和相应，如果需要的话。然而，其他的架构元素少或者排除直接访问HttpServletRequest或者 HttpServletResponse的需要。   
**易测性**   
 测试Struts 1的主要障碍是execute方法暴露了Servlet API。第三方的扩展，Struts测试用例，提供Struts 1的集合对象。   
 Struts 2的Action可以通过实例化Action测试，设置属性，然后调用方法。依赖注入的支持也是测试变得更简单。   
**接受输入**   
 Struts 1使用ActionForm对象捕获输入。象Action一样，所有的ActionForm必须扩展基类。因其他的JavaBean不能作 ActionForm使用，开发者经常创建多余的类捕获输入。DynaBeans可以被用来作替代ActionForm的类创建。但是开发者可以重新描述 已经存在的JavaBean。   
 Struts 2 Action属性作输入属性，排除第二个输入对象的需要。输入属性可能有丰富的对象类型这些类型有他们自己的属性。Action的属性可以通过标签库访 问。Struts 2也支持ActionForm形式。丰富的对象类型，包含业务或者域对象，可以被当作输入或者输出对象使用。馍型驱动特性简化标签对POJO输入对象的引 用。   
**表达式语言**   
 Struts 1整和JSTL，所以它使用JSTL的表达式语言。表达式语言有基本的图形对象移动，但是相对很弱的集合和被索引的属性支持。   
 Struts 2使用JSTL，但是框架也支持更大和更灵活的表达式，叫做“对象图形符号语言”（OGNL）。   
**将值绑定要视图上**   
 Struts 1使用标准JSP机制来绑定对象到页面上下文。   
 Struts 2使用“ValueStack”技术了标签库可以不用链接你的视图到对象的表现类型访问值。ValueStack策略允许重用视图。   
**类型转换**   
 Struts 1的ActionForm属性经常都是String的。Struts 1使用Commons-Beanutils类型转换。转换每一个类，不是每一个实例配置。   
 Struts 2使用OGNL类型转换。框架包含转换器基本的和共同的对象类型和原始类型。   
**验证**   
 Struts 1支持手动验证凭借ActionForm的validate方法，或者通过扩展的公用验证器。类可以有不同的验证上下文未相同的类，但是不能不能包括验证 子对象。   
 Struts 2支持手动验证凭借validate方法和XWork验证框架。Xwork验证框架支持一连串的验证子属性使用的验证了属性类的类型和严正上下文而定义。   
**Action执行的控制**   
 Struts 1支持独立的请求处理器对于每一个模型，但是所有在模型中的Action必须共享同一个生命周期。   
 Struts 2支持在每一个Action基础上凭借拦截栈创建不同的生命周期。自定义栈可以被创建且使用不同的所需 的Action。

### 说说Struts2的输入校验流程

* Struts2校验框架进行校验时，将执行以下流程：

A：类型转换器负责对字符串的请求参数执行类型转换，并将这些值设置成Action的属性值

B：在执行类型转换过程中可能出现异常，如果出现异常，将异常信息保存到ActionContext中，convertionError拦截器将负责将其封装到fieldError里，如果没有异常，直接进入第3步

C：调用Struts2的内置校验规则进行输入校验

D：通过反射调用validateXXX()方法

E：调用Action类中的validate()方法

F：如果上面的几步中没有出FiledError，就调用Acton中的逻辑处理方法，如果有，则进入input视图

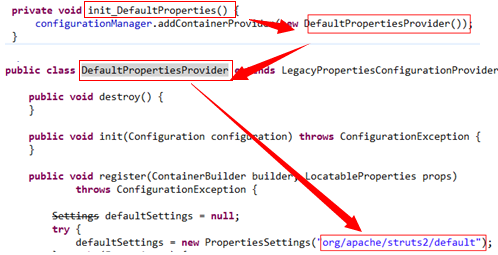
所以，在进行校验时，别忘记在Action中的配置名为input的结果如：

<result name=“input”>validate.jsp</result>

### Struts2配置文件的加载顺序



此处，我们以及清晰了看到了该类加载配置文件的顺序，我们依次围绕标号查看对应方法产生的文件即可。



**对应产生文件依次如下：**

init\_DefaultProperties(); // [1]

---- org/apache/struts2/**default.properties**

init\_TraditionalXmlConfigurations(); // [2]

--- **struts-default.xml**,**struts-plugin.xml**,**struts.xml**

init\_LegacyStrutsProperties(); // [3] --- 自定义**struts.properties**

init\_CustomConfigurationProviders(); // [5] ----- 自定义配置提供

init\_FilterInitParameters() ; // [6] ----- **web.xml**

init\_AliasStandardObjects() ; // [7] ---- Bean加载

结论 ：【前三个是默认的，不用关注，后面三个需要注意】

① **default.properties** 该文件保存在 struts2-core-2.3.7.jar 中 org.apache.struts2包里面 （常量的默认值）

② **struts-default.xml** 该文件保存在 struts2-core-2.3.7.jar （Bean、拦截器、结果类型 ）

③ **struts-plugin.xml** 该文件保存在struts-Xxx-2.3.7.jar （在插件包中存在 ，配置插件信息 ）struts-config-browser-plugin-2.3.7.jar里面有

④ struts.xml 该文件是web应用默认的struts配置文件 （实际开发中，通常写struts.xml ）

⑤ struts.properties 该文件是Struts的默认配置文件 (配置常量 )

⑥ web.xml 该文件是Web应用的配置文件 (配置常量 )

### 说说action的相关配置？

1）必须要为<action>元素 配置<package>元素 （struts2 围绕package进行Action的相关配置 ）

配置package 三个常用属性

<package name="default" namespace="/" extends="struts-default">

①name包名称，在struts2的配置文件中，包名不能重复，name并不是真正包名，只是为了管理Action

②namespace和 <action>的name属性，决定 Action的访问路径 （以/开始 ）

③extends继承哪个包，通常开发中继承struts-default包 （struts-default包在 struts-default.xml定义 ）【可以使用包中默认的拦截器和结果集】

2）Action是通过<action>元素配置

<action name="hello" class="cn.itcast.struts2.demo1.HelloAction">

<action>的name和 <package>的namespace属性共同决定 Action的访问路径

例如：

<package name="default" namespace="/user" extends="struts-default">

<action name="hello" class="cn.itcast.struts2.demo1.HelloAction">

<result name="success">/demo1/success.jsp</result>

</action>

</package>

此时的访问路径 http://localhost:8080/Struts2/demo1/user/hello.action

3) <action> 元素配置默认值

<package> 的namespace 默认值 /

<action> 的class 默认值 ActionSupport 类

<result> 的 name 默认值 success

### Action如何接受请求参数

**第一种** ：Action 本身作为model对象，通过成员setter封装 （**属性驱动** ）

页面：

用户名 <input type="text" name="username" /> <br/>

Action ：

public class RegistAction1 extends ActionSupport {

private String username;

public void setUsername(String username) {

this.username = username;

}

}

问题一： Action封装数据，会不会有线程问题 ？

\* struts2 Action 是多实例 ，为了在Action封装数据 （struts1 Action 是单例的 ）

问题二： 在使用第一种数据封装方式，数据封装到Action属性中，不可能将Action对象传递给 业务层

\* 需要再定义单独JavaBean ，将Action属性封装到 JavaBean

**第二种** ：创建独立model对象，页面通过ognl表达式封装 （**属性驱动**）

页面：

用户名 <input type="text" name="user.username" /> <br/> ----- 基于OGNL表达式的写法

Action：

public class RegistAction2 extends ActionSupport {

private User user;

public void setUser(User user) {

this.user = user;

}

public User getUser() {

return user;

}

}

问题： 谁来完成的参数封装

<interceptor name="params"

class="com.opensymphony.xwork2.interceptor.ParametersInterceptor"/>

**第三种** ：使用ModelDriven接口，对请求数据进行封装 （**模型驱动** ） ----- 主流

页面：

用户名 <input type="text" name="username" /> <br/>

Action ：

public class RegistAction3 extends ActionSupport implements ModelDriven<User> {

private User user = new User(); // 必须手动实例化

public User getModel() {

return user;

