Struts2

1. 谈谈你 mvc 的理解

MVC 是 Model—View—Controler 的简称。即模型—视图—控制器。MVC 是一种设计模式，它强制性的把应用程序的输入、处理和输出分开。MVC 中的模型、视图、控制器它们分别担负着不同的任务。

视图：视图是用户看到并与之交互的界面。视图向用户显示相关的数据，并接受用户的输入。视图不进行任何业务逻辑处理。

模型：模型表示业务数据和业务处理。相当于 JavaBean。一个模型能为多个视图提供数据。这提高了应用程序的重用性

控制器：当用户单击 Web 页面中的提交按钮时，控制器接受请求并调用相应的模型去处理请求。然后根据处理的结果调用相应的视图来显示处理的结果。

MVC 的处理过程：首先控制器接受用户的请求，调用相应的模型来进行业务处理，并返回数据给控制器。控制器调用相应的视图来显示处理的结果。并通过视图呈现给用户。

2. Struts2 和 Struts1 关系

struts1 和 struts2.0 的对比

a、Action 类：

struts1 要求 Action 类继承一个基类。struts2Action 要求继承 ActionSupport 基类

b、线程模式

struts1Action 是单例模式的并且必须是线程安全的，因为仅有一个 Action 的实例来处理所有的请求。

单例策略限制了 Struts1.2 Action 能做的事情，并且开发时特别小心。Action 资源必须是线程安全的或同步的。

struts2Action 为每一个请求产生一个实例，因此没有线程安全问题。

c、Servlet 依赖

struts1 的 Action 依赖于 Servlet API，因为当一个 Action 被调用时 HttpServletRequest 和 HttpServletResponse 被传递给 execut 方法。

struts2Action 不依赖于容器，允许 Action 脱离容器单独测试。如果需要，Struts2 Action 仍然可以访问初始的 Request 和 Response。但是，其他的元素减少或者消除了直接访问 HttpServletRequest 和 HttpServletResponse 的必要性。

d、可测性

测试 struts1 Action 的一个主要问题是 execute 方法暴露了 Servlet API（这使得测试要依赖于容器）。一个第三方扩展：struts TestCase 提供了一套 struts1.2 的模拟对象来进行测试。

Struts2Action 可以通过初始化、设置属性、调用方法来测试，“依赖注入”也使得测试更容易。

3. struts2 的拦截器使用了哪种设计模式 ？

拦截器采用责任链编程，是一种新的编程设计理念,面向切面编程 AOP 前端控制器

4. Struts 的运行流程是什么

①. 请求发送给 StrutsPrepareAndExecuteFilter

②. StrutsPrepareAndExecuteFilter 判定该请求是否是一个 Struts2 请求

③. 若该请求是一个 Struts2 请求,则 StrutsPrepareAndExecuteFilter 把请求的处理交给 ActionProxy

④. ActionProxy 创建一个 ActionInvocation 的实例，并进行初始化

⑤. ActionInvocation 实例在调用 Action 的过程前后,涉及到相关拦截器（Intercepter）的调用。

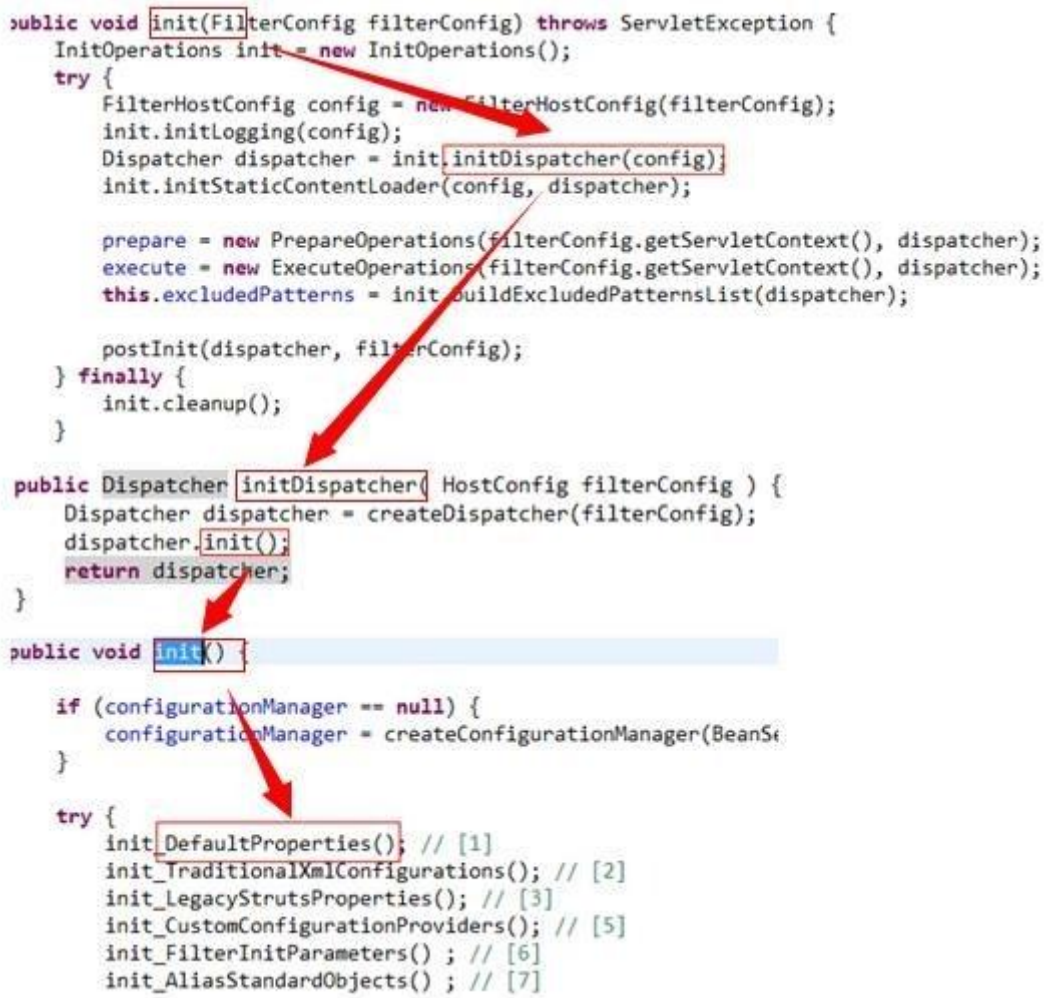
⑥. Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果。调用结果的 execute 方法，渲染结果。

⑦. 执行各个拦截器 invocation.invoke() 之后的代码

⑧. 把结果发送到客户端

5. Struts2 配置文件加载顺序

通过查看 StrutsPrepareAndExecuteFilter 源码可以得到



```
public void init(FilterConfig filterConfig) throws ServletException {
    InitOperations init = new InitOperations();
    try {
        FilterHostConfig config = new FilterHostConfig(filterConfig);
        init.initLogging(config);
        Dispatcher dispatcher = init.initDispatcher(config);
        init.initStaticContentLoader(config, dispatcher);

        prepare = new PrepareOperations(filterConfig.getServletContext(), dispatcher);
        execute = new ExecuteOperations(filterConfig.getServletContext(), dispatcher);
        this.excludedPatterns = init.buildExcludedPatternsList(dispatcher);

        postInit(dispatcher, filterConfig);
    } finally {
        init.cleanup();
    }

public Dispatcher initDispatcher( HostConfig filterConfig ) {
    Dispatcher dispatcher = createDispatcher(filterConfig);
    dispatcher.init();
    return dispatcher;
}

public void init() {

    if (configurationManager == null) {
        configurationManager = createConfigurationManager(BeanSe
    }

    try {
        init_DefaultProperties(); // [1]
        init_TraditionalXmlConfigurations(); // [2]
        init_LegacyStrutsProperties(); // [3]
        init_CustomConfigurationProviders(); // [5]
        init_FilterInitParameters() ; // [6]
        init_AliasStandardObjects() ; // [7]
```

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 ）

6. Action 是如何接受请求参数的？

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

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

第三种 ：使用 ModelDriven 接口，对请求数据进行封装 （模型驱动 ） ----- 企业开发的主流(模型驱动有很多特性)

7. struts2 的国际化信息文件有哪几类？

1） 全局范围

src 下编写 messages.properties

2） 包范围

action 所在包或其父包下编写 package.properties

3） action 范围

action 所在包下编写 action 类名.properties

4） 临时指定

jsp 页面中使用<s:i18n name=""><s:text name=""></s:text></s:i18n>

8. struts2 的 Action 中如何使用 ServletAPI？

1）解耦合的方式

getContext() 返回 ActionContext 实例对象

get(key) 相当于 HttpServletRequest 的 getAttribute(String name)方法

put(String,Object)

getApplication() 返回一个 Map 对象，存取 ServletContext 属性

getSession() 返回一个 Map 对象，存取 HttpSession 属性

getParameters() 类似调用 HttpServletRequest 的 getParameterMap()方法

2）接口注入方式

实现接口 ServletContextAware 向 Action 中注入 ServletContext 对象

实现接口 ServletRequestAware 向 Action 中注入 ServletRequest 对象

实现接口 ServletResponseAware 向 Action 中注入 ServletResponse 对象

3）ServletActionContext

直接使用 servlet API

9. struts2 中有哪些常用结果类型？你用过哪些？

Chain Result-->type="chain"用来处理 Action 链
Dispatcher Result -->type="dispatcher"用来转向页面，通常处理 JSP
Redirect Result -->type="redirect"重定向到一个 URL
Redirect Action Result -->type="redirectAction"重定向到一个 Action
Stream Result -->type="stream"向浏览器发送 InputSream 对象，通常用来处理文件下载
FreeMarker Result -->type="freemarker"处理 FreeMarker 模板
HttpHeader Result -->type="httpheader"用来控制特殊的 Http 行为
Velocity Result -->type="velocity"处理 Velocity 模板
XLST Result -->type="xslt"处理 XML/XLST 模板
PlainText Result -->type="plainText"

