MiniDao使用指南

作者: 张代浩

2013/08/16

目录

- 培训对象
  - 使用MiniDao进行开发的开发人员
- 培训目的
  - 使开发人员掌握通过MiniDao访问Mysql数据库的用法和技巧

目录

- 培训内容
  - MiniDao简介及特征
  - MiniDao的安装及基本概念
  - MiniDao的使用介绍
- 参考资料
  - Spring(IOC/AOP/JDBC)
  - Freemarker
  - Hibernate

MiniDao简介及特征

- MiniDao是Jeecg自己的持久化解决方案，集成Hibernate
- 实体维护和Mybaits SQL分离的两大优点。具有以下特征

  - O/R mapping不用设置xml，零配置便于维护
  - 不需要了解JDBC的知识
  - SQL语句和java代码的分离
  - 可以自动生成SQL语句
  - 接口和实现分离，不用写持久层代码，用户只需写接口，以及某些接口方法对应的SQL。它会通过AOP自动生成实现类
  - 支持自动事务处理和手动事务处理
  - 支持与hibernate轻量级无缝集成
  - MiniDao整合了Hibernate+mybatis的两大优势，支持实体维护和SQL分离
  - SQL支持脚本语言
  - Sql 性能优于Mybatis

※向下兼容Hibernate实体维护方式,实体的增删改查SQL自动生成

&Vs Mybatis

相同点：

- SQL语句和java代码的分离

不同点：

- O/R mapping不用设置xml，零配置，简单易用
- 可以自动生成SQL语句
- 接口和实现分离，不用写持久层代码，用户只需写接口，以及某些接口方法对应的SQL。它会通过AOP自动生成实现类
- 支持与hibernate轻量级无缝集成
- SQL支持更强大的脚本语言，可以写逻辑处理
- Sql 性能优于Mybatis
- Sql支持传递多个参数Map/Object/List/包装类型都可以
- Mybatis只支持一个参数<Map/Object>

SQL性能对比

（MiniDao SQL内容采用文件存储）

MiniDao Sql 耗时: 54 毫秒（SQL模板第一从文件读取，第二次从缓存读取）方法第一次执行的时候加载sql到缓存里
MiniDao Sql 耗时: 4 毫秒
MiniDao Sql 耗时: 4 毫秒
MiniDao Sql 耗时: 5 毫秒

（MiniDao SQL内容采用@Sql标签)

MiniDao Sql 耗时: 6 毫秒
MiniDao Sql 耗时: 1 毫秒
MiniDao Sql 耗时: 1 毫秒
MiniDao Sql 耗时: 2 毫秒

(Mybatis 在Sesion 初始化的 时候，加载Xml到缓存里，所以第一执行比MiniDao快)

Mybatis Sql 耗时: 18 毫秒　Mybatis Session初始化的时候，加载Xml到缓存里
Mybatis Sql 耗时: 6 毫秒
Mybatis Sql 耗时: 5 毫秒
Mybatis Sql 耗时: 9 毫秒

(Spring jdbc)

Springjdbc Sql 耗时: 10 毫秒
Springjdbc Sql 耗时: 1 毫秒
Springjdbc Sql 耗时: 1 毫秒
Springjdbc Sql 耗时: 1 毫秒

MiniDao支持SQL分离写法

- 第一步： EmployeeDao.java 接口定义(不需要实现)

```
@MiniDao
public interface EmployeeDao {

    @Arguments("employee")
    public List<Map> getAllEmployees(Employee employee);

    @Arguments("empno")
    Employee getEmployee(String empno);

    @Arguments({"empno","name"})
    Map getMap(String empno,String name);

    @Sql("SELECT count(*) FROM employee")
    Integer getCount();

    @Arguments("employee")
    int update(Employee employee);

    @Arguments("employee")
    void insert(Employee employee);
}
```