}

}

\* struts2 有很多围绕模型驱动的特性

\* <interceptor name="modelDriven"

class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/> 为模型驱动提供了更多特性

对比第二种、第三种 ： 第三种只能在Action中指定一个model对象，第二种可以在Action中定义多个model对象

<input type="text" name="user.username" />

<input type="text" name="product.info" />

### 值栈的内部结构？如何向值栈保存数据？为什么EL也能访问值栈中的数据？

①值栈由两部分组成

ObjectStack: Struts 把动作和相关对象压入 ObjectStack 中--List

ContextMap: Struts 把各种各样的映射关系(一些 Map 类型的对象) 压入 ContextMap 中

Struts 会把下面这些映射压入 ContextMap 中

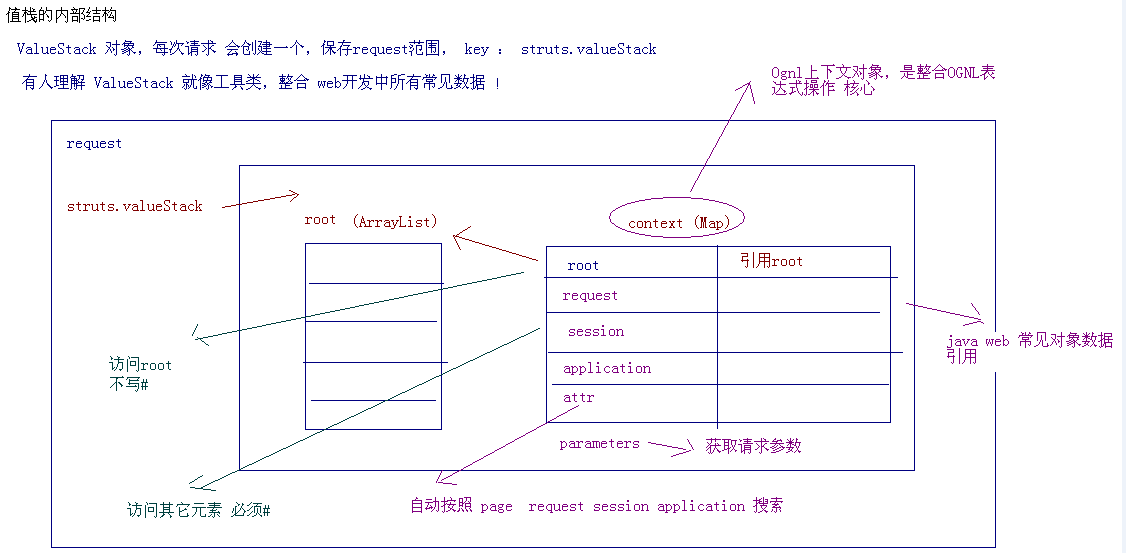
parameters: 该 Map 中包含当前请求的请求参数

request: 该 Map 中包含当前 request 对象中的所有属性

session: 该 Map 中包含当前 session 对象中的所有属性

application:该 Map 中包含当前 application 对象中的所有属性

attr: 该 Map 按如下顺序来检索某个属性: request, session, application



②两种方式

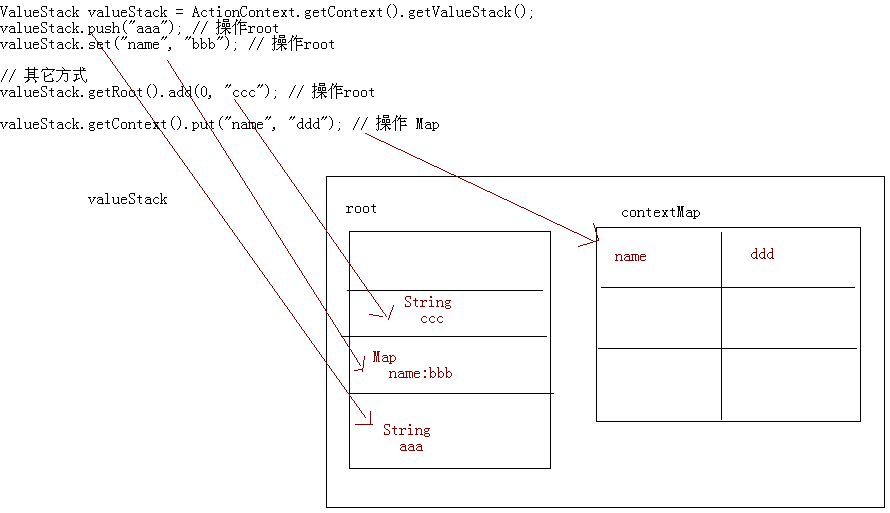
// 将数据保存root的索引0位置，放置到第一个元素 ArrayList add(0,element);

valueStack.push("itcast");

// 在值栈创建参数map， 将数据保存到map中

valueStack.set("company", "传智播客");

在jsp中 通过 <s:debug /> 查看值栈的内容



③StrutsPreparedAndExecuteFilter的doFilter代码中 request = prepare.wrapRequest(request);

\* 对Request对象进行了包装 ，StrutsRequestWrapper

\* 重写request的 getAttribute

Object attribute = super.getAttribute(s);

if (attribute == null) {

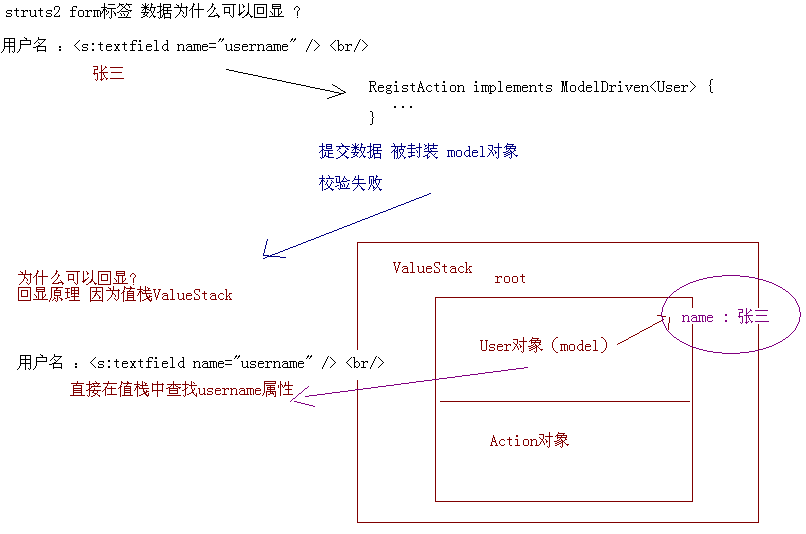
attribute = stack.findValue(s);

