Java私塾《深入浅出学Hibernate4》
——系列精品教程

配套视频和源码请到私塾在线下载.

http://sishuok.com

整体课程概览

- 系统的、循序渐进的学习Hibernate4的开发知识
- 第一部分：Hibernate入门
  包括：是什么、ORMapping原理、能干什么、有什么、体系结构、怎么做、HelloWorld
- 第二部分：Hibernate的基本配置
  包括：cfg配置、mapping配置、类型映射、组件映射、事务配置、缓存配置等
- 第三部分：Hibernate基本开发
  包括：持久类、Hibernate的对象状态、使用Hibernate实现增删改查、HQL语句、条件查询、Native SQL、批量处理、过滤器等
- 第四部分：关系映射
  包括：集合映射、关联关系映射、父子关系、复杂的映射等
- 第五部分：JavaEE的事务
  包括：事务产生的动机、事务的模型、分布式事务、声明性事务、事务传播性、编程性事务、事务的隔离级

整体课程概览

- 第六部分：Hibernate的事务和并发
- 第七部分：性能提升和二级缓存
  包括：抓取策略、集合的性能、二级缓存、查询缓存、管理缓存等
- 第八部分：基本实现原理
  包括：分增删改查的操作说明Hibernate基本的实现原理
- 第九部分：最佳实践
- 第十部分：零配置

本节课程概览

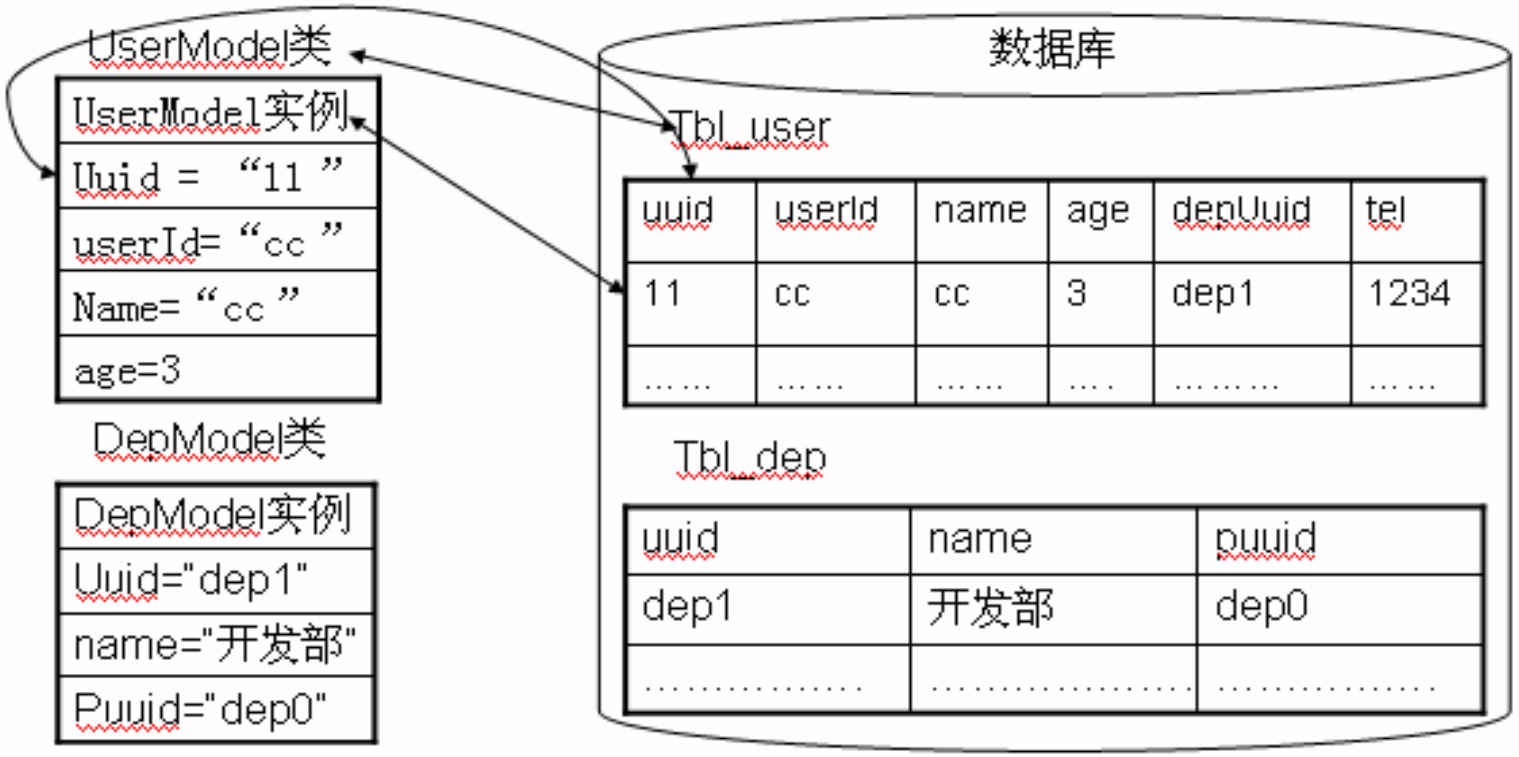
- Hibernate的基本概念
  Hibernate是什么

- ORMaping的基本理论

第一部分：Hibernate入门

Hibernate是什么-1

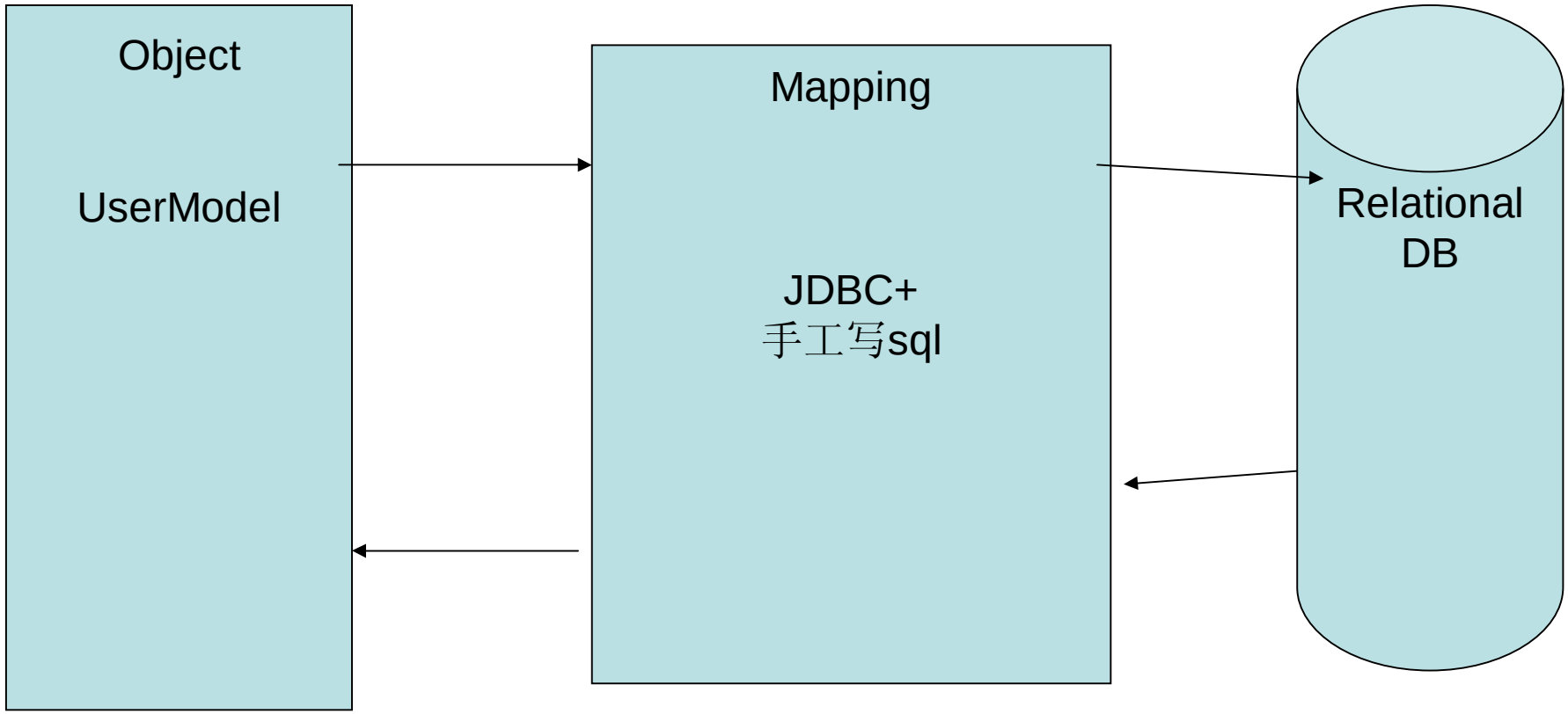
- Hibernate是什么
  Hibernate是一个轻量级的ORMapping框架
- ORMapping原理（Object Relational Mapping）



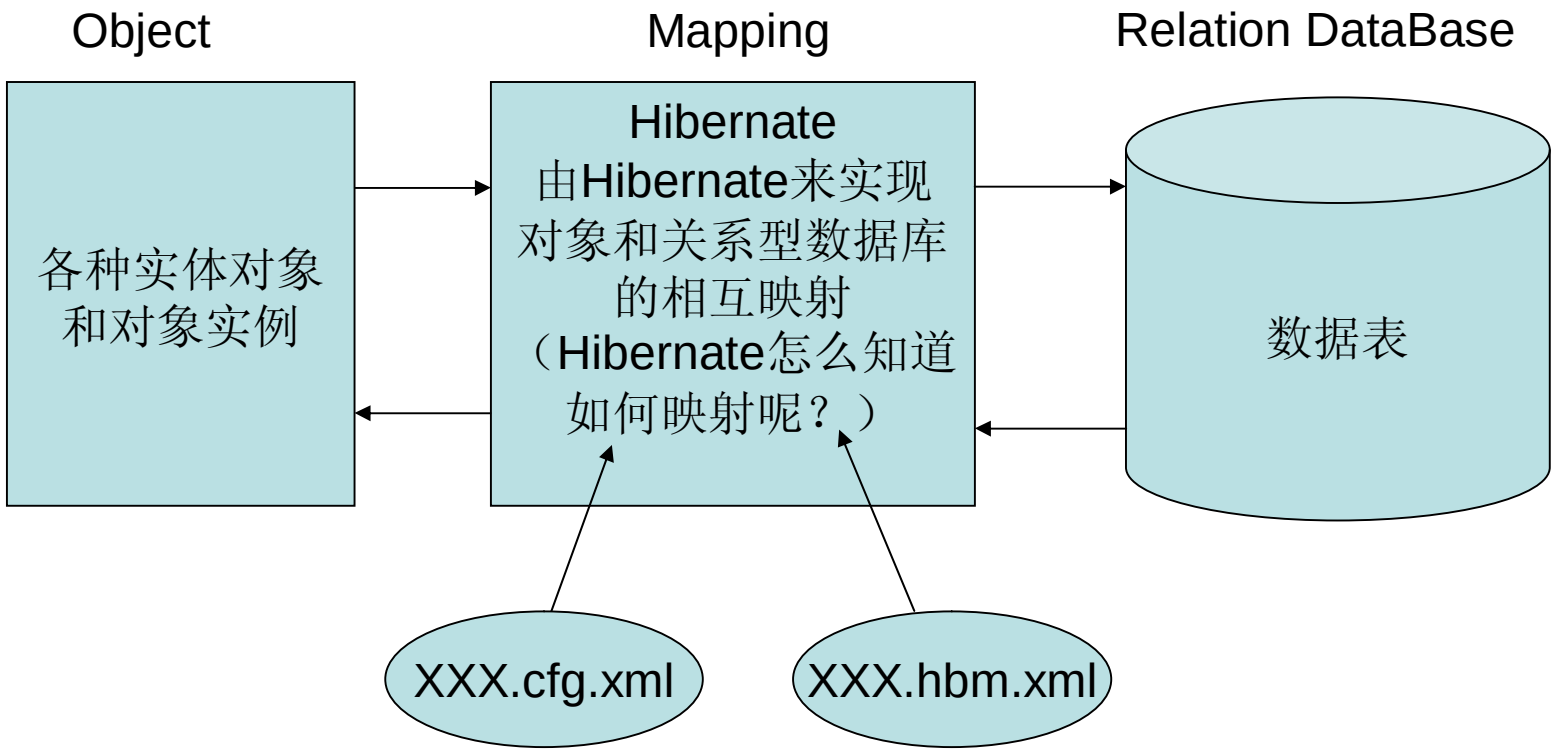
Hibernate是什么-2

- ORMapping基本对应规则：
  1：类跟表相对应
  2：类的属性跟表的字段相对应
  3：类的实例与表中具体的一条记录相对应
  4：一个类可以对应多个表，一个表也可以对应对个类
  5：DB中的表可以没有主键，但是Object中必须设置主键字段
  6：DB中表与表之间的关系（如：外键）映射成为Object之间的关系
  7：Object中属性的个数和名称可以和表中定义的字段个数和名称不一样

- ORMapping的基本实现方式：
  使用JDBC，用SQL来操作数据库，只是看动态生成还是人工写代码来实现。
  大家想想，我们实现过ORMapping吗？


Object
UserModel
Mapping
JDBC+
手工写sql
Relational
DB

Hibernate是什么-3



本节课程小结

- Hibernate是什么
- ORMapping的基本理论
- 作业：复习和掌握这些理论知识

本节课程概览

- Hibernate的基本概念
  Hibernate能干什么、Hibernate有什么

第一部分：Hibernate入门

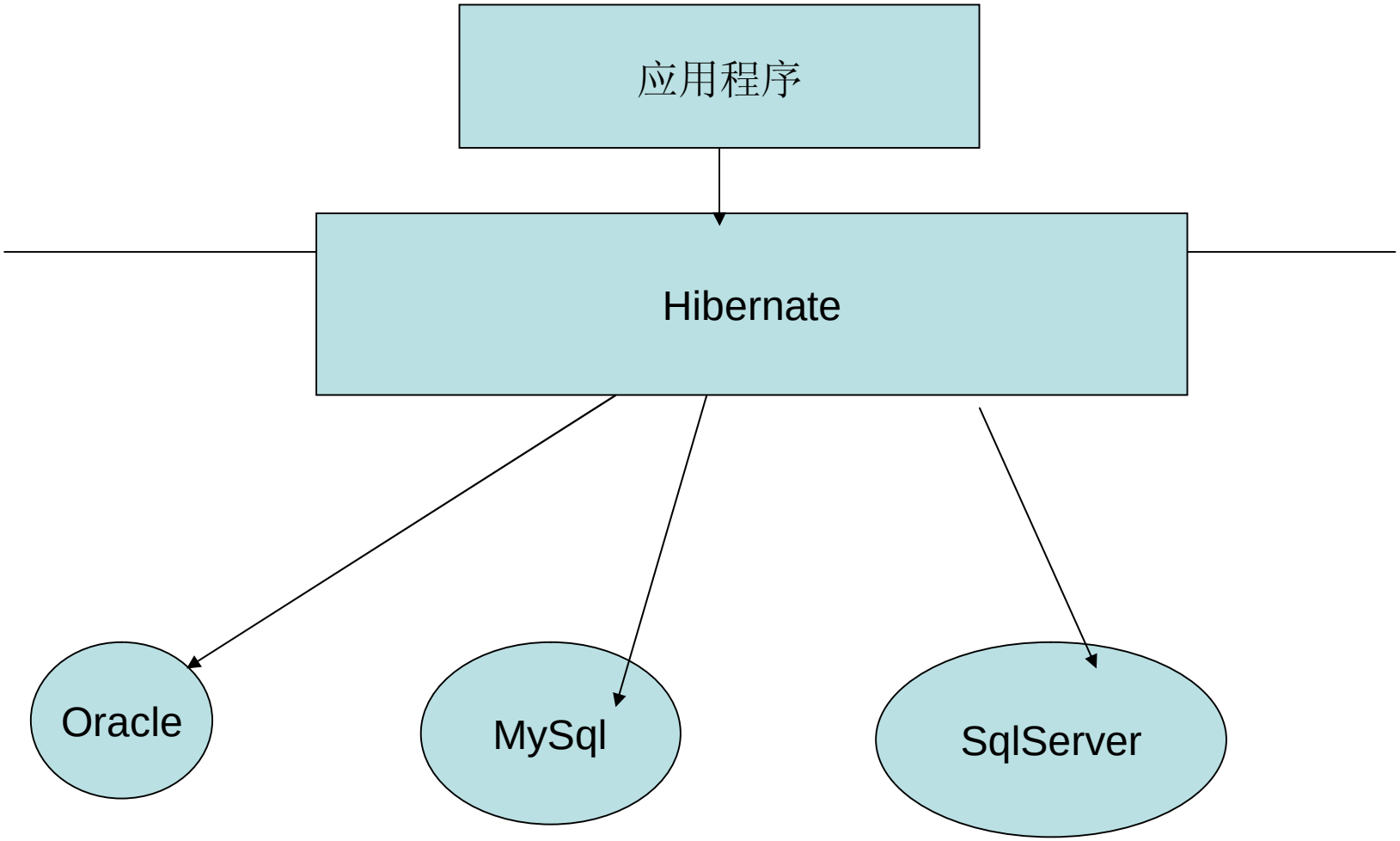
Hibernate能干什么

- Hibernate能干什么：

  Hibernate主要用来实现Java对象和表之间的映射，除此之外还提供还提供数据查询和获取数据的方法，可以大幅度减少开发时人工使用SQL和JDBC处理数据的时间。

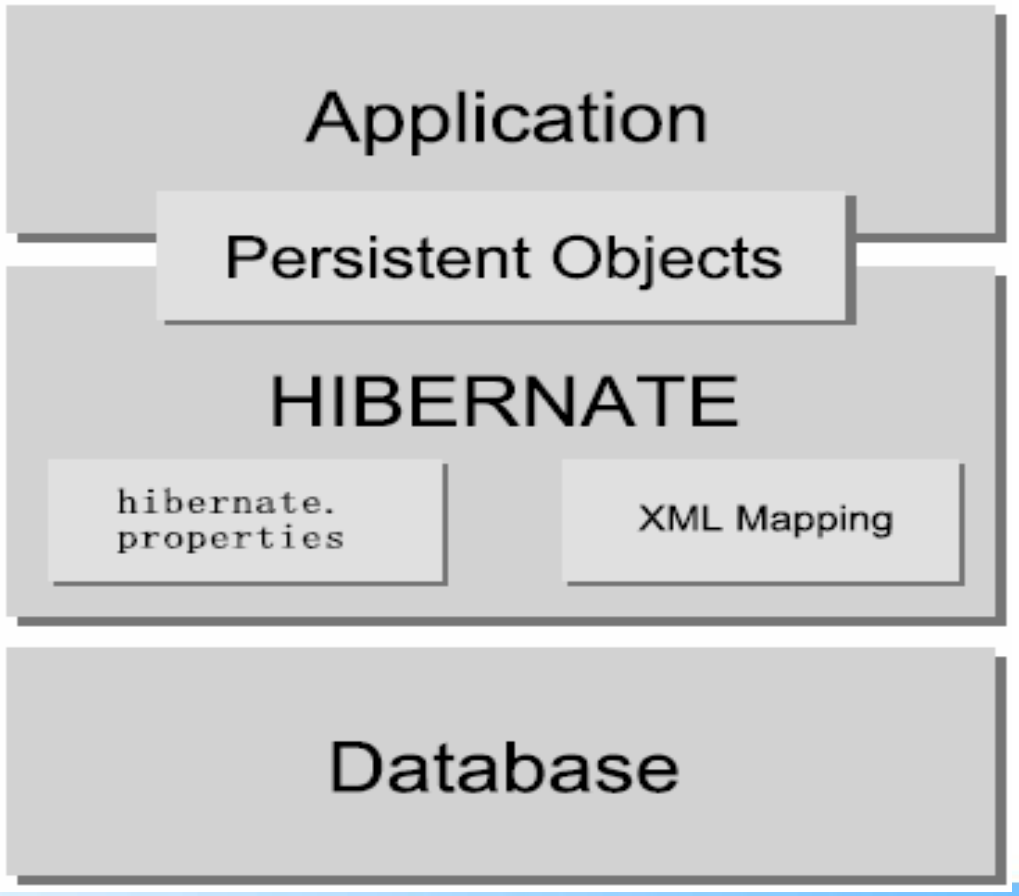
  Hibernate的目标是对于开发者通常的数据持久化相关的编程任务，解放其中的95%。对于以数据为中心的程序来说,它们往往只在数据库中使用存储过程来实现商业逻辑,Hibernate可能不是最好的解决方案;对于那些在基于Java的中间层应用中，它们实现面向对象的业务模型和商业逻辑的应用，Hibernate是最有用的。

  Hibernate可以帮助你消除或者包装那些针对特定厂商的SQL代码，并且帮你把结果集从表格式的表示形式转换到一系列的对象去。


应用程序
Hibernate
Oracle
MySql
SqlServer

Hibernate有什么-1

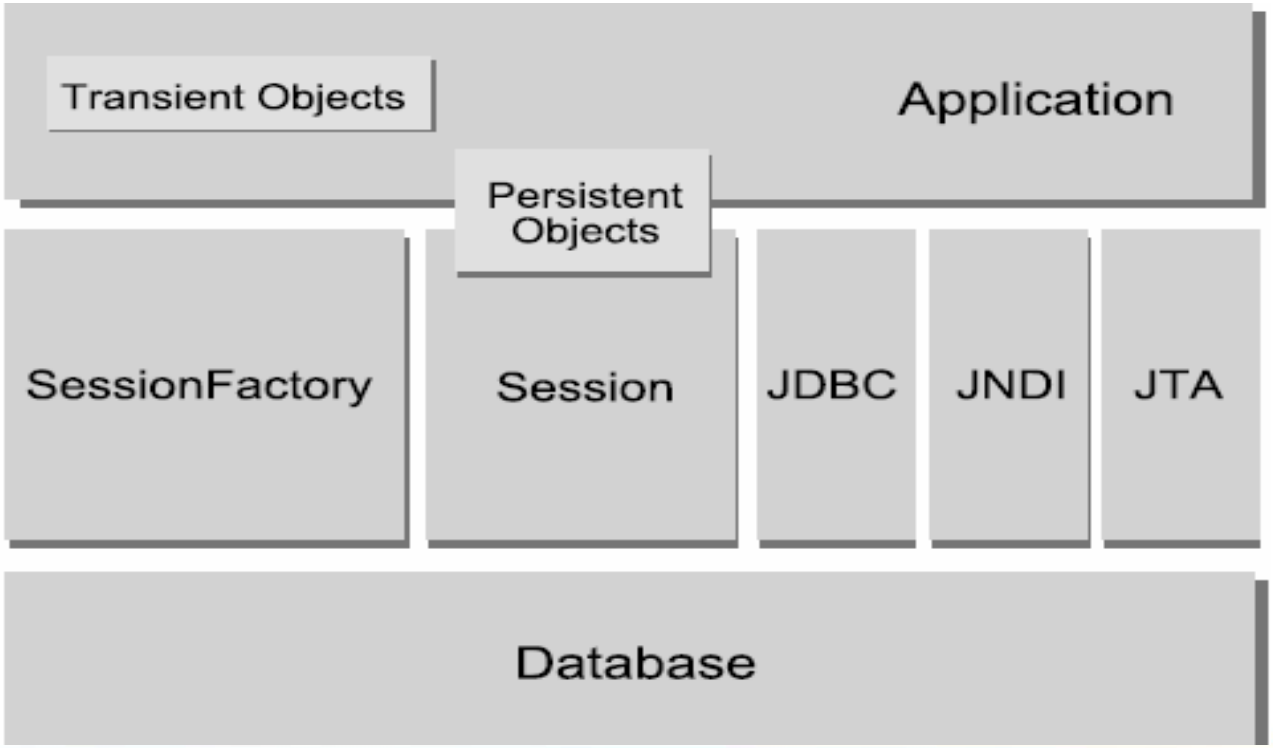
- 一个非常简要的Hibernate体系结构的高层概要图



Hibernate有什么-2

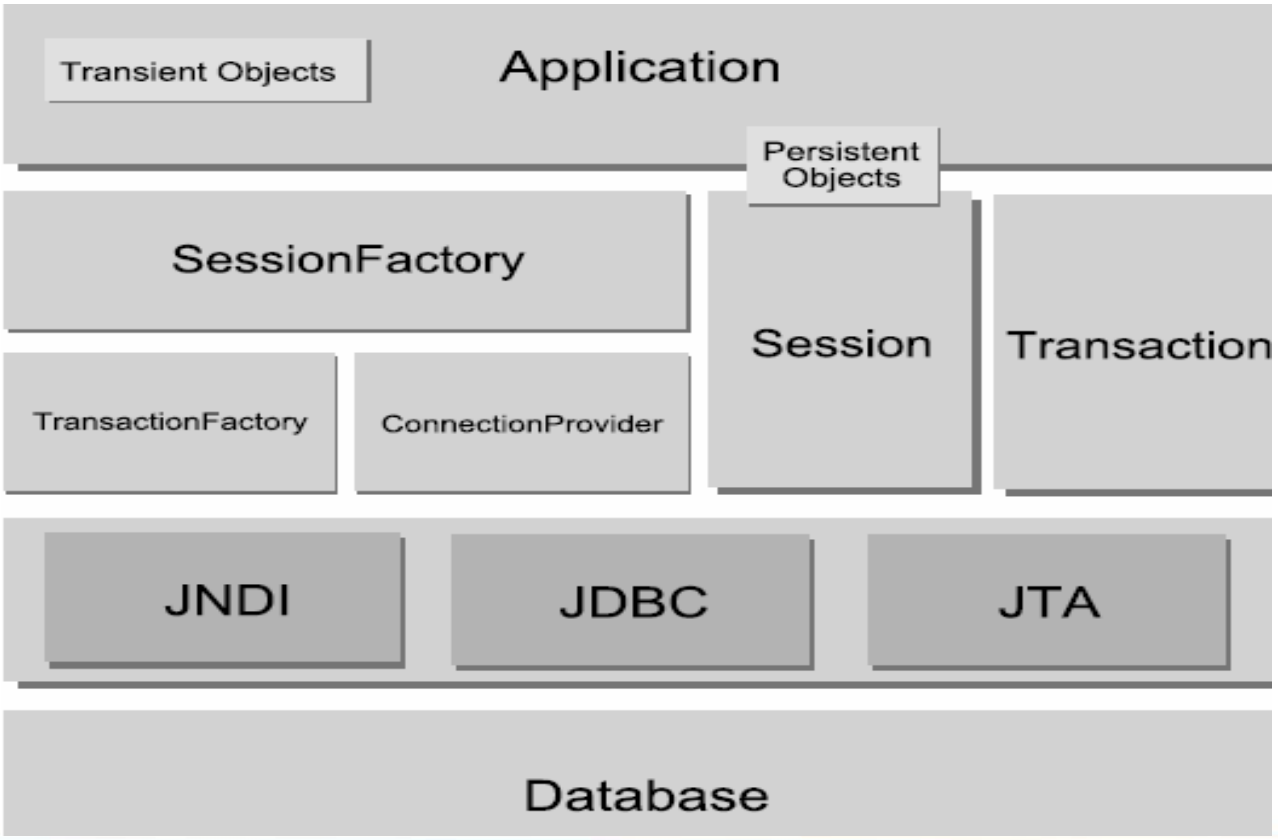
- Hibernate运行时体系结构

  “最小”的体系结构方案，要求应用程序提供自己的 JDBC 连接并管理自己的事务。这种方案使用了Hibernate API 的最小子集.



Hibernate有什么-3

- “全面解决”的体系结构方案，将应用层从底层的JDBC/JTA API中抽象出来，而让Hibernate来处理这些细节。



Hibernate有什么-4

- SessionFactory (org.hibernate.SessionFactory)
  针对单个数据库映射关系经过编译后的内存镜像，是线程安全的（不可变）。 它是生成Session的工厂，本身要用到ConnectionProvider。
- Session (org.hibernate.Session)
  表示应用程序与持久储存层之间交互操作的一个单线程对象，此对象生存期很短，隐藏了JDBC连接，也是Transaction的工厂。
- Transaction (org.hibernate.Transaction)
  应用程序用来指定原子操作单元范围的对象，它是单线程的，生命周期很短。它通过抽象将应用从底层具体的JDBC、JTA以及CORBA事务隔离开。
- ConnectionProvider (org.hibernate.connection.ConnectionProvider)
  生成JDBC连接的工厂（有连接池的作用）。它通过抽象将应用从底层的Datasource或DriverManager隔离开。仅供开发者扩展/实现用，并不暴露给应用程序使用。
- TransactionFactory (org.hibernate.TransactionFactory)
  生成Transaction对象实例的工厂。仅供开发者扩展/实现用，并不暴露给应用程序使用。

本节课程小结

- Hibernate能干什么
- Hibernate有什么、Hibernate的体系结构
- 作业：复习和掌握这些理论知识

本节课程概览

- Hibernate的HelloWorld

第一部分：Hibernate入门

HelloWorld-1

- 要研究怎么做，先得搞清楚需要做什么
  根据刚才的学习，做基本的Hibernate应用程序，要完成下面的工作：Object、数据库的表、两种配置文件、客户端程序来调用Hibernate的接口进行操作。
- 构建环境
  最简单的方法：把hibernate-release-4.0.0.Beta4.zip包里面lib/required下的jar包全部添加到工程的library里面，另外还需添加slf4j的实现包slf4j-log4j12-1.5.8.jar和log4j的实现包log4j-1.2.16.jar，还有别忘了把JDBC的驱动jar包也加入到library里面
- Object怎么做

1：就是前面学过的vo的写法（规则同样是那四点）
2：要求必须有一个public为空参的构造方法，现在写vo一般不写构造方法，默认就有一个，但是写构造方法的时候要注意写上一个public为空参的构造方法
3：要求提供一个标识属性(identifier)
4：使用非final的类（因为要使用代理来延迟实体的装载）
5：设若构建一个对象：cn.javass.h4.hello.UserModel，有四个属性：uuid,userId,name,age

HelloWorld-2

- 在数据库中建表
  设若构建一个表为tbl_user，字段：uuid,userId,name,age
- 配置xxx.cfg.xml

1：缺省名称为 hibernate.cfg.xml

2：存放在当前classes的根目录下，开发的时候在src根下就可以了

3：主要有如下四部分配置 ：

（1）与DB的连接

（2）可选配置

（3）资源文件注册

（4）二级缓存

4：配置的时候可以到Hibernate发行包里面找个hibernate.cfg.xml的例子，比如可以用“\project\hibernate-documentation\quickstart\tutorials\basic\src\test\resources”下面的hibernate.cfg.xml作例子

5：示例如下：

HelloWorld-3

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
    name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
        <property name="connection.username">test</property>
        <property name="connection.password">test</property>

        <property name="connection.pool_size">2</property>
        <property name="dialect">org.hibernate.dialect.OracleDialect</property>
        <property name="show_sql">true</property>

        <mapping resource="cn/javass/h4/hello/UserModel.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

HelloWorld-4

- 配置xxx.hbm.xml

1：与被描述的类同名 ，如：UserModel.hbm.xml

2：存放位置与所描述类存放在同一文件夹下

3：主要有如下四部分配置 ：

（1）类和表的映射

（2）主键的映射

（3）类的属性和DB中字段的映射

（4）关系的映射

4：配置的时候可以到hibernate发行包里面找个例子，比如可以用“\project\hibernate-core\src\test\java\org\hibernate\test\cid”下面的Customer.hbm.xml作例子

5：示例如下：

HelloWorld-5

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        '-//Hibernate/Hibernate Mapping DTD 3.0//EN'
        'http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd'>
<hibernate-mapping>
    <class name="cn.javass.h4.hello.UserModel" table="tbl_user">
        <id name="uuid">
            <generator class="assigned"/>
        </id>
        <property name= “userId” ></property>
        <property name= “name” ></property>
        <property name="age"></property>
    </class>
</hibernate-mapping>
```

HelloWorld-6

客户端文件：

```
package cn.javass.h4.hello;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class Client {
public static void main(String[] args) {
    SessionFactory sf = new
       Configuration().configure().buildSessionFactory();
    Session s = null;
    Transaction t = null;
```

HelloWorld-7

```
        try{
            //准备数据
            UserModel um = new UserModel();
            um.setUuid("1");
            um.setUserId("id1");
            um.setName("name1");
            um.setAge(1);
            s = sf.openSession();
            t = s.beginTransaction();
            s.save(um);
            t.commit();
        }catch(Exception err){
            t.rollback();
            err.printStackTrace();
        }finally{
             s.close();
        }
    }
}
```

HelloWorld-8

测试：直接在Elipse里面运行Client文件即可，运行结束，你将会看到在console输出：“Hibernate: insert into tbl_user (userId, name, age, uuid) values (?, ?, ?, ?)”，打开数据库的数据表，你会看到一条值已经加入了。

说明：

1：SessionFactory sf = new Configuration().configure().buildSessionFactory();
这句话的意思是读取hibernate.cfg.xml，创建Session工厂，是线程安全的。
默认是”hibernate.cfg.xml”，不用写出来，如果文件名不
是”hibernate.cfg.xml”，那么需要显示指定，如下：

```
SessionFactory sf = new
    Configuration().configure(“javass.cfg.xml”).buildSessionFactory();
```

2：Session是应用程序主要使用的Hibernate接口，约相当于JDBC的
Connection+Statement/PreparedStatement的功能，是线程不安全的

HelloWorld-9

3：在Hibernate4里面，已经不推荐使用Configuration类了，而改为使用ServiceRegistryBuilder和MetadataSources来代替，新的写法大致如下：

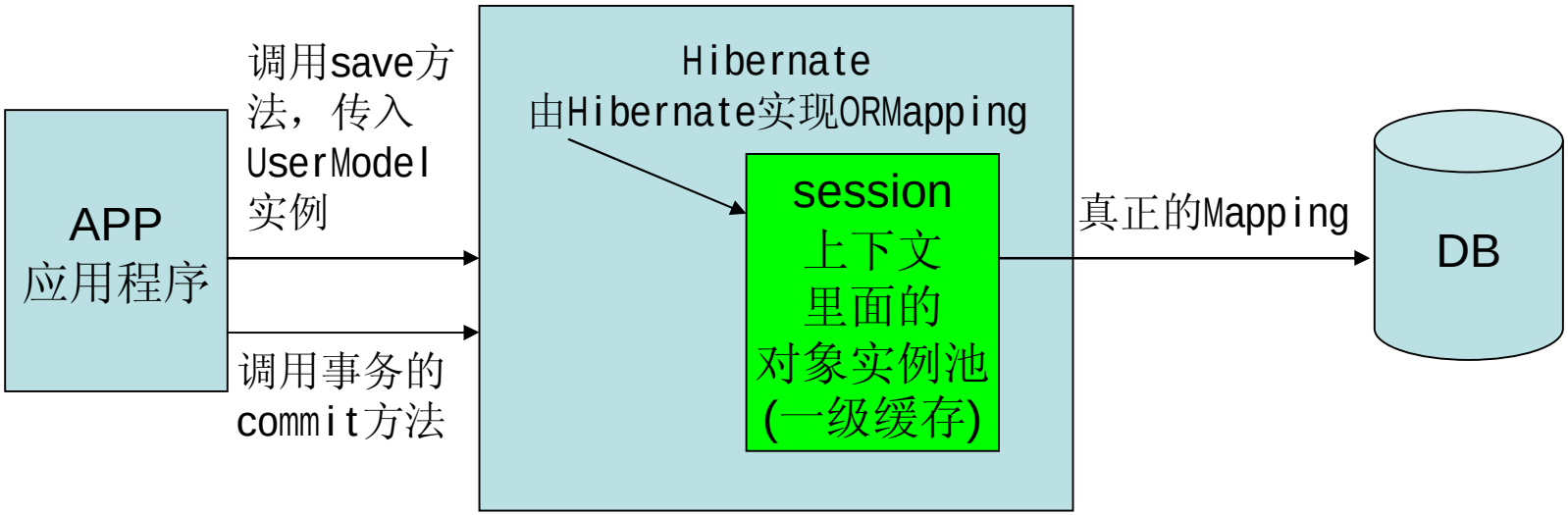
```
ServiceRegistryBuilder builder = new ServiceRegistryBuilder().configure();
builder.applySetting("connection.driver_class", "oracle.jdbc.driver.OracleDriver");
builder.applySetting("connection.url", "jdbc:oracle:thin:@localhost:1521:orcl");
builder.applySetting("connection.username", "ztb");
builder.applySetting("connection.password", "ztb");
builder.applySetting("connection.pool_size", "2");
builder.applySetting("hibernate.dialect", "org.hibernate.dialect.OracleDialect");
builder.applySetting("show_sql", "true");

MetadataSources sources = new MetadataSources( builder.buildServiceRegistry() );
sources.addResource("cn/javass/h4/hello/UserModel.hbm.xml");

MetadataImpl metadata = (MetadataImpl) sources.buildMetadata();
SessionFactory sf = metadata.getSessionFactoryBuilder().buildSessionFactory();
```

这种写法，现在还没有实现完全，不太好用，所以官方给出的示例里面还是采用以前的方式，大家先了解一下。

4：这里使用的事务Transaction是Hibernate的Transaction，需要有，不能去掉。

HelloWorld-10

为什么必须有这个Hibernate的事务呢？以HelloWorld为例来看看：



本节课程小结

- Hibernate的HelloWorld
  需要做些什么、具体每步怎么做

- 作业：按照讲述和演示，去构建Hibernate的开发和运行环境，并实现HelloWorld的功能。

本节课程概览

- Hibernate的基本配置
  XXX.cfg.xml的配置

第二部分：Hibernate的基本配置

可编程的配置方式-1

- 如果在配置cfg.xml的时候，不想在里面配置hbm.xml怎么办呢？可在程序里使用可编程的配置方式，也就是使用程序来指定在cfg.xml里面的配置信息，不推荐这种方式。如下：

```
Configuration cfg = new Configuration()
    .addResource("Item.hbm.xml")
    .addResource("Bid.hbm.xml");
```

一个替代方法（有时是更好选择）是，指定被映射的类，让Hibernate帮你寻找映射定义文件:

```
Configuration cfg = new Configuration()
    .addClass(org.hibernate.auction.Item.class)
    .addClass(org.hibernate.auction.Bid.class);
```

这种方式消除了任何对文件名的硬编码

可编程的配置方式-2

还可以通过编程的方式来指定配置属性：

```
Configuration cfg = new Configuration()
     .addClass(org.hibernate.auction.Item.class)
     .setProperty("hibernate.dialect",
    "org.hibernate.dialect.MySQLInnoDBDialect")
     .setProperty("hibernate.connection.datasource", "java:comp/env/jdbc/test")
     .setProperty("hibernate.order_updates", "true");
```

- 其他可以传入配置属性的方式：
  1：properties文件
  2：xml文件
  3：设置Java的系统属性，形如：java –Dproperty=value
- 另外要注意一点：
  org.hibernate.cfg.Configuration 实例被设计成启动期间（startup-time）对象，一旦SessionFactory 创建完成它就被丢弃了。

xxx.cfg.xml配置-与数据库连接-1

- 与数据库连接的配置有两种方法，一种是JDBC，一种是DataSource
- JDBC的配置

Hibernate JDBC属性

属性名	用途
hibernate.connection.driver_class	*jdbc驱动类*
hibernate.connection.url	*jdbc URL*
hibernate.connection.username	*数据库用户*
hibernate.connection.password	*数据库用户密码*
hibernate.connection.pool_size	*连接池容量上限数目*

- 示例：

```
<property
    name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
<property name="connection.username">javass</property>
<property name="connection.password">javass</property>
```

xxx.cfg.xml配置-与数据库连接-2

- 数据源的配置

Hibernate数据源属性

属性名	用途
hibernate.connection.datasource	数据源JNDI名字
hibernate.jndi.url	JNDI提供者的URL（可选）
hibernate.jndi.class	JNDI InitialContextFactory类（可选）
hibernate.connection.username	数据库用户（可选）
hibernate.connection.password	数据库用户密码（可选）

- 示例如下：

```
<property name="connection.datasource">java:/javassDs</property>
```

xxx.cfg.xml配置-与数据库连接-3

- 连接池c3p0的配置

由于Hibernate自己实现的连接池不太好，在项目中，建议使用工业级的连接池，比如：c3p0，Hibernate发行包中带有c3p0，下面就是其基本配置示例：

```
<property
   name="connection.driver_class">oracle.jdbc.driver.OracleDriver</proper
   ty>
        <property
   name="connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
        <property name="connection.username">javass</property>
        <property name="connection.password">javass</property>

        <property name="c3p0.min_size">5</property>
        <property name="c3p0.max_size">20</property>
        <property name="c3p0.timeout">180</property>
        <property name="c3p0.max_statements">50</property>
```

xxx.cfg.xml配置-可选配置-1

Hibernate配置属性

属性名	用途
hibernate.dialect	一个Hibernate Dialect类名允许Hibernate针对特定的关系数据库生成优化的SQL. 取值 full.classname.of.Dialect
hibernate.show_sql	输出所有SQL语句到控制台. 有一个另外的选择是把org.hibernate.SQL这个log category设为debug。 eg. true | false
hibernate.format_sql	在log和console中打印出更漂亮的SQL。 取值 true | false
hibernate.default_schema	在生成的SQL中, 将给定的schema/tablespace附加于非全限定名的表名上. 取值 SCHEMA_NAME

xxx.cfg.xml配置-可选配置-2

hibernate.default_catalog	在生成的SQL中，将给定的catalog附加于非全限定名的表名上. 取值 CATALOG_NAME
hibernate.session_factory_name	SessionFactory创建后，将自动使用这个名字绑定到JNDI中. 取值 jndi/composite/name
hibernate.max_fetch_depth	为单向关联(一对一，多对一)的外连接抓取（outer join fetch）树设置最大深度. 值为0意味着将关闭默认的外连接抓取. 取值 建议在0到3之间取值
hibernate.default_batch_fetch_size	为Hibernate关联的批量抓取设置默认数量. 取值 建议的取值为4，8，和16
hibernate.default_entity_mode	为由这个SessionFactory打开的所有Session指定默认的实体表现模式. 取值 dynamic-map，dom4j，pojo

数据库的catalog和schema

- 为了解决数据库中元素命名冲突的问题，引入catalog和schema来解决。从概念上说，一个数据库系统包含多个Catalog，每个 Catalog又包含多个Schema，而每个Schema又包含多个数据库对象（表、视图、字段等）。
- 比较简单而常用的实现方式是使用数据库名作为Catalog名，使用用户名作为Schema名，各种数据库系统对Catalog和Schema的支持具体可参见下表：

供应商	Catalog支持	Schema支持
Oracle	不支持	Oracle User ID
MySQL	不支持	数据库名
MS SQL Server	数据库名	对象属主名，2005版开始有变
DB2	指定数据库对象时，Catalog部分省略	Catalog属主名
Sybase	数据库名	数据库属主名
Informix	不支持	不需要
PointBase	不支持	数据库名

xxx.cfg.xml配置-可选配置-3

hibernate.order_updates	强制Hibernate按照被更新数据的主键，为SQL更新排序。这么做将减少在高并发系统中事务的死锁。 取值 true | false
hibernate.generate_statistics	如果开启，Hibernate将收集有助于性能调节的统计数据. 取值 true | false
hibernate.use_identifer_rollback	如果开启，在对象被删除时生成的标识属性将被重设为默认值. 取值 true | false
hibernate.use_sql_comments	如果开启，Hibernate将在SQL中生成有助于调试的注释信息，默认值为false. 取值 true | false

xxx.cfg.xml配置-JDBC和连接属性-1

Hibernate JDBC和连接(connection)属性

属性名	用途
hibernate.jdbc.fetch_size	非零值，指定JDBC抓取数量的大小（调用Statement.setFetchSize()).
hibernate.jdbc.batch_size	非零值，允许Hibernate使用JDBC2的批量更新. 取值 建议取5到30之间的值
hibernate.jdbc.batch_versioned_data	如果你想让你的JDBC驱动从executeBatch()返回正确的行计数，那么将此属性设为true(开启这个选项通常是安全的). 同时，Hibernate将为自动版本化的数据使用批量DML. 默认值为false. eg. true | false
hibernate.jdbc.factory_class	选择一个自定义的Batcher. 多数应用程序不需要这个配置属性. eg. classname.of.Batcher

xxx.cfg.xml配置-JDBC和连接属性-2

hibernate.jdbc.use_scrollable_resultset	允许Hibernate使用JDBC2的可滚动结果集. 只有在使用用户提供的JDBC连接时，这个选项才是必要的，否则Hibernate会使用连接的元数据. 取值 true | false
hibernate.jdbc.use_streams_for_binary	在JDBC读写binary（二进制）或serializable（可序列化）的类型时使用流(stream)(系统级属性). 取值 true | false
hibernate.jdbc.use_get_generated_keys	在数据插入数据库之后，允许使用JDBC3 PreparedStatement.getGeneratedKeys() 来获取数据库生成的key(键)。需要JDBC3+驱动和JRE1.4+，如果你的数据库驱动在使用Hibernate的标 识生成器时遇到问题，请将此值设为false. 默认情况下将使用连接的元数据来判定驱动的能力. 取值 true|false
hibernate.connection.provider_class	自定义ConnectionProvider的类名，此类用来向Hibernate提供JDBC连接.取值 classname.of.ConnectionProvider

xxx.cfg.xml配置-JDBC和连接属性-3

hibernate.connection.isolation	设置JDBC事务隔离级别. 查看java.sql.Connection来了解各个值的具体意义, 但请注意多数数据库都不支持所有的隔离级别. 取值 1, 2, 4, 8
hibernate.connection.autocommit	允许被缓存的JDBC连接开启自动提交(autocommit)(不建议). 取值 true | false
hibernate.connection.<propertyName>	将JDBC属性propertyName传递到DriverManager.getConnection()中去.
hibernate.jndi.<propertyName>	将属性propertyName传递到JNDI InitialContextFactory中去.

xxx.cfg.xml配置-JDBC和连接属性-4

hibernate.connection.release_mode	指定Hibernate在何时释放JDBC连接. 默认情况下,直到Session被显式关闭或被断开连接时,才会释放JDBC连接. 对于应用程序服务器的JTA数据源,你应当使用after_statement, 这样在每次JDBC调用后, 都会主动的释放连接. 对于非JTA的连接, 使用after_transaction在每个事务结束时释放连接是合理的. auto将为JTA和CMT事务策略选择after_statement, 为JDBC事务策略选择after_transaction. 取值 on_close | after_transaction | after_statement | auto

xxx.cfg.xml配置-缓存属性-1

Hibernate缓存属性

属性名	用途
hibernate.cache.provider_class	自定义的CacheProvider的类名. 取值 classname.of.CacheProvider
hibernate.cache.use_minimal_puts	以频繁的读操作为代价，优化二级缓存来最小化写操作. 在Hibernate3中，这个设置对的集群缓存非常有用，对集群缓存的实现而言，默认是开启的. 取值 true|false
hibernate.cache.use_query_cache	允许查询缓存，个别查询仍然需要被设置为可缓存的. 取值 true|false

xxx.cfg.xml配置-缓存属性-2

hibernate.cache.use_second_level_cache	能用来完全禁止使用二级缓存. 对那些在类的映射定义中指定<cache>的类，会默认开启二级缓存. 取值 true|false
hibernate.cache.query_cache_factory	自定义实现QueryCache接口的类名，默认为内建的StandardQueryCache. 取值 classname.of.QueryCache
hibernate.cache.region_prefix	二级缓存区域名的前缀. 取值 prefix
hibernate.cache.use_structured_entries	强制Hibernate以更人性化的格式将数据存入二级缓存. 取值 true|false

xxx.cfg.xml配置-事务属性-1

Hibernate事务属性

属性名	用途
hibernate.transaction.factory_class	一个TransactionFactory的类名，用于Hibernate Transaction API（默认为JDBCTransactionFactory）. 取值 classname.of.TransactionFactory
jta.UserTransaction	一个JNDI名字，被JTATransactionFactory用来从应用服务器获取JTA UserTransaction. 取值 jndi/composite/name
hibernate.transaction.manager_lookup_class	一个TransactionManagerLookup的类名 - 当使用JVM级缓存，或在JTA环境中使用hilo生成器的时候需要该类. 取值 classname.of.TransactionManagerLookup
hibernate.transaction.flush_before_completion	如果开启，session在事务完成后将被自动清洗(flush)。现在更好的方法是使用自动session上下文管理。请参见第 2.5 节 “上下文相关的（Contextual）Session”。 取值 true | false

xxx.cfg.xml配置-事务属性-2

hibernate.transaction.auto_close_session	如果开启，session在事务完成后将被自动关闭。现在更好的方法是使用自动session上下文管理。请参见第 2.5 节 “上下文相关的（Contextual）Session”。 取值 true | false

xxx.cfg.xml配置-事务策略-1

- 为了让应用在JDBC事务和JTA事务环境中可以移植，建议使用可选的Hibernate Transaction API，它包装并隐藏了底层系统
- 通过设置Hibernate配置属性hibernate.transaction.factory_class来指定 一个Transaction实例的工厂类
- 有三个标准(内建)的选择：

1：委托给数据库(JDBC)事务（默认）

Hibernate4以前:org.hibernate.transaction.JDBCTransactionFactory

Hibernate4里面:org.hibernate.engine.transaction.internal.jdbc.JDBCTransactionFactory

2：JTA事务，如果在上下文环境中存在运行着的事务(如，EJB会话Bean的方法)，则委托给容器管 理的事务，否则，将启动一个新的事务，并使用Bean管理的事务.

Hibernate4以前: org.hibernate.transaction.JTATransactionFactory

Hibernate4里面: org.hibernate.engine.transaction.internal.jta.JTATransactionFactory

3：委托给容器管理的JTA事务

Hibernate4以前: org.hibernate.transaction.CMTTransactionFactory

Hibernate4里面:org.hibernate.engine.transaction.internal.jta.CMTTransactionFactory

也可以定义属于你自己的事务策略 (如，针对CORBA的事务服务)

xxx.cfg.xml配置-事务策略-2

JTA TransactionManagers

Transaction工厂类	应用程序服务器
org.hibernate.transaction.JBossTransactionManagerLookup	JBoss
org.hibernate.transaction.WeblogicTransactionManagerLookup	Weblogic
org.hibernate.transaction.WebSphereTransactionManagerLookup	WebSphere
org.hibernate.transaction.WebSphereExtendedJTATransactionLookup	WebSphere 6
org.hibernate.transaction.OrionTransactionManagerLookup	Orion
org.hibernate.transaction.ResinTransactionManagerLookup	Resin
org.hibernate.transaction.JOTMTransactionManagerLookup	JOTM
org.hibernate.transaction.JOnASTransactionManagerLookup	JOnAS
org.hibernate.transaction.JRun4TransactionManagerLookup	JRun4
org.hibernate.transaction.BESTransactionManagerLookup	Borland ES

xxx.cfg.xml配置-其它属性-1

其他属性

属性名	用途
hibernate.current_session_context_class	为"当前" Session指定一个(自定义的)策略。关于内置策略的详情，请参见第 2.5 节 “上下文相关的（Contextual）Session” 。 eg. jta | thread | custom.Class
hibernate.query.factory_class	选择HQL解析器的实现. 取值 org.hibernate.hql.ast.ASTQueryTranslatorFactory or org.hibernate.hql.classic.ClassicQueryTranslatorFactory
hibernate.query.substitutions	将Hibernate查询中的符号映射到SQL查询中的符号（符号可能是函数名或常量名字）. 取值 hqlLiteral=SQL_LITERAL, hqlFunction=SQLFUNC

xxx.cfg.xml配置-其它属性-2

hibernate.hbm2ddl.auto	在SessionFactory创建时，自动检查数据库结构，或者将数据库schema的DDL导出到数据库. 使用create-drop时，在显式关闭SessionFactory时，将drop掉数据库schema. 取值 validate | update | create | create-drop
hibernate.cglib.use_reflection_optimizer	开启CGLIB来替代运行时反射机制(系统级属性). 反射机制有时在除错时比较有用. 注意即使关闭这个优化，Hibernate还是需要CGLIB. 你不能在hibernate.cfg.xml中设置此属性. 取值 true | false

xxx.cfg.xml配置-方言-1

RDBMS	Dialect
DB2	org.hibernate.dialect.DB2Dialect
DB2 AS/400	org.hibernate.dialect.DB2400Dialect
DB2 OS390	org.hibernate.dialect.DB2390Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL	org.hibernate.dialect.MySQLDialect
MySQL with InnoDB	org.hibernate.dialect.MySQLInnoDBDialect
MySQL with MyISAM	org.hibernate.dialect.MySQLMyISAMDialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle 9i	org.hibernate.dialect.Oracle9iDialect
Oracle 10g	org.hibernate.dialect.Oracle10gDialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect

xxx.cfg.xml配置-方言-2

SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Ingres	org.hibernate.dialect.IngresDialect
Progress	org.hibernate.dialect.ProgressDialect
Mckoi SQL	org.hibernate.dialect.MckoiDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Pointbase	org.hibernate.dialect.PointbaseDialect
FrontBase	org.hibernate.dialect.FrontbaseDialect
Firebird	org.hibernate.dialect.FirebirdDialect

xxx.cfg.xml配置-日志

1：Hibernate使用SLF4J来做日志记录，可以根据你选择的绑定把日志输出到日志框架（NOP、Simple、log4j version 1.2、JDK 1.4 logging、JCL 或 logback）上。

2：需要在 classpath 里加入 slf4j-api.jar 和你选择的绑定的 JAR 文件（使用 Log4J 时加入slf4j-log4j12.jar），当然别忘了加入log4j自己的jar包。

Hibernate日志类别

类别	功能
org.hibernate.SQL	在所有SQL DML语句被执行时为它们记录日志
org.hibernate.type	为所有JDBC参数记录日志
org.hibernate.tool.hbm2ddl	在所有SQL DDL语句执行时为它们记录日志
org.hibernate.pretty	在session清洗(flush)时，为所有与其关联的实体(最多20个)的状态记录日志
org.hibernate.cache	为所有二级缓存的活动记录日志
org.hibernate.transaction	为事务相关的活动记录日志
org.hibernate.jdbc	为所有JDBC资源的获取记录日志
org.hibernate.hql.AST	在解析查询的时候,记录HQL和SQL的AST分析日志
org.hibernate.secure	为JAAS认证请求做日志
org.hibernate	为任何Hibernate相关信息做日志 (信息量较大,但对查错非常有帮助)

本节课程小结

- Hibernate的基本配置
  XXX.cfg.xml的配置

- 作业：复习和练习这些配置，尤其是上课要求掌握的配置。

本节课程概览

- Hibernate的基本配置
  XXX.hbm.xml的配置

第二部分：Hibernate的基本配置

xxx.hbm.xml配置-hibernate-mapping元素-1

- <hibernate-mapping>元素

这个元素是xxx.hbm.xml配置的根元素，定义如下：