10. 你是否在 struts2 开发中 自定义过拦截器，实现什么功能？

权限控制
表单重复提交
请求参数解析
文件上传
cookie 处理
Action 运行时间监控

11. OGNL 表达式中值栈

问题一 ： 什么是值栈 ValueStack ？

ValueStack 实际是一个接口,在 Struts2 中利用 OGNL 时，实际上使用的是实现了该接口的 OgnlValueStack 类,这个类是 Struts2 利用 OGNL 的基础 -- 值栈对象（OGNL 是从值栈中获取数据的）

*每个 Action 实例都有一个 ValueStack 对象 （一个请求对应一个 ValueStack 对象）

*在其中保存当前 Action 对象和其他相关对象（值栈中是有 Action 引用的）

*Struts 框架把 ValueStack 对象保存在名为 “struts.valueStack” 的请求属性中,request 中 （值栈对象 是 request 一个属性）

问题二 ： 值栈的内部结构 ？

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

12. xml 进行 struts2 请求参数校验时，指定方法的校验文件和所有方法校验文件命名规则是什么？

RegisterAction-validation.xml

RegisterAction-customer_add-validation.xml

11. Struts2 中的默认包 struts-default 有什么作用？

1、 struts-default 包是 struts2 内置的，它定义了 struts2 内部的众多拦截器和 Result 类型，而 Struts2 很多核心的功能都是通过这些内置的拦截器实现，如：从请求中把请求参数封装到 action、文件上传和数据验证等等都是通过拦截器实现的。当包继承了 struts-default 包才能使用 struts2 为我们提供的这些功能。

2、.struts-default 包是在 struts-default.xml 中定义，struts-default.xml 也是 Struts2 默认配置文件。Struts2 每次都会自动加载 struts-default.xml 文件。

3、. 通常每个包都应该继承 struts-default 包。

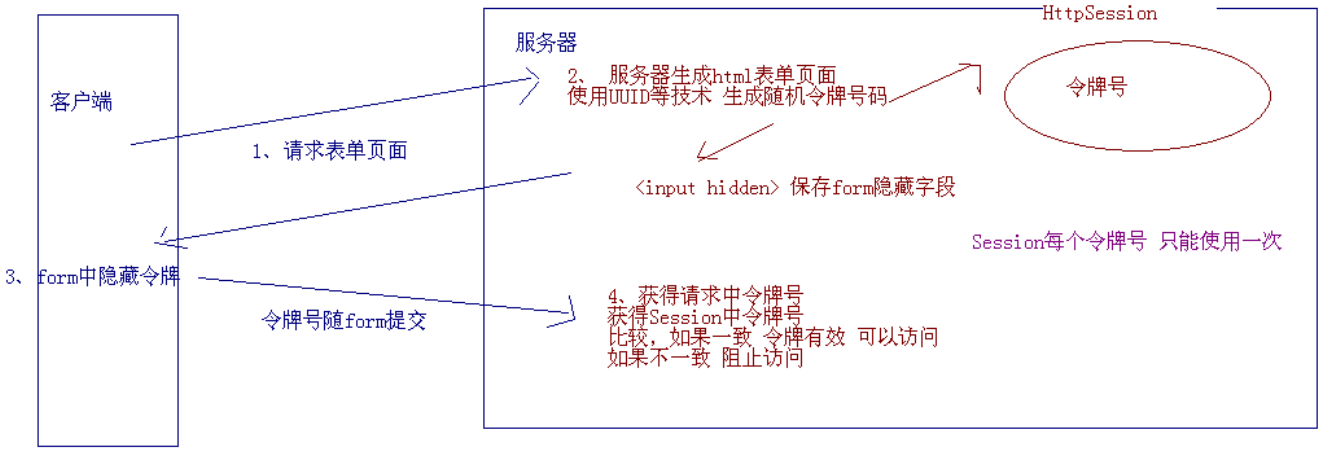
12. Struts2 中如何防止表单重复提交

设置 Struts 2 的预防表单重复提交的功能

Struts 2 标签中的 token 标签，可以用来生成一个独一无二的标记。这个标记必须嵌套在 form 标签中使用，它会在表单里插入一个隐藏字段并把标记保存到 HttpSession 对象里。toke 标签必须与 Token 或 Token Session 拦截器配合使用，两个拦截器都能对 token 标签进行处理。Token 拦截器遇到重复提交表单的情况，会返回一 个"invalid.token"结果并加上一个动作级别的错误。Token Session 拦截器扩展了 Token 拦截器并提供了一种更复杂的服务，它采取的做法与 Token 拦截器不同，它只是阻断了后续的提交，这样用户不提交多 少次，就好像只是提交了一次。

防止表单重复提交 ----- 令牌机制

哪些情况会导致重复提交
1、 服务器处理服务后，转发页面 ， 客户端点击刷新 （重定向）
2、 客户端网络过慢 按钮连续点击 （按钮点击一次后，禁用按钮）



13. action 是单实例还是多实例，为什么？

action 是单实例的。当多个用户访问一个请求的时候，服务器内存中只有一个与之对应的 action 类对象。

因为当服务器第一次加载 struts 的配置文件的时候，创建了一个 Action 后，每发送一个请求，服务器都会先去检索相应的范围内(request, session)是否存在这样一个 action 实例，如果存在，则使用这个实例，如果不存在，则创建一个 action 实例。

14. struts2 UI 主题有哪些？你用过哪些底层实现是什么？

simple, xhtml

15. struts2 中如何使用 Ajax ？

导入 struts2 解压目录 lib / struts2-json-plugin-2.3.7.jar

1、 返回 SUCCESS 结果集

2、 在 struts.xml 定义 <package> extends="json-default"

json-default extends struts-default 对功能进行扩展

3、 定义结果集，type="json"

16. struts2 的 Action 有几种书写方式？

POJO 类， implements Action， extends ActionSupport

17. addFieldError、addActionError 和 addActionMessage 有何区别？

一个字段的错误信息
一个 action 的错误信息
普通的 action 回显信息

Hibernate

1. 什么是 Hibernate 的并发机制？怎么去处理并发问题？

Hibernate 并发机制：

a、Hibernate 的 Session 对象是非线程安全的，对于单个请求，单个会话，单个的工作单元(即单个事务，单个线程)，它通常只使用一次，然后就丢弃。如果一个 Session 实例允许共享的话，那些支持并发运行的，例如 Http request，session beans 将会导致出现资源争用。如果在 Http Session 中有 hibernate 的 Session 的话，就可能会出现同步访问 Http Session。只要用户足够快的点击浏览器的“刷新”， 就会导致两个并发运行的线程使用同一个 Session。

b、多个事务并发访问同一块资源，可能会引发第一类丢失更新，脏读，幻读，不可重复读，第二类丢失更新一系列的问题。

解决方案：设置事务隔离级别。

Serializable：串行化。隔离级别最高
Repeatable Read：可重复读
Read Committed：已提交数据读
Read Uncommitted：未提交数据读。隔离级别最差

设置锁：乐观锁和悲观锁。

乐观锁：使用版本号或时间戳来检测更新丢失，在<class>的映射中设置 optimistic-lock="all"可以在没有版本或者时间戳属性映射的情况下实现 版本检查，此时 Hibernate 将比较一行记录的每个字段的状态 行级悲观锁：Hibernate 总是使用数据库的锁定机制，从不在内存中锁定对象！只要为 JDBC 连接指定一下隔 离级别，然后让数据库去搞

定一切就够了。类 LockMode 定义了 Hibernate 所需的不同的锁定级别：LockMode.UPGRADE,LockMode.UPGRADE_NOWAIT,LockMode.READ;

2. 什么是 Hibernate

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。

Hibernate 的核心接口一共有 5 个，分别为:Session、SessionFactory、Transaction、Query 和 Configuration。这 5 个核心接口在任何开发中都会用到。通过这些接口，不仅可以对持久化对象进行存取，还能够进行事务控制。下面对这五个核心接口分别加以介绍。

·Session 接口:Session 接口负责执行被持久化对象的 CRUD 操作(CRUD 的任务是完成与数据库的交流，包含了很多常见的 SQL 语句。)。但需要注意的是 Session 对象是非线程安全的。同时，Hibernate 的 session 不同于 JSP 应用中的 HttpSession。这里当使用 session 这个术语时，其实指的是 Hibernate 中的 session，而以后会将 HttpSesion 对象称为用户 session。

·SessionFactory 接口:SessionFactroy 接口负责初始化 Hibernate。它充当数据存储源的代理，并负责创建 Session 对象。这里用到了工厂模式。需要注意的是 SessionFactory 并不是轻量级的，因为一般情况下，一个项目通常只需要一个 SessionFactory 就够，当需要操作多个数据库时，可以为每个数据库指定一个 SessionFactory。

·Configuration 接口:Configuration 接口负责配置并启动 Hibernate，创建 SessionFactory 对象。在 Hibernate 的启动的过程中，Configuration 类的实例首先定位映射文档位置、读取配置，然后创建 SessionFactory 对象。

·Transaction 接口:Transaction 接口负责事务相关的操作。它是可选的，开发人员也可以设计编写自己的底层事务处理代码。

·Query 和 Criteria 接口:Query 和 Criteria 接口负责执行各种数据库查询。它可以使用 HQL 语言或 SQL 语句两种表达方式。

3. Hibernate 与 jdbc 的联系

hibernate 是 jdbc 的轻量级封装，包括 jdbc 的与数据库的连接（用 hibernate.property 的配置文件实现当然本质是封装了 jdbc 的 forname），和查询，删除等代码，都用面向对象的思想用代码联系起来，hibernate 通过 hbm 配置文件把 po 类的字段和数据库的字段关联起来比如数据库的 id,在 po 类中就是 pravite Long id; public Long getId() ;public setId(Long id);然后 hql 语句也是面向对象的,它的查询语句不是查询数据库而是查询类的,这些实现的魔法就是 xml 文件，其实 hibernate=封装的 jdbc+xml 文件