}

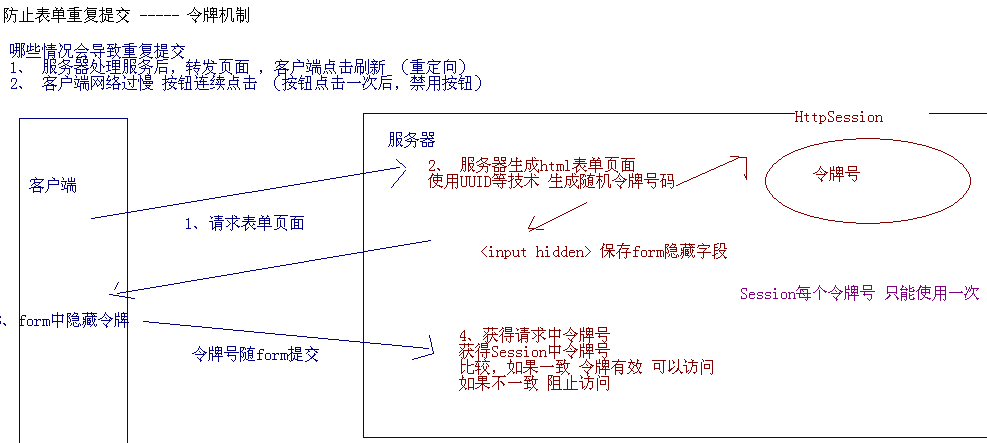
访问request范围的数据时，如果数据找不到，去值栈中找

\* request对象 具备访问值栈数据的能力 （查找root的数据）

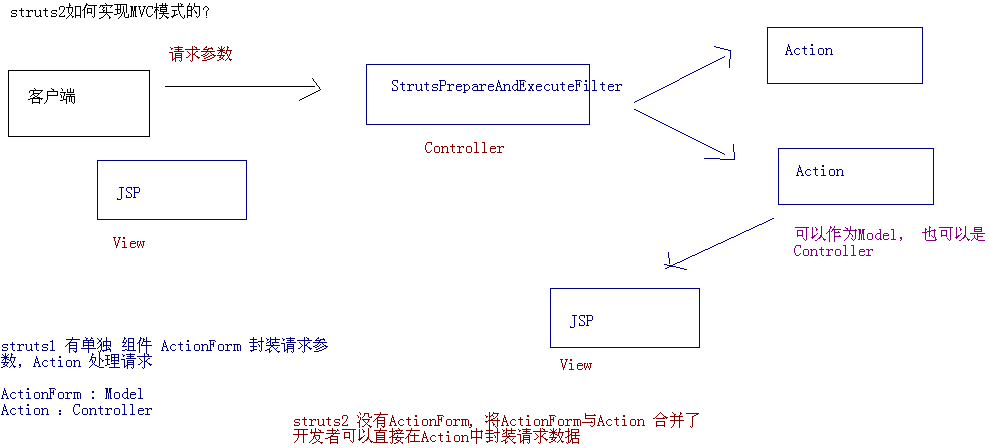
### Struts2 form标签数据为什么可以回显？



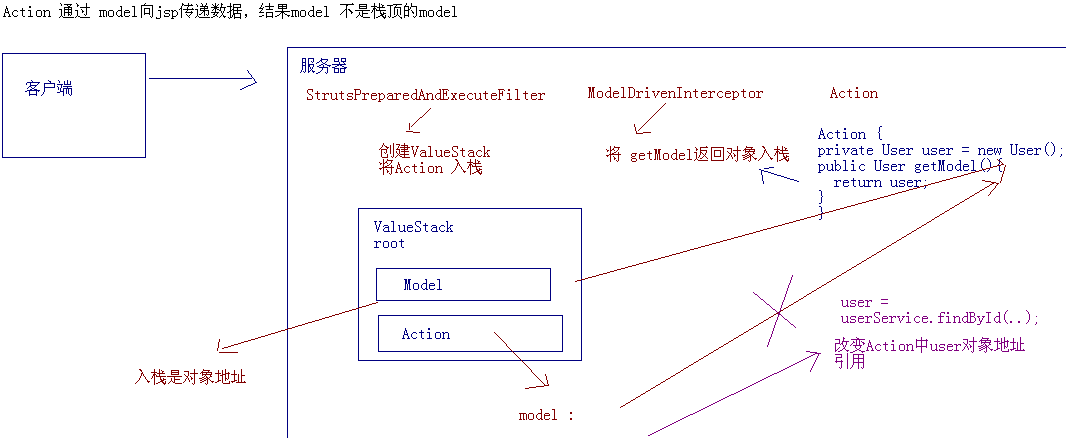
### 如何防止表单重复提交？



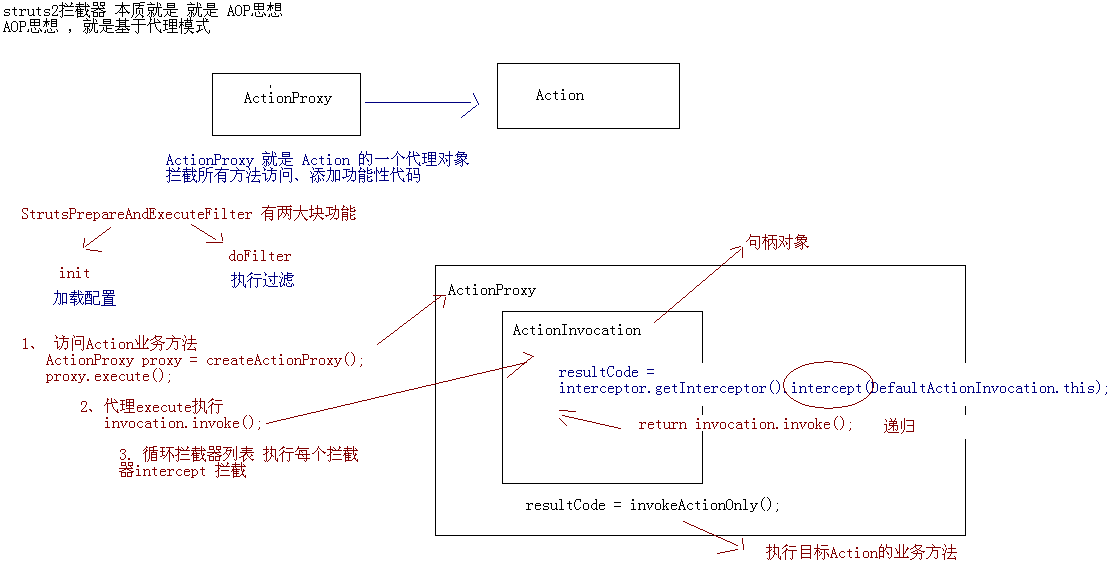
### Struts2如何实现MVC？



### Action通过model向jsp传递数据，结果model不是栈顶model



### ActionProxy运行原理分析



## Hibernate

### Hibernate 工作原理及为什么要用？

**原理：**

hibernate，通过对jdbc 进行封装，对java 类和 关系数据库进行mapping，实现了对关系数据库的面向对象方式的操作，改变了传统的jdbc + sql 操作数据的方式，从而使开发人员可以话更多精力进行对象方面的开发

1.读取并解析配置文件

2.读取并解析映射信息，创建SessionFactory

3.打开Sesssion

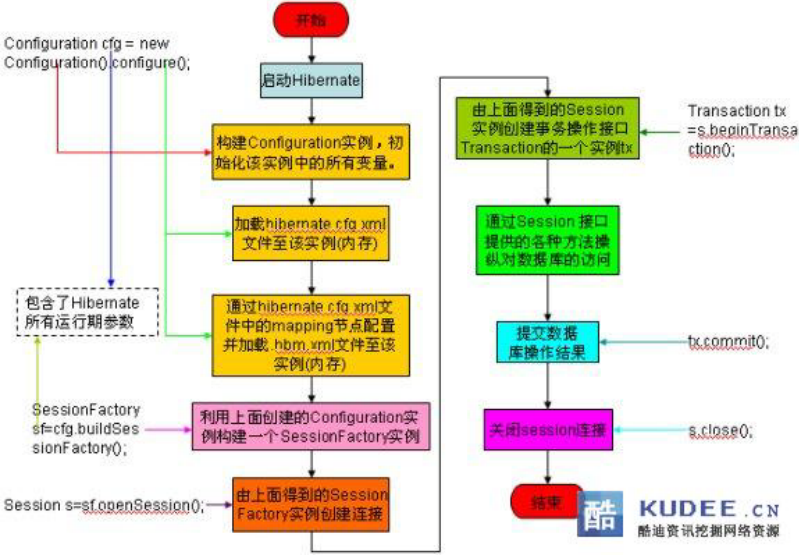
4.创建事务Transation

5.持久化操作

6.提交事务

7.关闭Session

8.关闭SesstionFactory



**为什么要用：**

1.对JDBC访问数据库的代码做了封装，大大简化了数据访问层繁琐的重复性代码。

2.Hibernate 是一个基于JDBC 的主流持久化框架，是一个优秀的ORM 实现。他很大程度

的简化DAO 层的编码工作

3.hibernate 的性能非常好，因为它是个轻量级框架。映射的灵活性很出色。它支持各

种关系数据库，从一对一到多对多的各种复杂关系。

### Hibernate开发流程



### 什么是 Hibernate 延迟加载?

延迟加载机制是为了避免一些无谓的性能开销而提出来的，所谓延迟加载就是当在真正需要

数据的时候，才真正执行数据加载操作。在Hibernate中提供了对实体对象的延迟加载以及

对集合的延迟加载，另外在Hibernate3中还提供了对属性的延迟加载。

### Hibernate 中类之间的关联关系有几种?(如：一对多、多对多的关系)

many-to-one、one-to-many、many-to-many、 one-to-one

### 说下 Hibernate的缓存机制

A：hibernate一级缓存

（1）hibernate支持两个级别的缓存，默认只支持一级缓存；

（2）每个Session内部自带一个一级缓存；

（3）某个Session被关闭时，其对应的一级缓存自动清除；

B：hibernate二级缓存

(1) 二级缓存独立于session，默认不开启；

### Hibernate 的查询方式

本地SQL查询、Criteria、Hql

### 如何优化 Hibernate？

1.使用双向一对多关联，不使用单向一对多

2.不用一对一，用多对一取代

3.配置对象缓存，不使用集合缓存

### hibernate 中get()与 load()区别

请注意如果没有匹配的数据库记录，load()方法可能抛出无法恢复的异常(unrecoverable

exception)。 如果类的映射使用了代理(proxy)，load()方法会返回一个未初始化的代理，直到你调用该代理的某方法时才会去访问数据库。若你希望在某对象中创建一个指向另一个对象的关联，又不想在从数据库中装载该对象时同时装载相关联的那个对象，那么这种操作方式就用得上的了。 如果为相应类映射关系设置了batch-size， 那么使用这种操作方式允许多个对象被一批装载（因为返回的是代理，无需从数据库中抓取所有对象的数据）。 如果你不确定是否有匹配的行存在，应该使用 get()方法，它会立刻访问数据库，如果没有对应的行，会返回null。

### 说说在 hibernate中使用Integer做映射和使用int做映射之间有什么差别？

Integer code和int code的区别:

Integer是对象. code=null; 对象可以为空.

int 是普通类型, 不可能=null.

根据你的数据库code是可以空的,故应该映射成Integer.

你没理由hbm.xml里写 Integer,类里却写int

### SQL和HQL有什么区别？

sql 面向数据库表查询

hql 面向对象查询

hql：from 后面跟的 类名＋类对象 where 后 用 对象的属性做条件

sql：from 后面跟的是表名 where 后 用表中字段做条件

**查询**

在Hibernate中使用查询时，一般使用Hql查询语句。

HQL（Hibernate Query Language），即Hibernate的查询语言跟SQL非常相像。不过HQL与SQL的最根本的区别，就是它是面向对象的。

使用HQL时需要注意以下几点：

1.大小写敏感

因为HQL是面向对象的，而对象类的名称和属性都是大小写敏感的，所以HQL是大小写敏感的。

HQL语句：from Cat as cat where cat.id > 1;与from Cat as cat where cat.ID > 1;是不一样的，这点与SQL不同。

2.from子句

from Cat，该句返回Cat对象实例，开发人员也可以给其加上别名，eg. from Cat as cat，对于多表查询的情况，可参考如下：

from Cat as cat, Dog as dog

其它方面都与SQL类似，在此不再赘述。

### 如何配置单向多对一关联？

<class name="Jd" table="TBL\_JD">

<id name="jdid" column="jdid" type="long">

<generator class="identity" />

</id>

<property name="jdname" column="jd" type="string" />

**<many-to-one name="qx" class="Qx" column="qxid" />**

</class>

### 如何配置单向一对多关联？

<class name="Qx" table="TBL\_QX">

<id name="qxid" column="qxid" type="long">

<generator class="native" />

</id>

<property name="qxname" column="qx" type="string" />

**<set name="jds" >**

**<key column="qxid" />**

**<one-to-many class="Jd" />**

**</set>**

</class>

### 如何配置双向一对多关联？

<class name="Jd" table="TBL\_JD">

<id name="jdid" column="jdid" type="long">

<generator class="identity" />

</id>

<property name="jdname" column="jd" type="string" />

<many-to-one name="qx" class="Qx" column="qxid" />

</class>

<class name="Qx" table="TBL\_QX">

<id name="qxid" column="qxid" type="long">

<generator class="native" />

</id>

<property name="qxname" column="qx" type="string" />

<set name="jds" >

<key column="qxid" />

<one-to-many class="Jd" />

</set>

</class>

### Hibernate中Java对象的状态以及对应的特征有哪些？

**A、临时状态（transient）**

特征：

1】不处于Session 缓存中

2】数据库中没有对象记录

Java如何进入临时状态

1】通过new语句刚创建一个对象时

2】当调用Session 的delete()方法，从Session 缓存中删除一个对象时。

**B、.持久化状态(persisted)**

特征：

1】处于Session 缓存中

2】持久化对象数据库中设有对象记录

3】Session 在特定时刻会保持二者同步

Java如何进入持久化状态

1】Session 的save()把临时－》持久化状态

2】Session 的load(),get()方法返回的对象

3】Session 的find()返回的list集合中存放的对象

4】Session 的update(),saveOrupdate()使游离－》持久化

**C、.游离状态(detached)**

特征：

1】不再位于Session 缓存中

2】游离对象由持久化状态转变而来，数据库中可能还有对应记录。

Java如何进入持久化状态－》游离状态

1】Session 的close()方法

2】Session 的evict()方法，从缓存中删除一个对象。提高性能。少用。

### 如何优化Hibernate？

1.使用双向一对多关联，不使用单向一对多   
 2.灵活使用单向一对多关联   
 3.不用一对一，用多对一取代   
 4.配置对象缓存，不使用集合缓存   
 5.一对多集合使用Bag,多对多集合使用Set   
 6. 继承类使用显式多态   
 7. 表字段要少，表关联不要怕多，有二级缓存撑腰

### hibernate分页查询？

例如：从数据库中的第20000条数据开始查后面100条记录

Query q = session.createQuery("from Cat as c");;

q.setFirstResult(20000);;

q.setMaxResults(100);;

List l = q.list();;

### Hibernate中怎样处理事务?

Hibernate是对JDBC的轻量级对象封装，Hibernate本身是不具备Transaction 处理功能的，Hibernate的Transaction实际上是底层的JDBC Transaction的封装，或者是JTA Transaction的封装，下面我们详细的分析：

Hibernate可以配置为JDBCTransaction或者是JTATransaction，这取决于你在hibernate.properties中的配置:

#hibernate.transaction.factory\_class net.sf.hibernate.transaction.JTATransactionFactory

#hibernate.transaction.factory\_class net.sf.hibernate.transaction.JDBCTransactionFactory

如果你什么都不配置，默认情况下使用JDBCTransaction，如果你配置为：

hibernate.transaction.factory\_class net.sf.hibernate.transaction.JTATransactionFactory

将使用JTATransaction

不管你准备让Hibernate使用JDBCTransaction，还是JTATransaction，我的忠告就是什么都不配，将让它保持默认状态，如下：

#hibernate.transaction.factory\_class net.sf.hibernate.transaction.JTATransactionFactory

#hibernate.transaction.factory\_class net.sf.hibernate.transaction.JDBCTransactionFactory

在下面的分析中我会给出原因。

一、JDBC Transaction

看看使用JDBC Transaction的时候我们的代码例子：

Session session = sf.openSession();

Transaction tx = session.beginTransactioin();

...

session.flush();

tx.commit();

session.close();

这是默认的情况，当你在代码中使用Hibernate的Transaction的时候实际上就是JDBCTransaction。那么JDBCTransaction究竟是什么东西呢？来看看源代码就清楚了：

Hibernate2.0.3源代码中的类

net.sf.hibernate.transaction.JDBCTransaction:

public void begin() throws HibernateException {

...

if (toggleAutoCommit) session.connection().setAutoCommit(false);

...

}

这是启动Transaction的方法，看到 connection().setAutoCommit(false) 了吗？是不是很熟悉？

再来看

public void commit() throws HibernateException {

...

try {

if ( session.getFlushMode()!=FlushMode.NEVER ) session.flush();

try {

session.connection().commit();

committed = true;

}

...

toggleAutoCommit();

}

这是提交方法，看到connection().commit() 了吗？下面就不用我多说了，这个类代码非常简单易懂，通过阅读使我们明白Hibernate的Transaction都在干了些什么？我现在把用 Hibernate写的例子翻译成JDBC，大家就一目了然了：

Connection conn = ...; <--- session = sf.openSession();

conn.setAutoCommit(false); <--- tx = session.beginTransactioin();

... <--- ...

conn.commit(); <--- tx.commit(); (对应左边的两句)

conn.setAutoCommit(true);

conn.close(); <--- session.close();

看明白了吧，Hibernate的JDBCTransaction根本就是conn.commit而已，根本毫无神秘可言，只不过在Hibernate中， Session打开的时候，就会自动conn.setAutoCommit(false)，不像一般的JDBC，默认都是true，所以你最后不写 commit也没有关系，由于Hibernate已经把AutoCommit给关掉了，所以用Hibernate的时候，你在程序中不写 Transaction的话，数据库根本就没有反应。

## Spring

### 什么是Spring, 它有什么特点？ 包括哪些内容？

Spring是一个轻量级的控制反转(IoC)和面向切面(AOP)的容器框架。

◆ 轻量——从大小与开销两方面而言Spring都是轻量的。完整的Spring框架可以在一个大小只有1MB多的JAR文件里发布。并 且Spring所需的处理开销也是微不足道的。此外，Spring是非侵入式的：典型地，Spring应用中的对象不依赖于Spring的特定类。

◆ 控制反转——Spring通过一种称作控制反转（IoC）的技术促进了松耦 合。当应用了IoC，一个对象依赖的其它对象会通过被动的方式传递进来，而不是这个对象自己创建或者查找依赖对象。你可以认为IoC与JNDI相反——不 是对象从容器中查找依赖，而是容器在对象初始化时不等对象请求就主动将依赖传递给它。

◆ 面向切面——Spring提供了面向切面编程的丰富支持，允许通过分离应用的 业务逻辑与系统级服务（例如审计（auditing）和事务（）管理）进行内聚性的开发。应用对象只实现它们应该做的——完成业务逻辑——仅此而已。它们 并不负责（甚至是意识）其它的系统级关注点，例如日志或事务支持。

◆ 容器——Spring包含并管理应用对象的配置和生命周期，在这个意义上它是 一种容器，你可以配置你的每个bean如何被创建——基于一个可配置原型（prototype），你的bean可以创建一个单独的实例或者每次需要时都生 成一个新的实例——以及它们是如何相互关联的。然而，Spring不应该被混同于传统的重量级的EJB容器，它们经常是庞大与笨重的，难以使用。

◆ 框架——Spring可以将简单的组件配置、组合成为复杂的应用。在Spring中，应用对象被声明式地组合，典型地是在一个XML文件里。 Spring也提供了很多基础功能（事务管理、持久化框架集成等等），将应用逻辑的开发留给了你。

\* 核心技术 IoC 和 AOP

\* 数据访问 （持久层解决方案）

\* Web层解决方案 SpringMVC

\* 集成 （整合其他开源框架）

### 使用Spring有什么好处？

◆Spring能有效地组织你的中间层对象,无论你是否选择使用了EJB。如果你仅仅使用了Struts或其他的包含了J2EE特有APIs的 framework，你会发现Spring关注了遗留下的问题，。

◆Spring能消除在许多工程上对Singleton的过多使用。根据我的经验，这是一个主要的问题，它减少了系统的可测试性和面向对象特性。

◆Spring能消除使用各种各样格式的属性定制文件的需要,在整个应用和工程中，可通过一种 一致的方法来进行配置。曾经感到迷惑，一个特定类要查找迷幻般的属性关键字或系统属性,为此不得不读Javadoc乃至源编码吗？有了Spring，你可 很简单地看到类的JavaBean属性。倒置控制的使用(在下面讨论)帮助完成这种简化。

◆Spring能通过接口而不是类促进好的编程习惯，减少编程代价到几乎为零。

◆Spring被设计为让使用它创建的应用尽可能少的依赖于他的APIs。在Spring应用中的大多数业务对象没有依赖于Spring。

◆使用Spring构建的应用程序易于单元测试。

◆Spring能使EJB的使用成为一个实现选择,而不是应用架构的必然选择。你能选择用POJOs或local EJBs来实现业务接口，却不会影响调用代码。

◆Spring帮助你解决许多问题而无需使用EJB。Spring能提供一种EJB的替换物，它们适于许多web应用。例如,Spring能使用 AOP提供声明性事务而不通过使用EJB容器，如果你仅仅需要与单个的数据库打交道，甚至不需要JTA实现。

■Spring为数据存取提供了一致的框架,不论是使用JDBC或O/R mapping产品（如Hibernate）。

Spring确实使你能通过最简单可行的解决办法解决你的问题。这些特性是有很大价值的。

总结起来，Spring有如下优点：

◆低侵入式设计，代码污染极低

◆ 独立于各种应用服务器，可以真正实现Write Once,Run Anywhere的承诺

◆Spring的DI机制降低了业务对象替换的复杂性

◆Spring并不完全依赖于Spring，开发者可自由选用Spring框架的部分或全部

### spring中的核心类有那些，各有什么作用？

BeanFactory：产生一个新的实例，可以实现单例模式

BeanWrapper：提供统一的get及set方法

ApplicationContext:提供框架的实现，包括BeanFactory的所有功能

### spring常用的类和接口有哪些？

**A：ApplicationContextAware接口**

当一个类需要获取ApplicationContext实例时，可以让该类实现ApplicationContextAware接口。

**B：ApplicationEvent抽象类**

当需要创建自定义事件时，可以新建一个继承自ApplicationEvent抽象类的类。

**C：ApplicationListener接口**

当需要监听自定义事件时，可以新建一个实现ApplicationListener接口的类，并将该类配置到Spring容器中。

**D：BeanNameAware接口**

当bean需要获取自身在容器中的id/name时，可以实现BeanNameAware接口。

**E：InitializingBean接口**

当需要在bean的全部属性设置成功后做些特殊的处理，可以让该bean实现InitializingBean接口。

效果等同于bean的init-method属性的使用或者@PostContsuct注解的使用。

三种方式的执行顺序：先注解，然后执行InitializingBean接口中定义的方法，最后执行init-method属性指定的方法。

**F：DisposableBean接口**

当需要在bean销毁之前做些特殊的处理，可以让该bean实现DisposableBean接口。

效果等同于bean的destroy-method属性的使用或者@PreDestory注解的使用。

三种方式的执行顺序：先注解，然后执行DisposableBean接口中定义的方法，最后执行destroy-method属性指定的方法。

**G：BeanPostProcessor接口**

当需要对受管bean进行预处理时，可以新建一个实现BeanPostProcessor接口的类，并将该类配置到Spring容器中。

**H：BeanFactoryPostProcessor接口**

当需要对Bean工厂进行预处理时，可以新建一个实现BeanFactoryPostProcessor接口的类，并将该类配置到Spring容器中。

### spring事务的原理(从架构图回答)

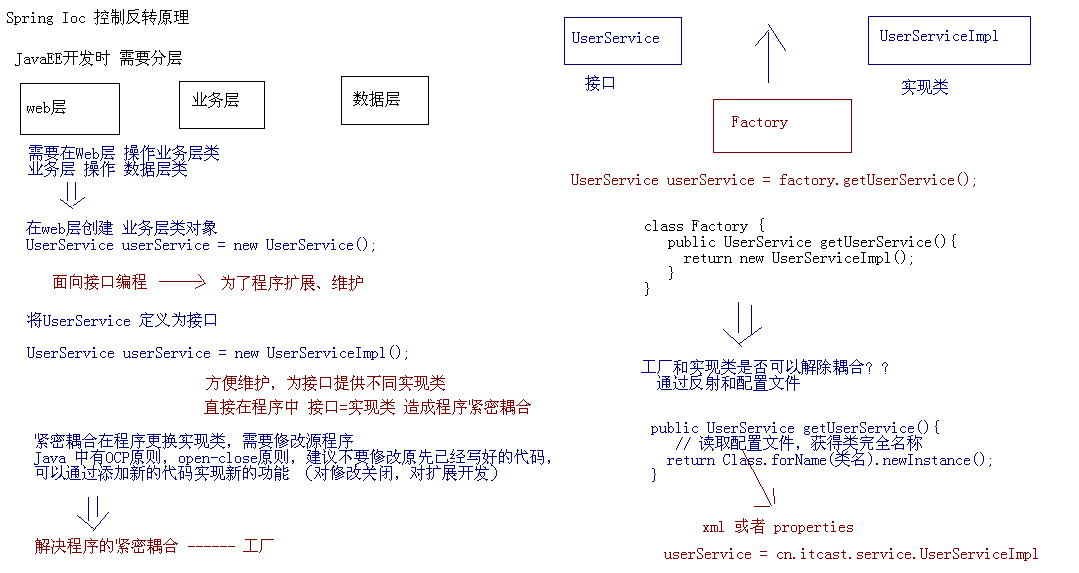
事务管理的方式：编程型和声明型，spring推荐使用后一种方式

声明型事务管理的优势非常明显：代码中无需关于关注事务逻辑，让Spring声明式事务管理负责事务逻辑，声明式事务管理无需与具体的事务逻辑耦 合，可以方便地在不同事务逻辑之间切换。

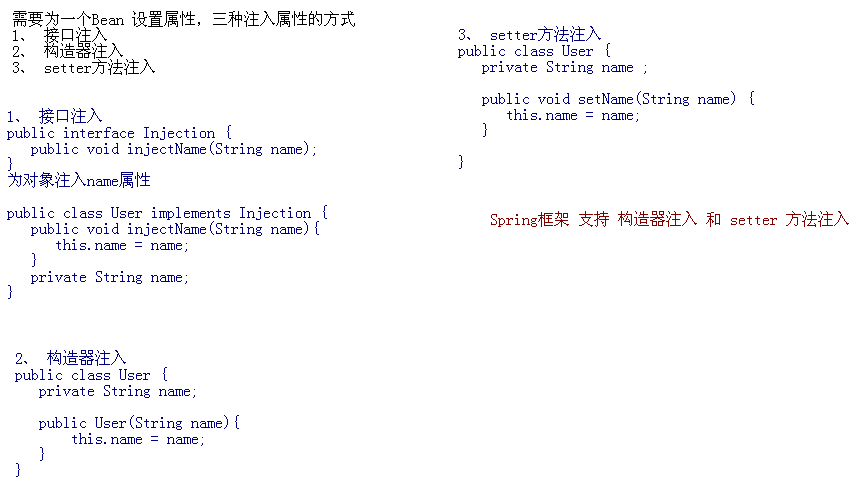
### 什么是IOC，什么又是DI，他们有什么区别？

依赖注入DI是一个程序设计模式和架构模型， 一些时候也称作控制反转，尽管在技术上来讲，依赖注入是一个IOC的特殊实现，依赖注入是指一个对象应用另外一个对象来提供一个特殊的能力，例如：把一个 数据库连接已参数的形式传到一个对象的结构方法里面而不是在那个对象内部自行创建一个连接。控制反转和依赖注入的基本思想就是把类的依赖从类内部转化到外 部以减少依赖