```
<hibernate-mapping
    schema="schemaName" (1)
    catalog="catalogName" (2)
    default-cascade="cascade_style" (3)
    default-access="field|property|ClassName" (4)
    default-lazy="true|false" (5)
    auto-import="true|false" (6)
    package="package.name" (7) />
```

(1)schema (可选)：数据库schema的名称，表名会加上所指定的schema的名字扩展为表的全限定名。

(2) catalog (可选)：数据库catalog的名称，表名会加上所指定的catalog的名字扩展为表的全限定名。

xxx.hbm.xml配置-hibernate-mapping元素-2

(3) default-cascade (可选 - 默认为 none): 默认的级联风格。指定了未明确注明cascade属性的Java属性和集合类，Hibernate会采取什么样的默认级联风格。auto-import属性默认让我们在查询语言中可以使用非全限定名的类名。

(4) default-access (可选 - 默认为 property): Hibernate用来访问所有属性的策略。可以通过实现PropertyAccessor接口自定义。

(5) default-lazy (可选 - 默认为 true): 指定了未明确注明lazy属性的Java属性和集合类， Hibernate会采取什么样的默认加载风格。

(6) auto-import (可选 - 默认为 true): 指定我们是否可以在查询语言中使用非全限定的类名（仅限于本映射文件中的类）。

(7) package (可选): 指定一个包前缀，如果在映射文档中没有指定全限定的类名， 就使用这个作为包名。

假若你有两个持久化类，它们的非全限定名是一样的（就是两个类的名字一样，所在的包不一样）， 你应该设置auto-import="false"。如果你把一个“import过”的名字同时对应两个类， Hibernate会抛出一个异常。

Lazy和1+N次查询

- 1+N次查询的问题

  如果设置了装载策略为lazy，那么可能会带来有名的1+N次查询的问题，1+N有两种典型的体现，一个是以Iterator为代表，一个是以关系映射为代表。

- 以Iterator为代表的1+N次查询

1：第一次查询的时候：如果是设置了lazy的对象，Hibernate会只装载主键的值

2：那么以后每次真正调用一个对象的时候，Hibernate发现这个对象没有值，只有主键，那么Hibernate会用主键做条件重新查询一次。

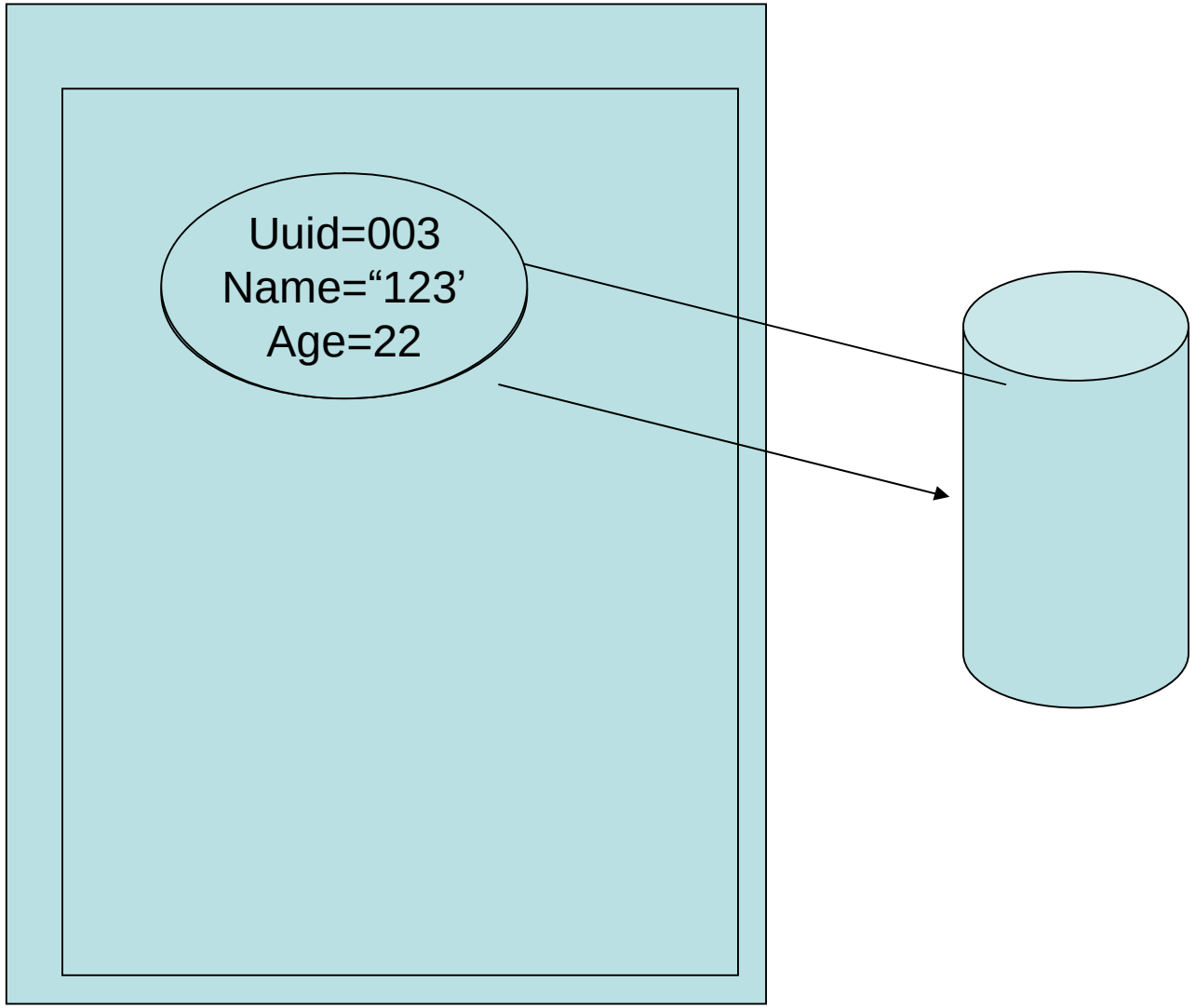
3：设若N条记录后来都访问了，那么总的查询次数就是1+N次

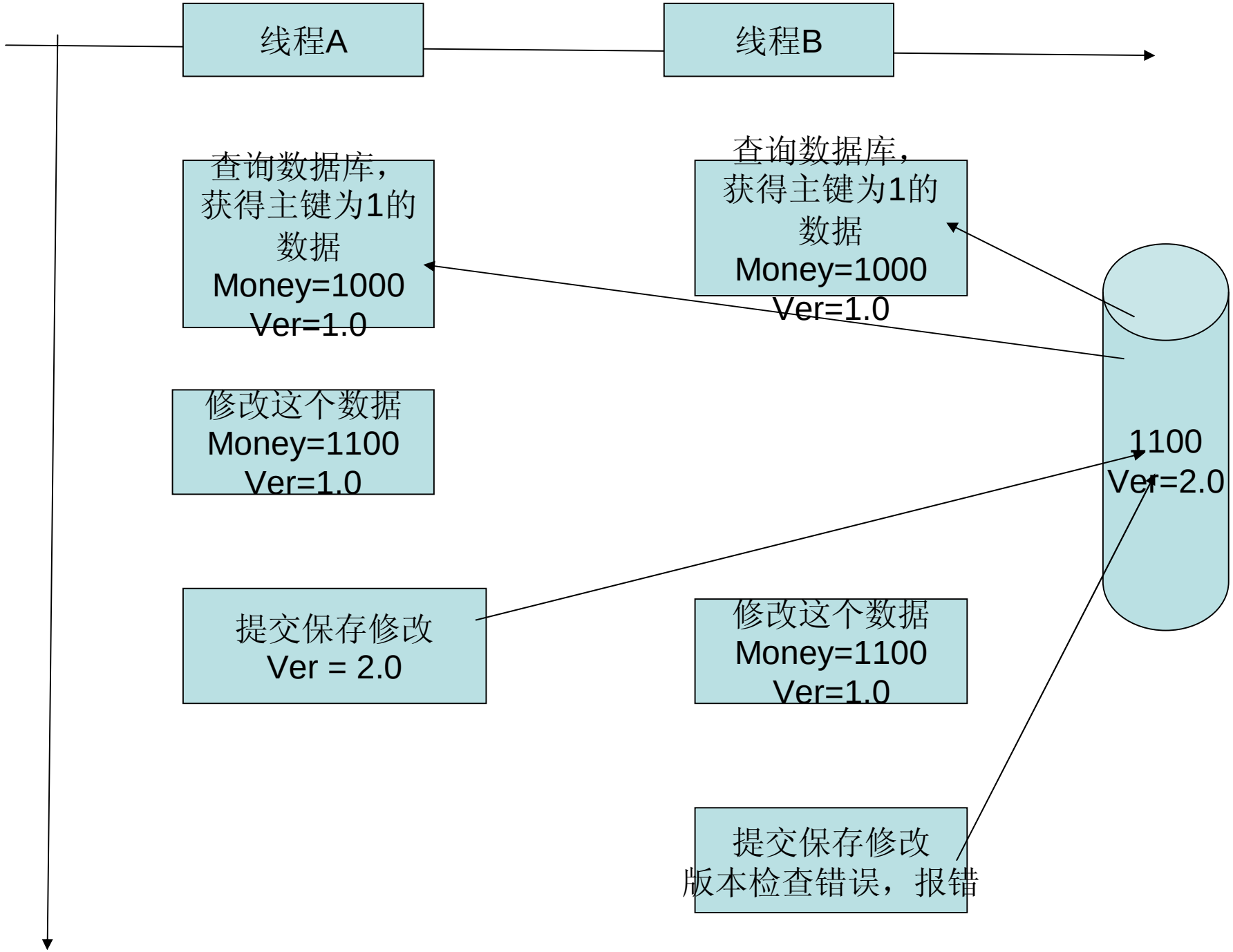
- 以关系映射为代表的1+N次查询

1：第一次查询出有N条Parnet对象

2：当访问每个Parent对象里面的Child对象或Child对象集合的时候，会重新用一条sql去访问Child对象

3：设若N条Parent对象都访问了里面的Child，那么总的查询次数就是1+N次


Uuid=003
Name=“123’
Age=22


线程A
线程B
查询数据库，
获得主键为1的
数据
Money=1000
Ver=1.0
查询数据库，
获得主键为1的
数据
Money=1000
Ver=1.0
修改这个数据
Money=1100
Ver=1.0
1100
Ver=2.0
提交保存修改
Ver = 2.0
修改这个数据
Money=1100
Ver=1.0
提交保存修改
版本检查错误，报错

xxx.hbm.xml配置-class元素-1

n <class>元素

使用class元素来定义一个持久化类：