4. Hibernate 与 spring 的联系

hibernate 中的一些对象可以给 Spring 来管理,让 Spring 容器来创建 hibernate 中一些对象实例化。例如：SessionFactory，HibernateTemplate 等。

Hibernate 本来是对数据库的一些操作,放在 DAO 层,而 Spring 给业务层的方法定义了事务,业务层调用 DAO 层的方法,很好的将 Hibernate 的操作也加入到事务中来了。

5. 为什么要使用 Hibernate 开发你的项目呢？Hibernate 的开发流程是怎么样的？

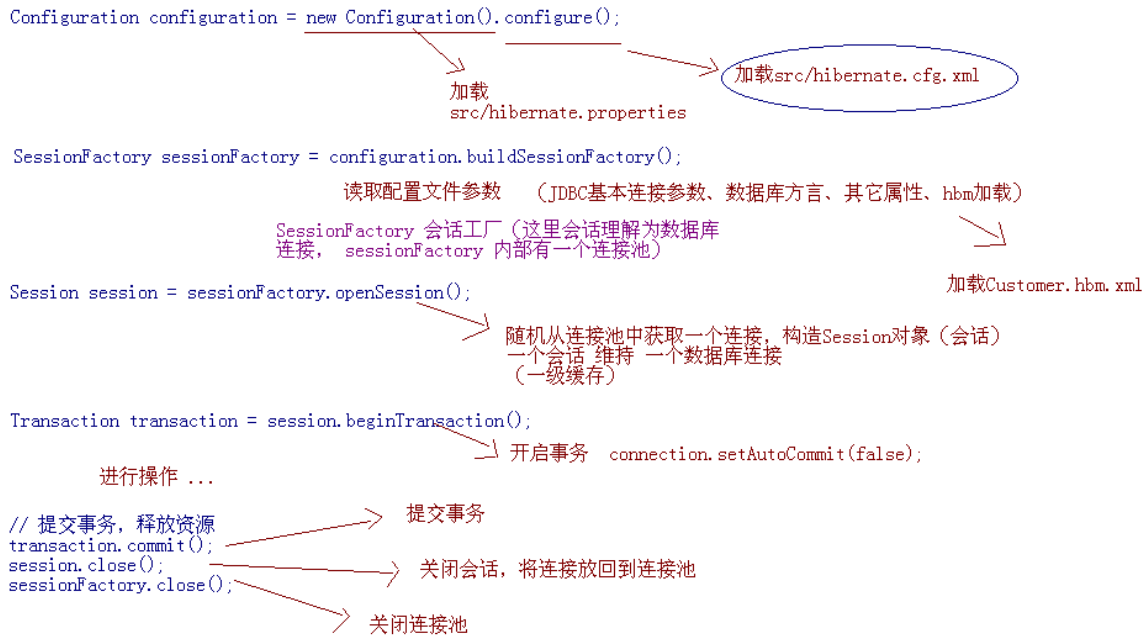
为什么要使用

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。

开发流程



6. 什么是延迟加载？

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。

7. Hibernate 中 GET 和 LOAD 的区别？。

session.get 方法， 查询立即执行 ， 返回 Customer 类对象

session.load 方法，默认采用延迟加载数据方式，不会立即查询，返回 Customer 类子类对象 （动态生成代理对象）

session 的 get 方法如果查询对象不存在则返回 null 使用 load 对象不存在则报错

* 如果 PO 类使用 final 修饰，load 无法创建代理对象，返回目标对象本身 （load 效果和 get 效果 相同 ）

8. Hibernate 自带的分页机制是什么？如果不使用 Hibernate 自带的分页，则采用什么方式分页？

1、hibernate 自带的分页机制：获得 Session 对象后，从 Session 中获得 Query 对象。用 Query.setFirstResult()：设置要显示的第一行数据,

Query.setMaxResults()：设置要显示的最后一行数据。

2、不使用 hibernate 自带的分页,可采用 sql 语句分页,

Oracle 分页语句和 mysql 分页语句自己写。

9. hibernate 的对象的三种持久化状态，并给出解释？以及状态之间的转换

* transient 瞬时态(临时态、自由态） ： 不存在持久化标识 OID，尚未与 Hibernate Session 关联对象，被认为处于瞬时态，失去引用将被 JVM 回收

OID 就是 对象中 与数据库主键 映射 属性 ，例如 Customer 类 id 属性

* persistent 持久态 ： 存在持久化标识 OID，与当前 session 有关联，并且相关联的 session 没有关闭 ,并且事务未提交

* detached 脱管态(离线态、游离态） ： 存在持久化标识 OID，但没有与当前 session 关联，脱管状态改变 hibernate 不能检测到

10. hibernate 的三种状态之间如何转换

1）瞬时态对象 通过 new 获得

瞬时--持久 save 、 saveOrUpdate （都是 Session）

瞬时--脱管 book.setId(1)； 为瞬时对象设置 OID

2） 持久态对象 get/load 、Query 查询获得

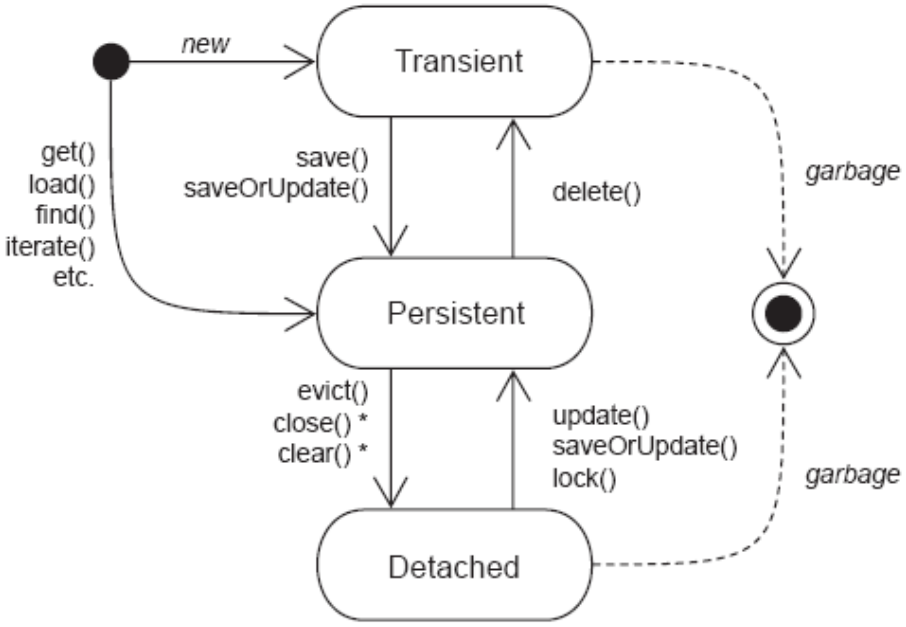
持久--瞬时 delete （被删除持久化对象 不建议再次使用 ）

持久--脱管 evict(清除一级缓存中某一个对象)、close(关闭 Session，清除一级缓存)、clear(清除一级缓存所有对象 ）

3） 脱管态对象 无法直接获得

脱管--瞬时 book.setId(null)；删除对象 OID

脱管--持久 update、saveOrUpdate、 lock(过时)



11. update（）和 saveOrUpdate（）的区别？

update()和 saveOrUpdate()是用来对跨 Session 的 PO 进行状态管理的。

update()方法操作的对象必须是持久化了的对象。也就是说，如果此对象在数据库中不存在的话，就不能使用 update()方法。

saveOrUpdate()方法操作的对象既可以使持久化了的，也可以使没有持久化的对象。如果是持久化了的对象调用 saveOrUpdate()则会

更新数据库中的对象；如果是未持久化的对象使用此方法,则 save 到数据库中。

12. Hibernate 的 inverse 属性和 Cascade 的作用？

1)明确 inverse 和 cascade 的作用

inverse 决定是否把对对象中集合的改动反映到数据库中，所以 inverse 只对集合起作用，也就是只对 one-to-many 或 many-to-many 有效（因为只有这两种关联关系包含集合，而 one-to-one 和 many-to-one 只含有关系对方的一个引用）。

cascade 决定是否把对对象的改动反映到数据库中，所以 cascade 对所有的关联关系都起作用（因为关联关系就是指对象之间的关联关系）。

2)inverse 和 cascade 的区别

作用的范围不同：

Inverse 是设置在集合元素中的。

Cascade 对于所有涉及到关联的元素都有效。

<many-to-one/><ont-to-many/>没有 inverse 属性，但有 cascade 属性

执行的策略不同

Inverse 会首先判断集合的变化情况，然后针对变化执行相应的处理。

Cascade 是直接对集合中每个元素执行相应的处理

执行的时机不同

Inverse 是在执行 SQL 语句之前判断是否要执行该 SQL 语句

Cascade 则在主控方发生操作时用来判断是否要进行级联操作

执行的目标不同

Inverse 对于<ont-to-many>和<many-to-many>处理方式不相同。

对于<ont-to-many>，inverse 所处理的是对被关联表进行修改操作。

对于<many-to-many>，inverse 所处理的则是中间关联表

Cascade 不会区分这两种关系的差别，所做的操作都是针对被关联的对象。

总结：

<one-to-many>

<one-to-many>中，建议 inverse=”true”，由“many”方来进行关联关系的维护

<many-to-many>中，只设置其中一方 inverse=”false”，或双方都不设置

Cascade，通常情况下都不会使用。特别是删除，一定要慎重。

13. hibernate 拒绝连接、服务器崩溃的原因？最少写 5 个

1. db 没有打开
2. 网络连接可能出了问题
3. 连接配置错了
4. 驱动的 driver，url 是否都写对了
5. LIB 下加入相应驱动，数据连接代码是否有误
6. 数据库配置可能有问题
7. 当前联接太多了，服务器都有访问人数限制的
8. 服务器的相应端口没有开，即它不提供相应的服务