应用控制反转，对象在被创建的时候，由一个调控系统内所有对象的外界实体，将其所依赖的对象的引用，传递给它。也可以说，依赖被注入到对象中。所 以，控制反转是，关于一个对象如何获取他所依赖的对象的引用，这个责任的反转。



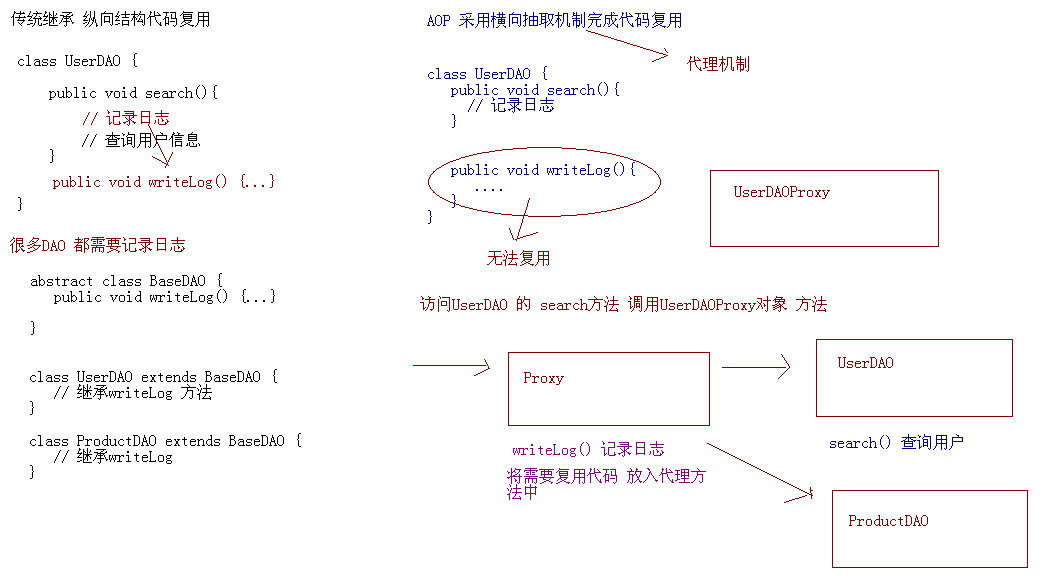
### Bean注入属性的几种方式？



### 什么是aop，aop的作用是什么?

面向切面编程（AOP）提供另外一种角度来思考程序结构，通过这种方式弥补了面向对象编程（OOP）的不足，除了类（classes）以外，AOP提供了切面。切面对关注点进行模块化，例如横切多个类型和对象的事务管理

Spring的一个关键的组件就是AOP框架，可以自由选择是否使用AOP 提供声明式企业服务，特别是为了替代EJB声明式服务。最重要的服务是声明性事务管理，这个服务建立在Spring的抽象事物管理之上。允许用户实现自定义切面，用AOP来完善OOP的使用,可以把Spring AOP看作是对Spring的一种增强



### AOP里面重要的几个名词概念解释：

**切面（Aspect）：** 一个关注点的模块化，这个关注点可能会横切多个对象。事务管理是J2EE应用中一个关于横切关注点的很好的例子。 在Spring AOP中，切面可以使用通用类（基于模式的风格） 或者在普通类中以 @Aspect 注解（@AspectJ风格）来实现。

**连接点（Joinpoint）：** 在程序执行过程中某个特定的点，比如某方法调用的时候或者处理异常的时候。 在Spring AOP中，一个连接点 总是 代表一个方法的执行。 通过声明一个org.aspectj.lang.JoinPoint类型的参数可以使通知（Advice）的主体部分获得连接点信息。

**通知（Advice）：** 在切面的某个特定的连接点（Joinpoint）上执行的动作。通知有各种类型，其中包括“around”、“before”和“after”等通知。 通知的类型将在后面部分进行讨论。许多AOP框架，包括Spring，都是以拦截器做通知模型， 并维护一个以连接点为中心的拦截器链。

**切入点（Pointcut）：** 匹配连接点（Joinpoint）的断言。通知和一个切入点表达式关联，并在满足这个切入点的连接点上运行（例如，当执行某个特定名称的方法时）。 切入点表达式如何和连接点匹配是AOP的核心：Spring缺省使用AspectJ切入点语法。

**引入（Introduction）：** （也被称为内部类型声明（inter-type declaration））。声明额外的方法或者某个类型的字段。 Spring允许引入新的接口（以及一个对应的实现）到任何被代理的对象。 例如，你可以使用一个引入来使bean实现 IsModified 接口，以便简化缓存机制。

**目标对象（Target Object）：** 被一个或者多个切面（aspect）所通知（advise）的对象。也有人把它叫做 被通知（advised） 对象。 既然Spring AOP是通过运行时代理实现的，这个对象永远是一个 被代理（proxied） 对象。

**AOP代理（AOP Proxy）：** AOP框架创建的对象，用来实现切面契约（aspect contract）（包括通知方法执行等功能）。 在Spring中，AOP代理可以是JDK动态代理或者CGLIB代理。 注意：Spring 2.0最新引入的基于模式（schema-based）风格和@AspectJ注解风格的切面声明，对于使用这些风格的用户来说，代理的创建是透明的。

**织入（Weaving）：** 把切面（aspect）连接到其它的应用程序类型或者对象上，并创建一个被通知（advised）的对象。 这些可以在编译时（例如使用AspectJ编译器），类加载时和运行时完成。 Spring和其他纯Java AOP框架一样，在运行时完成织入。

**通知的类型：**

**前置通知（Before advice）：** 在某连接点（join point）之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。

**返回后通知（After returning advice）：** 在某连接点（join point）正常完成后执行的通知：例如，一个方法没有抛出任何异常，正常返回。

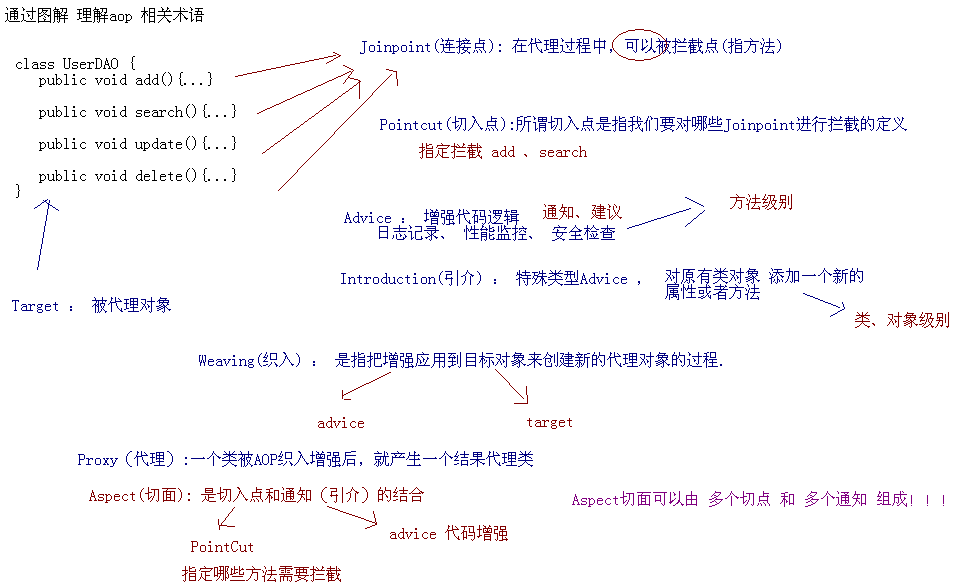
**抛出异常后通知（After throwing advice）：** 在方法抛出异常退出时执行的通知。