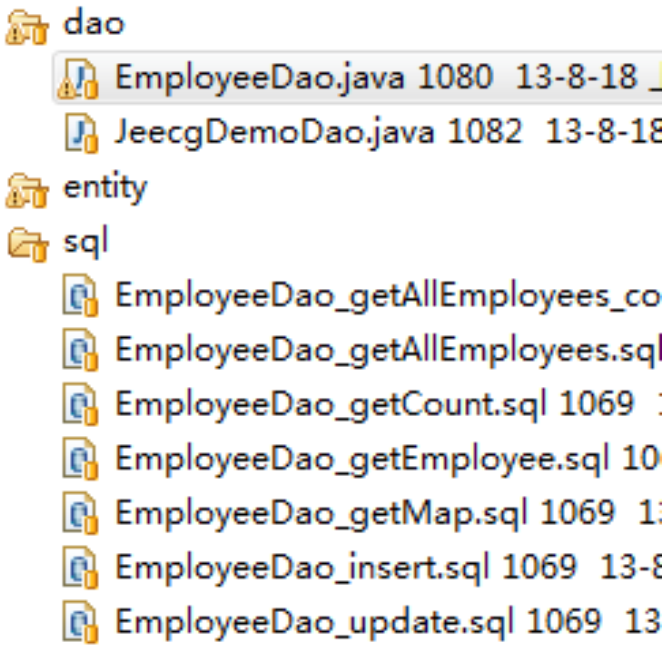
MiniDao支持SQL分离写法

- 第二步：接口方法对应SQL文件创建

Sql文件定位到dao接口的方法，dao接口的每个方法对应一个sql文件
SQL文件命名规则：{Dao接口名} _{方法名}.sql

简单SQL，也可以采用方法注释标签

```
@Sql("SELECT count(*) FROM employee")
    Integer getCount();
```



MiniDao支持SQL分离写法

- 第三步：SQL文件

SQL文件采用模板语言Freemarker语法，可以灵活运用，甚至可以写脚本语言，宏处理等；

示例：

```
SELECT * FROM employee where 1=1
<#if employee.age ?exists>
        and age = '${employee.age}'
</#if>
<#if employee.name ?exists>
        and name = '${employee.name}'
</#if>
<#if employee.empno ?exists>
        and empno = '${employee.empno}'
</#if>
```

MiniDao支持实体方式维护

- 第一步：自定义接口继承MiniDaoSupportHiber接口

- 示例：
- public interface JeecgDemoDao extends MiniDaoSupportHiber<JeecgDemo>{

说明：JeecgDemo：持久化对象

JeecgDemoDao 用户自定义接口

SQL参数传递两种方式

方式一: 支持采用占位符，格式字段前加冒号【: 字段名】

- **优点:**
- 防止sql注入；sql执行计划只解析一次；字段值根据类型自动转换不需要手工处理

- **缺点:**只能传参数原生态值；参数为List情况循环体不适用
- **示例：**

```
SELECT * FROM employee where 1=1
<#if employee.age ?exists>
and age = :employee.age
</#if>
<#if employee.name ?exists>
and name = :employee.name
</#if>
<#if employee.empno ?exists>
and empno = :employee.empno
</#if>
```

SQL参数传递两种方式

方式二:模板语言方式，格式【${字段名}】

- **缺点:**

  Sql直接拼装，有SQL注入风险；参数值需根据类型手工转换；

- **优点:**
- 可以对参数值进行脚本处理；参数为List对象，循环体对象必须用该方式；
- （用户体验没有变化，直接将${}改为: 即可）
- 特点:支持多参数，支持参数多层，参数为list必须采用模板语言方式
- 示例:

```
SELECT * FROM employee where 1=1
<#if employee.age ?exists>
        and age = '${employee.age}'
</#if>
<#if employee.name ?exists>
        and name = '${employee.name}'
</#if>
<#if employee.empno ?exists>
        and empno = '${employee.empno}'
</#if>
```

MiniDao支持SQL分离写法

- 第四步：@Arguments 注释标签讲解

```
/**
*(SQL模板参数名)
* 1. [注释标签参数]必须和[方法参数]，保持顺序一致
* 2. [注释标签参数]的参数数目不能大于[方法参数]的参数数目
* 3. 只有在[注释标签参数]标注的参数，才会传递到SQL模板里
* 4. 如果[方法参数]只有一个，如果用户不设置 [注释标签参数]，则默认参数名为miniDto
* @date 20130817
* @version V1.0
*/
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Arguments {
    String[] value() default {};
}
```

- 示例：

```
@Arguments({"empno","name"})
Map getMap(String empno,String name);
```

MiniDao支持实体方式维护

- 第二步：JavaBean实体定义，采用JPA方式进行注解
  ※ 同Hibernate实体配置一样，支持一对多，多对多等复杂配置关系

- 示例：

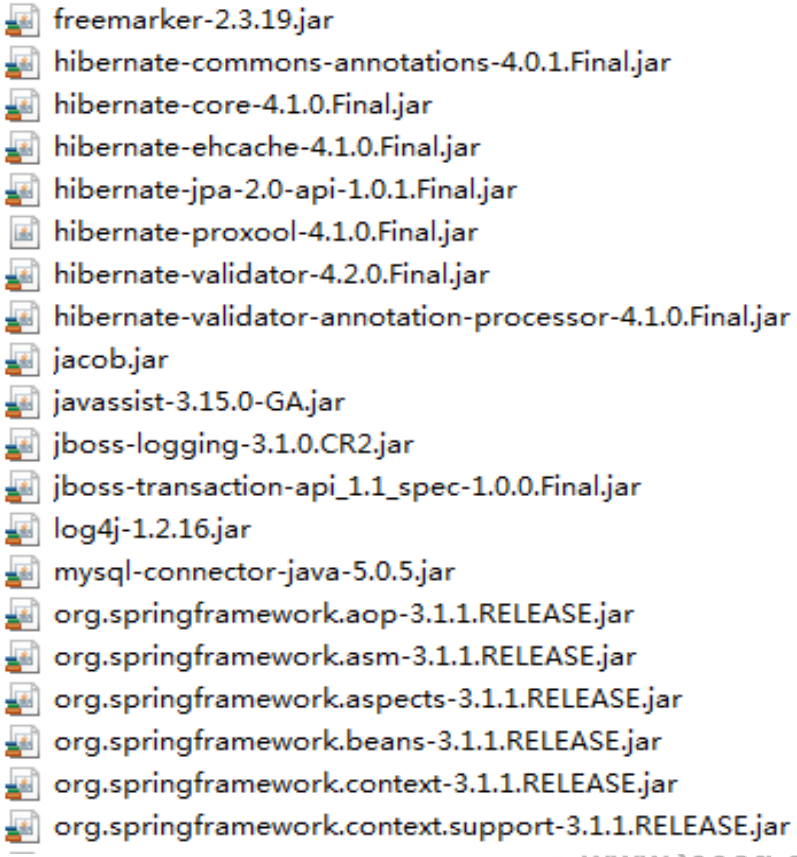
```
@Entity
@Table(name = "jeecg_demo")
@Inheritance(strategy = InheritanceType.JOINED)
 public class JeecgDemo extends IdEntity implements java.io.Serializable {
        private static final long serialVersionUID = 1L;

        @Id
        @GeneratedValue(generator = "hibernate-uuid")
        @GenericGenerator(name = "hibernate-uuid", strategy = "uuid")
        private String id;
        /**手机号码*/
        @Column(name ="mobilePhone",nullable=true)
        private java.lang.String mobilePhone;
        /**办公电话*/
        @Column(name ="officePhone",nullable=true)
        private java.lang.String officePhone;
        /**电子邮箱*/
        @Column(name ="email",nullable=true)
        private java.lang.String email;
```

MiniDao的安装及基本概念

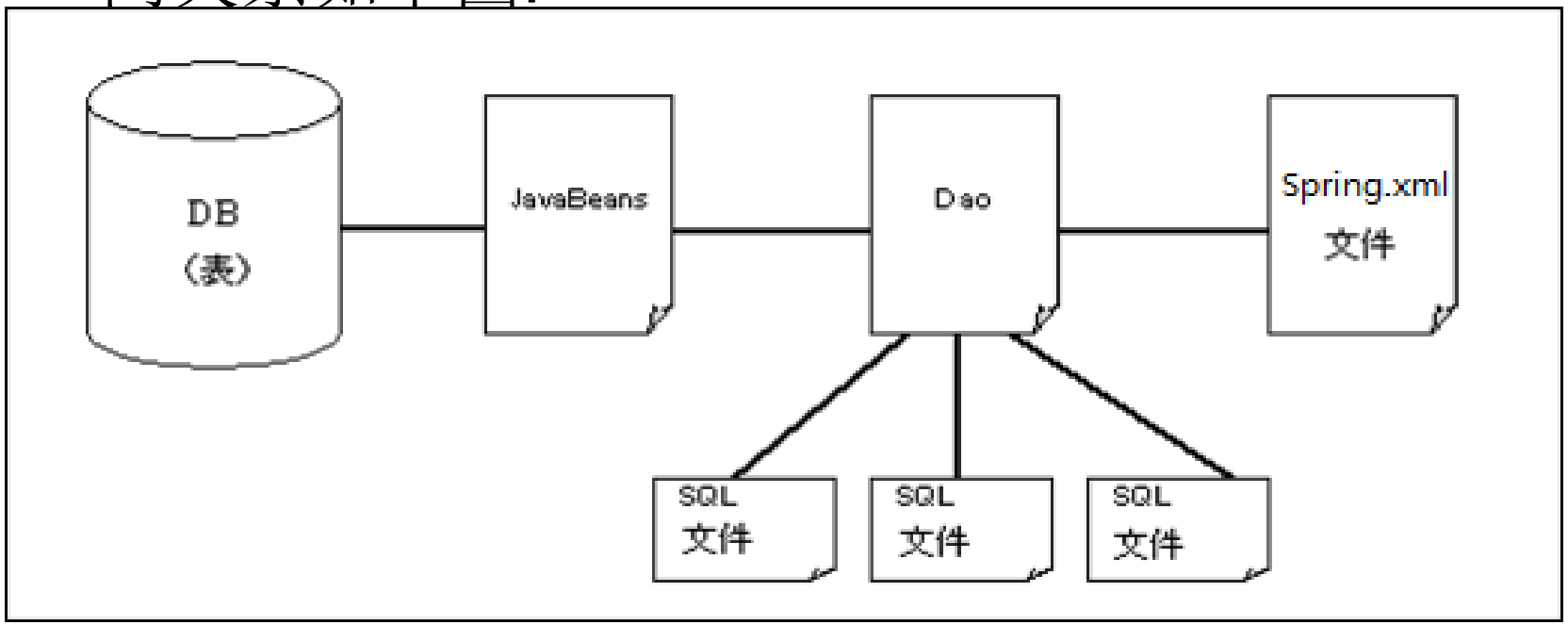
- MiniDao的安装
  - 与Jeecg同样，MiniDao需要JDK1.5以上的系统环境
  - 需要引入必要的lib文件
  - 引入必要的配置文件

  Spring.xml，log4j.properties
  - 使用MiniDao时必须作成的文件：JavaBeans、Dao(.java)、SQL文件(.sql)



MiniDao的安装及基本概念(2)

- MiniDao的基本概念
  - 使用MiniDao功能时，作成的JavaBeans，Dao(.java)，spring.xml文件，SQL文件(.sql)之间关系如下图:



MiniDao的安装及基本概念(2)

MiniDao配置文件

```
<!-- MiniDao动态代理类 -->
<bean id="miniDaoHandler"
class="org.jeecgframework.minidao.aop.MiniDaoHandler">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>

<!-- 注册MiniDao接口 -->
<bean class="org.jeecgframework.minidao.factory.MiniDaoBeanFactory">
<property name="packagesToScan">
<list>
<value>examples.dao.*</value>
</list>
</property>
</bean>
```

MiniDao的安装及基本概念(3)

- MiniDao的基本概念
  - JavaBeans，采用JPA注解注解方式：

    JavaBeans用来和表进行关联。 为了将JavaBeans和表进行关联，必须进行以下的常量声明和方法的实装:
  - 和表关联的常量声明 (TABLE注释)
  - 和列项关联的常量声明 (COLUMN注释)
  - 和其他的表结合时指定为键(key)的常量声明(N:1映射)
  - getter/setter方法的实装

MiniDao的安装及基本概念(4)

- MiniDao的基本概念
  - Dao(Data Access Object):

    Dao作为接口而作成。Dao本来的目的，就是通过把持久化的数据和处理逻辑相分离，来维持Bean的持久化。 Dao和JavaBeans的关系是1:1的关系,也即，有一个JavaBeans，就要作成一个Dao。 通过调用Dao的方法(method)，来执行与方法(method)相对应的SQL文件中的SQL指令。 在作成Dao的时候，必须注意以下几点:

  - 与JavaBeans关联的常量声明(BEAN注释)
  - 方法(method)的定义

MiniDao的安装及基本概念(5)

- MiniDao的基本概念
  - SQL文件:

    SQL文件里记述SQL检索，更新等指令。 一旦调用Dao里定义的方法(method)，就可以执行对应的SQL文件中记述的SQL指令。 请将作成的SQL文件与Dao放在同一个命名空间下。

MiniDao的安装及基本概念(6)

- MiniDao的基本概念
  - Spring.xml文件:

    在xml文件进行Dao配置，把Dao作为组件(component)注册到Spring容器(container)中。要使用Dao功能，对已注册的Dao，必须进行AOP的应用。Dao实体配置文件部分内容如下所示：

```
<!-- 注册JeecgDemoDao接口 -->
    <bean id="jeecgDemoDao" class="org.springframework.aop.framework.ProxyFactoryBean">
      <property name="proxyInterfaces" value="examples.dao.JeecgDemoDao" />
      <property name="interceptorNames">
                <list>
                        <value>miniDaoHandler</value>
                </list>
      </property>
    </bean>
```

MiniDao的安装及基本概念(7)

- MiniDao的基本概念
  - MiniDao的执行:

执行Dao的基本方法如下所示:

1. 以spring.xml文件中配置需要管理的Dao接口,将Dao注册进Spring容器中
2. 从Spring容器中调用getBean，取得已注册的Dao
3. 执行所得到的Dao的方法(method)

MiniDao的安装及基本概念(8)

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import examples.dao.JeecgDemoDao;
import examples.entity.JeecgDemo;

public class Client {
        public static void main(String args[]) {
            BeanFactory factory = new ClassPathXmlApplicationContext(
                                    "applicationContext.xml");
            JeecgDemoDao jeecgDemoDao = (JeecgDemoDao)
        factory.getBean("jeecgDemoDao");
            JeecgDemo entity = new JeecgDemo();
            entity.setId("402880e7408c7c5001408c7c52400000");
            entity.setAge(20999);
            entity.setUserName("zhangdaihao 12121");
            JeecgDemo s =
        jeecgDemoDao.getByHiber("402880e7408c9b1601408c9b179a0000");
            System.out.println(s.getUserName());
        }
}
```

MiniDao的使用介绍

- JavaBeans
  - 和表关联的常量声明（TABLE注释）

要和表进行关联，使用TABLE注释。 TABLE注释使用以下的形式进行常量声明：

```
@Table(name = "jeecg_demo")
```

这也可以用于定义schema。schema名为“SCOTT”的场合，声明如下所示：

```
@Table(name = "jeecg_demo",schema="scott")
```

MiniDao的使用介绍(2)

- JavaBeans
  - 和列项关联的常量声明 (COLUMN注释)

要和表的列项进行关联，使用COLUMN注释。

```
@Column(name="USER_NAME",nullable=false)
  public java.lang.String getUserName(){
    return this.userName;
  }
```

MiniDao的使用介绍(3)

- JavaBeans
  - 和其他的表结合时指定为键(key)的常量声明 (N:1映射)
  - @ManyToOne(fetch=FetchType,cascade=CascadeType)
    可选
    @ManyToOne表示一个多对一的映射,该注解标注的属性通常是数据库表的外键
    optional:是否允许该字段为null,该属性应该根据数据库表的外键约束来确定,默认为true
    fetch:表示抓取策略,默认为FetchType.EAGER
    cascade:表示默认的级联操作策略,可以指定为ALL,PERSIST,MERGE,REFRESH和REMOVE中的若干组合,默认为无级联操作
    targetEntity:表示该属性关联的实体类型.该属性通常不必指定,ORM框架根据属性类型自动判断targetEntity.
    示例:

```
//订单Order和用户User是一个ManyToOne的关系
//在Order类中定义
@ManyToOne()
@JoinColumn(name="USER")
public User getUser() {
   return user;
}
```

MiniDao的使用介绍(4)

- JavaBeans
  - 对应于列项(column)的实例(instance)变量声明

列项的值用Bean的实例变量表示。实例变量的声明方式有两种。

- 作为JavaBeans的属性的声明方式

getter method

public 类型 get属性名()

setter method

public void set属性名(参数)

- 作为public字段的声明方式

MiniDao的使用介绍(5)

- Dao
  - 方法(method)的定义

虽然通过调用Dao里定义的方法(method)，可以执行相应的SQL文件中记述的SQL指令，但是在更新处理和检索处理中需要遵循各自的方法命名规约。方法(method)名必须以右表中列出的单词开头

处理	名称
插入	insert,add,create
更新	update,modify,store
删除	delete,remove
批处理	batch
检索	以上各单词之外

MiniDao的使用介绍(6)

- Dao
  - INSERT处理

进行INSERT处理的方法名，必须以insert,add,create开头。 返回值可以指定为void或者int。 int的场合，返回值为更新的行数。参数类型与实体(Entity)的类型要一致。

- public void insert(Department department);
- public int addDept(Department department);
- public void createDept(Department department);

MiniDao的使用介绍(7)

- Dao
  - UPDATE处理

进行UPDATE处理的方法名，必须以update,modify,store开头。 返回值可以指定为void或者int。int的场合，返回值为更新的行数。参数类型与实体(Entity)的类型要一致。

- public int update(Department department);
- public int modifyDept(Department department);
- public void storeDept(Department department);

MiniDao的使用介绍(10)

- Dao
  - DELETE处理

进行DELETE处理的方法名，必须以delete,remove开头。 返回值可以为void或者int类型。 int的场合，返回值为更新的行数。参数类型与实体(Entity)的类型要一致。

– public void delete(Department department);
– public int removeDept(Department department);

MiniDao的使用介绍(11)

- Dao
  - 检索(SELECT)处理

进行检索处理的场合，要指定返回值的类型。返回值的类型是java.util.List的实装的场合，SELECT指令将返回实体(Entity)的列表(List)。返回值是实体(Entity)型的数组(array)的场合，返回实体数组(Entity array)。返回值的类型是实体(Entity)的场合，将返回实体(Entity)。

– public List selectList(int deptno);
– public Department[] selectArray(int deptno);

MiniDao的使用介绍(12)

- Dao
  - 检索(SELECT)处理

除了实体(Entity)以外,还可以利用DTO或者Map作为检索处理的返回值。 返回值为DTO类型的列表(List<Dto>)的场合，将返回DTO的列表(List)。 返回值为DTO类型的数组(Dto[])的场合，将返回DTO的数组(array)。 返回值为Map类型的列表(List<Map>)的场合，将返回Map的列表(List)。 返回值为Map类型的数组(Map[])的场合，将返回Map的数组(array)。

- public List<EmpDto> selectAsDtoList(int deptno);
- public EmpDto[] selectAsDtoArray(int deptno);
- public List<Map> selectAsMapList(int deptno);
- public Map[] selectAsMapArray(int deptno);

MiniDao的使用介绍(13)

- Dao
  - 检索(SELECT)处理

除此以外的场合，MiniDao还想定了这样一种情况，也即，像SELECT count(*) FROM emp这样的指令，返回值为1行只有一个列项值的情况。

– public int selectCountAll();

MiniDao的使用介绍(14)

- Dao
  - @Arguments注释标签

使用@Arguments注释指定方法(method)的参数名，这样就可以在SQL指令中引用方法(method)的参数。

```
@Arguments({"empno","name"})
Map getMap(String empno,String name);
```

方法(method)的参数与表的列名相对应的场合，在参数名中指定表的列名。

例如:方法(method)的参数名是empno，表的列名是employeeno的场合，就指定为employeeno。

如果是有复数个参数的场合，则用逗号分隔。

MiniDao的使用介绍(15)

- SQL文件(支持Freemarker语法)
  - IF注解(comment)

使用IF注解，可以根据相应的条件改变要执行的SQL指令。IF注解的记法如下:

```
<#if condition>...
<#elseif condition2>...
<#elseif condition3>.....
<#else>..
```

例：

```
<#if employee.empno ?exists>
    and empno = '${employee.empno}'
</#if>
```

作为IF注解的条件为假的处理部分，使用ELSEIF注解。 条件为假的场合，使用<#else>..之后的部分

MiniDao的使用介绍(16)

- MiniDao实体Bean调用方法

```
BeanFactory factory = new ClassPathXmlApplicationContext(
                        "applicationContext.xml");

EmployeeDao employeeDao = (EmployeeDao)
factory.getBean("employeeDao");
Employee employee = new Employee();
List<Map> list =  employeeDao.getAllEmployees(employee);
for(Map mp:list){
        System.out.println(mp.get("id"));
        System.out.println(mp.get("name"));
        System.out.println(mp.get("empno"));
        System.out.println(mp.get("age"));
        System.out.println(mp.get("birthday"));
        System.out.println(mp.get("salary"));
}
```

MiniDao的使用介绍(17)

- MiniDao测试分离SQL

```
BeanFactory factory = new ClassPathXmlApplicationContext(
                        "applicationContext.xml");

EmployeeDao employeeDao = (EmployeeDao)
factory.getBean("employeeDao");
Employee employee = new Employee();
List<Map> list =  employeeDao.getAllEmployees(employee);
for(Map mp:list){
        System.out.println(mp.get("id"));
        System.out.println(mp.get("name"));
        System.out.println(mp.get("empno"));
        System.out.println(mp.get("age"));
        System.out.println(mp.get("birthday"));
        System.out.println(mp.get("salary"));
}
```

MiniDao的使用介绍(18)

- MiniDao测试实体Bean维护

```
public class HiberClient {
    public static void main(String args[]) {
        BeanFactory factory = new ClassPathXmlApplicationContext(
                                "applicationContext.xml");
        JeecgDemoDao jeecgDemoDao = (JeecgDemoDao)
        factory.getBean("jeecgDemoDao");
        JeecgDemo s =
        jeecgDemoDao.getByIdHiber(JeecgDemo.class,"402880e7408f53a40
    1408f53a5aa0000");
        if(s!=null){
                System.out.println(s.getUserName());
        }
    }
}
```

参考资料

- 技术论坛：http://www.jeecg.org
- 作者：张代浩
- 联系方式：zhangdaiscott@163.com
- QQ交流群：325978980，143858350

JEECG 智能开发平台