```
<class
    name="ClassName" (1)
    table="tableName" (2)
    discriminator-value="discriminator_value" (3)
    mutable="true|false" (4)
    schema="owner" (5)
    catalog="catalog" (6)
    proxy="ProxyInterface" (7)
    dynamic-update="true|false" (8)
    dynamic-insert="true|false" (9)
    select-before-update="true|false" (10)
    polymorphism="implicit|explicit" (11)
    where="arbitrary sql where condition" (12)
    persister="PersisterClass" (13)
    batch-size="N" (14)
    optimistic-lock="none|version|dirty|all" (15)
    lazy="true|false" (16)
    entity-name="EntityName" (17)
    check="arbitrary sql check condition" (18)
    rowid="rowid" (19)
    subselect="SQL expression" (20)
    abstract="true|false" (21)
/>
```

xxx.hbm.xml配置-class元素-2

(1)name (可选): 持久化类（或者接口）的Java全限定名。如果这个属性不存在，Hibernate将假定这是一个非 POJO的实体映射。若指明的持久化类实际上是一个接口，这也是可以的，可以用元素<subclass>来指定该接口的实际实现类。

(2) table (可选 - 默认是类的非全限定名): 对应的数据库表名

(3) discriminator-value (可选 - 默认和类名一样): 一个用于区分不同的子类的值，在多态行为时使用。它可以接受的值包括 null 和 not null

(4) mutable (可选，默认值为true): 表明该类的实例是可变的或者不可变的。不可变类，mutable="false"不可以被应用程序更新或者删除。

(5) schema (optional): 覆盖在根元素<hibernate-mapping>中指定的schema

(6) catalog (optional):覆盖在根元素<hibernate-mapping>中指定的catalog

(7) proxy (可选): 指定一个接口，在延迟装载时作为代理使用。

可选的proxy属性允许延迟加载类的持久化实例。 Hibernate开始会返回实现了这个命名接口的CGLIB代理。当代理的某个方法被实际调用的时候， 真实的持久化对象才会被装载。

xxx.hbm.xml配置-class元素-3

(8) dynamic-update (可选，默认为 false)：指定用于UPDATE 的SQL将会在运行时动态生成，并且只更新那些改变过的字段。

(9) dynamic-insert (可选，默认为 false)：指定用于INSERT的 SQL 将会在运行时动态生成，并且只包含那些非空值字段

(10) select-before-update (可选，默认为 false)：指定Hibernate除非确定对象真正被修改了，否则不会执行SQL UPDATE操作。在特定场合（实际上，它只在一个瞬时对象（transient object）关联到一个 新的session中时执行的update()中生效），这说明Hibernate会在UPDATE 之前执行一次额外的SQL SELECT操作，来决定是否应该执行 UPDATE。使用select-before-update通常会降低性能。

(11) polymorphism（多态） (可选，默认值为 implicit (隐式))：界定是隐式还是显式的使用多态查询，这只在Hibernate的具体表继承策略中用到。
Implicit隐式多态是指：如果查询时给出的是任何超类、该类实现的接口或者该类的名字，都会返回这个类的实例；如果查询中给出的是子类的名字，则会返回子类的实例。Explicit显式多态是指：只有在查询时给出明确的该类名字时才会返回这个类的实例；同时只有在这个<class>的定义中作为<subclass> 或者<joined-subclass>出现的子类，才可能被返回。在大多数情况下，默认的polymorphism="implicit"都是合适的。

xxx.hbm.xml配置-class元素-4

(12) where (可选) 指定一个附加的SQLWHERE 条件， 在抓取这个类的对象时会一直增加这个条件

(13) persister (可选): 指定一个定制的ClassPersister

(14) batch-size (可选,默认是1) 指定一个用于 根据标识符（identifier）抓取实例时使用的“batch size”（批次抓取数量）

(15) optimistic-lock（乐观锁定）(可选，默认version): 决定乐观锁定的策略。如果你打开了dynamic-update，你可以选择几种乐观锁定的策略：

1）version（版本检查） 检查version/timestamp字段

2）all（全部） 检查全部字段

3）dirty（脏检查）只检察修改过的字段

4）none（不检查）不使用乐观锁定

非常强烈建议你在Hibernate中使用version/timestamp字段来进行乐观锁定。 对性能来说，这是最好的选择，并且这也是唯一能够处理在session外进行操作的策略

xxx.hbm.xml配置-class元素-5

(16) lazy (可选): 延迟加载（Lazy fetching）

(17) entity-name (可选，默认为类名): Hibernate4允许一个类进行多次映射（前提是映射到不同的表），并且允许使用Maps或XML代替Java层次的实体映射（也就是实现动态领域模型，不用写持久化类）

(18) check (可选): 这是一个SQL表达式，用于为自动生成的schema添加多行约束检查

(19) rowid (可选): Hibernate可以使用数据库支持的所谓的ROWIDs，例如：Oracle数据库，如果你设置这个可选的rowid， Hibernate可以使用额外的字段rowid实现快速更新。

(20) subselect (可选): 它将一个不可变（immutable）并且只读的实体映射到一个数据库的 子查询中。当你想用视图代替一张基本表的时候，这是有用的，但最好不要这样做

(21) abstract (可选): 用于在<union-subclass>的继承结构中标识抽象超类

xxx.hbm.xml配置-多次映射同一个类

- 如果想要多次映射同一个类，可以采用如下的方式：

  对特定的持久化类，映射多次是允许的。这种情形下，你必须指定 entity name 来区别不同映射实体的对象实例。默认情况下，实体名字和类名是相同的。 Hibernate 在操作持久化对象、编写查询条件，或者把关联映射到指定实体时，允许你指定这个 entity name（实体名字）。

xxx.hbm.xml配置-version元素-1

- <version>元素

<version> 元素是可选的，表明表中包含附带版本信息的数据。

```
<version
    column="version_column"(1)
    name="propertyName"(2)
    type="typename"(3)
    access="field|property|ClassName"(4)
    unsaved-value="null|negative|undefined"(5)
    generated="never|always"(6)
    insert="true|false"(7)
/>
```

(1)column（可选 — 默认为属性名）：指定持有版本号的字段名。
(2)name：持久化类的属性名。
(3)type（可选 — 默认是 integer）：版本号的类型。
(4)access（可选 — 默认为 property）：Hibernate 用来访问属性值的策略。

xxx.hbm.xml配置-version元素-2

(5)unsaved-value（可选 — 默认是 undefined）：用于标明某个实例是刚刚被实例化的（尚未保存）版本属性值，依靠这个值就可以把这种情况 和已经在先前的 session 中保存或装载的脱管（detached）实例区分开来。（undefined 指明应被使用的标识属性值。）

(6)generated（可选 — 默认是 never）：表明此版本属性值是否实际上是由数据库生成的。

(7)insert（可选 — 默认是 true）：表明此版本列应该包含在 SQL 插入语句中。只有当数据库字段有默认值 0 的时候，才可以设置为 false。

乐观锁示例-1

1：首先在数据库表中添加一个字段：version ，类型为number

2：在UserModel里面添加一个字段：version，类型为int，也要对应的getter和setter方法

3：在UserModel.hbm.xml中添加配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        '-//Hibernate/Hibernate Mapping DTD 3.0//EN'
        'http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd'>
<hibernate-mapping>
    <class name="cn.javass.h3.hello.UserModel" table="tbl_user"   optimistic-lock="version">
        <id name="uuid">
            <generator class="assigned"/>
        </id>
        <version name="version"/>
        <property name="userId"></property>
    <property name="name"></property>
    <property name="age"></property>
    </class>
</hibernate-mapping>
```

乐观锁示例-2

4：然后把数据库tbl_user中的数据全部清除掉，这样好观察数据变化

5：运行原来的client，在数据库中生成一条数据，值是：uuid=1，userId=id1，name=name1，age=1，version=0

6：注意：version字段为Hibernate自己维护，程序中不需要操作这个字段

7：然后写新的客户端进行测试，如下：

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class Client {
    public static void main(String[] args) {
        SessionFactory sf = new Configuration().configure().buildSessionFactory();
        Session session1 = null;
        Session session2 = null;
        try{
```

乐观锁示例-3

```
                // 有使用者1开启了一个session1
                session1 = sf.openSession();
                // 在这之后，马上有另一个使用者2开启了session2
                session2 = sf.openSession();
                // 使用者1查询数据
                UserModel userV1 = (UserModel) session1.load(UserModel.class, "1");
                // 使用者2查询同一条数据
                UserModel userV2 = (UserModel) session2.load(UserModel.class, "1");
                // 此时两个版本号是相同的
System.out.println("v1="+ userV1.getVersion() + ",v2="+ userV2.getVersion());
                Transaction tx1 = session1.beginTransaction();
                Transaction tx2 = session2.beginTransaction();
                // 使用者1更新数据
                userV1.setAge(111);
                tx1.commit();
                // 此时由于数据更新，数据库中的版本号递增了
                // 两笔数据版本号不一样了
                    System.out.println("v1="+userV1.getVersion()+",v2="+
                    userV2.getVersion());
```

乐观锁示例-4

```
            // userV2 的 age 数据还是旧的
            // 数据更新
            userV2.setName("version test");
            // 因版本号比数据库中的旧
            // 修改会失败，抛出StableObjectStateException例外
            tx2.commit();
        }catch(Exception err){
            err.printStackTrace();
        }finally{
            session1.close();
            session2.close();
        }
    }
}
```

本节课程小结

- Hibernate的基本配置
  XXX.hbm.xml的配置

- 作业：复习和练习这些配置，尤其是上课要求掌握的配置。

本节课程概览

- Hibernate的基本配置
  XXX.hbm.xml的配置

第二部分：Hibernate的基本配置

xxx.hbm.xml配置-id元素

被映射的类必须定义对应数据库表主键字段。大多数类有一个JavaBeans风格的属性， 为每一个实例包含唯一的标识。<id> 元素定义了该属性到数据库表主键字段的映射。

```
<id
    name="propertyName" (1)
    type="typename" (2)
    column="column_name" (3)
    unsaved-value="null|any|none|undefined|id_value" (4)
    access="field|property|ClassName"> (5)
    <generator class="generatorClass"/>
</id>
```

(1) name (可选): 标识属性的名字

(2) type (可选): 标识Hibernate类型的名字

(3) column (可选 - 默认为属性名): 主键字段的名字

(4) unsaved-value (可选 - 默认为一个切合实际（sensible）的值): 一个特定的标识属性值，用来标志该实例是刚刚创建的，尚未保存。 unsaved-value 属性在Hibernate4中几乎不再需要。

(5) access (可选 - 默认为property)：Hibernate用来访问属性值得策略

Hibernate的类型-1

- Hibernate的类型

  Hibernate在进行映射的时候，既不使用Java的类型，也不使用数据库的类型，而是使用自己提供的类型， Hibernate提供大量的内建类型，也支持自定义类型。

- Hibernate的内建类型

内置的 basic mapping types 可以大致地分类为：

1：integer, long, short, float, double, character, byte, boolean, yes_no, true_false

这些类型都对应Java的原始类型或其封装类，来符合特定厂商的SQL 字段类型。

2：boolean,yes_no和true_false

是Java中boolean或者Boolean的另外说法。

3：string

从 java.lang.String 到 VARCHAR（或者 Oracle 的 VARCHAR2）的映射。

4：date, time, timestamp

从java.util.Date和其子类到SQL类型 DATE，TIME 和 TIMESTAMP（或等价类型）的映射。

5：calendar, calendar_date

从java.util.Calendar到SQL类型TIMESTAMP和DATE（或等价类型）的映射。

Hibernate的类型-2

6：big_decimal, big_integer

从 java.math.BigDecimal 和 java.math.BigInteger 到 NUMERIC（或者 Oracle 的 NUMBER类型）的映射。

7：locale, timezone, currency

从 java.util.Locale，java.util.TimeZone 和 java.util.Currency 到 VARCHAR（或者 Oracle 的VARCHAR2 类型）的映射。Locale 和 Currency 的实例被映射为它们的 ISO 代码。TimeZone 的实例被影射为它的 ID。

8：class

从java.lang.Class 到 VARCHAR（或者 Oracle 的 VARCHAR2 类型）的映射。Class 被映射为它的全限定名。

9：binary

把字节数组（byte arrays）映射为对应的 SQL 二进制类型。

10：text

把长 Java 字符串映射为 SQL 的 CLOB 或者 TEXT 类型。

Hibernate的类型-3

11：serializable

把可序列化的 Java 类型映射到对应的 SQL 二进制类型。你也可以为一个并非默认为基本类型的可序列化 Java 类或者接口指定 Hibernate 类型 serializable。

12：clob, blob

JDBC 类 java.sql.Clob 和 java.sql.Blob的映射。某些程序可能不适合使用这个类型，因为blob 和 clob 对象可能在一个事务之外是无法重用的。（而且，驱动程序对这种类型的支持充满着补丁和前后矛盾。）

13：imm_date, imm_time, imm_timestamp, imm_calendar, imm_calendar_date, imm_serializable, imm_binary

一般来说，映射类型被假定为是可变的 Java 类型，只有对不可变 Java 类型，Hibernate 会采取特定的优化措施，应用程序会把这些对象作为不可变对象处理。比如，你不应该对作为 imm_timestamp 映射的 Date 执行 Date.setTime()。要改变属性的值，并且保存这一改变，应用程序必须对这一属性重新设置一个新的（不一样的）对象。

在 org.hibernate.Hibernate 中，定义了基础类型对应的 Type 常量。比如，Hibernate.STRING 代表 string 类型。

xxx.hbm.xml配置-composite-id元素

- composite-id元素

如果表使用联合主键，你可以映射类的多个属性为标识符属性。这种做法现在是非常不推荐的，但可能有些遗留系统使用了双主键或多主键。

配置如下示例：

```
<composite-id>
    <key-property name= “name"/>
    <key-property name="deptment"/>
</composite-id>
```

就表示是一个双主键，由name和deptment联合来做主键

使用Composite-id元素的时候，你的持久化类必须覆盖 equals() 和 hashCode() 方法，来实现组合的标识符的相等判断。实现Serializable 接口也是必须的。

xxx.hbm.xml配置-generator元素-1

可选的<generator>子元素是一个Java类的名字，用来为该持久化类的实例生成唯一的标识。如果这个生成器实例需要某些配置值或者初始化参数， 用<param>元素来传递。如下示例：

```
<id name="id" type="long" column="person_id">
    <generator class="sequence">
        <param name="sequence">person_id_sequence</param>
    </generator>
</id>
```

下面是一些Hibernate内置生成器：

- **n** increment

  用于为long, short或者int类型生成 唯一标识。只有在没有其他进程往同一张表中插入数据时才能使用。 在集群下不要使用。

- **n** identity

  对DB2,MySQL, MS SQL Server, Sybase和HypersonicSQL的内置标识字段提供支持。 返回的标识符是long, short 或者int类型的。

xxx.hbm.xml配置-generator元素-2

- Sequence

  在DB2,PostgreSQL, Oracle, SAP DB, McKoi中使用sequence， 而在Interbase中使用生成器(generator)。返回的标识符是long, short或int类型的

- hilo

  使用一个高/低位算法高效的生成long, short 或者 int类型的标识符。给定一个表和字段（默认分别是 hibernate_unique_key 和next_hi）作为高位值的来源。 高/低位算法生成的标识符只在一个特定的数据库中是唯一的。

- Seqhilo

  使用一个高/低位算法来高效的生成long, short 或者 int类型的标识符，给定一个数据库序列（sequence)的名字。

- Uuid

  用一个128-bit的UUID算法生成字符串类型的标识符， 这在一个网络中是唯一的（使用了IP地址）。UUID被编码为一个32位16进制数字的字符串。

- guid

  在MS SQL Server 和 MySQL 中使用数据库生成的GUID字符串

xxx.hbm.xml配置-generator元素-3

- native
  根据底层数据库的能力选择identity, sequence 或者hilo中的一个。
- Assigned
  让应用程序在save()之前为对象分配一个标示符。这是 <generator>元素没有指定时的默认生成策略。
- select
  通过数据库触发器选择一些唯一主键的行并返回主键值来分配一个主键。
- foreign
  使用另外一个相关联的对象的标识符。通常和<one-to-one>联合起来使用

- 推荐使用UUID
  UUID 包含：IP 地址、JVM 的启动时间（精确到 1/4 秒）、系统时间和一个计数器值（在 JVM 中唯一）。

xxx.hbm.xml配置-property元素-1

<property>元素为类定义了一个持久化的,JavaBean风格的属性。

```
<property
    name="propertyName" (1)
    column="column_name" (2)
    type="typename" (3)
    update="true|false" (4)
    insert="true|false" (5)
    formula="arbitrary SQL expression" (6)
    access="field|property|ClassName" (7)
    lazy="true|false" (8)
    unique="true|false" (9)
    not-null="true|false" (10)
    optimistic-lock="true|false" (11)
    generated="never|insert|always" (12)
/>
```

xxx.hbm.xml配置-property元素-2

(1) name: 属性的名字,以小写字母开头

(2) column (可选 - 默认为属性名字): 对应的数据库字段名。也可以通过嵌套的<column>元素指定。

(3) type (可选): 一个Hibernate类型的名字。

typename可以是如下几种:

A: Hibernate基本类型名（比如: integer, string, character, date, timestamp, float, binary, serializable, object, blob）。

B: 一个Java类的名字，这个类属于一种默认基础类型 (比如: int, float,char, java.lang.String, java.util.Date, java.lang.Integer, java.sql.Clob)。

C: 一个可以序列化的Java类的名字。

D: 一个自定义类型的类的名字。（比如: cn.javass.type.MyCustomType)。

如果你没有指定类型，Hibernarte会使用反射来得到这个名字的属性，以此来猜测正确的Hibernate类型。 Hibernate会按照规则B,C,D的顺序对属性读取器(getter方法）的返回类进行解释。然而，这还不够。 在某些情况下你仍然需要type属性。（比如，为了区别Hibernate.DATE 和Hibernate.TIMESTAMP,或者为了指定一个自定义类型。）

xxx.hbm.xml配置-property元素-3

(4) update, insert (可选 - 默认为 true) : 表明用于UPDATE 和/或 INSERT 的SQL语句中是否包含这个被映射了的字段。这二者如果都设置为false 则表明这是一个“外源性（derived）”的属性，它的值来源于映射到同一个（或多个）字段的某些其他属性，或者通过一个trigger(触发器）或其他程序生成。

(5) formula (可选): 一个SQL表达式，定义了这个计算属性的值。计算属性没有和它对应的数据库字段，相当于是一个子查询。

(6) access (可选 - 默认值property): Hibernate访问属性值的策略
access属性用来让你控制Hibernate如何在运行时访问属性。在默认情况下，Hibernate会使用属性的get/set方法对。如果你指明access="field", Hibernate会忽略get/set方法对，直接使用反射来访问成员变量。你也可以指定你自己的策略， 这就需要你自己实现org.hibernate.property.PropertyAccessor接口， 再在access中设置你自定义策略类的名字。

(7) lazy (可选 - 默认为 false): 指定 指定实例变量第一次被访问时，这个属性是否延迟抓取（fetched lazily）（ 需要运行时字节码增强）。

xxx.hbm.xml配置-property元素-4

(8) unique (可选): 使用DDL为该字段添加唯一的约束。 同样，允许它作为property-ref引用的目标。

(9) not-null (可选): 使用DDL为该字段添加可否为空（nullability）的约束。

(10) optimistic-lock (可选 - 默认为 true): 指定这个属性在做更新时是否需要获得乐观锁定（optimistic lock）。 换句话说，它决定这个属性发生脏数据时版本（version）的值是否增长。

(11) generated (可选 – 默认值never): 表明此属性值是否由数据库生成

组件映射-1

- <component>元素

<component> 元素把子对象的一些元素与父类对应的表的一些字段映射起来。然后组件可以定义它们自己的属性、组件或者集合。