**后通知（After (finally) advice）：** 当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。

**环绕通知（Around Advice）：** 包围一个连接点（join point）的通知，如方法调用。这是最强大的一种通知类型。 环绕通知可以在方法调用前后完成自定义的行为。它也会选择是否继续执行连接点或直接返回它们自己的返回值或抛出异常来结束执行。

环绕通知是最常用的一种通知类型。大部分基于拦截的AOP框架，例如Nanning和JBoss4，都只提供环绕通知。

切入点（pointcut）和连接点（join point）匹配的概念是AOP的关键，这使得AOP不同于其它仅仅提供拦截功能的旧技术。 切入点使得定位通知（advice）可独立于OO层次。 例如，一个提供声明式事务管理的around通知可以被应用到一组横跨多个对象中的方法上（例如服务层的所有业务操作）。



### 请介绍一下Spring框架中Bean的生命周期和作用域

(1)  bean定义

在配置文件里面用<bean></bean>来进行定义。

(2)  bean初始化

    有两种方式初始化:

A.在配置文件中通过指定init-method属性来完成

B.实现org.springframwork.beans.factory.InitializingBean接口

(3)  bean调用

有三种方式可以得到bean实例，并进行调用

(4)  bean销毁

销毁有两种方式

A.使用配置文件指定的destroy-method属性

B.实现org.springframwork.bean.factory.DisposeableBean接口

##作用域

**singleton**

当一个bean的作用域为singleton, 那么Spring IoC容器中只会存在一个共享的bean实例，并且所有对bean的请求，只要id与该bean定义相匹配，则只会返回bean的同一实例。

**prototype**

Prototype作用域的bean会导致在每次对该bean请求（将其注入到另一个bean中，或者以程序的方式调用容器的getBean() 方法）时都会创建一个新的bean实例。根据经验，对所有有状态的bean应该使用prototype作用域，而对无状态的bean则应该使用 singleton作用域

**request**

在一次HTTP请求中，一个bean定义对应一个实例；即每次HTTP请求将会有各自的bean实例， 它们依据某个bean定义创建而成。该作用 域仅在基于web的Spring ApplicationContext情形下有效。

**session**

在一个HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring

ApplicationContext情形下有效。

**global session**

在一个全局的HTTP Session中，一个bean定义对应一个实例。典型情况下，仅在使用portlet

context的时候有效。该作用域仅在基于 web的Spring ApplicationContext情形下有效。

### Bean的初始化方式有哪些？

有两种方式初始化Bean。   
1、在配置文档中通过指定init-method 属性来完成   
 在Bean的类中实现一个初始化Bean属性的方法，如init()，如：   
 public class HelloWorld{   
 public String msg=null;   
 public Date date=null;   
  
 public void init() {   
 msg=”HelloWorld”;   
 date=new Date();   
 }   
 ……   
 }   
 然后，在配置文件中设置init-mothod属性：   
<bean id=”HelloWorld” class=”com.pqf.beans.HelloWorld” init-mothod=”init” >   
</bean>   
  
2、实现 org.springframwork.beans.factory.InitializingBean接口   
Bean实现InitializingBean接口，并且增加 afterPropertiesSet() 方法：   
 public class HelloWorld implement InitializingBean {   
 public String msg=null;   
 public Date date=null;   
  
 public void afterPropertiesSet() {   
 msg=”向全世界问好！”;   
 date=new Date();   
 }   
 ……   
 }   
那么，当这个Bean的所有属性被Spring的BeanFactory设置完后，会自动调用afterPropertiesSet()方法对 Bean进行初始化，于是，配置文件就不用指定 init-method属性了。

### Bean的调用方式有哪些？

有三种方式可以得到Bean并进行调用：   
**1、使用BeanWrapper**   
 HelloWorld hw=new HelloWorld();   
 BeanWrapper bw=new BeanWrapperImpl(hw);   
 bw.setPropertyvalue(”msg”,”HelloWorld”);   
 system.out.println(bw.getPropertyCalue(”msg”));   
**2、使用BeanFactory**   
 InputStream is=new FileInputStream(”config.xml”);   
 XmlBeanFactory factory=new XmlBeanFactory(is);   
 HelloWorld hw=(HelloWorld) factory.getBean(”HelloWorld”);   
 system.out.println(hw.getMsg());   
**3、使用ApplicationConttext**   
 ApplicationContext actx=new FleSystemXmlApplicationContext(”config.xml”);   
 HelloWorld hw=(HelloWorld) actx.getBean(”HelloWorld”);   
 System.out.println(hw.getMsg());

### Bean的销毁方式有哪些？

1、使用配置文件中的 destory-method 属性   
 与初始化属性 init-methods类似，在Bean的类中实现一个撤销Bean的方法，然后在配置文件中通过 destory-method指定，那么当bean销毁时，Spring将自动调用指定的销毁方法。   
2、实现 org.springframwork.bean.factory.DisposebleBean接口   
 如果实现了DisposebleBean接口，那么Spring将自动调用bean中的Destory方法进行销毁，所以，Bean中必须提供 Destory方法。

### 介绍一下Spring的事务管理：

事务就是对一系列的数据库操作（比如插入多条数据）进行统一的提交或回滚操作，如果插入成功，那么一起成功，如果中间有一条出现异常，那么回滚之 前的所有操作。

这样可以防止出现脏数据，防止数据库数据出现问题。

开发中为了避免这种情况一般都会进行事务管理。Spring中也有自己的事务管理机制，一般是使用TransactionMananger进行管 理，可以通过Spring的注入来完成此功能。

spring提供了几个关于事务处理的类：

TransactionDefinition //事务属性定义

TranscationStatus //代表了当前的事务，可以提交，回滚。

PlatformTransactionManager这个是spring提供的用于管理事务的基础接口，其下有一个实现的抽象类 AbstractPlatformTransactionManager,我们使用的事务管理类例如 DataSourceTransactionManager等都是这个类的子类。

一般事务定义步骤：

TransactionDefinition td = new TransactionDefinition();

TransactionStatus ts = transactionManager.getTransaction(td);

try

{ //do sth

transactionManager.commit(ts);

}

catch(Exception e){transactionManager.rollback(ts);}

spring提供的事务管理可以分为两类：编程式的和声明式的。编程式的，比较灵活，但是代码量大，存在重复的代码比较多；声明式的比编程式的更 灵活。

编程式主要使用transactionTemplate。省略了部分的提交，回滚，一系列的事务对象定义，需注入事务管理对象.

void add(){

transactionTemplate.execute( new TransactionCallback(){

pulic Object doInTransaction(TransactionStatus ts)

{ //do sth}

}

}

声明式：

使用TransactionProxyFactoryBean:

PROPAGATION\_REQUIRED PROPAGATION\_REQUIRED PROPAGATION\_REQUIRED,readOnly

围绕Poxy的动态代理 能够自动的提交和回滚事务

org.springframework.transaction.interceptor.TransactionProxyFactoryBean

PROPAGATION\_REQUIRED–支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

PROPAGATION\_SUPPORTS–支持当前事务，如果当前没有事务，就以非事务方式执行。

PROPAGATION\_MANDATORY–支持当前事务，如果当前没有事务，就抛出异常。

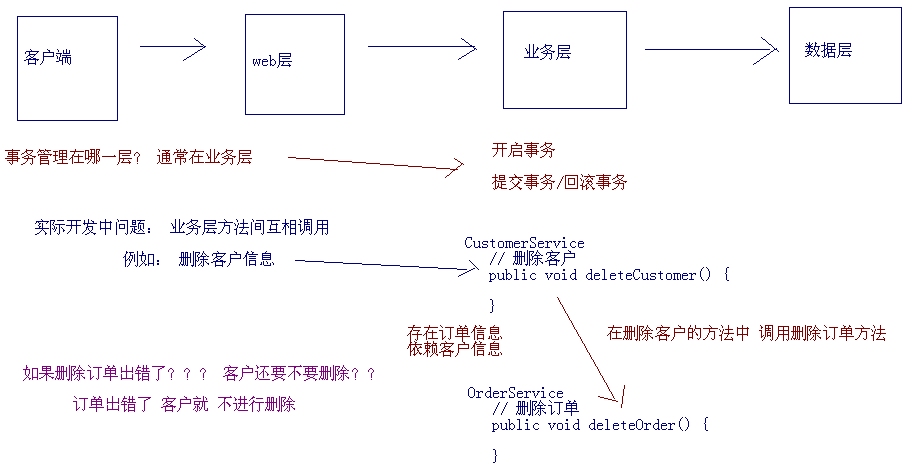
PROPAGATION\_REQUIRES\_NEW–新建事务，如果当前存在事务，把当前事务挂起。

