wind-orm框架文档

|  |  |
| --- | --- |
| **文档名称**： | wind-orm框架主文档 |
| **技术类型**： | ORM |
| **作者**： | 胡璐璐 |
| **网站URL：** | http://www.tcin.cn |
| **当前版本号**： | V1.3.2（JDK1.8.x） |
| **修改时间**： | 2023年11月8日 18:29:05 |

**目录**

[一. 前言 10](#_Toc167303696)

[1.1什么是ORM？ 10](#_Toc167303697)

[1.2说明 11](#_Toc167303698)

[1.2.1 核心类（Table） 11](#_Toc167303699)

[1.2.2 辅助工具类（TableUtil） 12](#_Toc167303700)

[1.2.3 分页类（Page） 12](#_Toc167303701)

[1.2.4 数据源配置类（Config） 14](#_Toc167303702)

[1.2.5 数据源配置类\_写（Config\_Write） 14](#_Toc167303703)

[1.2.6 数据源配置类\_读（Config\_Read） 14](#_Toc167303704)

[1.2.7 数据源配置类\_通用参数（Config\_global） 15](#_Toc167303705)

[二. 配置 16](#_Toc167303706)

[2.0配置文件 16](#_Toc167303707)

[2.1支持的数据库 18](#_Toc167303708)

[2.2注解(Annotation) 18](#_Toc167303709)

[2.2.1作用范围：类 19](#_Toc167303710)

[2.2.1.1 @Tables 19](#_Toc167303711)

[2.2.1.2 @DataSource 19](#_Toc167303712)

[2.2.1.3 @Lock 20](#_Toc167303713)

[2.2.2作用范围：字段（成员属性/变量） 21](#_Toc167303714)

[2.2.2.1 @Column 21](#_Toc167303715)

[2.2.2.2 @ForeignKey 22](#_Toc167303716)

[2.2.2.3 @Id 23](#_Toc167303717)

[2.2.2.4 @OrderBy 24](#_Toc167303718)

[2.2.2.5 @DateTime 24](#_Toc167303719)

[2.2.2.6 @Null 25](#_Toc167303720)

[2.2.2.7 @Value 25](#_Toc167303721)

[2.2.2.8 @Default 26](#_Toc167303722)

[三. 核心方法（Table） 27](#_Toc167303723)

[3.1 对象方法 27](#_Toc167303724)

[3.1.1查询 27](#_Toc167303725)

[3.1.1.1 find() 27](#_Toc167303726)

[3.1.1.2 find(String,Page) 27](#_Toc167303727)

[3.1.1.3 findOne(boolean) 28](#_Toc167303728)

[3.1.1.4 find(String,List,boolean,Page) 28](#_Toc167303729)

[3.1.1.5 findSize() 29](#_Toc167303730)

[3.1.1.6 findSizeMap(String) 30](#_Toc167303731)

[3.1.2保存（插入、更新） 30](#_Toc167303732)

[3.1.2.1 save() 30](#_Toc167303733)

[3.1.2.2 save(boolean,boolean) 31](#_Toc167303734)

[3.1.3删除 31](#_Toc167303735)

[3.1.3.1 delete() 31](#_Toc167303736)

[3.1.4其他 32](#_Toc167303737)

[3.1.4.1 setOrderFieldName(String) 32](#_Toc167303738)

[3.1.4.2 getOrderFieldName() 32](#_Toc167303739)

[3.1.4.3 setGroupFieldName(String) 32](#_Toc167303740)

[3.1.4.4 getGroupFieldName() 33](#_Toc167303741)

[3.1.4.5 addSpecifiedFieldName(String…) 33](#_Toc167303742)

[3.1.4.6 clearSpecifiedFieldName() 33](#_Toc167303743)

[3.1.4.7 getSpecifiedColumnSet() 33](#_Toc167303744)

[3.1.4.8 setOrderSQL(String) 34](#_Toc167303745)

[3.1.4.9 getOrderSQL() 34](#_Toc167303746)

[3.1.4.10 setGroupSQL(String) 34](#_Toc167303747)

[3.1.4.11 getGroupSQL() 35](#_Toc167303748)

[3.1.4.12 setDataSource(String) 35](#_Toc167303749)

[3.1.4.13 getDataSource() 35](#_Toc167303750)

[3.1.4.14 setTable(String) 35](#_Toc167303751)

[3.1.4.15 getTable() 36](#_Toc167303752)

[3.2 静态方法 36](#_Toc167303753)

[3.2.1 getInstance(Class) 36](#_Toc167303754)

[3.2.2查询 36](#_Toc167303755)

[3.2.2.1 findById(Class,Object,String…) 36](#_Toc167303756)

[3.2.2.2 findAll(Class,boolean) 37](#_Toc167303757)

[3.2.2.3 findAll(Class,boolean,Page) 37](#_Toc167303758)

[3.2.2.4 findAll(Class,boolean,Page,String,String,String…) 37](#_Toc167303759)

[3.2.2.5 find(Class,String,List,boolean,Page) 38](#_Toc167303760)

[3.2.2.6 find(Class,String,List,boolean,Page,String,String,String…) 39](#_Toc167303761)

[3.2.2.7 findOne(Class,String,Object[],boolean) 40](#_Toc167303762)

[3.2.2.8 findOne(Class,String,Object[],boolean,String…) 40](#_Toc167303763)

[3.2.2.9 findOne(Class,String,List,boolean,String…) 41](#_Toc167303764)

[3.2.2.10 findByIdList(Class,List, String…) 42](#_Toc167303765)

[3.2.2.11 findByNotIdList(Class,List, String...) 42](#_Toc167303766)

[3.2.2.12 findAllSize(Class) 43](#_Toc167303767)

[3.2.2.13 findDate(Class) 43](#_Toc167303768)

[3.2.2.14 findIdList(Class,String,List) 44](#_Toc167303769)

[3.2.2.15 findAllIdList(Class) 44](#_Toc167303770)

[3.2.2.16 findSpecifiedList(Class,String,List,String) 44](#_Toc167303771)

[3.2.2.17 find(Class,String,List) 45](#_Toc167303772)

[3.2.2.18 findSize(Class,String,List) 46](#_Toc167303773)

[3.2.2.19 findSizeMap(Class,String,List,String) 47](#_Toc167303774)

[3.2.2.20 findAllForeign(Table,boolean,Page) 47](#_Toc167303775)

[3.2.2.21 findForeign(Table,boolean,Page,String...) 48](#_Toc167303776)

[3.2.2.22 findForeign(Table,String...) 48](#_Toc167303777)

[3.2.2.23 findForeign(List,boolean,Page,String[],String...) 49](#_Toc167303778)

[3.2.2.24 findForeign (List,String…) 50](#_Toc167303779)

[3.2.2.25 findAllForeign(List,boolean,Page) 50](#_Toc167303780)

[3.2.2.26 findOneForeign(Table,String,String...) 51](#_Toc167303781)

[3.2.2.27 findOneForeign(List,String,String...) 51](#_Toc167303782)

[3.2.3保存（插入、更新） 52](#_Toc167303783)

[3.2.3.1 save(List) 52](#_Toc167303784)

[3.2.3.2 save(List,boolean,boolean) 53](#_Toc167303785)

[3.2.3.3 save(Class,String,List) 54](#_Toc167303786)

[3.2.3.4 update(Class,String,String,List) 55](#_Toc167303787)

[3.2.4自增/自减 56](#_Toc167303788)

[3.2.4.1 increaseOrDecrease(Class,String,double,Object) 56](#_Toc167303789)

[3.2.4.2 increaseOrDecrease(Class,String,double,String,List) 56](#_Toc167303790)

[3.2.5删除 57](#_Toc167303791)

[3.2.5.1 delete(Class,Object) 57](#_Toc167303792)

[3.2.5.2 delete(Class) 58](#_Toc167303793)

[3.2.5.3 delete(Class,String,List) 58](#_Toc167303794)

[3.2.6 DDL（数据库模式定义语言） 59](#_Toc167303795)

[3.2.6.1 call(Class,String,Object[],Integer) 59](#_Toc167303796)

[3.2.6.2 callList(Class,String,Object[],Integer[]) 59](#_Toc167303797)

[3.2.6.3 create(Class,String) 60](#_Toc167303798)

[3.2.6.4 copy(Class,String,boolean) 60](#_Toc167303799)

[3.2.6.5 drop(Class,String) 61](#_Toc167303800)

[3.2.6.6 getTableSQL(Class) 61](#_Toc167303801)

[3.2.6.7 isTableExist (Class) 62](#_Toc167303802)

[3.2.6.8 isTableExist (Class,String) 62](#_Toc167303803)

[3.2.6.9 getColumn(Class) 62](#_Toc167303804)

[3.2.7数学函数（sum、avg、max、min等） 63](#_Toc167303805)

[3.2.7.1 sum(Class,String) 63](#_Toc167303806)

[3.2.7.2 sum(Class,String,List,String) 63](#_Toc167303807)

[3.2.7.3 sum(Class,String,List,String,String) 64](#_Toc167303808)

[3.2.7.4 avg(Class,String) 64](#_Toc167303809)

[3.2.7.5 avg(Class,String,List,String) 65](#_Toc167303810)

[3.2.7.6 avg(Class,String,List,String,String) 65](#_Toc167303811)

[3.2.7.7 max(Class,String) 66](#_Toc167303812)

[3.2.7.8 max(Class,String,List,String) 66](#_Toc167303813)

[3.2.7.9 max(Class,String,List,String,String) 67](#_Toc167303814)

[3.2.7.10 maxString(Class,String) 68](#_Toc167303815)

[3.2.7.11 maxString(Class,String,List,String) 68](#_Toc167303816)

[3.2.7.12 maxString(Class,String,List,String,String) 68](#_Toc167303817)

[3.2.7.13 min(Class,String) 69](#_Toc167303818)

[3.2.7.14 min(Class,String,List,String) 69](#_Toc167303819)

[3.2.7.15 min(Class,String,List,String,String) 70](#_Toc167303820)

[3.2.7.16 minString(Class,String) 71](#_Toc167303821)

[3.2.7.17 minString(Class,String,List,String) 71](#_Toc167303822)

[3.2.7.18 minString(Class,String,List,String,String) 72](#_Toc167303823)

[3.2.8其他 72](#_Toc167303824)

[3.2.8.1 setAutoCommit(Class,boolean) 72](#_Toc167303825)

[3.2.8.2 isAutoCommit(Class) 73](#_Toc167303826)

[3.2.8.3 commit(Class) 73](#_Toc167303827)

[3.2.8.4 setSavepoint(Class,String) 73](#_Toc167303828)

[3.2.8.5 releaseSavepoint(Class,Savepoint) 74](#_Toc167303829)

[3.2.8.6 rollback(Class)和rollback(Class,Savepoint) 74](#_Toc167303830)

[3.2.8.7 close(Class) 75](#_Toc167303831)

[3.2.8.8 setTimeout(Class,int) 75](#_Toc167303832)

[3.2.8.9 getTimeout(Class) 76](#_Toc167303833)

[四. 工具方法 76](#_Toc167303834)

[4.1 TableUtil（Table常用工具类） 76](#_Toc167303835)

[4.1.1 getMethodMap\_set(Class) 76](#_Toc167303836)

[4.1.2 getMethodMap\_get(Class) 77](#_Toc167303837)

[4.1.3 getMethodMap(Class,Class,String) 77](#_Toc167303838)

[4.1.4 getParameterNull(Method) 78](#_Toc167303839)

[4.1.5 get(Object,Map,Field) 78](#_Toc167303840)

[4.1.6 set(Object,Map,Field,Object) 79](#_Toc167303841)

[4.1.7 setPrimaryKeyValue(Object,Object) 80](#_Toc167303842)

[4.1.8 getPrimaryKeyValue(Object) 80](#_Toc167303843)

[4.1.9 getPrimaryKeyField(Class) 81](#_Toc167303844)

[4.1.10 cast(Object,Class) 81](#_Toc167303845)

[4.1.11 getDataSource(Class) 82](#_Toc167303846)

[4.1.12 getClass(Class,Class) 82](#_Toc167303847)

[4.1.13 getAnnotation(Class,Class) 83](#_Toc167303848)

[4.1.14 getConfig(Table) 83](#_Toc167303849)

[4.1.15 getConfig(Class) 84](#_Toc167303850)

[4.1.16 getConfig(String) 84](#_Toc167303851)

[4.1.17 getTable(Class) 84](#_Toc167303852)

[4.1.18 getPrimaryKey(Class) 85](#_Toc167303853)

[4.1.19 getGeneric(Field) 85](#_Toc167303854)

[4.1.20 copyProperties(Object,Object) 85](#_Toc167303855)

[4.1.21 getField(Class) 86](#_Toc167303856)

[4.1.22 getField(Class,Class) 86](#_Toc167303857)

[4.1.23 getField(Class,String) 87](#_Toc167303858)

[4.1.24 getMethod(Class) 87](#_Toc167303859)

[4.1.25 getMethod(Class,Class) 87](#_Toc167303860)

[4.1.26 getForeignKeyField(Class,Class) 88](#_Toc167303861)

[4.1.27 getIdList(List) 88](#_Toc167303862)

[4.1.28 getSpecifiedList(List,String) 88](#_Toc167303863)

[4.1.29 getForeignKey(Field) 89](#_Toc167303864)

[4.1.30 getColumn(Field) 89](#_Toc167303865)

[4.1.31 getColumnValueStr(Field) 90](#_Toc167303866)

[4.1.32 arrayConverList(Object[]) 90](#_Toc167303867)

[4.1.33 sort(List,List) 90](#_Toc167303868)

[4.1.34 isMatch(List) 91](#_Toc167303869)

[4.1.35 getPrimaryKey\_Database(Connection,String) 92](#_Toc167303870)

[4.1.36 getConnection(String,String,String,Integer) 92](#_Toc167303871)

[4.1.37 closeConnection(Connection) 93](#_Toc167303872)

[4.1.38 close(ResultSet,Statement) 93](#_Toc167303873)

[4.1.39 getColumn\_Database(Connection,String) 93](#_Toc167303874)

[4.1.40 listInit(List) 94](#_Toc167303875)

[4.1.41 getPlaceholder(List) 94](#_Toc167303876)

[4.1.42 getObject(Class,Object...) 94](#_Toc167303877)

[4.1.43 getObjectList(List,String) 95](#_Toc167303878)

[4.1.44 setObjectList(List,String,List) 95](#_Toc167303879)

[4.1.45 listToMap(List) 96](#_Toc167303880)

[4.1.46 getPack(Config,String) 96](#_Toc167303881)

[4.1.47 getPack(String,String) 97](#_Toc167303882)

[4.1.48 findForeign(List,boolean,Page,String[],String…) 97](#_Toc167303883)

[4.1.49 isNull(Field,Object) 98](#_Toc167303884)

[4.1.50 getInterfaces(Class<?>,Class<?>) 98](#_Toc167303885)

[4.2 ConfigUtil 99](#_Toc167303886)

[4.2.1 getDataSourceConfig() 99](#_Toc167303887)

[五. 连接池 99](#_Toc167303888)

[5.1 ConnectionPoolFactory（连接池工厂类） 99](#_Toc167303889)

[5.2 ConnectionPool（连接池接口类） 100](#_Toc167303890)

[5.3 ConnectionPoolImpl（连接池接口实现类） 100](#_Toc167303891)

[5.4 ConnectionManager（连接管理器接口类） 100](#_Toc167303892)

[5.5 ConnectionManagerImpl（连接管理器接口实现类） 101](#_Toc167303893)

[六. 异常 101](#_Toc167303894)

[6.1 ConnectionPoolException 101](#_Toc167303895)

[6.2 ConnectionPoolFactoryException 101](#_Toc167303896)

[6.3 SQLFactoryException 101](#_Toc167303897)

[6.5 TableDDLException 101](#_Toc167303898)

[6.6 TableDeleteException 102](#_Toc167303899)

[6.7 TableException 102](#_Toc167303900)

[6.8 TableSaveException 102](#_Toc167303901)

[6.9 TableSelectException 102](#_Toc167303902)

[七. 缓存 102](#_Toc167303903)

[八. 拦截器 102](#_Toc167303904)

# 前言

## 1.1什么是ORM？

**对象关系映射**（英语：(Object Relational Mapping，简称**ORM**，或O/RM，或O/R mapping），是一种程序技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换。从效果上说，它其实是创建了一个可在编程语言里使用的--“虚拟对象数据库”。

面向对象是从软件工程基本原则（如耦合、聚合、封装）的基础上发展起来的，而关系数据库则是从数学理论发展而来的，两套理论存在显著的区别。为了解决这个不匹配的现象，对象关系映射技术应运而生。

对象关系映射（Object-Relational Mapping）提供了概念性的、易于理解的模型化数据的方法。ORM方法论基于三个核心原则：

（1）、简单：以最基本的形式建模数据。

（2）、传达性：数据库结构被任何人都能理解的语言文档化。

（3）、精确性：基于数据模型创建正确标准化的结构。

典型地，建模者通过收集来自那些熟悉应用程序但不熟练的数据建模者的人的信息开发信息模型。建模者必须能够用非技术企业专家可以理解的术语在概念层次上与数据结构进行通讯。建模者也必须能以简单的单元分析信息，对样本数据进行处理。ORM专门被设计为改进这种联系

ORM是表到对象关系的映射，在Java里可以用bean代表数据库表的Model，即：

（1）、类（对象）—>表

（2）、变量（成员属性）—>列

本框架的映射技术，采用注解映射，如：

1. @Tables：代表数据库表的映射
2. @Column：代表数据库列的映射
3. @ForeignKey：代表绑定外键的映射（数据库里不需要强制绑定外键）

…………….

## 1.2说明

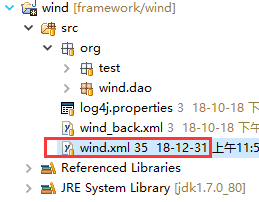
以**Table**类出发，操作：

（1）、添删改查

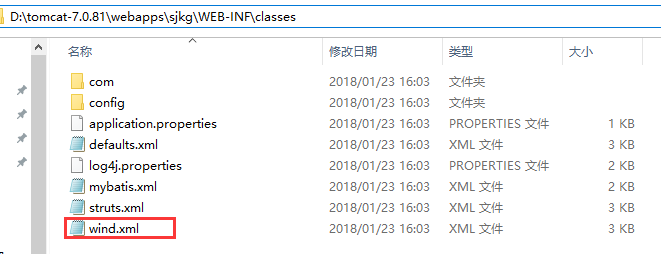
（2）、事务开启

（3）、Connection连接获取、释放

**wind-orm.xml**文件，目前只能放入工程的根目录下，MyEclipse为src目录，如图：



部署到Tomcat下则为：classes目录下，如图：



### 核心类（Table）

本框架的核心类：**Table(org.wind.orm.Table)**

以此为出发，操作所有的ORM方法，如：

//根据主键为**159**查询

Table.findById(Student.class,159);

//保存一批对象数据，根据第1个对象主键是否为null，决定是做【插入】还是【更新】

Table.save(List);

### 辅助工具类（TableUtil）

辅助工具类：**TableUtil(org.wind.orm.service.TableUtil)**

辅助绝大多数的ORM方法，如：生成占位符，获取主键Field、Method等

### 分页类（Page）

分页类：**Page(org.wind.orm.bean.Page)**

支持：普通分页、高效分页、超高效分页（目前只针对MySQL数据）

在调用一些findXXX(…)方法时用到

|  |  |
| --- | --- |
| 构造方法： | 1. **public** Page(**int** pageNew,**int** limit)： 高效分页 **pageNew**：跳转的页数（小于等于0时候。将自动设置为1 **limit**：每页多少条数据（不能为负数）,为负数会抛出异常TableSelectException 2. **public** Page(**int** pageNew,**int** limit,Object pageLastId)： 超高效分页 **pageNew** : 当前页的第几页（负数=前第几页；正数=后第几页） **limit**: 前（后）第几页多少条数据，为负数会抛出异常 **pageLastId**: 如果指定页是当前页的上页，则是当前页的最小ID，反之最大ID。若该值为NULL，则跟第1条一样的效果 3. **public** Page(**int** pageNew,**int** limit,Object pageLastId,Object pageLastId2) **前3个参数跟第2条一样**。 **pageLastId**2: 第2层的id（可空；主键，唯一），主要用于当第1个id参数（pageLastId）不是主键值的时候，需要该参数做为唯一值来做超高效分页。本方法和上面的方法，目前只针对**MySQL**数据库 |
| 变量： | 1. pageNew：int类型，新的页数（跳转的目标页数） 2. limit：int类型，分页起始位置后面多少条数据、每页多少条（默认**100**条） 3. id：Object类型，当前页的最大或最小id主键（表为【字符串主键】的，则该值一般为：创建时间），如果指定页是当前页的上页，则是当前页的最小ID，反之最大ID（对于MySQL，可超高效分页） 4. id2：Object类型，第2层的排序id，一般在id不是主键值时（可能是【创建时间】）需要；该列的值必须为主键【唯一】 5. isEfficient：boolean类型，是否超高效分页（默认：false） |
| 方法： | 1. **public** **int** getPageNew()：获取当前跳转的页数 2. **public** **void** setPageNew(**int** pageNew)：设置当前跳转的页数 3. **public** **long** getBegin()：获取分页的起始位置。计算得出 4. **public** **int** getLimit()：获取当前页显示的数据条数。分页起始位置后面多少条数据、每页多少条 5. **public** **void** setLimit(**int** limit)：设置当前页显示的数据条数 6. **public** **long** getEnd()：获取分页的结束位置。计算得出 7. **public** **boolean** isEfficient()：判断当前分页是否超高效，false=不是 8. **public** Object getId()：获取当前若是超高效分页时的最小或最大id。如果指定页是当前页的上页，则是当前页的最小ID，反之最大ID 9. **public** Object getId2()：获取当前若是超高效分页时，若上面的id参数不是主键，则该属性参数必须为表的主键 10. **public** **void** setId(Object id)：设置当前若是超高效分页时的最小或最大id 11. **public** **void** setId2(Object id2)：设置第2个主键id（必须是主键值） 12. **public** **long** getPageCount(**long** size)：获取总页数 **size** : 查询的数据总数（不是带分页的数据） |

### 数据源配置类（Config）

数据源主入口配置类：**Config (org.wind.orm.bean.Config)**

数据源配置主类，含有：读写分离的配置信息。写和读均可以继承主类的配置信息，如果写和读有配置自己的信息，则优先用自己的

|  |  |
| --- | --- |
| 构造方法： | 所有**成员属性**的参数 |
| 变量： | 继承Config\_global类，拥有该父类的成员变量  //写  **private** Config\_Write configWrite;  //读  **private** Config\_Read configRead; |
| 方法： | 成员属性相应的get和set方法 |

### 数据源配置类\_写（Config\_Write）

数据源配置类\_写：**Config (org.wind.orm.bean.Config\_Write)**

读写分离的【写】配置信息。

|  |  |
| --- | --- |
| 构造方法： | 所有**成员属性**的参数 |
| 变量： | 继承Config\_global类，拥有该父类的成员变量 |
| 方法： |  |

### 数据源配置类\_读（Config\_Read）

数据源配置类\_读：**Config (org.wind.orm.bean.Config\_Read)**

读写分离的【读】配置信息。

|  |  |
| --- | --- |
| 构造方法： | 所有**成员属性**的参数 |
| 变量： | 继承Config\_global类，拥有该父类的成员变量 |
| 方法： |  |

### 数据源配置类\_通用参数（Config\_global）

数据源通用参数配置类：**Config\_global (org.wind.orm.bean.Config\_global)**

wind-orm.xml下所有的数据源配置信息（如：max=最大保持Connection连接数）

|  |  |
| --- | --- |
| 构造方法： | 所有**成员属性**的参数 |
| 变量： | //数据源ID  **protected** String id;  //数据库驱动字符串（默认：com.mysql.jdbc.Driver）  **protected** String driver;  //数据库连接字符串  **protected** String url;  //数据库名称  **protected** String dataBase;  //数据库用户名  **protected** String userName;  //数据库密码  **protected** String passWord;  //最大连接数（连接池）  **protected** Integer max;  //初始化连接数  **protected** Integer init;  //数据库类型（mysql【默认】、sql server、oracle等）  **protected** String type;  //是否打印SQL  **protected** Boolean isPrintSQL;  //全局执行SQL超时（单位：秒，默认为：30秒）  **protected** Integer timeout;  //生成、获取connection超时数（单位：秒，默认：5秒）  **protected** Integer connectionTimeout;  //空间时间；超过该数将关闭（单位：秒，默认：1800秒）  **protected** Integer freeTime;  //是否在执行SQL的时候，指定数据库名（默认：true）  **protected** Boolean isSpecifyDatabaseName; |
| 方法： | 成员属性相应的get和set方法 |

# 配置

## 2.0配置文件

本技术框架的配置文件只有：**wind-orm.xml**,内容及说明如下：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <dataSources url=*"http://www.tcin.cn"*>  <!-- 【SQL】 -->  <dataSource id=*"default"*>  <!-- 数据库驱动字符串 -->  <driver>com.mysql.jdbc.Driver</driver>  <!-- 数据库连接字符串 -->  <url>jdbc:mysql://localhost:3306/test</url>  <!-- 数据库名称（不填，则为自动获取） -->  <!-- <dataBase>test</dataBase> -->  <!-- 数据库用户名 -->  <userName>root</userName>  <!-- 数据库密码 -->  <passWord>123456</passWord>  <!-- 连接池最大保持连接数（为空或小于等于0，则为1个） -->  <max>3</max>  <!-- 初始化：连接数（最多初始化max数，为空或小于等于0，则为1个） -->  <init>1</init>  <!-- 数据库类型（mysql【默认】、sql server、oracle——不填，则自动获取） -->  <!-- <type>mysql</type> -->  <!-- 是否打印SQL -->  <print\_sql>true</print\_sql>  <!-- 执行SQL超时（单位：秒，默认为：30秒） -->  <timeout></timeout>  <!-- 生成、获取connection超时数（单位：秒，默认：5秒） -->  <connectionTimeout></connectionTimeout>  <!-- 空闲时间；超过该数将关闭（单位：秒，默认：1800秒） -->  <freeTime></freeTime>  <!-- 是否在执行SQL的时候，指定数据库名（默认：true） -->  <isSpecifyDatabaseName>true</isSpecifyDatabaseName>  <!-- 读写分离\_写 -->  <write>  <url>jdbc:mysql://localhost:3306/test</url>  </write>  <!-- 读写分离\_读 -->  <read>  <url>jdbc:mysql://localhost:3306/test</url>  <!-- 数据库用户名 -->  <userName>user\_read</userName>  <!-- 数据库密码 -->  <passWord>123456789</passWord>  </read>  </dataSource>  </dataSources> |

**说明**：

（1）、根据：【<dataSoure id=*"default"*>】，的**id不同设置,**可配置多个数据源

（2）、findXXX（）的一些方法的条件SQL，主表的别名为：t

**参数说明**：

|  |  |
| --- | --- |
| 参数名称 | 说明 |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?> | xml标准规范头部 |
| <dataSources url=*"http://www.tcin.cn"*> | 本技术框架的头部，url部分代表作者的个人网站URL |
| <dataSource id=*"default"*> | 数据源头的配置，id为当前数据源的唯一标识（不设置则为默认的：default）,本框架若没有找到数据源，则取【id=”default”】的数据源  可配置多个数据源，只需要id值不同 |
| <driver> | 数据库驱动字符串，如：mysql的—>【com.mysql.jdbc.Driver】 |
| <url> | 数据库连接字符串，如：mysql的—>【jdbc:mysql://192.168.1.156:3306/test】 |
| <dataBase> | 数据库名称（不填，则为自动获取） |
| <userName> | 数据库用户名 |
| <passWord> | 数据库密码 |
| <max> | 连接池最大保持连接数（为空或小于等于0，则为1个） |
| <init> | 初始化：连接数（最多初始化max数，为空或小于等于0，则为1个） |
| <type> | 数据库类型  （mysql【默认】、sql server、oracle等，详细支持看下面） |
| <print\_sql> | 是否打印SQL  以控制台的方式打印 |
| <timeout> | 执行SQL超时  （单位：秒，默认为：30秒） |
| <connectionTimeout> | 生成/获取connection超时数  （单位：秒，默认：5秒） |
| <freeTime> | 空间时间；超过该数将关闭  （单位：秒，默认：1800秒）  可自行实现定时任务：  ConnectionFreeTask |
| <isSpecifyDatabaseName> | 是否在执行SQL的时候，指定数据库名（默认：true） |
| <write> | **数据源**：读写分离\_写。  它的子节点跟父节点的配置一样，可继承 |
| <read> | **数据源**：读写分离\_读。  它的子节点跟父节点的配置一样，可继承 |

## 2.1支持的数据库

本框架目前支持以下数据库：

|  |  |  |
| --- | --- | --- |
| 数据库 | 备注 | 表和列  包装字符 |
| MySQL  （默认） | **\*推荐**  支持：普通、高效、超高效分页 | **`** |
| SQL Server | 支持：普通、高效分页 | [] |
| Oracle | 支持：普通、高效分页 | “” |

**表和列包装字符**：每种数据库都有自己的特殊包装字符，用以区分数据库和兼容关键词

**说明**：对于没有在此列表展示的数据库，将全部使用**MySQL**的形式

## 2.2注解(Annotation)

------------------------------------------------------------**使用例子**-------------------------------------------------

**使用格式**：

@Column("date") 或 @Column(name=”使用日期”,value="date")

**示例**：

@Column("date")

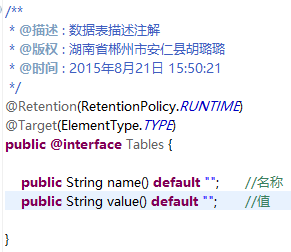
**private** String date;

### 2.2.1作用范围：类

#### 2.2.1.1 @Tables

【类—>表】的映射，加上该注解，则框架里的添删改查方法优先取该注解的value值，否则取类名代表表名, 数据库为Oracle时，会将表名全部转成大写。

详情如下图：

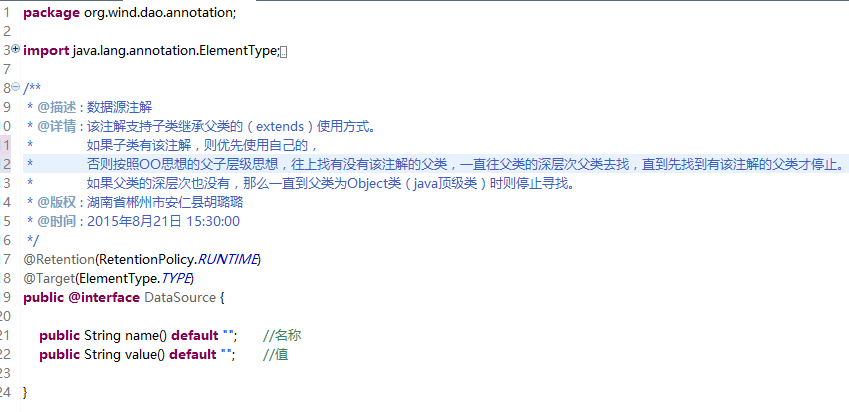


#### 2.2.1.2 @DataSource

数据源注解，标记该ORM类绑定了wind-orm.xml配置的哪个数据源，value值为指定数据源的id，不加该注解，则取默认的【id=”**default**”】的数据源。

**重要**：该注解支持子类继承父类的（extends）使用方式。如果子类有该注解，则优先使用自己的，否则按照OO思想的父子层级思想，往上找有没有该注解的父类，一直往父类的深层次父类去找，直到先找到有该注解的父类才停止寻找。如果父类的深层次也没有，那么一直到父类为Object类（java顶级类）时则停止寻找。

详情如下图：

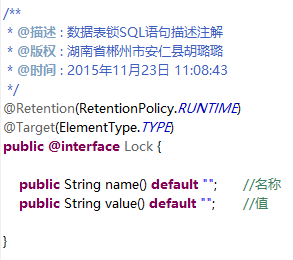


#### 2.2.1.3 @Lock

数据表锁SQL语句描述注解。该注解目前仅对【SQL Server】的数据库有效。使用方式：



详情如下图：



### 2.2.2作用范围：字段（成员属性/变量）

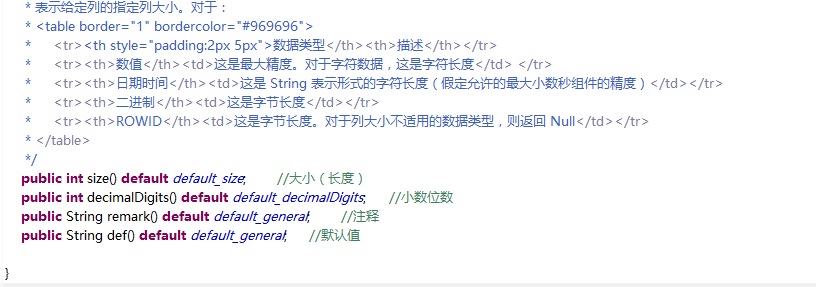
#### 2.2.2.1 @Column

【成员属性（字段、变量）—>列】的映射

加上该注解，则框架里的添删改查方法优先取该注解的value值，否则取字段名（变量、属性名）代表列名，数据库为Oracle时，会将列名全部转成大写。

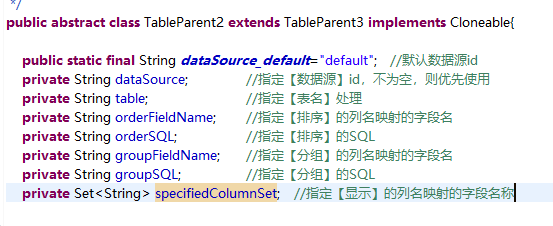
详情如下图：





**注意**：字段名不能有跟父类【Table】及之上的字段名有重复重复，如：table

（下图列表）



#### 2.2.2.2 @ForeignKey

标记该成员属性（字段、变量）为外键关系，需要该成员变量的类型必须为继承**Table**类。列名取@Column注解的value值，否则取字段所属Class的@Id注解的字段的代表列名。详细说明看下面。数据库为Oracle时，会将列名全部转成大写

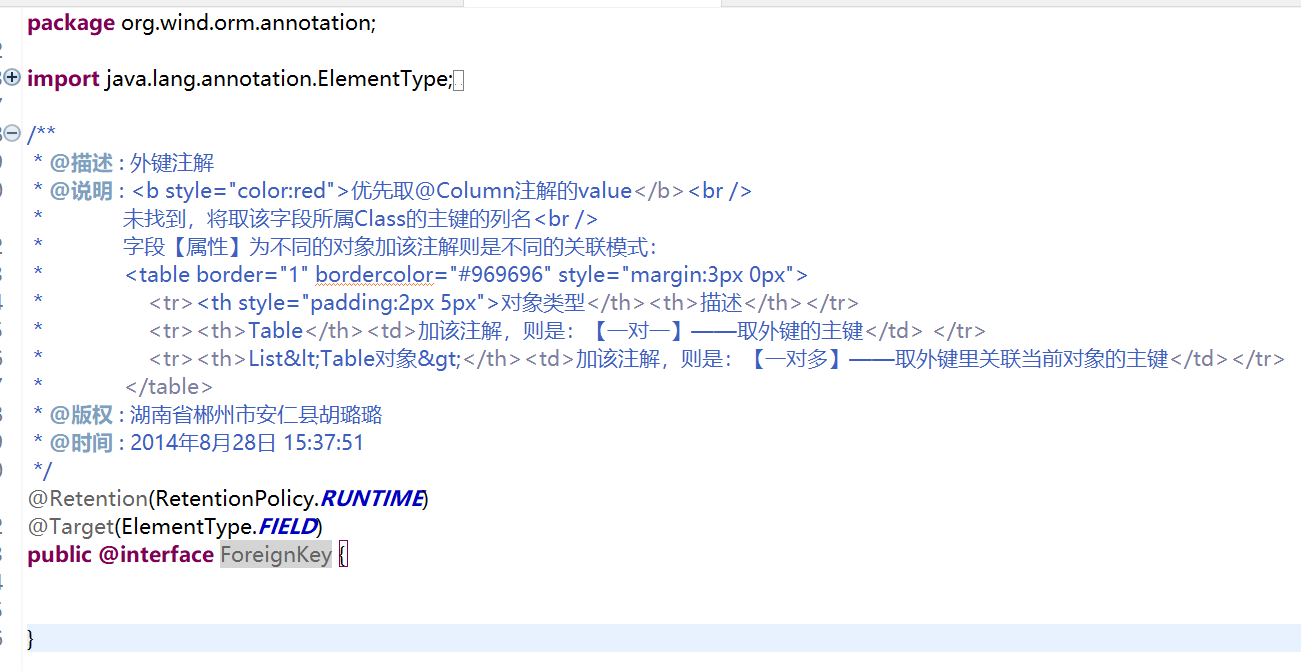
**外键关系**：

（1）、变量为单个自定义对象则为【一对一】的关系；

（2）、List类型的变量则为【一对多】

**注**： 加上该注解，则框架里的添删改查方法取@Column注解的value值做列名，未找到则取该变量的类的**@Id**注解代表列名

详情如下图：



#### 2.2.2.3 @Id

主键注解，标记该变量为主键字段。value值为主键生成策略，不设置value值，则取默认的策略：数据库自增长。**使用在字段（成员属性/变量）上**。详情如下图：



**使用方式**：

**@Id(Id.UUID)@Column("ID\_UUID")**

**主键生成策略**：

1. **autoIncrement**：默认。数据库自增
2. **UUID**：用于字符串类型，常见的唯一标识符，如： **c4275605-54de-4192-b493-757a8935abb2**
3. **sequence()**：指定的索引名（自增时，针对不支持自增，但支持索引的数据库），调用该索引的函数生成（例：oracle）

#### 2.2.2.4 @OrderBy

排序注解，表明当前字段做为默认的排序字段（分页时可用到），不加则取主键做为排序列。

详情如下图：

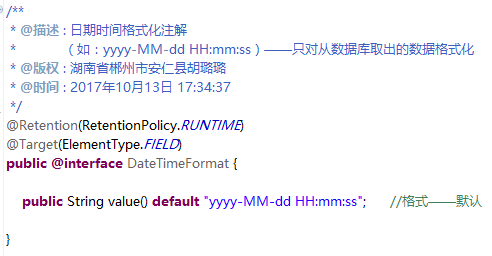


#### 2.2.2.5 @DateTime

日期时间注解——如果是日期时间相关的注解，请务必加上，并且接受者必须是String类型，否则将会有不可预支的错误。可对从数据库取出的数据格式化 ，格式为常见的如：yyyy-MM-dd HH:mm:ss、yyyy-MM-dd、HH:mm:ss等

原类名为：**DateTimeFomat**

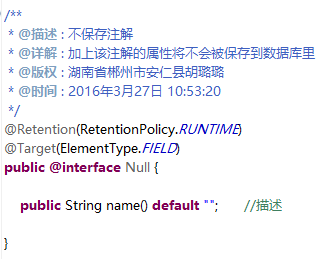
详情如下图：



#### 2.2.2.6 @Null

不保存注解，在进行save()方法操作时，不保存该注解绑定的成员变量（属性、字段）映射的列。目前该注解暂时没任何作用

详情如下图：



#### 2.2.2.7 @Value

值注解。主要针对于判断条件、插入、更新、删除的值SQL的替换。

如：Value内容为：TO\_DATE(?, 'YYYY-MM-DD HH24:MI:SS')

**条件** : where create\_time>TO\_DATE(?, 'YYYY-MM-DD HH24:MI:SS')

**插入** : insert into student(create\_time) values(TO\_DATE(?, 'YYYY-MM-DD HH24:MI:SS'))

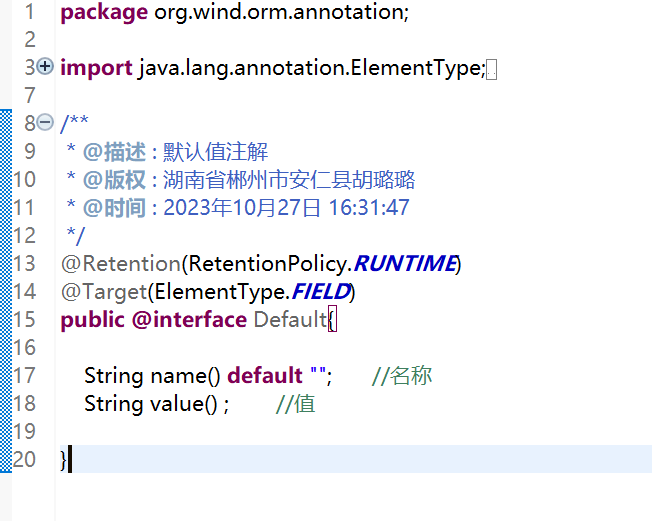
详情如下图：



**注意**：若字段加了@Id注解，则该注解失效

#### 2.2.2.8 @Default

默认值注解，Config相关类的字段配置默认值



# 核心方法（Table）

## 3.1 对象方法

生成1个对象（如：Student t\_obj=new Student();），通过t\_obj变量去调用下面的方法

### 3.1.1查询

#### 3.1.1.1 find()

查询：根据ORM的字段是否存在值来加条件（为null的不加入，不排除字符串为空字符串的）；排序：升序，不分页

|  |  |
| --- | --- |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回泛型，根据接收值的类型动态返回，列表式 |
| 示例： | Student t\_obj=new Student();  t\_obj.setName(“张三”);  t\_obj.setAge(26);  List<Student> objList = t\_obj.find(); |

#### 3.1.1.2 find(String,Page)

查询：根据ORM的字段是否存在值来加条件（为null的不加入，不排除字符串为空字符串的）；

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、isDesc： boolean类型，是否降序  2、page：Page类型，分页对象 |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回泛型，根据接收值的类型动态返回，列表式 |
| 示例： | Student t\_obj=new Student();  t\_obj.setName(“张三”);  t\_obj.setAge(26);  List<Student> objList = t\_obj.find(true,new Page(1,10)); |

#### 3.1.1.3 findOne(boolean)

查询：根据ORM的字段是否存在值来加条件，只查1个，其他说明和find()一样

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、isDesc： boolean类型，是否降序 |
| 返回值： | <T **extends** Table> T |
| 说明： | 返回泛型，根据接收值的类型动态返回，单个对象式 |
| 示例： | Student t\_obj=new Student();  t\_obj.setName(“张三”);  t\_obj.setAge(26);  Student obj\_new = t\_obj.findOne(false); |

#### 3.1.1.4 find(String,List,boolean,Page)

查询：根据条件（如果有），不根据对象的属性是否有值。

可设置排序（{@link #setOrderFieldName(String) }）。

可设置分组（{@link #setGroupFieldName(String)}）

可设置显示的字段（{@link #addSpecifiedFieldName(String...)}）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. conditionsSQL：String类型，条件SQL 2. conditionsList：List<? extends Object>类型，条件值 3. isDesc：是否降序 4. page：分页对象 |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回泛型，根据接收值的类型动态返回，单个对象式 |
| 示例： | Student t\_obj=new Student();  t\_obj. setOrderFieldName(“age”); //年龄排序  t\_obj. setGroupFieldName (“name”); //姓名分组  t\_obj. addSpecifiedFieldName (“name”,”age”,”weight”,”height”); //显示：姓名、年龄、体重、身高。若要显示更多的，则继续以字符串往后面添加  String conditionsSQL=” birthday=? and class =?”;  List<Object> conditionsList=new ArrayList<Object>();  conditionsList.add(“1986-06-13”); //出生日期  conditionsList.add(“157”); //班级  List<Student> objList=t\_obj.find(conditionsSQL,conditionsList,true,new Page(3,25)); |

#### 3.1.1.5 findSize()

查询 : 总数，根据ORM的字段是否存在值来加条件

|  |  |
| --- | --- |
| 返回值： | **long** |
| 说明： | 返回long型，根据接收值的类型动态返回，单个对象式 |
| 示例： | Student t\_obj=new Student();  t\_obj.setName(“张三”);  t\_obj.setAge(26);  long size = t\_obj.findSize(); |

#### 3.1.1.6 findSizeMap(String)

查询 : 分组总数，根据ORM的字段是否存在值来加条件

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、groupFieldName：String类型，指定的分组字段（属性、变量）名称 |
| 返回值： | <T> Map<T,Long> |
| 说明： | 返回泛型Map，根据groupFieldName对应的字段类型做key，value为总数 |
| 示例： | Student t\_obj=new Student();  t\_obj.setName(“张三”);  Map<String,Long> sizeMap=t\_obj.findSizeMap(“name”); |

### 3.1.2保存（插入、更新）

#### 3.1.2.1 save()

保存：（更新：当主键变量存在值时；插入：主键变量为null时候。插入成功后，会把主键【设置】到该【对象映射】的【主键变量】里）——对象式

|  |  |
| --- | --- |
| 返回值： | int |
| 说明： | 返回影响的行数，同数据库插入、更新操作 |
| 示例： | //插入  Student t\_obj=new Student();  t\_obj.setName(“张三”);  int result=t\_obj.save();  //更新（因为是同1个对象，且已插入成功，有主键值了）  t\_obj.setName(“李四”);  result=t\_obj.save(); |

#### 3.1.2.2 save(boolean,boolean)

保存：手动选择式

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. isAdd：boolean类型，是否为添加（插入），反之为更新 2. isInsertAll：boolean类型，插入时，是否插入全部数据（含：主键） |
| 返回值： | int |
| 说明： | 返回影响的行数，同数据库插入、更新操作 |
| 示例： | //插入  Student t\_obj=new Student();  t\_obj.setName(“张三199”);  int result=t\_obj.save();result=t\_obj.save(true,true); |

### 3.1.3删除

#### 3.1.3.1 delete()

删除：根据当前ORM对象的变量是否有值

|  |  |
| --- | --- |
| 返回值： | int |
| 说明： | 返回影响的行数，同数据库删除操作 |
| 示例： | Student t\_obj=new Student();  t\_obj.setName(“张三”);  int result=t\_obj.delete(); //删除name为“张三”的所有数据 |

### 3.1.4其他

#### 3.1.4.1 setOrderFieldName(String)

设置排序的字段

orderFieldName=ORM类的字段（成员属性）名称

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、orderFieldName：String类型，指定的排序字段（属性、变量）名称 |
| 返回值： | void |
| 说明： | 执行对象式查询时，单独设置的排序字段映射的列，因为有@OrderBy默认排序注解，所以此处设置的优先使用 |
| 示例： | Student t\_obj=new Student();  t\_obj.setOrderFieldName(“age”); //对age字段排序 |

#### 3.1.4.2 getOrderFieldName()

获取当前调用者对象设置的排序字段名称，此方法配合了上面的setOrderFieldName(String)方法使用

#### 3.1.4.3 setGroupFieldName(String)

设置分组的字段

groupFieldName=ORM类的字段（成员属性）名称

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、groupFieldName：String类型，指定的分组字段（属性、变量）名称 |
| 返回值： | void |
| 说明： | 执行对象式查询时，单独设置的分组字段映射的列 |
| 示例： | Student t\_obj=new Student();  t\_obj.setGroupFieldName(“name”); //对name字段分组 |

#### 3.1.4.4 getGroupFieldName()

获取当前调用者对象设置的分组字段名称，此方法配合了上面的setGroupFieldName(String)方法使用

#### 3.1.4.5 addSpecifiedFieldName(String…)

一次性加入多个显示的字段

specifiedFieldNameArr=ORM类的字段（成员属性）名称——可变式数组

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、specifiedFieldNameArr：String…类型，指定的一批显示的字段（属性、变量）名称 |
| 返回值： | void |
| 说明： | 在执行查询时，不会显示指定的这些字段映射的列（主键除外）  可重复多次调用该方法。不会覆盖前面已加入的。并且会去重，内部是用Set<String>装入 |
| 示例： | Student t\_obj=new Student();  t\_obj.addSpecifiedFieldName(“name”,”age”,”weight”,“school”); |

#### 3.1.4.6 clearSpecifiedFieldName()

清除所有指定显示的列名映射的字段（成员属性）名称

#### 3.1.4.7 getSpecifiedColumnSet()

获取当前对象设置的指定显示的字段的名称列表

|  |  |
| --- | --- |
| 返回值： | Set<String> |
| 示例： | Student t\_obj=new Student();  t\_obj.addSpecifiedFieldName(“name”,”age”,”weight”,“school”);  Set<String> t\_set=t\_obj.getSpecifiedColumnSet(); //返回的是上面的字段 |

#### 3.1.4.8 setOrderSQL(String)

设置排序的SQL（同功能时优先）

orderSQL=排序部分的SQL代码，不需要order by关键字

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、orderSQL：String类型，指定的排序部分的SQL，不需要order by关键字 |
| 返回值： | void |
| 说明： | 执行对象式查询时，单独设置的排序部分的SQL |
| 示例： | Student t\_obj=new Student();  t\_obj.setOrderSQL(“age desc,sex asc”); |

#### 3.1.4.9 getOrderSQL()

获取当前对象设置的排序SQL，此方法配合了上面的setOrderSQL(String)方法使用

#### 3.1.4.10 setGroupSQL(String)

设置分组的SQL（同功能时优先）

groupSQL=排序部分的SQL代码，不需要group by关键字

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、groupSQL：String类型，指定的分组部分的SQL，不需要group by关键字 |
| 返回值： | void |
| 说明： | 执行对象式查询时，单独设置的分组部分的SQL |
| 示例： | Student t\_obj=new Student();  t\_obj.setGroupSQL(“age,sex”); //分组SQL全文：group by age,sex |

#### 3.1.4.11 getGroupSQL()

获取当前对象设置的分组SQL，此方法配合了上面的setGroupSQL(String)方法使用

#### 3.1.4.12 setDataSource(String)

设置当前对象使用的数据源ID（同功能时优先）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、dataSource：String类型，指定的分组部分的SQL，不需要group by关键字 |
| 返回值： | void |
| 说明： | 执行对象式查询时，所使用的数据源 |
| 示例： | Student t\_obj=new Student();  t\_obj. setDataSource(“db\_sys”); |

#### 3.1.4.13 getDataSource()

获取当前对象所使用的数据源ID，此方法配合了上面的setDataSource(String)方法使用

#### 3.1.4.14 setTable(String)

设置当前对象使用的表名（默认：当前类的表名）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、table：String类型，指定使用的表名 |
| 返回值： | void |
| 示例： | Student t\_obj=new Student();  t\_obj. setTable(“student\_123”); |

#### 3.1.4.15 getTable()

获取当前对象使用的表名（默认：当前类的表名），此方法配合了上面的setTable(String)方法使用

## 3.2 静态方法

直接通过**Table**类调用，如：Table.findById(Student.class,1579);

### 3.2.1 getInstance(Class)

获取动态实例化Table对象，并返回

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1、tableClass：Class<? extends Table> 类型，继承Table的Model类 |
| 返回值： | Table |
| 说明： | 返回Table对象，实际类为tableClass参数指定的类 |
| 示例： | Table t\_obj=getInstance(Student.class); |

### 3.2.2查询

#### 3.2.2.1 findById(Class,Object,String…)

查询：根据tableClass所属对象映射的表的ID的1个对象，根据主键查询

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. id：Object类型，主键值 3. specifiedFieldNameArr：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要显示的列（属性、变量）映射的字段名称。**可不填，主键是必定会返回的** |
| 返回值： | <T **extends** Table> T |
| 说明： | 返回tableClass指定的类的1个对象 |
| 示例： | Student t\_obj=Table.findById(Student.class, 2,”name”,”age”,”height”); |

#### 3.2.2.2 findAll(Class,boolean)

查询：根据tableClass所属对象映射的表的所有对象，List装入

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. isDesc：boolean类型，是否降序 |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回tableClass指定的类的所有对象 |
| 示例： | List<Student> t\_objList=Table.findAll(Student.class,true); |

#### 3.2.2.3 findAll(Class,boolean,Page)

查询：根据tableClass所属对象映射的表的所有对象，带分页，List装入

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. isDesc：boolean类型，是否降序 3. page：Page类型，分页对象 |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回tableClass指定的类的，指定页数的对象 |
| 示例： | Page page=new Page(2,25);  List<Student> t\_objList=Table.findAll(Student.class,true,page); |

#### 3.2.2.4 findAll(Class,boolean,Page,String,String,String…)

查询：根据tableClass所属对象映射的表的所有对象，带分页，List装入

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. isDesc：boolean类型，是否降序 3. page：Page类型，分页对象 4. orderFieldName：String类型，指定排序的列名映射的字段名，为空则取主键列名 5. groupFieldName：String类型，指定分组的列名映射的字段名 6. specifiedFieldNameArr：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要显示的列（属性、变量）映射的字段名称。**可不填，主键是必定会返回的** |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回tableClass指定的类的，指定页数，指定显示的字段的对象 |
| 示例： | Page page=new Page(2,25);  List<Student> t\_objList=Table.findAll(Student.class,true,page,”age”,”name”,”name”,”age”,”height”,”width”); |

#### 3.2.2.5 find(Class,String,List,boolean,Page)

查询：根据传来的条件语句，List装入，带Page分页对象、条件；isDesc=是否降序；page=分页对象；

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<Object>类型，条件占位符的值 4. isDesc：boolean类型，是否降序 5. page：Page类型，分页对象 |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回tableClass指定的类的，指定条件，指定页数的对象 |
| 示例： | Page page=new Page(2,25);  String tjSQL=”name like ? and age=?”;  List<Object> tjList=new ArrayList<Object>();  tjList.add(“%王五%”);  tjList.add(27);  List<Student> t\_objList=Table.find(Student.class,tjSQL,tjList,false,page); |

#### 3.2.2.6 find(Class,String,List,boolean,Page,String,String,String…)

查询：根据传来的条件语句，List装入，带Page分页对象、条件；isDesc=是否降序；page=分页对象；orderFieldName=排序字段；groupFieldName=分组字段；specifiedFieldNameArr=显示的字段（非必须）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<Object>类型，条件占位符的值 4. isDesc：boolean类型，是否降序 5. page：Page类型，分页对象 6. orderFieldName：String类型，指定排序的列名映射的字段名，为空则取主键列名 7. groupFieldName：String类型，指定分组的列名映射的字段名 8. specifiedFieldNameArr：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要显示的列（属性、变量）映射的字段名称。**可不填，主键是必定会返回的** |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回tableClass指定：类、条件、页数、显示的字段的对象 |
| 示例： | Page page=new Page(2,25);  String tjSQL=”name like ? and age=?”;  List<Object> tjList=new ArrayList<Object>();  tjList.add(“%王五%”);  tjList.add(27);  List<Student> t\_objList=Table.find(Student.class, tjSQL, tjList,true,page,”age”,”name”,”name”,”age”,”height”,”width”); |

#### 3.2.2.7 findOne(Class,String,Object[],boolean)

查询：1个对象，根据指定的继承Table的Model类，指定的条件

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsArr：