

MiniDao-PE

使用指南

版 本：1.6.2
作 者：张代浩
日 期：2017/08/23

前言

■ 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

■ 源码下载

- 地址: <http://git.oschina.net/jeecg/minidao>

目 录

■ 大纲

- 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

一、MiniDao简介及特性

1、MiniDao简介及特征

MiniDao 是一款超轻量的JAVA持久层框架，基于 SpringJdbc + freemarker 实现，具备Mybatis一样的SQL分离灵活性和标签逻辑。最大优点：可无缝集成Hibernate项目，支持事务统一管理，有效解决Hibernate项目，实现灵活的SQL分离问题。

MiniDao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
- 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
- SQL支持脚本语言
- Sql 性能优于Mybatis

2、MiniDao支持SQL分离写法

第一步： EmployeeDao. java 接口定义(不需要实现)

```
@Repository
public interface EmployeeDao {

    @Sql("select * from employee where id = :id")
    Employee get(@Param("id") String id);

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

    void insert(@Param("employee") Employee employee);

    @ResultType(Employee.class)
    public MiniDaoPage<Employee> getAll(@Param("employee") Employee
employee,@Param("page") int page,@Param("rows") int rows);

    @Sql("delete from employee where id = :id")
    public void delete(@Param("id") String id);

}
```

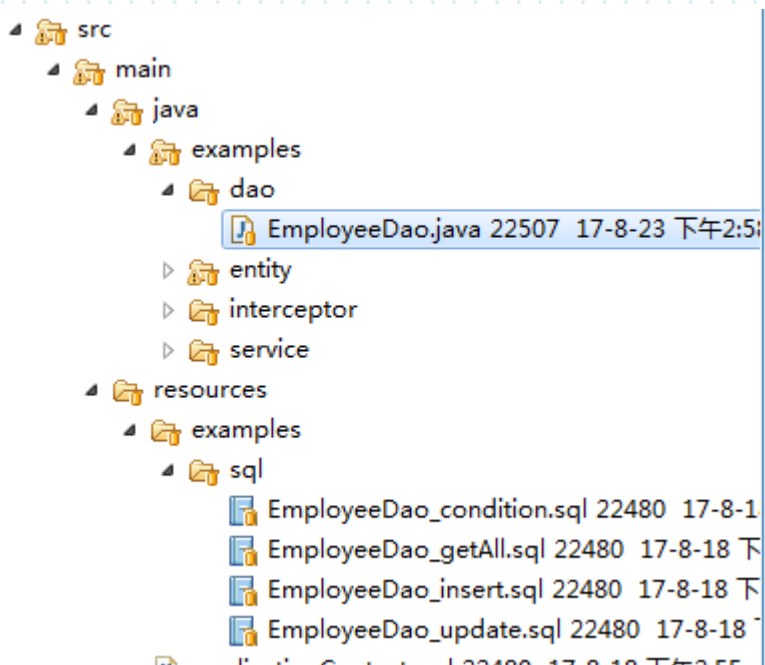
2、MiniDao支持SQL分离写法

第二步：接口方法对应SQL文件创建

[1]、分离Sql文件

SQL文件M目录规则，与minidao接口保持相同目录

SQL文件命名规则：{Dao接口名}_{方法名}.sql



[2]、SQL注解标签

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

2、MiniDao支持SQL分离写法

第三步：SQL文件

SQL文件采用模板语言Freemarker作为解析引擎，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>
```


■ 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>
```

2、MiniDao支持SQL分离写法

第四步：@Arguments 参数标签

➤ 注解定义：

```
/**
 * (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
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Arguments {
    String[] value() default {};
}
```

➤ 用法示例：

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

2、MiniDao支持SQL分离写法

第四步：@Param 参数标签

➤ 注解定义：

```
/**
 * minidao参数注解
 */
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Param {
    String value();
}
```


























➤ 用法示例：

```
Map<String, Object> getMap2(@Param("empno") String empno, @Param("name")String
name);
```

二、Mini-Dao的安装及基本概念

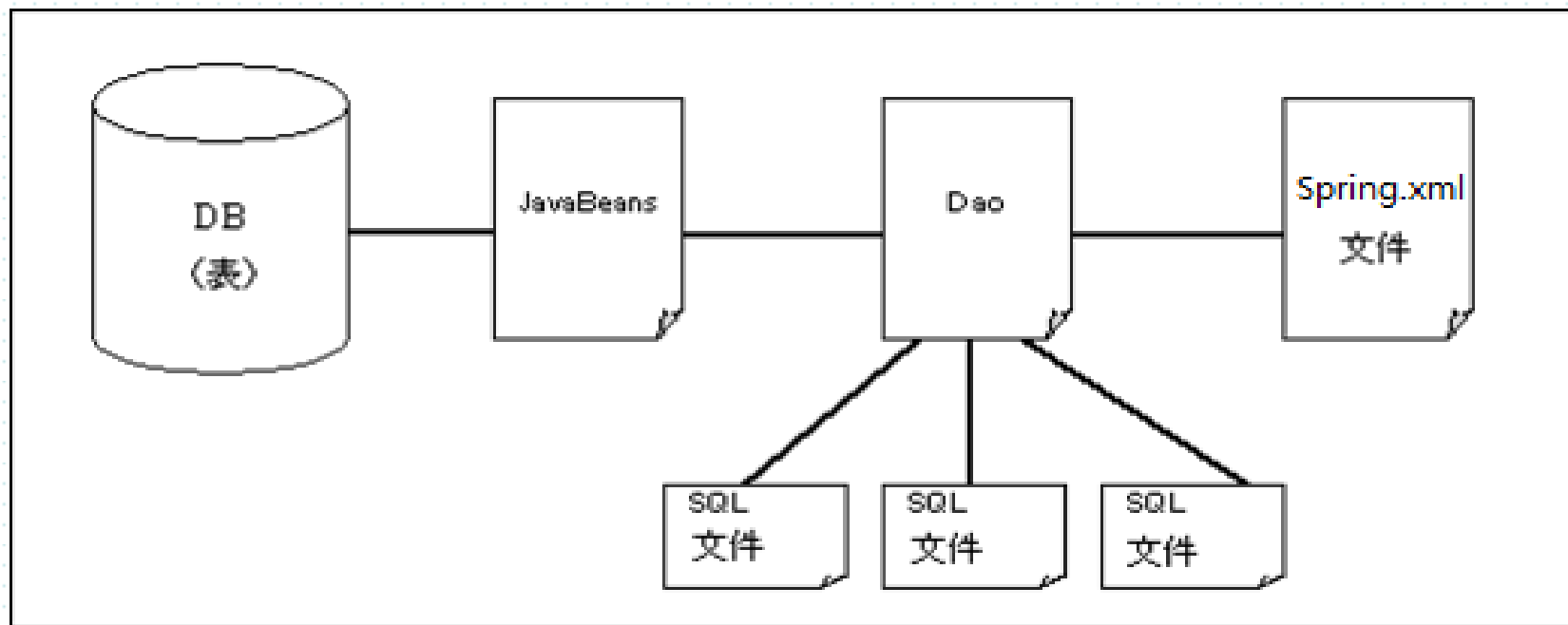
1、MiniDao的安装

- 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)

-  aopalliance-1.0.jar
-  aspectjrt-1.6.9.jar
-  aspectjweaver-1.6.9.jar
-  com.springsource.org.junit-4.9.0.jar
-  commons-beanutils-1.7.0.jar
-  commons-collections-3.2.1.jar
-  commons-lang-2.6.jar
-  commons-logging-1.0.4.jar
-  dom4j-1.6.1.jar
-  druid-0.2.9.jar
-  freemarker-null-2.3.19.jar
-  javassist-3.15.0-GA.jar
-  log4j-1.2.16.jar
-  minidao-pe-1.6-SNAPSHOT.jar
-  mysql-connector-java-5.0.5.jar
-  ognl-2.6.11.jar
-  spring-aop-4.0.9.RELEASE.jar
-  spring-beans-4.0.9.RELEASE.jar
-  spring-context-4.0.9.RELEASE.jar
-  spring-context-support-4.0.9.RELEASE.jar
-  spring-core-4.0.9.RELEASE.jar
-  spring-expression-4.0.9.RELEASE.jar
-  spring-jdbc-4.0.9.RELEASE.jar
-  spring-test-4.0.9.RELEASE.jar
-  spring-tx-4.0.9.RELEASE.jar

■MiniDao 轻量级持久层解决方案

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：



2、MiniDao的配置文件

2.1 MiniDao配置文件

```
<!-- MiniDao扫描类 -->
<bean class="org.jeecgframework.minidao.factory.MinidaoBeanScannerConfigurer">
    <!-- 是使用什么字母做关键字Map的关键字默认值origin 即和sql保持一致,lower小写(推荐),upper 大写 -->
    <property name="keyType" value="lower"></property>
    <!-- 格式化sql -->
    <property name="formatSql" value="true"></property>
    <!-- 输出sql -->
    <property name="showSql" value="true"></property>
    <!-- 数据库类型 -->
    <property name="dbType" value="mysql"></property>
    <!-- dao地址,配置符合spring方式 -->
    <property name="basePackage" value="examples.dao"></property>
    <!-- 使用的注解,默认是Minidao,推荐 Repository -->
    <property name="annotation" value="org.springframework.stereotype.Repository"></property>
</bean>
```


2、MiniDao的配置文件

2.2 MiniDao配置文件 - 参数说明

keyType	Map的关键字大小写配置	lower小写(推荐) upper 大写
formatSql	是否格式化SQL	true/false
showSql	日志是否打印SQL	true/false
dbType	数据库类型（重要）	oracle/mysql/sqlserver/postgres/db2
<i>basePackage</i>	扫描路径	Dao接口扫描路径，多个逗号隔开
<i>annotation</i>	Minidao接口注解	默认是Minidao, 推荐 Repository
<i>emptyInterceptor</i>	Minidao 拦截器	拦截实现自己的功能

2、MiniDao的安装

2.3 MiniDao配置文件 - 拦截器配置

说明：自定义拦截器需要实现接口EmptyInterceptor，实现onInsert和onUpdate方法

```
<!-- minidao拦截器 -->
<bean name="minidaoInterceptor" class="org.jeecgframework.minidao.aspect.MinidaoInterceptor"></bean>
<!-- MiniDao扫描类 -->
<bean class="org.jeecgframework.minidao.factory.MinidaoBeanScannerConfigurer">
  <!-- 是使用什么字母做关键字Map的关键字 默认值origin 即和sql保持一致,lower小写(推荐),upper 大写 -->
  <property name="keyType" value="lower"></property>
  <!-- 格式化sql -->
  <property name="formatSql" value="true"></property>
  <!-- 输出sql -->
  <property name="showSql" value="true"></property>
  <!-- 数据库类型 -->
  <property name="dbType" value="mysql"></property>
  <!-- dao地址,配置符合spring方式 -->
  <property name="basePackage" value="examples.dao"></property>
  <!-- 使用的注解,默认是Minidao,推荐 Repository -->
  <property name="annotation" value="org.springframework.stereotype.Repository"></property>
  <!-- Minidao拦截器配置 -->
  <property name="emptyInterceptor" ref="minidaoInterceptor"></property>
</bean>
```

2、MiniDao的配置文件

2.4 MiniDao配置文件 - 拦截器自定义示例

参考代码：

```
/**
 * minidao拦截器实现【自动填充：创建人，创建时间，修改人，修改时间】
 */
@Service
public class MinidaoInterceptor implements EmptyInterceptor {

    @Override
    public boolean onInsert(Field[] fields, Object obj) {
        Map<Object, Object> map = new HashMap<Object, Object>();
        for (int j = 0; j < fields.length; j++) {
            fields[j].setAccessible(true);
            String fieldName = fields[j].getName();
            //获取登录用户
            LoginUser loginUser = ContextHolderUtils.getLoginSessionUser();
            if (loginUser != null) {
                if ("createBy".equals(fieldName)) {
                    map.put("createBy", loginUser.getUserName());
                }
                if ("createDate".equals(fieldName)) {
                    map.put("createDate", new Date());
                }
            }
            try {
                //回写Value值
                setFieldValue(map, obj);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return false;
    }
}
```

3、MiniDao的基本概念

3.1 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

3、MiniDao的基本概念

3.2 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。**

3、MiniDao的基本概念

3.3 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:

```
<!-- MiniDao动态代理类 -->
<bean id="miniDaoHandler" class="org.jeecgframework.minidao.factory.MinidaoBeanScannerConfigurer">
  <!-- 是使用什么字母做关键字Map的关键字 默认值origin 即和sql保持一致,lower小写(推荐),upper 大写 -->
  <property name="keyType" value="lower"></property>
  <!-- 格式化sql -->
  <property name="formatSql" value="false"></property>
  <!-- 输出sql -->
  <property name="showSql" value="false"></property>
  <!-- 数据库类型 -->
  <property name="dbType" value="mysql"></property>
  <!-- dao地址,配置符合spring方式 -->
  <property name="basePackage" value="org.jeecgframework.web,com.jeecg"></property>
  <!-- 使用的注解,默认是Minidao,推荐 Repository-->
  <property name="annotation" value="org.springframework.stereotype.Repository"></property>
</bean>
```

MiniDao Spring配置代码

配置文件: spring-minidao.xml

```
<!-- MiniDao动态代理类 -->
<bean id="miniDaoHandler"
class="org.jeecgframework.minidao.factory.MinidaoBeanScannerConfigurer">
    <!-- 是使用什么字母做关键字Map的关键字 默认值origin 即和sql保持一致, lower小写(推荐), upper 大写 -->
    <property name="keyType" value="lower"></property>
    <!-- 格式化sql -->
    <property name="formatSql" value="false"></property>
    <!-- 输出sql -->
    <property name="showSql" value="false"></property>
    <!-- 数据库类型 -->
    <property name="dbType" value="mysql"></property>
    <!-- dao地址, 配置符合spring方式 -->
    <property name="basePackage" value="org.jeecgframework.web.com.jeecg"></property>
    <!-- 使用的注解, 默认是Minidao, 推荐 Repository-->
    <property name="annotation"
value="org.springframework.stereotype.Repository"></property>
</bean>
```

3、MiniDao的基本概念

3.4 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)

3、MiniDao的基本概念

```
import java.util.Date;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import examples.dao.EmployeeDao;
import examples.entity.Employee;

public class ClientDao {
    public static void main(String args[]) {
        BeanFactory factory = new
        ClassPathXmlApplicationContext("applicationContext.xml");
        EmployeeDao employeeDao = (EmployeeDao) factory.getBean("employeeDao");
        Employee employee = new Employee();
        employee.setId("8");
        employee.setBirthday(new Date());
        employee.setName("雇员张三");
        //调用minidao方法
        employeeDao.insert(employee);
    }
}
```

三、MiniDao的使用介绍

1、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);`

1、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);`

1、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();`

2、Dao参数注解

■ @Param注释标签

使用@Param注释指定方法(method)的参数名**别名**,这样就可以在SQL指令中通过**别名**引用方法(method)的参数。

方法:

```
Map<String,Object> getMap2(@Param("empnokey") String  
empno,@Param("namekey")String name);
```

SQL:

```
SELECT * FROM employee WHERE empno = :empnokey and name = :namekey
```

注意:

通过@Param标记,方法的全部参数必须都设置别名,否则会提示错误,尽量别名和参数名保持一致

2、Dao参数注解

■ @Arguments注释标签

使用@Arguments注释指定方法(method)的参数的别名,这样就可以在SQL指令中通过别名引用方法(method)的参数。

方法:

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

SQL:

```
SELECT * FROM employee  
WHERE empno = :empnokey  
and name = :namekey
```

说明: 尽量别名和参数名保持一致。

3、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

&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
- 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Session 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

四、技术支持

■MiniDao 轻量级持久层解决方案

- ◆ 技术论坛: www.jeecg.org
- ◆ 作 者: 张代浩
- ◆ 联系方式: jeecg@sina.com
- ◆ QQ交流群: 325978980, 143858350

MiniDao

J2EE持久层轻量级解决方案

JEECG 开源社区

[Http://www.jeecg.org](http://www.jeecg.org)