PROPAGATION\_NOT\_SUPPORTED–以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

PROPAGATION\_NEVER–以非事务方式执行，如果当前存在事务，则抛出异常。

PROPAGATION\_NESTED–如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则进行与 PROPAGATION\_REQUIRED类似的操作。

### 为什么要有事务传播行为？



### Spring里面如何配置数据库驱动？

使用”org.springframework.jdbc.datasource.DriverManagerDataSource”数据源来配置数据库驱动。示例如下：

<bean id=”dataSource”>

    <property name=”driverClassName”>

        <value>org.hsqldb.jdbcDriver</value> </property>

    <property name=”url”>

        <value>jdbc:hsqldb:db/appfuse</value>

    </property>

    <property name=”username”><value>sa</value></property>

    <property name=”password”><value></value></property>

</bean>

### Spring里面applicationContext.xml文件能不能改成其他文件名？

ContextLoaderListener是一个ServletContextListener, 它在你的web应用启动的时候初始化。缺省情况下， 它会在WEB-INF/applicationContext.xml文件找Spring的配置。 你可以通过定义一个<context-param>元素名字为”contextConfigLocation”来改变Spring配置文件的 位置。示例如下：

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener

<context-param>

        <param-name>contextConfigLocation</param-name>

        <param-value>/WEB-INF/xyz.xml</param-value>

</context-param>

</listener-class>

</listener>

### 如何在web应用里面配置spring?

在web.xml中加入如下同容,在启动web服务器时加载/WEB-INF/applicationContext.xml中的内容。

<servlet>

<servlet-name>context</servlet-name>

<servlet-class>

org.springframework.web.context.ContextLoaderServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

通过如下类得到ApplicationContext实例

    WebApplicationContextUtils.getWebApplicationContext

### Spring里面如何定义hibernate mapping？

添加hibernate mapping 文件到web/WEB-INF目录下的applicationContext.xml文件里面。示例如下：

<property name=”mappingResources”>

    <list>

        <value>org/appfuse/model/User.hbm.xml</value>

    </list>

</property>

### spring中的BeanFactory与ApplicationContext的作用有哪些？

1. BeanFactory负责读取bean配置文档，管理bean的加载，实例化，维护bean之间的依赖关系，负责bean的声明周期。

2. ApplicationContext除了提供上述BeanFactory所能提供的功能之外，还提供了更完整的框架功能：

a. 国际化支持

b. 资源访问：Resource rs = ctx. getResource(”classpath:config.properties”), “file:c:/config.properties”

c. 事件传递：通过实现ApplicationContextAware接口

3. 常用的获取ApplicationContext的方法：

FileSystemXmlApplicationContext：从文件系统或者url指定的xml配置文件创建，参数为配置文件名或文件名数 组

ClassPathXmlApplicationContext：从classpath的xml配置文件创建，可以从jar包中读取配置文件

WebApplicationContextUtils：从web应用的根目录读取配置文件，需要先在web.xml中配置，可以配置监听器或者 servlet来实现

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<servlet>

<servlet-name>context</servlet-name>

<servlet-class>

org.springframework.web.context.ContextLoaderServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

这两种方式都默认配置文件为web-inf/applicationContext.xml，也可使用context-param指定配置文件

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/myApplicationContext.xml</param-value>

</context-param>

### BeanFactory 接口 和 ApplicationContext 接口区别 ？

\* ApplicationContext 接口继承 BeanFactory 接口 ，Spring核心工厂是BeanFactory ,BeanFactory 采取延迟加载，第一次getBean时才会初始化Bean , ApplicationContext 是会在加载配置文件时初始化Bean

\* ApplicationContext是对BeanFactory扩展

国际化处理

事件传递

Bean自动装配

各种不同应用层的Context实现

开发中基本都在使用ApplicationContext, web项目使用WebApplicationContext ，很少用到BeanFactory

BeanFactory beanFactory = new XmlBeanFactory(new ClassPathResource("applicationContext.xml"));

IHelloService helloService = (IHelloService) beanFactory.getBean("helloService");

helloService.sayHello();

### 如何在web环境中配置applicationContext.xml文件?

<listener>

   <listener-class>

    org.springframework.web.context.ContextLoaderListener

   </listener-class>

</listener>

或：

<servlet>

   <servlet-name>context</servlet-name>

    <servlet-class>

     org.springframework.web.context.ContextLoaderServlet

    </servlet-class>

   <load-on-startup>1</load-on-startup>

</servlet>

通过如下方法取出applicationContext实例:

ApplicationContext ac=WebApplicationContextUtils.

getWebApplicationContext(this.getServletContext);

### 如何在spring中实现国际化？

在applicationContext.xml加载一个bean

<bean id=”messageSource”

class=”org.springframework.context.support.ResourceBundleMessageSource”>

<property name=”basename”>

    <value>message</value>

</property>

</bean>

? 在src目录下建多个properties文件

? 对于非英文的要用native2ascii -encoding gb2312 源 目转化文件相关内容

? 其命名格式是message\_语言\_国家。

? 页面中的中显示提示信息，键名取键值。

? 当给定国家，系统会自动加载对应的国家的properties信息。

? 通过applictionContext.getMessage(“键名”,”参数”,”区域”)取出相关的信息。

### [Spring如何处理线程并发](G:\Users\é¦å¤´\Desktop\é¢è¯é¢\Spring如何处理线程并发 - OPEN开源论坛_files\Spring如何处理线程并发 - OPEN开源论坛.htm)问题？

**Spring使用ThreadLocal解决线程安全问题**

我们知道在一般情况下，只有无状态的Bean才可以在多线程环境下共享，在Spring中，绝大部分Bean都可以声明为singleton作用域。就是因为Spring对一些Bean(如RequestContextHolder、TransactionSynchronizationManager、LocaleContextHolder等)中非线程安全状态采用ThreadLocal进行处理，让它们也成为线程安全的状态，因为有状态的Bean就可以在多线程中共享了。

ThreadLocal和线程同步机制都是为了解决多线程中相同变量的访问冲突问题。

在同步机制中，通过对象的锁机制保证同一时间只有一个线程访问变量。这时该变量是多个线程共享的，使用同步机制要求程序慎密地分析什么时候对变量进行读写，什么时候需要锁定某个对象，什么时候释放对象锁等繁杂的问题，程序设计和编写难度相对较大。

而ThreadLocal则从另一个角度来解决多线程的并发访问。ThreadLocal会为每一个线程提供一个独立的变量副本，从而隔离了多个线程对数据的访问冲突。因为每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。ThreadLocal提供了线程安全的共享对象，在编写多线程代码时，可以把不安全的变量封装进ThreadLocal。

由于ThreadLocal中可以持有任何类型的对象，低版本JDK所提供的get()返回的是Object对象，需要强制类型转换。但JDK5.0通过泛型很好的解决了这个问题，在一定程度地简化ThreadLocal的使用。

概括起来说，对于多线程资源共享的问题，同步机制采用了“以时间换空间”的方式，而ThreadLocal采用了“以空间换时间”的方式。前者仅提供一份变量，让不同的线程排队访问，而后者为每一个线程都提供了一份变量，因此可以同时访问而互不影响。

### 简单叙述一下Spring 中BeanFactory与ApplicationContext的差别？

使用 BeanFactory 从 xml配置文件加载bean:

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.core.io.FileSystemResource;

public class XmlConfigWithBeanFactory {

public static void main(String[] args) {

XmlBeanFactory factory = new XmlBeanFactory(new FileSystemRe

"build/beans.xml"));

}

}

使用 ApplicationConText 从xml 配置文件加载 bean:

public class XmlConfigWithApplication{

public static void main(String[] args){

ApplicationContext application =

new ClassPathXmlApplicationContext(beans.xml"));

application.getBean("BeanName");

}

}

简而言之，BeanFactory 提供了配置框架和基本的功能， 而ApplicationContext 为它增加了更强的功能，这些功能中的一些或许更加接近J2EE 并且围绕企业级应用。一般来说，ApplicationContext是 BeanFactory 的完全超集， 任何 BeanFactory 功能和行为的描述也同样被认为适用于ApplicationContext

相对于 BeanFactory 而言,ApplicationContext 提供了以下扩展功能.

(a) 国际化支持

(b) 资源访问

(c) 事件传播

(d) 多实例加载