```
<component
    name="propertyName"(1)
    class="className"(2)
    insert="true|false"(3)
    update="true|false"(4)
    access="field|property|ClassName"(5)
    lazy="true|false"(6)
    optimistic-lock="true|false"(7)
    unique="true|false"(8)
>
    <property ...../>
    <many-to-one .... />
</component>
```

组件映射-2

(1)name：属性名。
(2)class（可选 — 默认为通过反射得到的属性类型）：组件（子）类的名字。
(3)insert：被映射的字段是否出现在 SQL 的 INSERT 语句中？
(4)update：被映射的字段是否出现在 SQL 的 UPDATE 语句中？
(5)access（可选 — 默认为 property）：Hibernate 用来访问属性值的策略。
(6)lazy（可选 — 默认是 false）：表明此组件应在实例变量第一次被访问的时候延迟加载（需要编译时字节码装置器）。
(7)optimistic-lock（可选 — 默认是 true）：表明更新此组件是否需要获取乐观锁。换句话说，当这个属性变脏时，是否增加版本号（Version）。
(8)unique（可选 — 默认是 false）：表明组件映射的所有字段上都有唯一性约束。

其 <property> 子标签为子类的一些属性与表字段之间建立映射。

组件映射示例-1

组件(Component)是一个被包含的对象，在持久化的过程中，它被当作值类型，而并非一个实体的引用，指的是对象的合成。使用示例如下：

1：新建一个PersonModel，包含字段：userId和name

2：修改UserModel，在里面去掉userId和name，然后添加一个字段：p，类型为PersonModel，也要对应的getter和setter方法

3：在UserModel.hbm.xml中修改配置如下：

```
<hibernate-mapping>
    <class name="cn.javass.h3.hello.UserModel" table="tbl_user">
        <id name="uuid">
            <generator class="assigned"/>
        </id>
    <component name="p" class="cn.javass.h3.hello.PersonModel">
          <property name="userId"></property>
          <property name="name"></property>
    </component>
    <property name="age"></property>
    </class>
</hibernate-mapping>
```

组件映射示例-2

4：相应的修改Client文件如下：

```
//准备数据
UserModel um = new UserModel();
um.setUuid("13");
um.setAge(1);
PersonModel pm = new PersonModel();
pm.setUserId("id1");
pm.setName("name1");
um.setP(pm);
```

5：运行测试

xxx.hbm.xml配置- join元素-1

- <join>元素

  使用 <join> 元素，假若在表之间存在一对一关联,可以将一个类的属性映射到多张表中。

```
<join
    table="tablename"(1)
    schema="owner"(2)
    catalog="catalog"(3)
    fetch="join|select"(4)
    inverse="true|false"(5)
    optional="true|false"(6)
>
    <key ... />
    <property ... />
</join>
```

xxx.hbm.xml配置- join元素-2

(1)table：被连接表的名称。
(2)schema（可选）：覆盖在根 <hibernate-mapping> 元素中指定的schema名字。
(3)catalog（可选）：覆盖在根<hibernate-mapping> 元素中指定的catalog名字。
(4)fetch（可选 — 默认是 join）：如果设置为默认值 join，Hibernate 将使用一个内连接来得到这个类或其超类定义的 <join>，而使用一个外连接来得到其子类定义的 <join>。如果设置为 select，则 Hibernate 将为子类定义的 <join> 使用顺序选择。这仅在一行数据表示一个子类的对象的时候才会发生。对这个类和其超类定义的 <join>，依然会使用内连接得到。
(5)inverse（可选 — 默认是 false）：如果打开，Hibernate 不会插入或者更新此连接定义的属性。
(6)optional（可选 — 默认是 false）：如果打开，Hibernate 只会在此连接定义的属性非空时插入一行数据，并且总是使用一个外连接来得到这些属性。

xxx.hbm.xml配置- join元素-3

例如，一个人（person）的地址（address）信息可以被映射到单独的表中（并保留所有属性的值类型语义）：

```
<class name="Person"
table="PERSON">
   <id name="id" column="PERSON_ID"
   >...</id>
   <join table="ADDRESS">
        <key column="ADDRESS_ID"/>
        <property name="address"/>
        <property name="zip"/>
        <property name="country"/>
   </join>
...
```

此特性常常对遗留数据模型有用

xxx.hbm.xml配置-properties映射-1

- <properties>元素

  <properties> 元素允许定义一个命名的逻辑分组（grouping）包含一个类中的多个属性。这个元素最重要的用处是允许多个属性的组合作为 property-ref 的目标（target）。这也是定义多字段唯一约束的一种方便途径。

- 基本定义如下：

```
<properties
   name="logicalName"    (1)
   insert="true|false"   (2)
   update="true|false"   (3)
   optimistic-lock="true|false" (4)
   unique="true|false"   (5)
>
   <property ...../>
   <many-to-one .... />
</properties>
```

xxx.hbm.xml配置-properties映射-2

(1)name：分组的逻辑名称 — 不是实际属性的名称。
(2)insert：被映射的字段是否出现在SQL的INSERT语句中
(3)update：被映射的字段是否出现在SQL的UPDATE语句中
(4)optimistic-lock（可选 — 默认是 true）：表明更新此组件是否需要获取乐观锁。换句话说，当这个属性变脏时，是否增加版本号（Version）。
(5)unique（可选—默认 false）：表明组件映射的所有字段上都有唯一性约束。

- <properties>元素的简单示例

```
<properties name="name"
   unique="true" update="false">
   <property name="firstName"/>
   <property name="initial"/>
   <property name="lastName"/>
</properties>
```

在其他元素里面，就可以使用属性参照来引用了，如property-ref="name"

xxx.hbm.xml配置-其他

- 与对象继承相关的配置
  鉴别器(discriminator)、子类、连接的子类、联合子类、连接、Any元素

- 与关系相关的配置
  多对一(many-to-one)、一对一(one-to-one)、key映射

- 与version类似的timestamp
  同样用在乐观锁的配置上，作为版本的替代。时间戳本质上是一种对乐观锁定的一种不是特别安全的实现，并不推荐使用。

- 自然ID(natural-id)
  在<natural-id>元素中列出自然键的属性。Hibernate会帮你生成必须的唯一键值和非空约束，但这一映射不是为了把自然键作为主键而准备的。

- 动态组件（dynamic-component）
  类似于Component元素，由Hibernate根据配置进行动态映射Map类型，不推荐。

本节课程小结

- Hibernate的基本配置
  XXX.hbm.xml的配置

- 作业：复习和练习这些配置，尤其是上课要求掌握的配置。

本节课程概览

- Hibernate的基本开发
  应用Hibernate提供的方法来实现CRUD

第三部分：Hibernate的基本开发

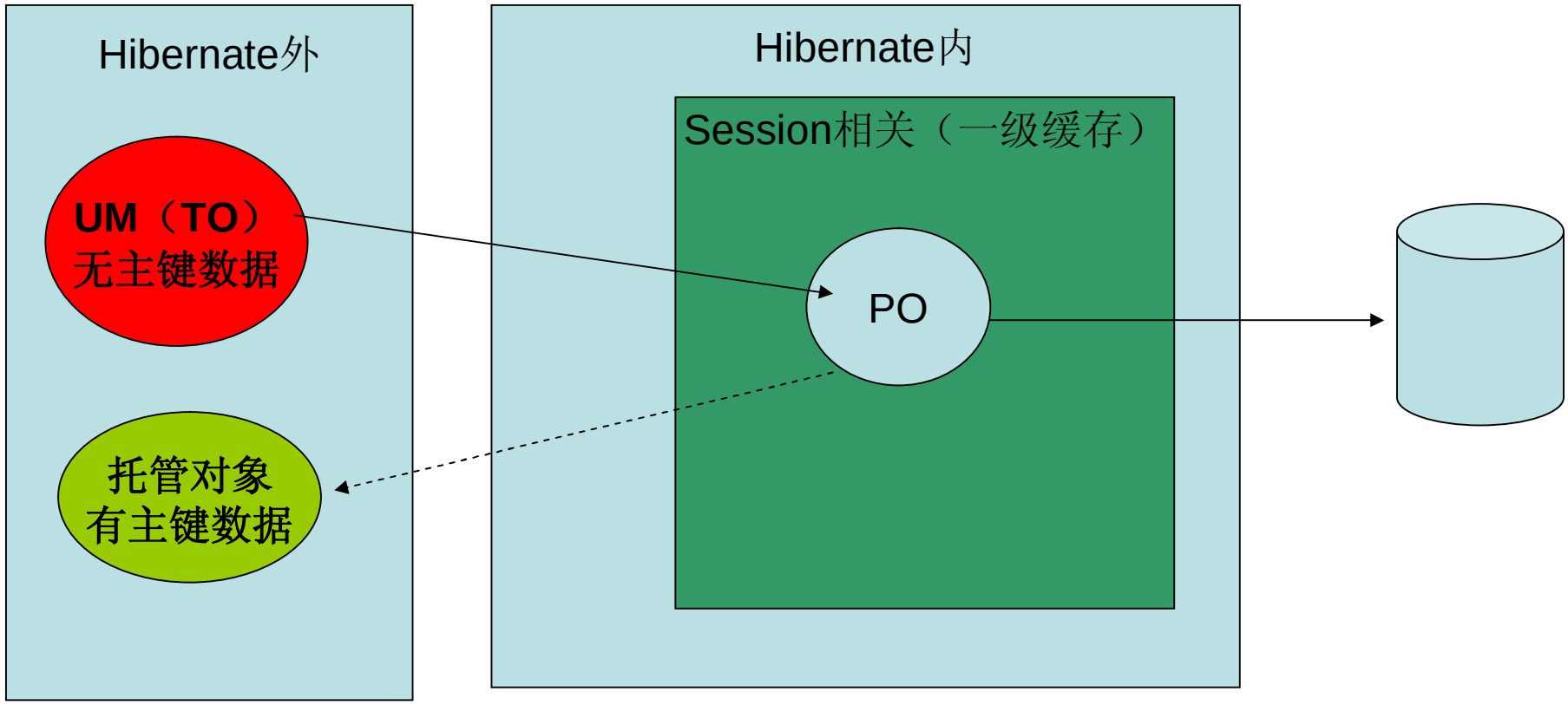
Hibernate对象状态

1：**瞬时**(Transient) - 由new操作符创建，且尚未与Hibernate Session 关联的对象被认定为瞬时的。瞬时对象不会被持久化到数据库中，也不会被赋予持久化标识(identifier)。 如果瞬时对象在程序中没有被引用，它会被垃圾回收器销毁。 使用Hibernate Session可以将其变为持久状态，Hibernate会自动执行必要的SQL语句。

2：**持久**(Persistent) - 持久的实例在数据库中有对应的记录，并拥有一个持久化标识。 持久的实例可能是刚被保存的，或刚被加载的，无论哪一种，按定义，它存在于相关联的Session作用范围内。 Hibernate会检测到处于持久状态的对象的任何改动，在当前操作单元执行完毕时将对象数据与数据库同步。开发者不需要手动执行UPDATE。将对象从持久状态变成瞬时状态同样也不需要手动执行DELETE语句。

3：**脱管**(Detached) - 与持久对象关联的Session被关闭后，对象就变为脱管的。 对脱管对象的引用依然有效，对象可继续被修改。脱管对象如果重新关联到某个新的Session上， 会再次转变为持久的，在脱管期间的改动将被持久化到数据库。

Hibernate对象状态



Hibernate对象操作CRUD-1

通过Session接口来操作Hibernate

- 新增——save方法、persist方法

  1：persist() 使一个临时实例持久化。然而，它不保证立即把标识符值分配给持久性实例，这会发生在flush的时候。persist() 也保证它在事务边界外调用时不会执行 INSERT 语句。这对于长期运行的带有扩展会话/持久化上下文的会话是很有用的。

  2：save() 保证返回一个标识符。如果需要运行 INSERT 来获取标识符（如 "identity" 而非"sequence" 生成器），这个 INSERT 将立即执行，不管你是否在事务内部还是外部。这对于长期运行的带有扩展会话/持久化上下文的会话来说会出现问题。

- 删除——delete方法

Hibernate对象操作CRUD-2

- 修改——有四种方法来做，分别是：

1：直接在Session打开的时候load对象，然后修改这个持久对象，在事务提交的时候，会自动flush到数据库中。

2：修改托管对象，可用update或merge方法

3：自动状态检测：saveOrUpdate方法

- update和merge方法

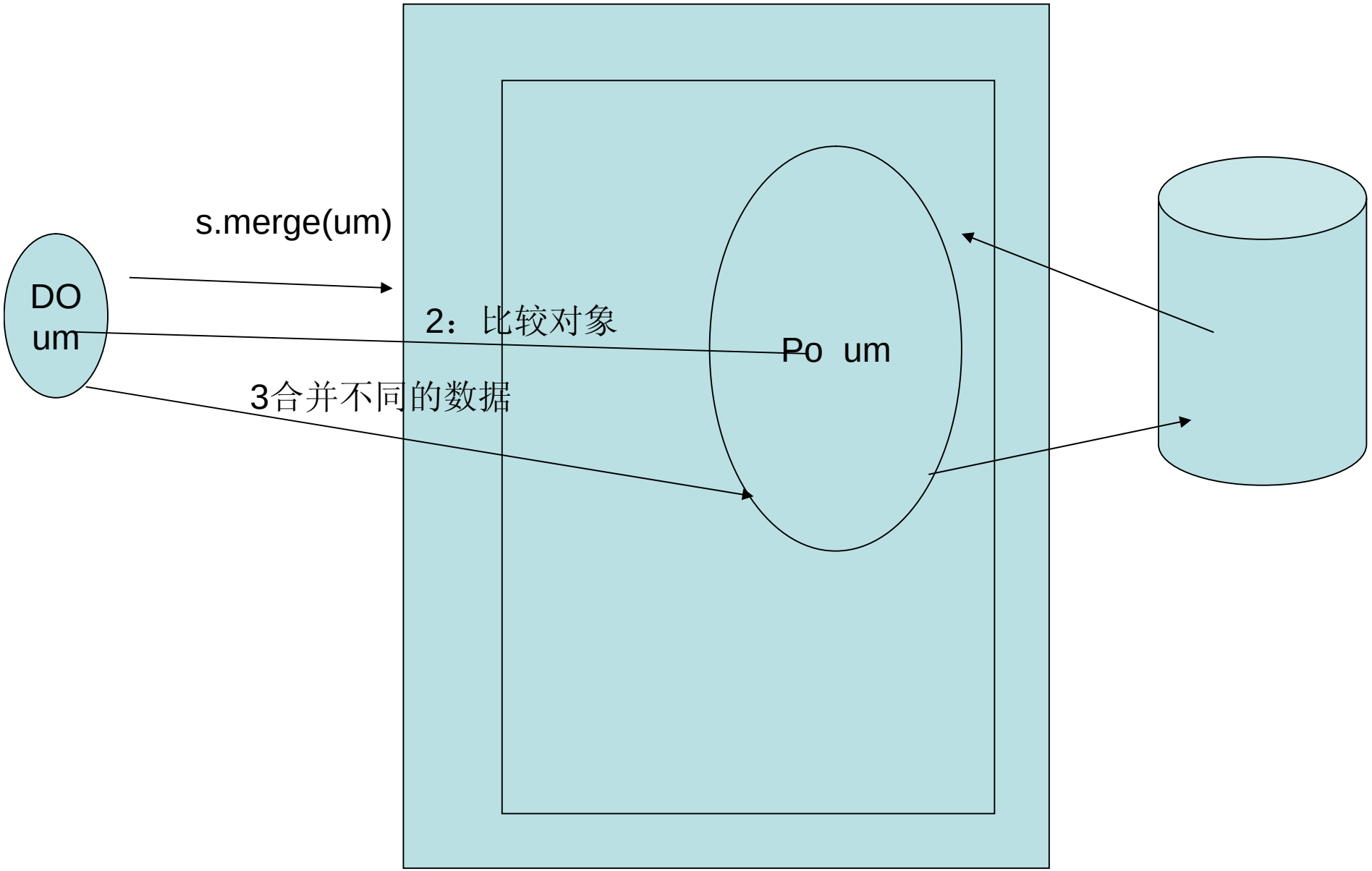
1：如果数据库里面存在你要修改的记录，update每次是直接执行修改语句；而merge是先在缓存中查找，缓存中没有相应数据，就到数据库去查询，然后再合并数据，如果数据是一样的，那么merge方法不会去做修改，如果数据有不一样的地方，merge才真正修改数据库。

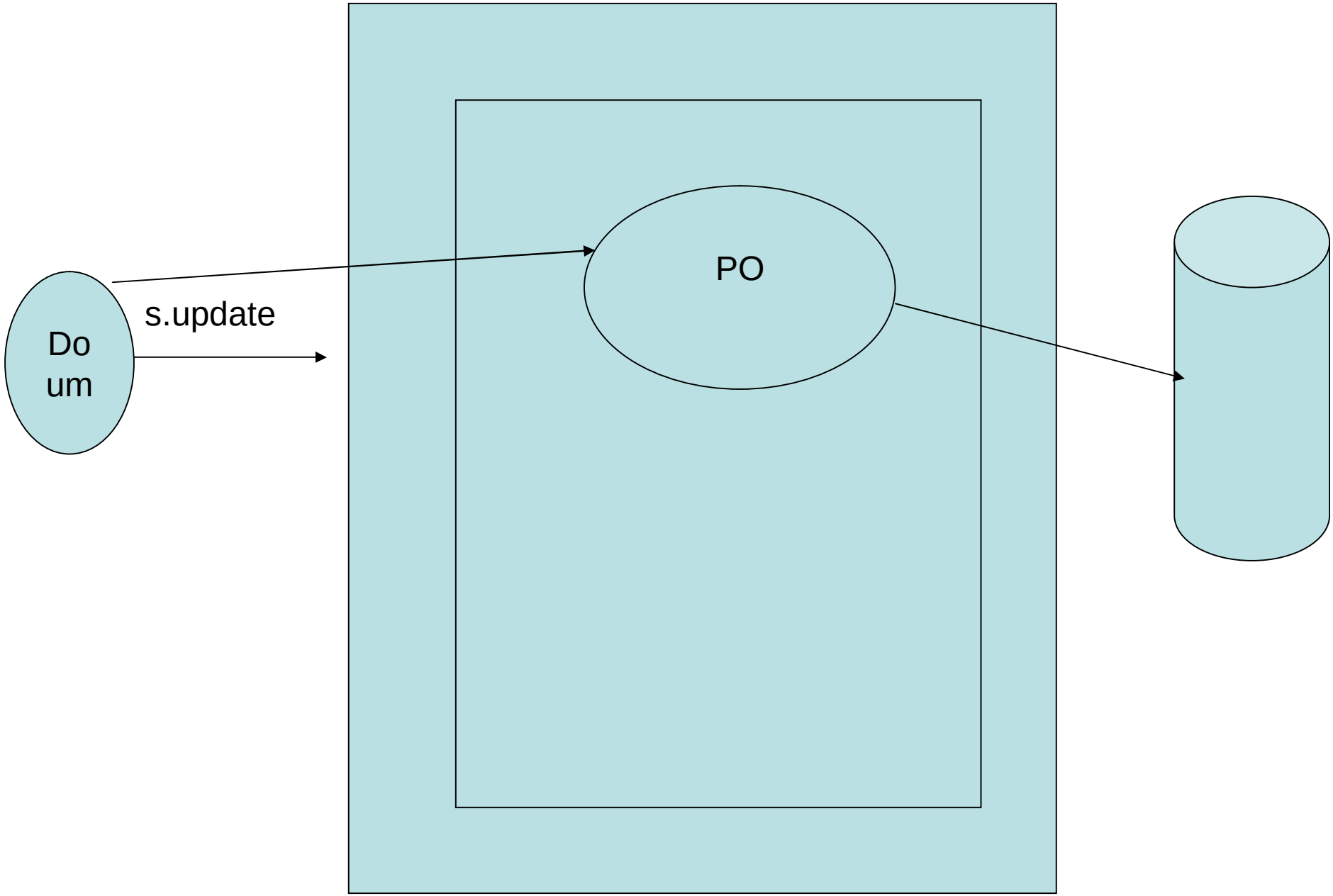
2：如果数据库中不存在你要修改的记录，update是报错；而merge方法是当作一条新增的值，向数据库中新增一条数据。

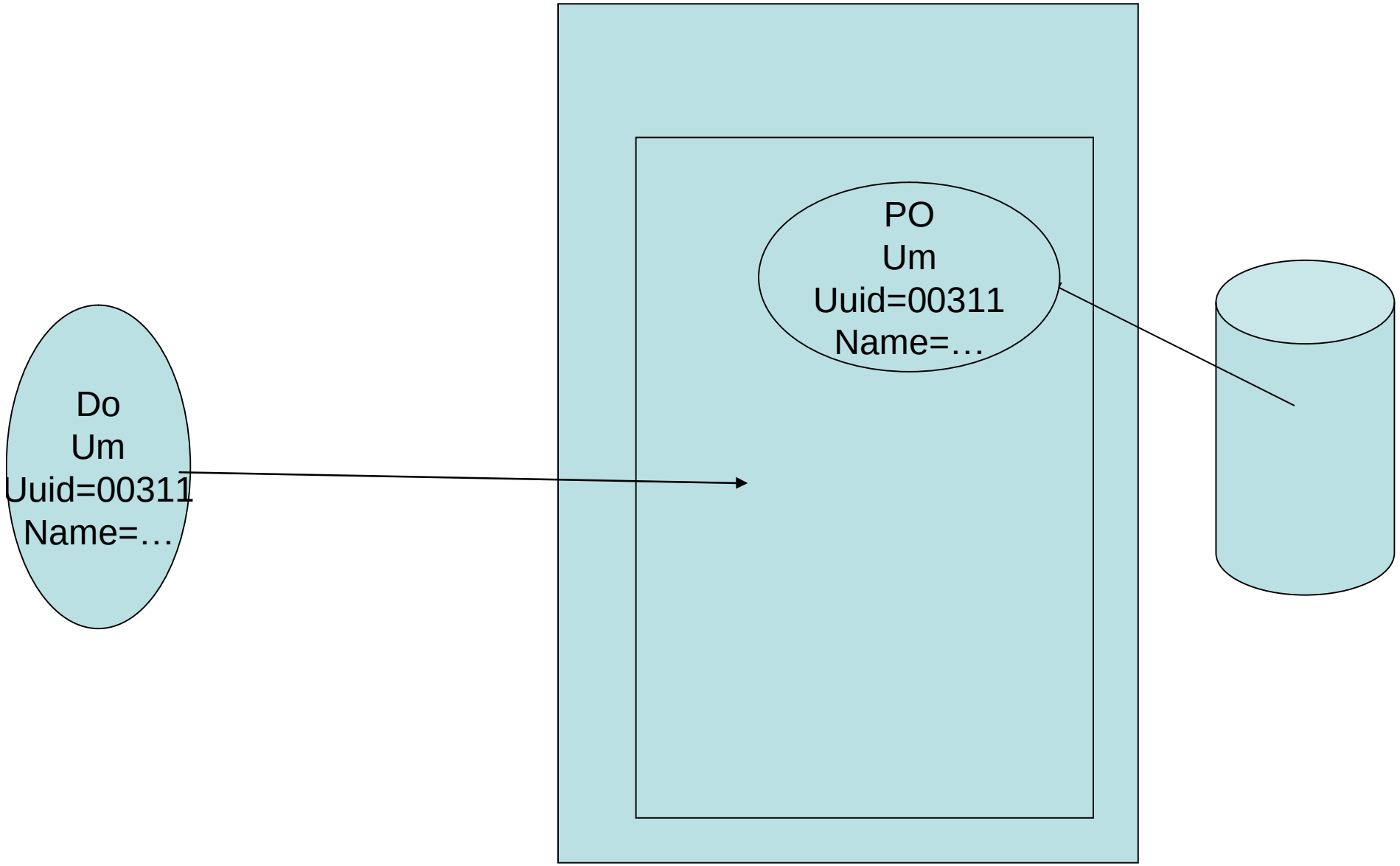
3：update后，传入的TO对象就是PO的了，而merge还是TO的。

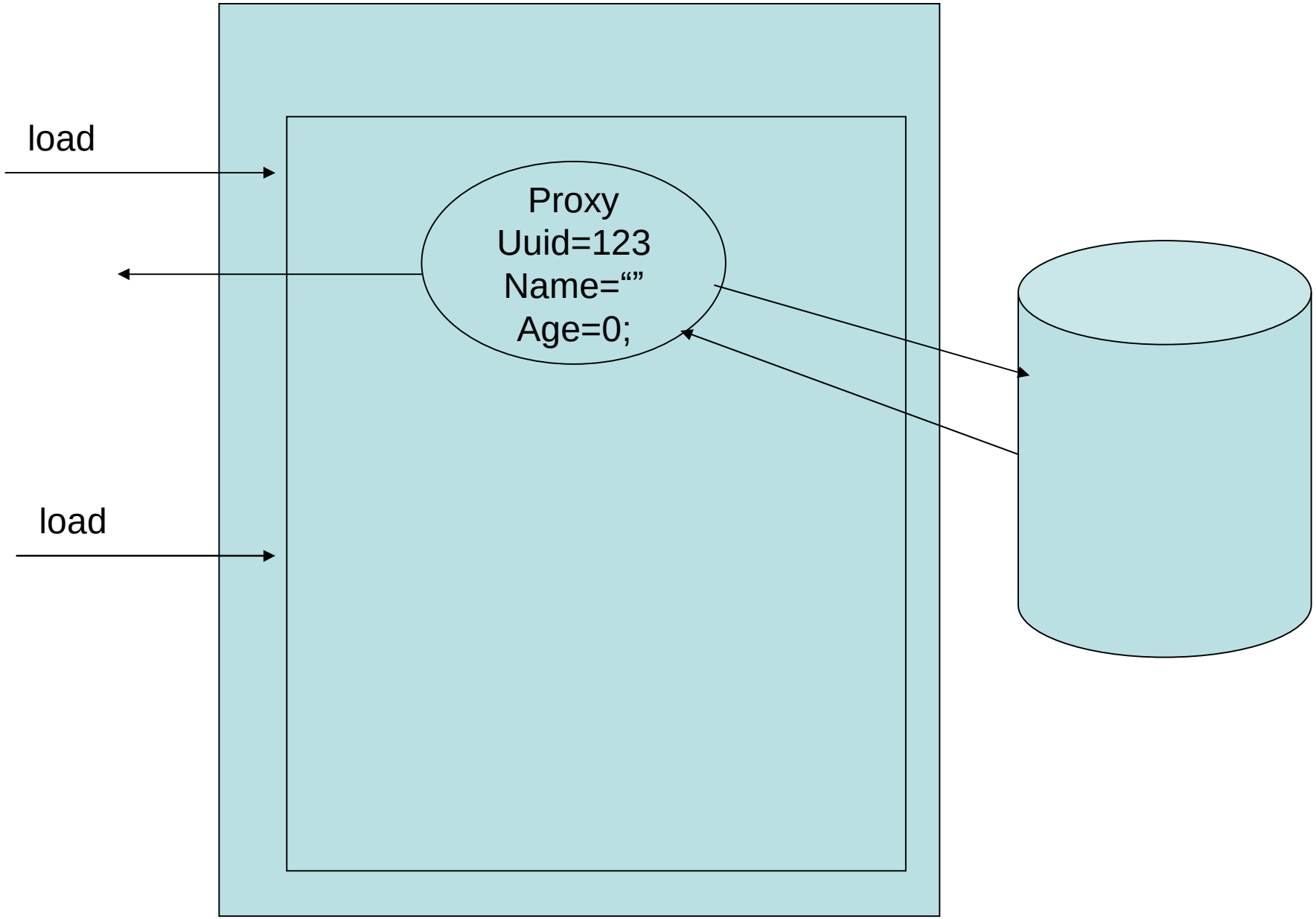
4：如果你确定当前session没有包含与之具有相同持久化标识的持久实例，使用update()。如果想随时合并改动而不考虑session的状态，使用 merge()。换句话说，在一个新 session 中通常第一个调用的是update()方法，以保证重新关联脱管对象的操作首先被执行。

5：请注意：使用update来把一个TO变成PO，那么不管是否修改了对象，都是要执行update sql语句的。


s.merge(um)
DO
um
2：比较对象
Po um
3合并不同的数据


Do
um
s.update
PO


Do
Um
Uuid=00311
Name=…
PO
Um
Uuid=00311
Name=…


load
Proxy
Uuid=123
Name=“”
Age=0;
load

Hibernate对象操作CRUD-3

- 通常下面的场景会使用 update() 或 saveOrUpdate()

1：程序在第一个 session 中加载对象
2：该对象被传递到表现层
3：对象发生了一些改动
4：该对象被返回到业务逻辑层
5：程序调用第二个session的update()方法持久这些改动

- saveOrUpdate方法做下面的事：

1：如果对象已经在本session中持久化了，不做任何事
2：如果另一个与本session关联的对象拥有相同的持久化标识，抛出一个异常
3：如果对象没有持久化标识属性，对其调用 save()
4：如果对象的持久标识表明其是一个新实例化的对象，对其调用 save()。
5：如果对象是附带版本信息的（通过 <version> 或 <timestamp>）并且版本属性的值表明其是一个新实例化的对象，save() 它。
6：否则update()这个对象

Hibernate对象操作CRUD-4

- merge做如下的事情

1：如果session中存在相同持久化标识的实例，用用户给出的对象的状态覆盖旧有的持久实例
2：如果session中没有相应的持久实例，则尝试从数据库中加载，或创建新的持久化实例
3：最后返回该持久实例
4：用户给出的这个对象没有被关联到 session 上，它依旧是脱管的

- 按主键查询

1：load方法：load的时候首先查询一级缓存，没有就创建并返回一个代理对象，等到使用的时候，才查二级缓存，如果二级缓存中没有数据就查数据库，如果数据库中没有，就抛例外
2：get方法：先查缓存，如果缓存中没有这条具体的数据，就查数据库，如果数据库没有值，就返回null，总之get方法不管用不用，都要拿到真实的数据

Hibernate对象操作CRUD-5

- Hibernate实现按条件查询的方式

1：最重要的按条件查询的方法是使用Query接口，使用HQL
2：本地查询（native sql）：就是使用标准的sql，也是通过Query接口来实现
3：按条件查询（Query By Criteria，QBC）：使用动态的，面向对象的方式来创建查询
4：按样例查询（Query By Example，简写QBE）：类似我们自己写的getByCondition
5：命名查询：在hbm.xml中配置hql语句，在程序里面通过名称来创建Query接口

- Query的list方法

一个查询通常在调用 list() 时被执行，执行结果会完全装载进内存中的一个集合，查询返回的对象处于持久状态。如果你知道的查询只会返回一个对象，可使用 list() 的快捷方式 uniqueResult()。

- Iterator和List

某些情况下，你可以使用iterate()方法得到更好的性能。 这通常是你预期返回的结果在session，或二级缓存(second-level cache)中已经存在时的情况。 如若不然，iterate()会比list()慢，而且可能简单查询也需要进行多次数据库访问：iterate()会首先使用1条语句得到所有对象的持久化标识(identifiers)，再根据持久化标识执行n条附加的select语句实例化实际的对象。

Hibernate对象操作CRUD-6

- 外置命名查询

可以在映射文件中定义命名查询(named queries)。

```
<query name="javass">
   <![CDATA[select Object(o) from UserModel o]]>
</query>
```

参数绑定及执行以编程方式完成：

```
List list = s.getNamedQuery("cn.javass.h3.hello.UserModel.javass").list();
```

注意要用全限定名加名称的方式进行访问

Hibernate对象操作CRUD-7

- flush方法

每间隔一段时间，Session会执行一些必需的SQL语句来把内存中对象的状态同步到JDBC连接中。这个过程被称为刷出(flush)，默认会在下面的时间点执行：

1：在某些查询执行之前

2：在调用org.hibernate.Transaction.commit()的时候

3：在调用Session.flush()的时候

涉及的 SQL 语句会按照下面的顺序发出执行：

1. 所有对实体进行插入的语句，其顺序按照对象执行save() 的时间顺序
2. 所有对实体进行更新的语句
3. 所有进行集合删除的语句
4. 所有对集合元素进行删除，更新或者插入的语句
5. 所有进行集合插入的语句
6. 所有对实体进行删除的语句，其顺序按照对象执行 delete() 的时间顺序

除非你明确地发出了flush()指令，关于Session何时会执行这些JDBC调用是完全无法保证的，只能保证它们执行的前后顺序。 当然，Hibernate保证，Query.list(..)绝对不会返回已经失效的数据，也不会返回错误数据。

Hibernate对象操作其它

- lock方法：也允许程序重新关联某个对象到一个新 session 上。不过，该脱管对象必须是没有修改过的。示例如：s.lock(um, LockMode.READ);
  注意：lock主要还是用在事务处理上，关联对象只是一个附带的功能

- 获取元数据
  Hibernate 提供了ClassMetadata接口和Type来访问元数据。示例如下：

```
ClassMetadata catMeta = sf.getClassMetadata(UserModel.class);
String[] propertyNames = catMeta.getPropertyNames();
Type[] propertyTypes = catMeta.getPropertyTypes();
for (int i = 0; i < propertyNames.length; i++) {
   System.out.println("name=="+propertyNames[i] + ", type== “
                      +propertyTypes[i]);
}
```

HQL-1

- HQL介绍

Hibernate配备了一种非常强大的查询语言，这种语言看上去很像SQL。但是不要被语法结构 上的相似所迷惑，HQL是非常有意识的被设计为完全面向对象的查询，它可以理解如继承、多态 和关联之类的概念。

看个示例，看看sql和HQL的相同与不同：

```
Sql：select    *      from tbl_user    where uuid= ‘123’
HQL：select Object(o) from UserModel o where o.uuid= ‘123’
```

- HQL特点

1：HQL对Java类和属性是大小写敏感的，对其他不是大小写敏感的。

2：基本上sql和HQL是可以转换的，因为按照Hibernate的实现原理，最终运行的还是sql，只不过是自动生成的而已。

3：HQL支持内连接和外连接

4：HQL支持使用聚集函数，如：count、avg、sum、min、max等

5：HQL支持order by 和 group by

6：HQL支持条件表达式，如：in、like、between等

HQL-2

- select子句

1：直接返回对象集合，形如：select o from UserModel o
2：返回某个特定类型的集合，形如：select o.name from UserModel o
3：返回Object[]，形如：select o.uuid,o.name from UserModel o
4：返回List，形如：select new List(o.uuid,o.name) from UserModel o
5：返回任意的对象，形如：select new cn.javass.h3.hello.A(o.uuid,o.name) from UserModel o ，这要求A对象有一个构造方法是传入这两个参数
6：返回Map类型，形如：select new Map(o.uuid as Id,o.name as N) from UserModel o ，返回的结果，以as后面的别名做map的key，对应的数据做值

- from子句

1：直接from对象，形如： from UserModel
2：可以分配别名，形如：from UserModel as um ， as关键字可以省略
3：如果from后面有多个对象，形如：from UserModel,DepModel ，相当于多表联合查询，返回他们的笛卡尔积

HQL-3

- 聚集函数

1：受支持的有avg,sum,min,max,count

2：关键字 distinct 与all 也可以使用，它们具有与 SQL 相同的语义，比如：

```
select count(distinct o.name) from UserModel o
```

- where子句

1：如果前面没有指派别名，那就直接使用属性名

2：如果指派了别名，必须使用别名.属性的方式

3：在where子句中允许使用的表达式包括大多数在 SQL中使用的表达式，包括：

(1)数学运算符 +，-，*，/

(2)二进制比较运算符 =，>=，<=，<>，!=，like

(3)逻辑运算符 and，or，not

(4)括号 ( )，表示分组

(5)in，not in，between，is null，is not null，is empty，is not empty，member of and not member of

(6)字符串连接符 ...||... or concat(...,...)

(7)current_date()，current_time()，and current_timestamp()

HQL-4

(8)second(...)、minute(...)、hour(...)、day(...)、month(...) 和 year(...)
(9)EJB-QL 3.0 定义的任何功能或操作符：substring(), trim(), lower(), upper(), length(),locate(), abs(), sqrt(), bit_length(), mod()
(10)coalesce() 和 nullif()
(11)str() 把数字或者时间值转换为可读的字符串
(12)cast(... as ...)，其第二个参数是某 Hibernate 类型的名字，以及 extract(... from ...)，只要 ANSI cast() 和 extract() 被底层数据库支持
(13)HQL index() 函数，作用于 join 的有序集合的别名。
(14)HQL 函数，把集合作为参数：size(), minelement(), maxelement(), minindex(), maxindex()，还有特别的 elements() 和 indices 函数，可以与数量词加以限定：some, all, exists, any, in。
(15)任何数据库支持的 SQL 标量函数，比如 sign(), trunc(), rtrim(), sin()
(16)JDBC 风格的参数传入 ?
(17)命名参数 :name，:start_date，:x1
(18)SQL 直接常量 'foo', 69, 6.66E+2, '1970-01-01 10:00:01.0'
(19)Java public static final 类型的常量 eg.Color.TABBY

HQL-5

- group by 子句

1：对于返回聚集值的查询，可以按照任何属性进行分组
2：可以使用having子句
3：sql中的聚集函数，可以出现在having子句中
4：group by 子句与 order by 子句中都不能包含算术表达式
5：不能group by 某个实体对象，必须明确的列出所有的聚集属性

- order by 子句

查询返回的列表（list）可以按照一个返回的类或组件（components）中的任何属性进行排序，可选的 asc 或 desc 关键字指明了按照升序或降序进行排序。

- 子查询

对于支持子查询的数据库，Hibernate 支持在查询中使用子查询。一个子查询必须被圆括号包围起来。

HQL-6

- 连接（join）

1：Hibernate可以在相关联的实体间使用join，类似于sql，支持inner join、left outer join、right outer join、full join（全连接，并不常用）。

2：inner join可以简写成join，left outer join 和right outer join在简写的时候可以把outer去掉。

- with

通过 HQL 的 with 关键字，你可以提供额外的 join 条件。

如：from Cat as cat left join cat.kittens as kitten with kitten.bodyWeight > 10.0

- fetch

可以要求立即返回关联的集合对象，如：

```
from Cat as cat
     inner join fetch cat.mate
     left join fetch cat.kittens
```

HQL-7

- 对于没有关联的实体，如何使用join呢？

  对于没有关联的实体，想要使用join，可以采用本地查询的方式，使用sql来实现，比如：

```
s.createSQLQuery("select um.*,dm.* from tbl_user2 um left join tbl_dep dm on um.age=dm.uuid")
.addEntity("um",UserModel.class).addEntity("dm",DepModel.class);
```

CRUD示例

1：新增

2：load、 get

3：修改

4：按条件查询

（1）传入条件值的方法：？或 :名称，索引从0开始

（2）给参数赋值

（3）返回对象

（4）返回多个属性，形成Object[]

（5）getByCondition的Hibernate版实现

5：删除

6：分页

```
Query q = sess.createQuery("from DomesticCat cat");
q.setFirstResult(20);
q.setMaxResults(10);
List cats = q.list();
```

本地查询-1

也可以使用你的数据库的Native SQL语言来查询数据。这对你在要使用数据库的某些特性的时候(比如说在查询提示或者Oracle中的 CONNECT关键字)，这是非常有用的。这就能够扫清你把原来直接使用SQL/JDBC 的程序迁移到基于 Hibernate应用的道路上的障碍。

- 使用SQLQuery

对原生SQL查询执行的控制是通过SQLQuery接口进行的，通过执行Session.createSQLQuery()获取这个接口。下面来描述如何使用这个API进行查询。

- 标量查询（Scalar queries）

```
s.createSQLQuery(“select uuid,name from tbl_user”).list();
```

它将返回一个Object[]组成的List，Hibernate会使用ResultSetMetadata来判定返回的标量值的实际顺序和类型。你也可以使用scalar来明确指明类型，如：

```
s.createSQLQuery("select * from tbl_user").addScalar("id",
    LongType.INSTANCE)
```

id字段就明确是long型，当然你也可以指定很多个字段的类型。

本地查询-2

- 实体查询(Entity queries)

上面的查询都是返回标量值的，也就是从resultset中返回的“裸”数据。下面展示如何通过addEntity()让原生查询返回实体对象。

(1) s.createSQLQuery("select * from tbl_user").addEntity(UserModel.class);

(2) s.createSQLQuery(“select uuid,userId from tbl_user2”).addEntity(UserModel.class); //一定要把表的所有字段罗列出来

(3) s.createSQLQuery("select {um}.uuid as {um.uuid},{um}.name as {um.name} from tbl_user {um}").addEntity("um",UserModel.class);

功能跟第二个差不多，也要把表的所有字段都罗列出来

(4)简单点的写法：s.createSQLQuery("select * from tbl_user2 um")
.addEntity("um",UserModel.class);

(5)添加条件的示例：

s.createSQLQuery("select * from tbl_user where uuid=? and name like ?").addEntity(UserModel.class).setString(0, "3").setString(1,"%na%");

本地查询-3

- 命名Sql查询

可以在映射文档中定义查询的名字,然后就可以象调用一个命名的 HQL 查询一样直接调用命名 SQL查询.在这种情况下，我们不 需要调用 addEntity() 方法。

在hbm.xml中配置，示例如下：

```
<sql-query name="users">
    <return alias="um" class="cn.javass.h3.hello.UserModel"/>
    select um.name as {um.name},
        um.age as {um.age},
        um.uuid as {um.uuid}
    from tbl_user um
    where um.name like :name
</sql-query>
```

注意：因为要返回一个对象，所以要把表的所有字段都罗列上，否则会报错“列名无效”，其实是在反射向对象赋值的时候，从sql的返回中得不到这个数据。

程序里面调用示例：

```
Query q = s.getNamedQuery(um.getClass().getName()
                          +".users").setString("name", "%n%");
```

本地查询-4

- 命名Sql查询--使用return-property

  使用 <return-property> 你可以明确的告诉 Hibernate 使用哪些字段别名，这取代了使用 {}-语法 来让 Hibernate 注入它自己的别名。

  在hbm.xml中配置，示例如下：

```
<sql-query name="users">
    <return alias="um" class="cn.javass.h3.hello.UserModel">
        <return-property name="name" column="umName"></return-property>
        <return-property name="uuid" column="uuid"></return-property>
        <return-property name="age" column="age"></return-property>
    </return>
    select um.name as umName,
    um.age as age,
    um.uuid as uuid
    from tbl_user um
    where um.name like :name
</sql-query>
```

条件查询-1

具有一个直观的、可扩展的条件查询API是Hibernate的特色。

- 创建一个Criteria 实例

  org.hibernate.Criteria接口表示特定持久类的一个查询。Session是 Criteria 实例的工厂。

```
Criteria crit = sess.createCriteria(Cat.class);
crit.setMaxResults(50);
List cats = crit.list();
```

- 限制结果集内容

  一个单独的查询条件是org.hibernate.criterion.Criterion 接口的一个实例org.hibernate.criterion.Restrictions类 定义了获得某些内置Criterion类型的工厂方法。

```
List list = s.createCriteria(UserModel.class)
        .add(Restrictions.eq("uuid", "3"))
        .add(Restrictions.like("name", "%n%"))
    .list();
```

约束可以按照逻辑分组，示例如下：

条件查询-2

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "Fritz%") )
    .add( Restrictions.or(
        Restrictions.eq( "age", new Integer(0) ),
        Restrictions.isNull("age")
          )
).list();
```

- 对结果集排序

可以使用 org.hibernate.criterion.Order 来为查询结果排序。示例如下：

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "F%")
    .addOrder( Order.asc("name") )
    .addOrder( Order.desc("age") )
.setMaxResults(50)
.list();
```

按样例查询

按样例查询Query By Example，功能类似我们自己写的getByCondition，但是它的功能不够完善和灵活，所以用途不是很大。

- 创建一个Eaxmple实例

```
Example exa = Example.create(p1); //传入的对象就是封装查询条件的对象
exa.enableLike();
exa.excludeNone();
exa.excludeZeroes();
```

比较常用的方法如下所示：

1.ignoreCase()：忽略模板类中所有String属性的大小写。

2.enableLike(MatchMode mode)：表示对模板类中所有的String属性进行like模糊匹配，enableLike()中的参数指明以何种方式进行匹配，比如MatchMode.ANYWHERE的方式是like“%变量%”。

3.excludeZeroes()：不把为0值的字段加入到where条件句中。

4.excludeProperty(String name)：不把属性为name的字段加入到where条件句中。

- 执行Eaxmple

```
List<Parent> list = s.createCriteria(Parent.class).add(exa).list();
```

批量处理-1

1：假如有如下程序，需要向数据库里面加如100000条数据：

```
Session session = sessionFactory.openSession();
 Transaction tx = session.beginTransaction();
 for ( int i=0; i<100000; i++ ) {
     Customer customer = new Customer(.....);
     session.save(customer);
}
tx.commit();
session.close();
```

这个程序很显然不能正常运行，会抛出内存溢出的例外。按照前面讲过的原理，Hibernate的save方法是先把数据放到内存里面，数据太多，导致内存溢出。

那么该如何解决呢？

批量处理-2

解决方案:

1：首先将 hibernate.jdbc.batch_size的批量抓取数量参数设置到一个合适值（比如，10 - 50 之间），同时最好关闭二级缓存，如果有的话。

2：批量插入，一个可行的方案如下：

```
for ( int i=0; i<100000; i++ ) {
    Customer customer = new Customer(.....);
    session.save(customer);
    if ( i % 20 == 0 ) {
        //将本批数据插入数据库，并释放内存
        session.flush();
        session.clear();
    }
}
```

批量更新的做法跟这个类似

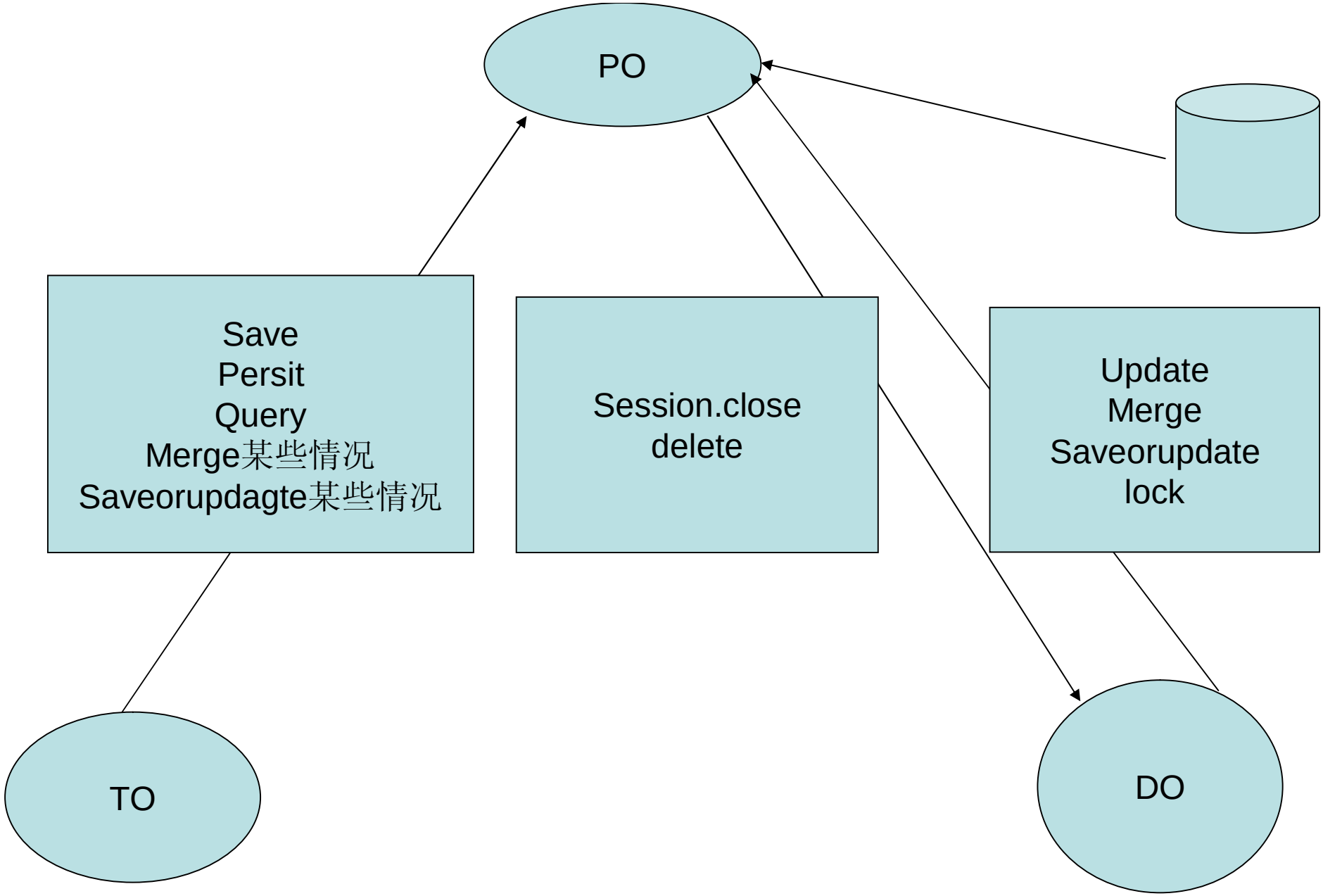
无状态Session接口

1：默认的Hibernate是有缓存的，称之为一级缓存。

2：也可以使用StatelessSession，来表示不实现一级缓存，也不和二级缓存和查询缓存交互

3：StatelessSession是低层的抽象，和底层JDBC相当接近

```
StatelessSession session = sessionFactory.openStatelessSession();
Transaction tx = session.beginTransaction();
ScrollableResults customers =
    session.getNamedQuery("GetCustomers") .scroll(ScrollMode.FORWARD_ONLY);
while ( customers.next() ) {
    Customer customer = (Customer) customers.get(0);
    customer.updateStuff(...);
    session.update(customer);
} tx.commit(); session.close();
```


PO
Save
Persit
Query
Merge某些情况
Saveorupdagte某些情况
Session.close
delete
Update
Merge
Saveorupdate
lock
TO
DO

本节课程小结

- Hibernate的基本开发
  掌握使用Hibernate来进行CRUD的各种方法

- 作业：
  1：复习和练习使用这些方法。
  2：从前面的项目里面，挑选一个模块，把DAO的实现改成使用Hibernate实现