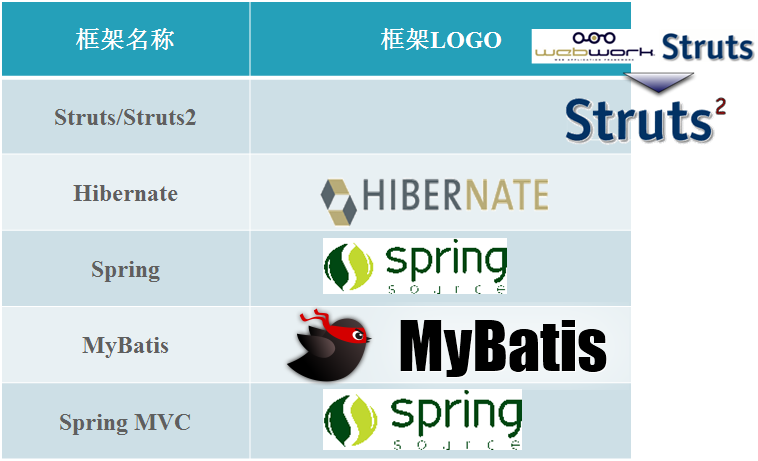
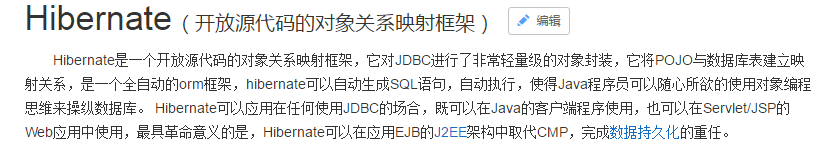
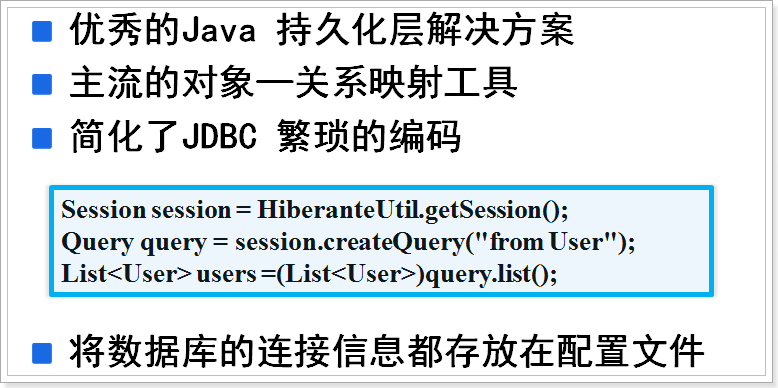
# JavaEE常用的开源框架



# Hibernate简介

Hibernate：冬眠





ORM: **O**bject **R**alationShip **M**apping 实体关系映射框架

table class

数据表-----> 实体类

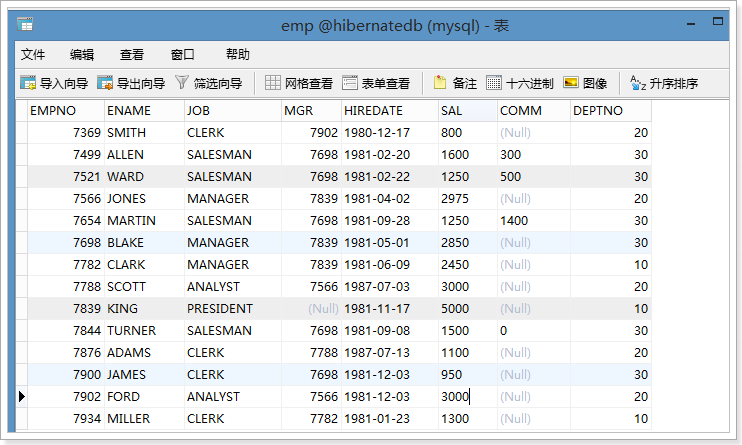
字段 ------> 属性

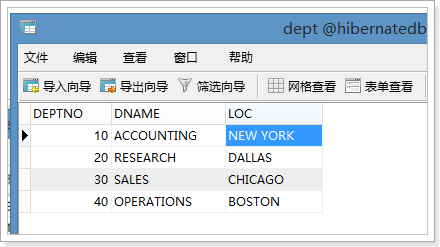
**Hibernate框架操作(增删改查 CRUD)数据库小菜一碟！！！ 全自动框架！！**

# Hibernate环境搭建

## 创建测试表

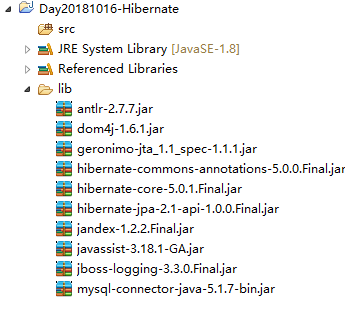
## 插入测试表





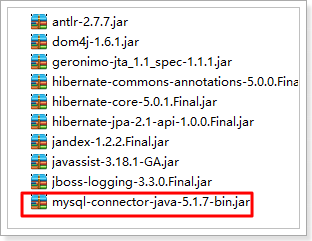
## 新建Java工程

* **传统的方式导入Jar包（不推荐！）**



* **Maven的方式导入Jar包（推荐！）**

## 传统方式导入Hibernate所需要的Jar

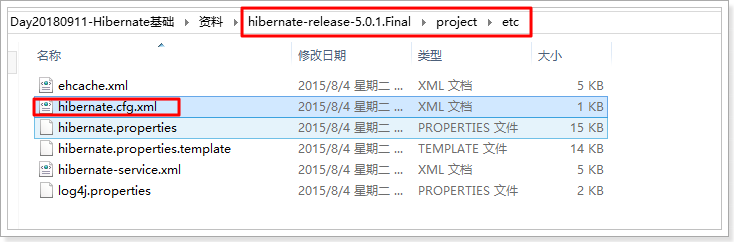


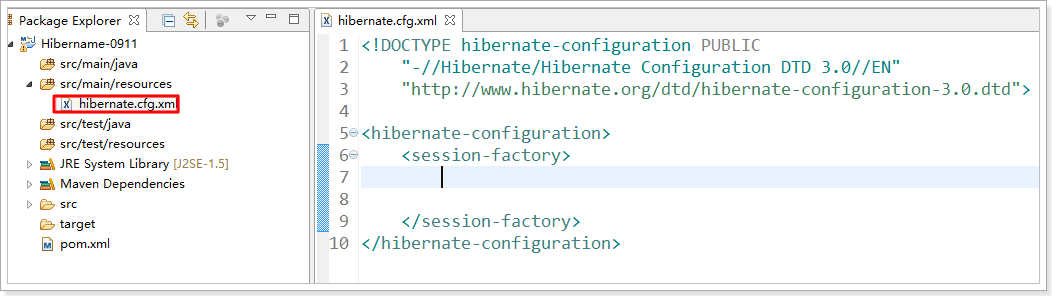
Pom.xml

|  |
| --- |
| <dependencies>  <!-- Hiberante核心包 -->  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-core</artifactId>  <version>5.0.1.Final</version>  </dependency>  <!-- mysql驱动包 -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.38</version>  </dependency>  </dependencies> |

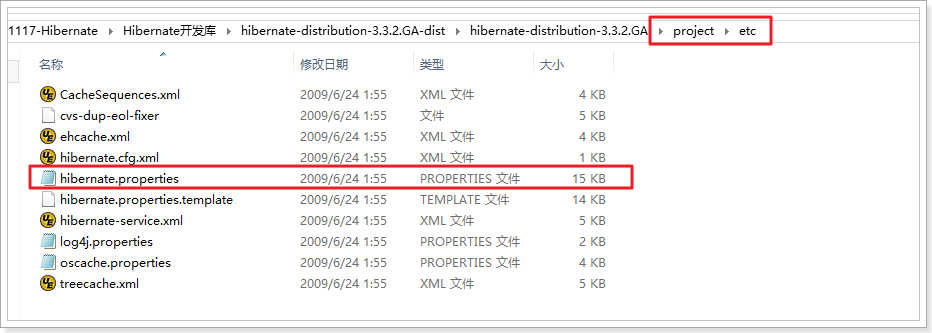
## 配置Hibernate的配置文件：hibernate.cfg.xml

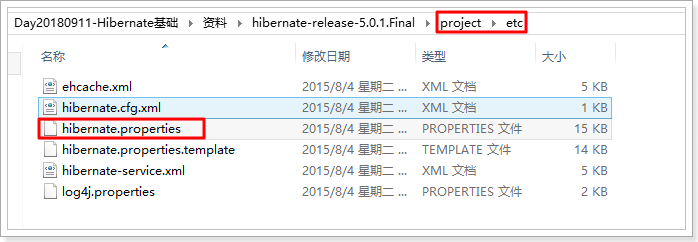
配置文件名字不能改！配置的是：Hibernate连接数据库的信息！





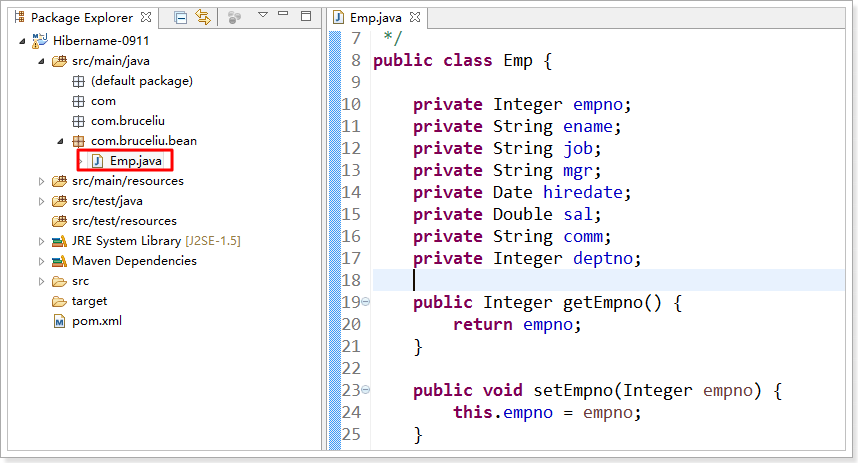
## 配置文件：hibernate.cfg.xml





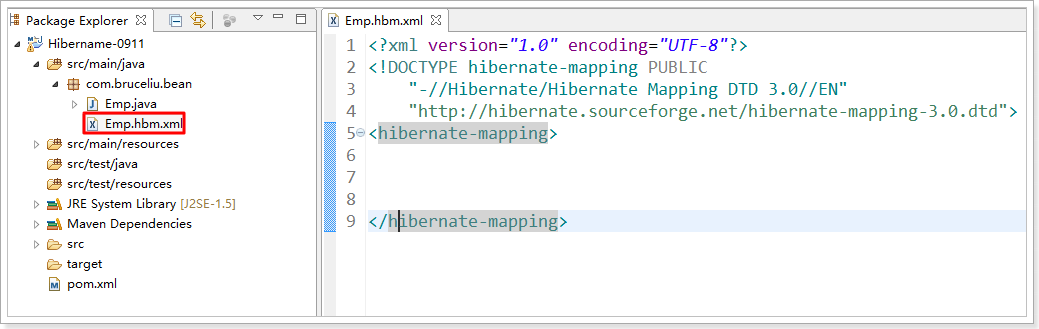
|  |
| --- |
| <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>    <!-- 1.hibernate的方言 告诉Hibernate连接何种数据库 -->  <property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>    <!-- 2.数据库连接字符串 -->  <property name=*"hibernate.connection.url"*>jdbc:mysql://192.168.132.128:3306/hibernatedb</property>    <!-- 3.数据库驱动类 -->  <property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>    <!-- 4.账号 -->  <property name=*"hibernate.connection.username"*>root</property>    <!-- 5.密码 -->  <property name=*"hibernate.connection.password"*>123</property>    </session-factory>  </hibernate-configuration> |

## 创建实体类



## 创建映射文件：Emp.hbm.xml

Hbm hibernate mapping 映射文件---> 告诉Hibernate 表和类之间的对应关系！



## 配置映射文件

|  |
| --- |
| <?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=*"com.bruceliu.bean.Emp"* table=*"EMP"*>    <id name=*"empno"* column=*"EMPNO"* type=*"java.lang.Integer"*>  <!-- 主键生成策略 native自增交给数据库处理-->  <generator class=*"native"*></generator>  </id>    <property name=*"ename"* column=*"ENAME"* type=*"java.lang.String"*/>  <property name=*"job"* column=*"JOB"* type=*"java.lang.String"*/>  <property name=*"mgr"* column=*"MGR"* type=*"java.lang.String"*/>  <property name=*"hiredate"* column=*"HIREDATE"* type=*"java.util.Date"*/>  <property name=*"sal"* column=*"SAL"* type=*"java.lang.Double"*/>  <property name=*"comm"* column=*"COMM"* type=*"java.lang.String"*/>  <property name=*"deptno"* column=*"DEPTNO"* type=*"java.lang.Integer"*/>    </class>  </hibernate-mapping> |

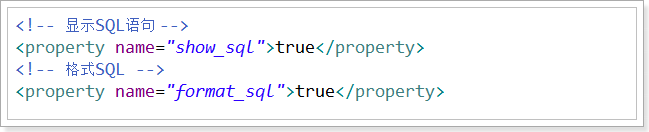


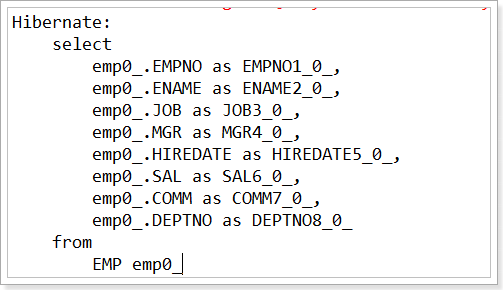
## 测试：

|  |
| --- |
| @Test  **public** **void** test1(){  //1.加载配置文件  Configuration configure = **new** Configuration().configure();  SessionFactory sessionFactory = configure.buildSessionFactory();  //3.获取session (与数据库的会话)  Session session = sessionFactory.openSession();  //4.获取一个查询接口实现  Query query = session.createQuery("from com.bruceliu.bean.Emp"); // HQL语句： Hibernate Query Lanauage  List<Emp> list = query.list();  **for** (Emp emp : list) {  System.***out***.println(emp);  }  session.close();  } |

## Hibernate自动生成SQL

* Hibernate在执行查询的时候 会自动生成对应数据库的SQL语句，默认SQL是不显示的！
* 如果想显示一下SQL语句，可以进行如下配置：

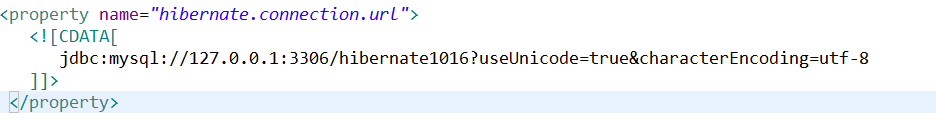




**注意：平时写项目的时候，可以显示SQL，方便调试程序！**

**关于配置XML中的特殊字符问题：**





# Hibernate基础使用

## 新增

* Hibernate执行增删改的时候需要开启事务支持，**Hibernate中的增删改要开启和提交事务！**

|  |
| --- |
| @Test  **public** **void** testAdd(){  Transaction tx = session.beginTransaction(); //开启事务  **try** {  Emp e=**new** Emp("猪八戒", "弼马温", 7369, **new** Date(), 2000.0, 200, 20);  Serializable id = session.save(e); //新增数据库,返回生成的主键值  System.***out***.println("新增成功,主键为："+id);  tx.commit();//提交事务  } **catch** (Exception e) {  tx.rollback();//回滚事务  e.printStackTrace();  }  } |

## 根据主键查询

* 方式1

|  |
| --- |
| @Test  **public** **void** testGetById(){  Query query = session.createQuery("from Emp where empno=?");  query.setParameter(0, 7936);  Emp e = (Emp) query.uniqueResult(); // 查询唯一的一条结果集 uniqueResult() 如果有多条将会报错！如果查询不到返回 null  System.***out***.println(e);  } |

* 方式2

|  |
| --- |
| @Test  **public** **void** testGetById2(){  Emp e = session.get(Emp.**class**, 7936); //根据主键查询  System.***out***.println(e);  }    @Test  **public** **void** testGetById3(){  Emp e = session.load(Emp.**class**, 7936); //根据主键查询  System.***out***.println(e);  } |

## 更新

|  |
| --- |
| @Test  **public** **void** testUpdate(){  Transaction tx = session.beginTransaction();  **try** {  Emp emp = session.get(Emp.**class**, 7936);  emp.setEname("猪无能");  emp.setDeptno(30);  emp.setJob("天棚元帅");  emp.setSal(5000.0);  session.update(emp); //更新  tx.commit();  System.***out***.println("更新成功~");  } **catch** (Exception e) {  tx.rollback();  e.printStackTrace();  }  } |

## 删除

* 方式1

|  |
| --- |
| @Test  **public** **void** testDelete(){  Transaction tx = session.beginTransaction();  **try** {  Query query = session.createQuery("delete from Emp where empno=?");  query.setParameter(0, 7936);  **int** count = query.executeUpdate();  System.***out***.println(count); //返回受影响的行数  tx.commit();  } **catch** (Exception e) {  tx.rollback();  e.printStackTrace();  }  } |

* 方式2

|  |
| --- |
| @Test  **public** **void** testDelete2(){  Transaction tx = session.beginTransaction();  **try** {  Emp e=**new** Emp();  e.setEmpno(7937);  session.delete(e); //根据主键查询  System.***out***.println("删除成功~~");  tx.commit();  } **catch** (Exception e) {  tx.rollback();  e.printStackTrace();  }  } |

# Hibernate查询

## 全查

* 方式1：

|  |
| --- |
| @Test  **public** **void** testQuery1(){  List<Emp> list = session.createQuery("from Emp").list(); //适合查询少量的数据  **for** (Emp emp : list) {  System.***out***.println(emp);  }  } |

* 方式2：

|  |
| --- |
| @Test  **public** **void** testQuery2(){  Iterator<Emp> it = session.createQuery("from Emp").iterate(); //性能高  **while**(it.hasNext()){  Emp emp = it.next();  System.***out***.println(emp);  }  } |

## 投影查询

* 查询表中的某几列

|  |
| --- |
| @Test  **public** **void** testQuery3(){  List<String> names = session.createQuery("select ename from Emp").list();  **for** (String s : names) {  System.***out***.println(s);  }  }    @Test  **public** **void** testQuery4(){  List<Object[]> values = session.createQuery("select ename,sal,hiredate from Emp").list();  **for** (Object[] o : values) {  System.***out***.println(o[0]+"---"+o[1]+"---"+o[2]);  }  }    @Test  **public** **void** testQuery5(){  // new Emp(ename,sal,hiredate) 调用Emp类的构造方法 让构造方法封装为对象  List<Emp> list = session.createQuery("select new Emp(ename,sal,hiredate) from Emp").list();  **for** (Emp e : list) {  System.***out***.println(e);  }  } |

## 聚合查询

* Max() 求最大值
* Min() 求最小值
* Avg() 求平均值
* Sum() 求综合
* Count() 求总数

|  |
| --- |
| /\*  \* 查询20号部门的最高薪资 最低薪资 平均薪资  \*/  @Test  **public** **void** testQuery6(){  Query query = session.createQuery("select max(sal),min(sal),avg(sal) from Emp where deptno=?");  query.setParameter(0, 20);  Object[] result = (Object[]) query.uniqueResult();  System.***out***.println(result[0]+"--"+result[1]+"--"+result[2]);  } |

## 分组查询

|  |
| --- |
| @Test  **public** **void** testQuery7(){  Query query=session.createQuery("select avg(sal),count(empno),deptno from Emp group by deptno having count(empno)>=5 order by deptno desc");  List<Object[]> list = query.list();  **for** (Object[] o : list) {  System.***out***.println(o[0]+"--"+o[1]+"--"+o[2]);  }  } |

## 分页查询

|  |
| --- |
| @Test  **public** **void** yesyQueryPage(){  //1.获取数据总量  Query query1 = session.createQuery("select count(empno) from Emp");  **int** totalCount = Integer.*parseInt*(query1.uniqueResult().toString());    //2.计算总页数  **int** pageIndex=5; //当前页面  **int** pageSize=3; //每页大小    **int** pageCount= (totalCount%pageSize==0)?(totalCount/pageSize):(totalCount/pageSize+1);    //3.每页数据  Query query2 = session.createQuery("from Emp");  query2.setFirstResult((pageIndex-1)\*pageSize); //设置开始查询的位置  query2.setMaxResults(pageSize); //设置每次查询的个数    List<Emp> list = query2.list();  **for** (Emp emp : list) {  System.***out***.println(emp);  }    System.***out***.println("当前页码:"+pageIndex);  System.***out***.println("每页大小:"+pageSize);  System.***out***.println("数据总量:"+totalCount);  System.***out***.println("总页数:"+pageCount);  } |

## 条数限制

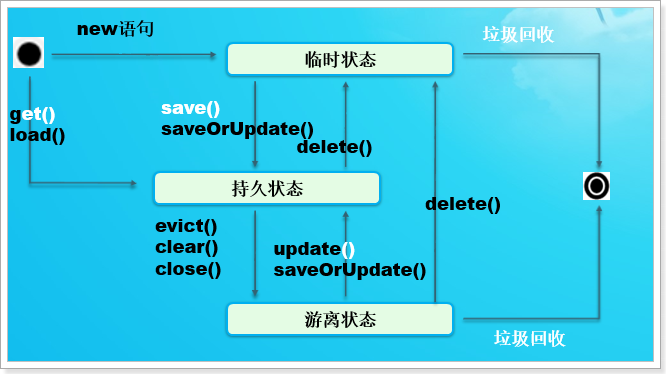
|  |
| --- |
| @Test  **public** **void** testQuery8(){  Query query = session.createQuery("from Emp order by sal desc");  query.setFirstResult(0);  query.setMaxResults(5);  List<Emp> list = query.list();  **for** (Emp emp : list) {  System.***out***.println(emp);  }  } |

# 6.Hibernate对象的三种状态（面试题）

**瞬时态**：一个Java对象创建之后，还没新增到数据库之前的状态

**持久态**：当调用save()或者是saveOrUpdate()方法之后的状态

**游离态**：当调用session的Close方法或者清空session之后对象的状态



# 7.Hibernate逆向生成表

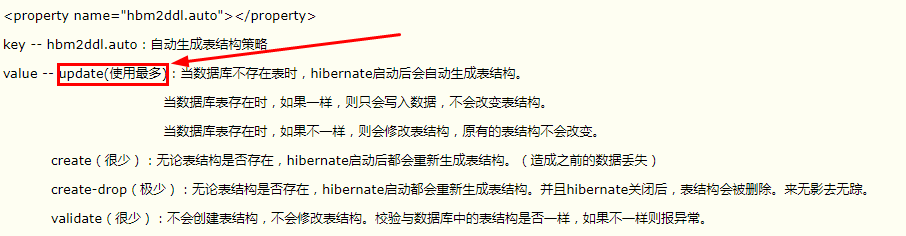
* 先有数据表---然后根据表配置 hbm.xml文件



* 先有hbm.xml文件然后根据配置文件自动生成表



但是配置文件中配置：





# 8.Hibernate类型