14. Hibernate 主键生成策略有哪些

Assigned

Assigned 方式由程序生成主键值，并且要在 save()之前指定否则会抛出异常

特点：主键的生成值完全由用户决定，与底层数据库无关。用户需要维护主键值，在调用 session.save()之前要指定主键值。

Hilo

Hilo 使用高低位算法生成主键，高低位算法使用一个高位值和一个低位值，然后把算法得到的两个值拼接起来作为数据库中的唯一主键。Hilo 方式需要额外的数据库表和字段提供高位值来源。默认请况下使用的表是

hibernate_unique_key，默认字段叫作 next_hi。next_hi 必须有一条记录否则会出现错误。

特点：需要额外的数据库表的支持，能保证同一个数据库中主键的唯一性，但不能保证多个数据库

之间主键的唯一性。Hilo 主键生成方式由 Hibernate 维护，所以 Hilo 方式与底层数据库无关，但不应该手动修改 hi/lo 算法使用的表的值，否则会引起主键重复的异常。

Increment

Increment 方式对主键值采取自动增长的方式生成新的主键值,但要求底层数据库的支持 Sequence。如 Oracle，DB2 等。需要在映射文件 xxx.hbm.xml 中加入 Increment 标志符的设置。

特点：由 Hibernate 本身维护，适用于所有的数据库，不适合多进程并发更新数据库，适合单一进程访问数据库。不能用于群集环境。

Identity

Identity 当时根据底层数据库，来支持自动增长，不同的数据库用不同的主键增长方式。

特点：与底层数据库有关，要求数据库支持 Identity，如 MySQl 中是 auto_increment, SQL Server 中是 Identity，支持的数据库有 MySql、SQL Server、DB2、Sybase 和 HypersonicSQL。 Identity 无需 Hibernate 和用户的干涉，使用较为方便，但不便于在不同的数据库之间移植程序。

Sequence

Sequence 需要底层数据库支持 Sequence 方式，例如 Oracle 数据库等

特点：需要底层数据库的支持序列，支持序列的数据库有 DB2、PostgreSql、Qracle、SAPDb 等在不同数据库之间移植程序，特别从支持序列的数据库移植到不支持序列的数据库需要修改配置文件

Native

Native 主键生成方式会根据不同的底层数据库自动选择 Identity、Sequence、Hilo 主键生成方式

特点：根据不同的底层数据库采用不同的主键生成方式。由于 Hibernate 会根据底层数据库采用不同的映射方式，因此便于程序移植，项目中如果用到多个数据库时，可以使用这种方式。

UUID

UUID 使用 128 位 UUID 算法生成主键，能够保证网络环境下的主键唯一性，也就能够保证在不同数据库及不同服务器下主键的唯一性。

特点:能够保证数据库中的主键唯一性，生成的主键占用比较多的存贮空间

Foreign GUID

Foreign 用于一对一关系中。GUID 主键生成方式使用了一种特殊算法，保证生成主键的唯一性，支持 SQL Server 和 MySQL

15. 缓存管理

问题较多分为多个问题

Hibernate 中提供了两级 Cache,第一级别的缓存是 Session 级别的缓存,它是属于事务范围的缓存。这一级别的缓存由 hibernate 管理的，一般情况下无需进行干预；第二级别的缓存是 SessionFactory 级别的缓存，它是属于进程范围或群集范围的缓存。这一级别的缓存可以进行配置和更改，并且可以动态加载和卸载。 Hibernate 还为查询结果提供了一个查询缓存，它依赖于第二级缓存。

1. **一级缓存和二级缓存的比较**:第一级缓存 第二级缓存 存放数据的形式 相互关联的持久化对象 对象的散装数据 缓存的范围 事务范围，每个事务都有单独的第一级缓存进程范围或集群范围，缓存被同一个进程或集群范围内的所有事务共享 并发访问策略由于每个事务都拥有单独的第一级缓存,不会出现并发问

题，无需提供并发访问策略由于多个事务会同时访问第二级缓存中相同数据，因此必须提供适当的并发访问策略，来保证特定的事务隔离级别 数据过期策略没有提供数据过期策略。处于一级缓存中的对象永远不会过期，除非应用程序显式清空缓存或者清除特定的对象必须提供数据过期策略，如基于内存的缓存中的对象的最大数目，允许对象处于缓存中的最长时间，以及允许对象处于缓存中的最长空闲时间 物理存储介质内存内存和硬盘。对象的散装数据首先存放在基于内在的缓存中，当内存中对象的数目达到数据过期策略中指定上限时，就会把其余的对象写入基于硬盘的缓存中。缓存的软件实现 在 Hibernate 的 Session 的实现中包含了缓存的实现由第三方提供，Hibernate 仅提供了缓存适配器(CacheProvider)。用于把特定的缓存插件集成到 Hibernate 中。启用缓存的方式只要应用程序通过 Session 接口来执行保存、更新、删除、加载和查询数据库数据的操作，Hibernate 就会启用第一级缓存，把数据库中的数据以对象的形式拷贝到缓存中，对于批量更新和批量删除操作，如果不希望启用第一级缓存，可以绕过 Hibernate API，直接通过 JDBC API 来执行指操作。用户可以在单个类或类的单个集合的粒度上配置第二级缓存。如果类的实例被经常读但很少被修改，就可以考虑使用第二级缓存。只有为某个类或集合配置了第二级缓存，Hibernate 在运行时才会把它的实例加入到第二级缓存中。 用户管理缓存的方式第一级缓存的物理介质为内存，由于内存容量有限，必须通过恰当的检索策略和检索方式来限制加载对象的数目。Session 的 evit()方法可以显式清空缓存中特定对象，但这种方法不值得推荐。 第二级缓存的物理介质可以是内存和硬盘，因此第二级缓存可以存放大量的数据，数据过期策略的 maxElementsInMemory 属性值可以控制内存中的对象数目。管理第二级缓存主要包括两个方面：选择需要使用第二级缓存的持久类，设置合适的并发访问策略：选择缓存适配器，设置合适的数据过期策略。

2. 一级缓存的管理： 当应用程序调用 Session 的 save()、update()、savaeOrUpdate()、get()或 load()，以及调用查询接口的 list()、iterate()或 filter()方法时,如果在 Session 缓存中还不存在相应的对象，Hibernate 就会把该对象加入到第一级缓存中。当清理缓存时，Hibernate 会根据缓存中对象的状态变化来同步更新数据库。 Session 为应用程序提供了两个管理缓存的方法： evict(Object obj)：从缓存中清除参数指定的持久化对象。 clear()：清空缓存中所有持久化对象。

3. 二级缓存的管理：

3.1. Hibernate 的二级缓存策略的一般过程如下：

1） 条件查询的时候，总是发出一条 select * from table_name where …. （选择所有字段）这样的 SQL 语句查询数据库，一次获得所有的数据对象。

2） 把获得的所有数据对象根据 ID 放入到第二级缓存中。

3） 当 Hibernate 根据 ID 访问数据对象的时候，首先从 Session 一级缓存中查；查不到，如果配置了二级缓存，那么从二级缓存中查；查不到，再查询数据库，把结果按照 ID 放入到缓存。

4） 删除、更新、增加数据的时候，同时更新缓存。

Hibernate 的二级缓存策略，是针对于 ID 查询的缓存策略，对于条件查询则毫无作用。为此，Hibernate 提供了针对条件查询的 Query Cache。

3.2. 什么样的数据适合存放到第二级缓存中？ 1 很少被修改的数据 2 不是很重要的数据，允许出现偶尔并发的数据 3 不会被并发访问的数据 4 参考数据,指的是供应用参考的常量数据，它的实例数目有限，它的实例会被许多其他类的实例引用，实例极少或者从来不会被修改。

3.3. 不适合存放到第二级缓存的数据？ 1 经常被修改的数据 2 财务数据,绝对不允许出现并发 3 与其他应用共享的数据。

3.4. 常用的缓存插件 Hibernater 的二级缓存是一个插件，下面是几种常用的缓存插件：

EhCache：可作为进程范围的缓存，存放数据的物理介质可以是内存或硬盘，对 Hibernate 的查询缓存提供了支持。

OSCache：可作为进程范围的缓存，存放数据的物理介质可以是内存或硬盘，提供了丰富的缓存数据过期策略，对 Hibernate 的查询缓存提供了支持。

SwarmCache：可作为群集范围内的缓存，但不支持 Hibernate 的查询缓存。

JBossCache：可作为群集范围内的缓存，支持事务型并发访问策略，对 Hibernate 的查询缓存提供了支持。

3.5. 配置二级缓存的主要步骤：

1） 选择需要使用二级缓存的持久化类，设置它的命名缓存的并发访问策略。这是最值得认真考虑的步骤。

2） 选择合适的缓存插件，然后编辑该插件的配置文件。

16. 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

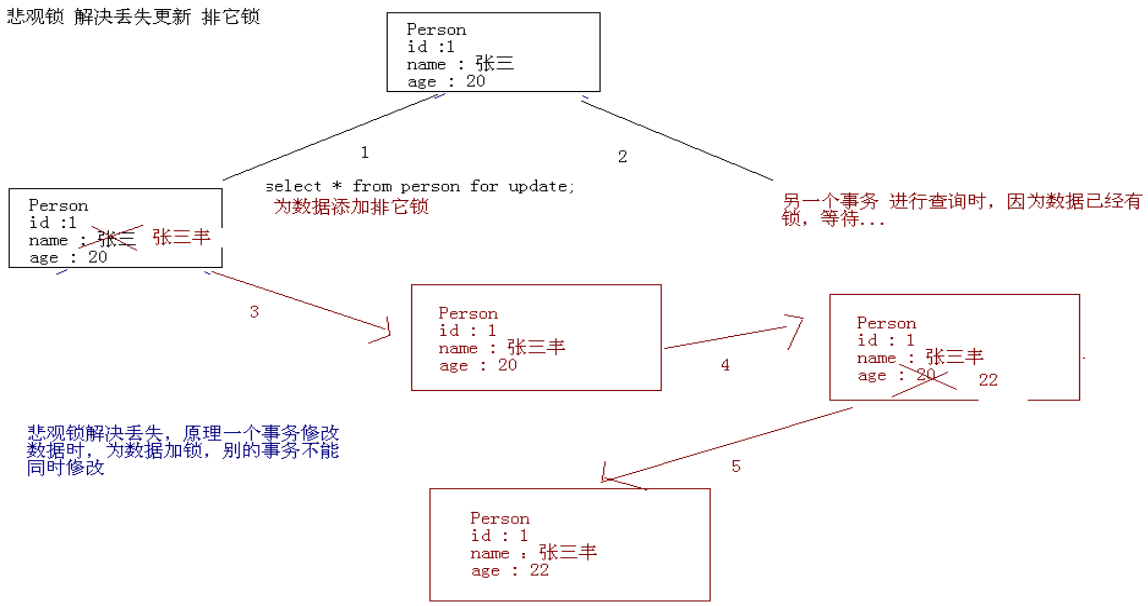
17、Hibernate 解决丢失更新 悲观锁 和乐观锁

悲观锁 和 乐观锁

悲观锁： 采用数据库内部锁机制，在一个事务操作数据时，为数据加锁，另一个事务无法操作

* 排它锁 （写锁），数据库中每张表只能添加一个排它锁，排它锁与其他锁互斥

* 在修改数据时，自动添加排它锁
* 在查询数据时 添加排它锁 select * from customers for update;



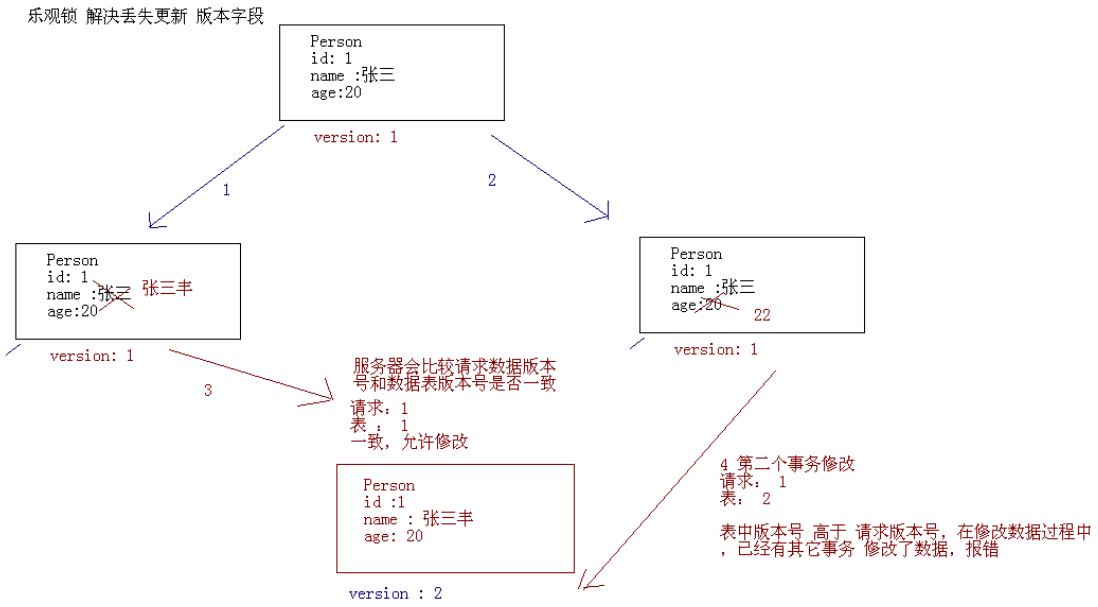
hibernate 中使用悲观锁

Customer customer = (Customer) session.load(Customer.class, 1,LockMode.UPGRADE); // MySQL

Customer customer = (Customer) session.load(Customer.class, 1,LockMode.*UPGRADE NOWAIT*);//oracle

* 悲观锁解决丢失更新，效率问题 ， 数据不能同时修改

乐观锁 ： 与数据库锁无关，在数据表中为数据添加 版本字段，每次数据修改都会导致版本号+1



hibernate 为 Customer 表 添加版本字段

1) 在 customer 类 添加 private Integer version; 版本字段
2) 在 Customer.hbm.xml 定义版本字段

```
<!-- 定义版本字段 -->
<!-- name 是属性名 -->
<version name="version"></version>
```

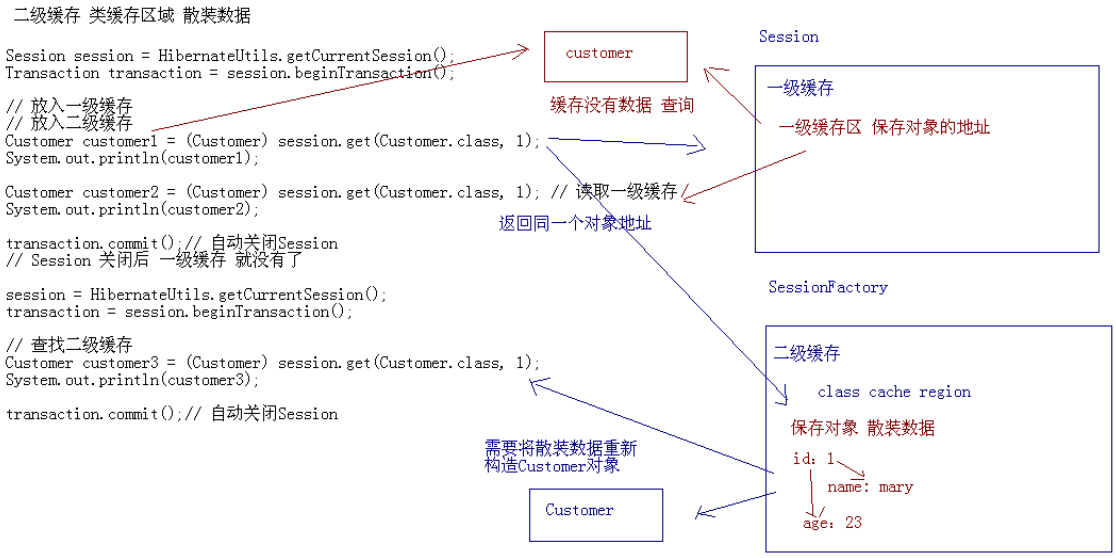
18Hibernate 二级缓存的内部结构

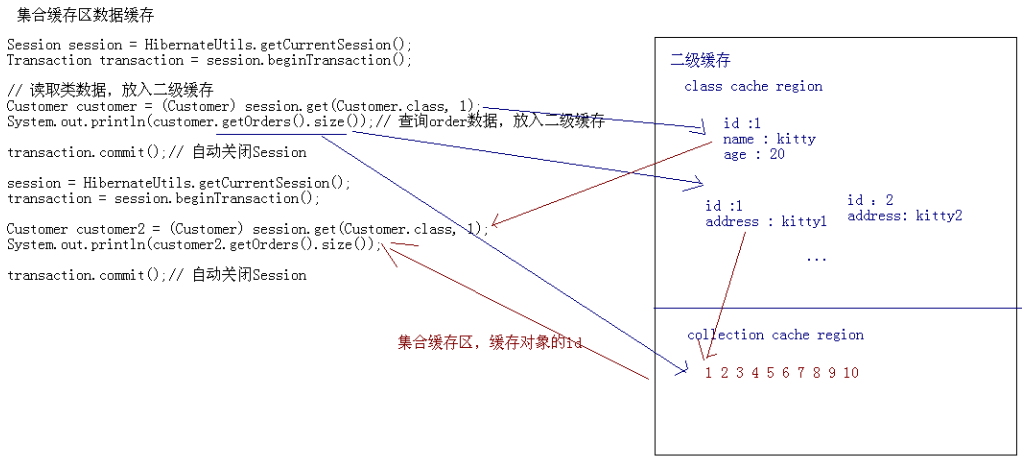
* 类缓存区域
* 集合缓存区域
* 更新时间戳区域
* 查询缓存区域

* 从二级缓存区 返回 数据每次 地址都是不同的 （散装数据 ）

每次查询二级缓存，都是将散装数据 构造为一个新的对象

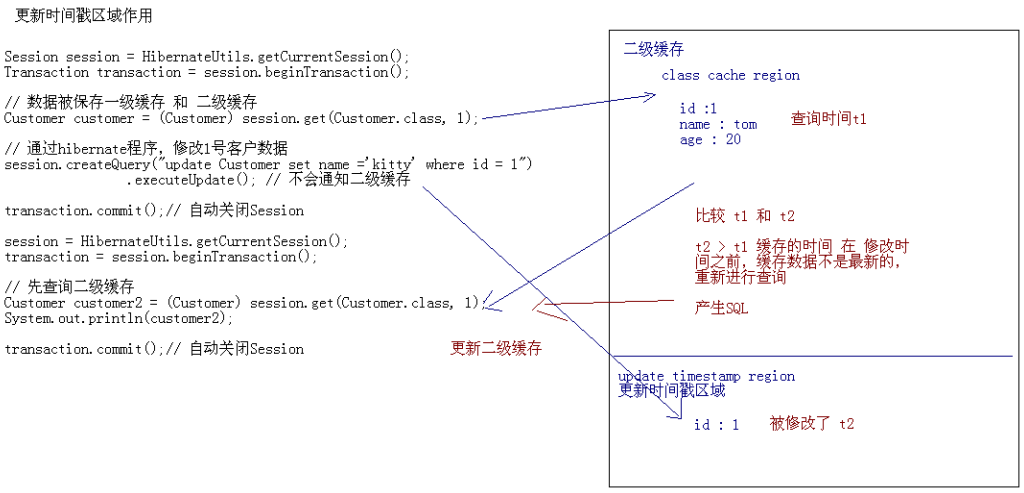
hibernate3_day4 图五 二级缓存类缓冲区特点





作用：记录数据最后更新时间，确保缓存数据是有效的

更新时间戳其余，记录数据最后更新时间，在使用二级缓存时，比较缓存时间 t1 与 更新时间 t2 ， 如果 t2 > t1 丢弃原来缓存数据，重新查询数据库



有人称查询缓存 为 hibernate 第三级缓存

* 二级缓存 缓存数据 都是类对象数据 ，数据都是缓存在 "类缓存区域"
二级缓存缓存 PO 类对象，条件(key)是 id

Spring

1. Spring 的优点

* 方便解耦，简化开发

Spring 就是一个大工厂，可以将所有对象创建和依赖关系维护，交给 Spring 管理

* AOP 编程的支持

Spring 提供面向切面编程，可以方便的实现对程序进行权限拦截、运行监控等功能

* 声明式事务的支持

只需要通过配置就可以完成对事务的管理，而无需手动编程

* 方便程序的测试

Spring 对 Junit4 支持，可以通过注解方便的测试 Spring 程序

* 方便集成各种优秀框架

Spring 不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如：Struts、Hibernate、MyBatis、Quartz 等）的直接支持

* 降低 JavaEE API 的使用难度

Spring 对 JavaEE 开发中非常难用的一些 API（JDBC、JavaMail、远程调用等），都提供了封装，使这些 API 应用难度大大降低

2. 简单描述 IOC 和 DI

一个类需要用到某个接口的方法，我们需要将类 A 和接口 B 的实现关联起来，最简单的方法是类 A 中创建一个对于接口 B 的实现 C 的实例，但这种方法显然两者的依赖（Dependency）太大了。而 IoC 的方法是只在类 A 中定义好用于关联接口 B 的实现的方法，将类 A，接口 B 和接口 B 的实现 C 放入 IoC 的 容器（Container）中，通过一定的配置由容器（Container）来实现类 A 与接口 B 的实现 C 的关联。

IoC Inverse of Control 反转控制的概念，就是将原本在程序中手动创建 HelloService 对象的控制权，交由 Spring 框架管理，简单说，就是创建 HelloService 对象控制权被反转到了 Spring 框架

DI：Dependency Injection 依赖注入，在 Spring 框架负责创建 Bean 对象时，动态的将依赖对象注入到 Bean 组件

3. spring 中的如何获取 bean

Spring 提供配置 Bean 三种实例化方式

1）使用类构造器实例化(默认无参数)

```
<bean id="bean1" class="cn.hzgg.spring.b_instance.Bean1"></bean>
```

2）使用静态工厂方法实例化(简单工厂模式)

```
<bean id="bean2" class="cn.hzgg.spring.b_instance.Bean2Factory" factory-method="getBean2"></bean>
```

3）使用实例工厂方法实例化(工厂方法模式)

```
<bean id="bean3Factory" class="cn.hzgg.spring.b_instance.Bean3Factory"></bean>
<bean id="bean3" factory-bean="bean3Factory" factory-method="getBean3"></bean>
```

4. spring 是什么？根据你的理解详细谈谈你的见解。

◆目的：解决企业应用开发的复杂性

◆功能：使用基本的 JavaBean 代替 EJB，并提供了更多的企业应用功能

◆范围：任何 Java 应用

简单来说，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。

所有 Spring 的这些特征使你能够编写更干净、更可管理、并且更易于测试的代码。它们也为 Spring 中的各种模块提供了基础支持。

5. 项目中如何体现 Spring 中的切面编程，举例说明。

面向切面编程：主要是横切一个关注点,将一个关注点模块化成一个切面。在切面上声明一个通知(Advice)和切入点(Pointcut)；通知：是指在切面的某个特定的连接点(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。)上执行的动作。通知中定义了要插入的方法。切入点：切入点的内容是一个表达式，以描述需要在哪些对象的哪些方法上插入通知中定义的方法。

项目中用到的 Spring 中的切面编程最多的地方：声明式事务管理。

a、定义一个事务管理器

b、配置事务特性(相当于声明通知。一般在业务层的类的一些方法上定义事务)

c、配置哪些类的哪些方法需要配置事务(相当于切入点。一般是业务类的方法上)

6. spring 中有几种方式完成依赖注入

第一种 构造器注入

通过 <constructor-arg> 元素完成注入

```
<bean id="car" class="cn.hzgg.spring.e_di.Car">
<!-- 通过构造器参数，完成属性注入 -->
<constructor-arg index="0" type="java.lang.String" value="保时捷"></constructor-arg>
 <!-- 第一个参数 String 类型参数 -->
<constructor-arg index="1" type="double" value="1000000"></constructor-arg>
    </bean>
```

第二种 setter 方法注入

```
<bean id="car2" class="cn.hzgg.spring.e_di.Car2">
```

```
        <!-- 通过 property 元素完成属性注入 -->
    <property name="name" value="宝马"></property>
    <property name="price" value="500000"></property>
</bean>
```

7. Spring Bean 的作用域之间有什么区别？

singleton：这种 bean 范围是默认的，这种范围确保不管接受到多少个请求，每个容器中只有一个 bean 的实例，单例的模式由 bean factory 自身来维护。

prototype：原形范围与单例范围相反，为每一个 bean 请求提供一个实例。

request：在请求 bean 范围内会每一个来自客户端的网络请求创建一个实例，在请求完成以后，bean 会失效并被垃圾回收器回收。

Session：与请求范围类似，确保每个 session 中有一个 bean 的实例，在 session 过期后，bean 会随之失效。

global-session：global-session 和 Portlet 应用相关。当你的应用部署在 Portlet 容器中工作时，它包含很多 portlet。如果你想要声明让所有的 portlet 共用全局的存储变量的话，那么这全局变量需要存储在 global-session 中。

全局作用域与 Servlet 中的 session 作用域效果相同

8. Spring 框架中的单例 Beans 是线程安全的么？

Spring 框架并没有对单例 bean 进行任何多线程的封装处理。关于单例 bean 的线程安全和并发问题需要开发者自行去搞定。但实际上，大部分的 Spring bean 并没有可变的状态(比如 Serview 类和 DAO 类)，所以在某种程度上说 Spring 的单例 bean 是线程安全的。如果你的 bean 有多种状态的话（比如 View Model 对象），就需要自行保证线程安全。

最浅显的解决办法就是将多态 bean 的作用域由“singleton”变更为“prototype”。

9. spring 在项目中如何充当粘合剂

1、在项目中利用 spring 的 IOC(控制反转或依赖注入),明确地定义组件接口(如 UserDAO)，开发者可以独立开发各个组件，然后根据组件间的依赖关系组装(UserAction 依赖于 UserBiz,UserBiz 依赖于 UserDAO)运行，很好的把 Struts(Action)和 hibernate(DAO 的实现)结合起来了。

2、 spring 的事务管理把 hibernate 对数据库的操作进行了事务配置。

10. Spring 的事务如何配置

spring 的声明式事务配置：

```
1. <!-- 配置 sessionFactory -->
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
```

```
        <property name="configLocation">
            <value>/WEB-INF/classes/hibernate.cfg.xml</value>
        </property>
    </bean>
```

2. 配置事务管理器

```
    <!-- 配置事务管理器 -->
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory">
            <ref local="sessionFactory" />
        </property>
    </bean>
```

3. 配置事务特性

```
    <tx:advice id="txAdvice"  transaction-manager="transactionManager">
        <tx:attributes>
                    <tx:method name="add*" propagation="REQUIRED"/>
                     <tx:method name="update*" propagation="REQUIRED"/>
                     <tx:method name="del*" propagation="REQUIRED"/>
                     <tx:method name="*" read-only="true"/>
                </tx:attributes>
    </tx:advice>
```

4. 配置哪些类的哪些方法配置事务

```
    <aop:config>
        <aop:pointcut id="allManagerMethod" ession="execution(* com.yyaccp.servi
ce.impl.*.*(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="allManagerMethod">
    </aop:config>
```

isolation 设定事务的隔离级别，事务管理器根据它来控制另外一个事务可以看到本事务内的哪些数据。

定义的 4 个不同的事务隔离级别：

DEFAULT：默认的隔离级别，使用数据库默认的事务隔离级别

READ_COMMITTED：保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据。这种事务隔离级别可以避免脏读出现，但是可能会出现不可重复读和幻像读。

READ_UNCOMMITTED：这是事务最低的隔离级别，它充许别外一个事务可以看到这个事务未提交的数据。这种隔离级别会产生脏读，不可重复读和幻像读。

REPEATABLE_READ：这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。它除了

保证一个事务不能读取另一个事务未提交的数据外，还保证了避免不可重复读。

SERIALIZABLE：这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。除了防止脏读，不可重复读外，还避免了幻像读。

propagation 定义了 7 个事务传播行为

REQUIRED： 如果存在一个事务，则支持当前事务。如果没有事务则开启一个新的事务。

SUPPORTS： 如果存在一个事务，支持当前事务。如果没有事务，则非事务的执行。但是对于事务同步的事务管理器，SUPPORTS 与不使用事务有少许不同。

REQUIRES_NEW 总是开启一个新的事务。如果一个事务已经存在，则将这个存在的事务挂起。

NOT_SUPPORTED 总是非事务地执行，并挂起任何存在的事务。

NEVER 总是非事务地执行，如果存在一个活动事务，则抛出异常

NESTED：如果一个活动的事务存在，则运行在一个嵌套的事务中. 如果没有活动事务, 则按 TransactionDefinition.PROPAGATION_REQUIRED 属性执行。

嵌套事务一个非常重要的概念就是内层事务依赖于外层事务。外层事务失败时，会回滚内层事务所做的动作。而内层事务操作失败并不会引起外层事务的回滚。

REQUIRED 应该是我们首先的事务传播行为。它能够满足我们大多数的事务需求。

11. transaction 有那几种实现(事务处理)(Spring)

在 Spring 中，事务处理主要有两种方式

(1) 代码控制事务

在程序中引入新的模版类，这个类封装了事务管理的功能

(2) 参数配置控制事务,在 Application-Context.xml 增加一个事务代理（UserDAOProxy）配置

12. Spring 中 advisor 和 aspect 区别 ?

advisor 是 spring 中 aop 定义切面，通常由一个切点和一个通知组成

aspect 是规范中切面 ， 允许由多个切点和 多个通知组成

13. Spring 框架中 bean 的生命周期

- Spring 容器读取 XML 文件中 bean 的定义并实例化 bean。
- Spring 根据 bean 的定义设置属性值。
- 如果该 Bean 实现了 BeanNameAware 接口，Spring 将 bean 的 id 传递给 setBeanName()方法。
- 如果该 Bean 实现了 BeanFactoryAware 接口，Spring 将 beanfactory 传递给 setBeanFactory()方法。
- 如果任何 bean BeanPostProcessors 和该 bean 相关，Spring 调用 postProcessBeforeInitialization()方法。
- 如果该 Bean 实现了 InitializingBean 接口，调用 Bean 中的 afterPropertiesSet 方法。如果 bean 有初始化函数声明，调用相应的初始化方法。

- 如果任何 bean BeanPostProcessors 和该 bean 相关，调用 postProcessAfterInitialization()方法。
- 如果该 bean 实现了 DisposableBean，调用 destroy()方法。

14. AOP 相关术语

- **1）解释 AOP**

  AOP Aspect Oriented Programing 面向切面编程，AOP 采取横向抽取机制，取代了传统纵向继承体系重复性代码,AOP 允许程序员模块化横向业务逻辑，或定义核心部分的功能，例如性能监视、事务管理、安全检查、缓存。

- **2)切面(Aspect)**

  AOP 的核心就是切面,它将多个类的通用行为封装为可重用的模块。该模块含有一组 API 提供 cross-cutting 功能。例如,日志模块称为日志的 AOP 切面。根据需求的不同,一个应用程序可以有若干切面。在 Spring AOP 中，切面通过带有@Aspect 注解的类实现。

- **3)连接点(Join point)**

  连接点代表应用程序中插入 AOP 切面的地点。它实际上是 Spring AOP 框架在应用程序中执行动作的地点。所谓连接点是指那些被拦截到的点。在 spring 中,这些点指的是方法,因为 spring 只支持方法类型的连接点.

- **5).通知(Advice)**

  通知表示在方法执行前后需要执行的动作。指拦截到 Joinpoint 之后所要做的事情,实际上它是 Spring AOP 框架在程序执行过程中触发的一些代码。

  Spring 切面可以执行一下五种类型的通知:

  before(前置通知)：在一个方法之前执行的通知。

  after(最终通知)：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）。

  after-returning(后置通知)：在某连接点正常完成后执行的通知。

  after-throwing(异常通知)：在方法抛出异常退出时执行的通知。

  around(环绕通知)：在方法调用前后触发的通知。

- **6)切入点(Pointcut)**

  切入点是一个或一组连接点，通知将在这些位置执行。可以通过表达式或匹配的方式指明切入点。

  所谓切入点是指我们要对哪些 Joinpoint 进行拦截的定义

- **7)什么是代理?**

  代理是将通知应用到目标对象后创建的对象。从客户端的角度看，代理对象和目标对象是一样的。

- **8)有几种不同类型的自动代理?**

  BeanNameAutoProxyCreator：bean 名称自动代理创建器

  DefaultAdvisorAutoProxyCreator：默认通知者自动代理创建器

  Metadata autoproxying：元数据自动代理

15. Spring 如何处理线程并发问题？

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。

16. 什么是 Spring 的 MVC 框架？

Spring 提供了一个功能齐全的 MVC 框架用于构建 Web 应用程序。Spring 框架可以很容易的和其他的 MVC 框架融合(如 Struts)，该框架使用控制反转(IOC)将控制器逻辑和业务对象分离开来。它也允许以声明的方式绑定请求参数到业务对象上。

17. SpringMVC 架构流程

1、 用户发起请求到前端控制器 DispatcherServlet

2、 DispatcherServlet 接收到请求后调用 HandlerMapping 处理器映射器

3、处理器映射器会根据 URL 找到具体的处理器，然后生成处理器对象以及处理器拦截器把这些对象一并返回到 DispatcherServlet

4、 DispatcherServlet 通过 HandlerAdpater 处理器适配器调用处理器

5、 执行处理器（Controller）也叫后端控制器

6、 Controller 执行完毕后返回 ModelAndView

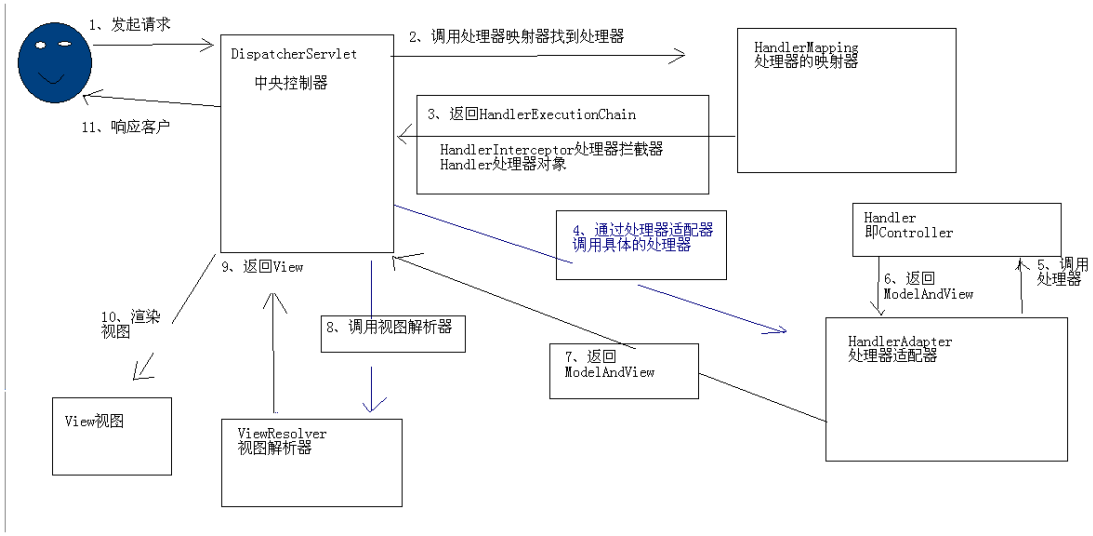
7、 HanlderAdpater 将 Controller 执行的结果 ModleAndView 返回到 DispatcherServlet

8、 DispatcherServlet 将 ModelAndView 传递给视图解析器(ViewReslover)

9、 ViewReslover 解析后返回具体的 View

10、DispatcherServlet 对 view 进行渲染视图(将数据填充到视图中)

11、DispatcherServlet 响应客户



18. 如何解决 POST 请求中文乱码问题，GET 的又如何处理呢？

在 web.xml 中加入：

```
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

以上可以解决 post 请求乱码问题。对于 get 请求中文参数出现乱码解决方法有两个：

修改 tomcat 配置文件添加编码与工程编码一致，如下：

```
<ConnectorURIEncoding="utf-8"  connectionTimeout="20000"  port="8080"  protocol="HTTP/1.1"
redirectPort="8443"/>
```

另外一种方法对参数进行重新编码：

```
String userName = new String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")
```

19. SpringMVC 与 Struts2 的主要区别？

①springmvc 的入口是一个 servlet 即前端控制器，而 struts2 入口是一个 filter 过虑器。

②springmvc 是基于方法开发，传递参数是通过方法形参，可以设计为单例或多例(建议单例)，struts2 是基于类开发，传递参数是通过类的属性，只能设计为多例。

③Struts 采用值栈存储请求和响应的数据，通过 OGNL 存取数据， springmvc 通过参数解析器是将 request 对象内容进行解析成方法形参，将响应数据和页面封装成 ModelAndView 对象，最后又将模型数据通过 request 对象传输到页面。 Jsp 视图解析器默认使用 jstl。

20.SpringAOP 的底层实现原理是什么？ 你知道哪些动态代理方式?这两种动态代理方式有什么区别？

AOP 面向切面编程 底层原理 代理！！！

JDK 动态代理原理， 为目标对象 接口生成代理对象 ，对于不使用接口的业务类，无法使用 JDK 动态代理

CGLIB(Code Generation Library)是一个开源项目！

是一个强大的,高性能,高质量的 Code 生成类库,它可以在运行期扩展 Java 类与实现 Java 接口。Hibernate 支持 CGlib 来实现 PO 字节码的动态生成。

CGlib 采用非常底层字节码技术，可以为一个类创建子类，解决无接口代理问题

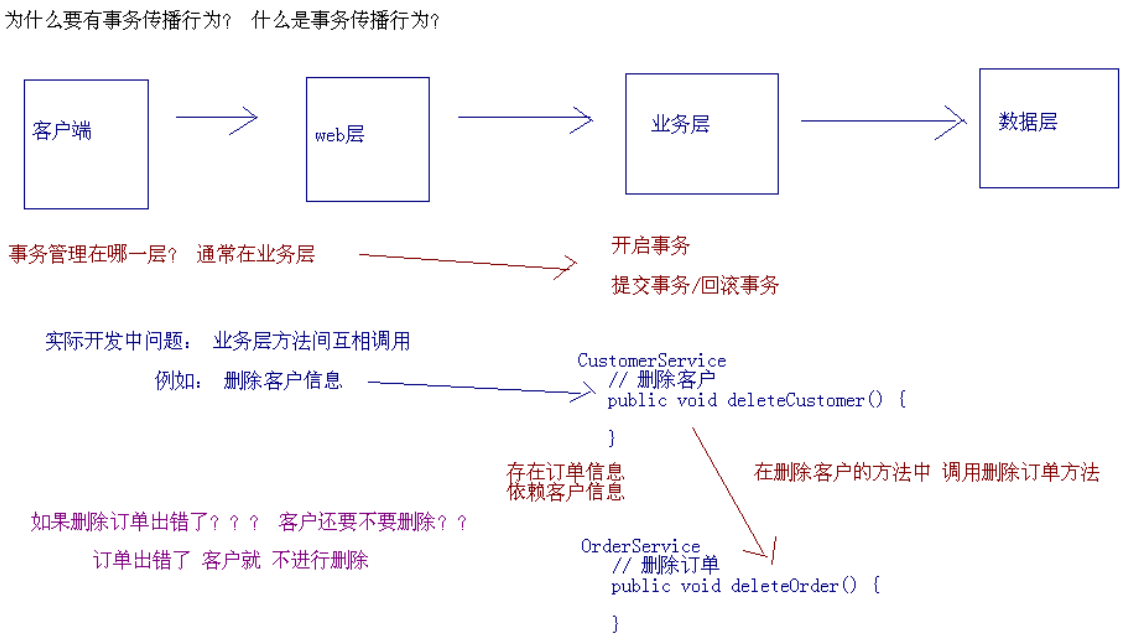
1）.若目标对象实现了若干接口，spring 使用 JDK 的 java.lang.reflect.Proxy 类代理。

2）.若目标对象没有实现任何接口，spring 使用 CGLIB 库生成目标对象的子类

程序中应优先对接口创建代理，便于程序解耦维护

21、什么是传播行为为什么要有传播行为

* 不是 JDBC 规范定义
* 传播行为 针对实际开发中问题



传播行为解决问题： 一个业务层事务 调用 另一个业务层事务，事务之间关系如何处理

七种传播行为

propagation _required 支持当前事务，如果不存在 就新建一个

* 删除客户 删除订单， 处于同一个事务，如果 删除订单失败，删除客户也要回滚

PROPAGATION_SUPPORTS 支持当前事务，如果不存在，就不使用事务

PROPAGATION_ mandatory 支持当前事务，如果不存在，抛出异常

PROPAGATION_REQUIRES_NEW 如果有事务存在，挂起当前事务，创建一个新的事务

* 生成订单， 发送通知邮件， 通知邮件会创建一个新的事务，如果邮件失败， 不影响订单生成

PROPAGATION_NOT_SUPPORTED 以非事务方式运行，如果有事务存在，挂起当前事务

PROPAGATION_NEVER 以非事务方式运行，如果有事务存在，抛出异常

PROPAGATION_NESTED 如果当前事务存在，则嵌套事务执行

* 依赖于 JDBC3.0 提供 SavePoint 技术

* 删除客户 删除订单，在删除客户后，设置 SavePoint，执行删除订单，删除订单和删除客户在同一个事务 ，删除订单失败， 事务回滚 SavePoint ，由用户控制是事务提交 还是 回滚

重点：

PROPAGATION_REQUIRED 一个事务， 要么都成功，要么都失败
PROPAGATION_REQUIRES_NEW 两个不同事务，彼此之间没有关系 一个事务失败了 不影响另一个事务
PROPAGATION_NESTED 一个事务， 在 A 事务 调用 B 过程中， B 失败了， 回滚事务到 之前 SavePoint ， 用户可以选择提交事务或者回滚事务

MyBatis

1. JDBC 编程有哪些不足之处，MyBatis 是如何解决这些问题的？

① 数据库链接创建释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。
解决：在 SqlMapConfig.xml 中配置数据链接池，使用连接池管理数据库链接。
② Sql 语句写在代码中造成代码不易维护，实际应用 sql 变化的可能较大，sql 变动需要改变 java 代码。
解决：将 Sql 语句配置在 XXXXmapper.xml 文件中与 java 代码分离。
③ 向 sql 语句传参数麻烦，因为 sql 语句的 where 条件不一定，可能多也可能少，占位符需要和参数一一对应。
解决： Mybatis 自动将 java 对象映射至 sql 语句。
④ 对结果集解析麻烦，sql 变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成 pojo 对象解析比较方便。
解决：Mybatis 自动将 sql 执行结果映射至 java 对象。

2. MyBatis 编程步骤是什么样的？

① 创建 SqlSessionFactory
② 通过 SqlSessionFactory 创建 SqlSession
③ 通过 sqlsession 执行数据库操作
④ 调用 session.commit()提交事务
⑤ 调用 session.close()关闭会话

3. MyBatis 与 Hibernate 有哪些不同？

Mybatis 和 hibernate 不同，它不完全是一个 ORM 框架，因为 MyBatis 需要程序员自己编写 Sql 语句，不过 mybatis 可以通过 XML 或注解方式灵活配置要运行的 sql 语句，并将 java 对象和 sql 语句映射生成最终执行的 sql，最后将 sql 执行的结果再映射生成 java 对象。

Mybatis 学习门槛低，简单易学，程序员直接编写原生态 sql，可严格控制 sql 执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。但是灵活的前提是 mybatis 无法做到数据库无关性，如果需要实现支持多种数据库的软件则需要自定义多套 sql 映射文件，工作量大。

Hibernate 对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件（例如需求固定的定制化软件）如果用 hibernate 开发可以节省很多代码，提高效率。但是 Hibernate 的缺点是学习门槛高，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，以及怎样用好 Hibernate 需要具有很强的经验和能力才行。

总之，按照用户的需求在有限的资源环境下只要能做出维护性、扩展性良好的软件架构都是好架构，所以框架只有适合才是最好。

4. 使用 MyBatis 的 mapper 接口调用时有哪些要求？

① Mapper 接口方法名和 mapper.xml 中定义的每个 sql 的 id 相同

② Mapper 接口方法的输入参数类型和 mapper.xml 中定义的每个 sql 的 parameterType 的类型相同

③ Mapper 接口方法的输出参数类型和 mapper.xml 中定义的每个 sql 的 resultType 的类型相同

④ Mapper.xml 文件中的 namespace 即是 mapper 接口的类路径。

5. SqlMapConfig.xml 中配置有哪些内容？

SqlMapConfig.xml 中配置的内容和顺序如下：

properties（属性）

settings（配置）

typeAliases（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境集合属性对象）

environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

mappers（映射器）

6. 简单的说一下 MyBatis 的一级缓存和二级缓存？

Mybatis 首先去缓存中查询结果集，如果没有则查询数据库，如果有则从缓存取出返回结果集就不走数据库。Mybatis 内部存储缓存使用一个 HashMap，key 为 hashCode+sqlId+Sql 语句。value 为从查询出来映射生成的 java 对象

Mybatis 的二级缓存即查询缓存，它的作用域是一个 mapper 的 namespace，即在同一个 namespace 中查询 sql 可以从缓存中获取数据。二级缓存是可以跨 SqlSession 的。

7. Mapper 编写有哪几种方式？

①接口实现类继承 SqlSessionDaoSupport

使用此种方法需要编写 mapper 接口，mapper 接口实现类、mapper.xml 文件

1、 在 sqlMapConfig.xml 中配置 mapper.xml 的位置

```
<mappers>
    <mapper resource="mapper.xml 文件的地址"  />
     <mapper resource="mapper.xml 文件的地址"  />
</mappers>
```

2、定义 mapper 接口

3、实现类集成 SqlSessionDaoSupport

mapper 方法中可以 this.getSqlSession()进行数据增删改查。

4、spring 配置

```
<bean id=" "  class="mapper 接口的实现">
      <property  name="sqlSessionFactory"  ref="sqlSessionFactory"></property>
</bean>
```

②使用 org.mybatis.spring.mapper.MapperFactoryBean

1、在 sqlMapConfig.xml 中配置 mapper.xml 的位置

如果 mapper.xml 和 mappre 接口的名称相同且在同一个目录，这里可以不用配置

```
<mappers>
     <mapper resource="mapper.xml 文件的地址"  />
   <mapper resource="mapper.xml 文件的地址"  />
</mappers>
```

2、定义 mapper 接口

注意

a)mapper.xml 中的 namespace 为 mapper 接口的地址

b)mapper 接口中的方法名和 mapper.xml 中的定义的 statement 的 id 保持一致

3、 Spring 中定义

```
<bean id=""  class="org.mybatis.spring.mapper.MapperFactoryBean">
      <property  name="mapperInterface"     value="mapper 接口地址"  />
    <property  name="sqlSessionFactory"  ref="sqlSessionFactory"  />
</bean>
```

③使用 mapper 扫描器

1、mapper.xml 文件编写，

注意：

mapper.xml 中的 namespace 为 mapper 接口的地址

mapper 接口中的方法名和 mapper.xml 中的定义的 statement 的 id 保持一致

如果将 mapper.xml 和 mapper 接口的名称保持一致则不用在 sqlMapConfig.xml 中进行配置

2、定义 mapper 接口

注意 mapper.xml 的文件名和 mapper 的接口名称保持一致，且放在同一个目录

3、配置 mapper 扫描器

```
1. <bean  class="org.mybatis.spring.mapper.MapperScannerConfigurer">
2.         <property  name="basePackage"  value="mapper 接口包地址"></property>
```

```
3.          <property  name="sqlSessionFactoryBeanName"  value="sqlSessionFactory"/>
4.  </bean>
```

4、使用扫描器后从 spring 容器中获取 mapper 的实现对象

扫描器将接口通过代理方法生成实现对象，要 spring 容器中自动注册，名称为 mapper 接口的名称。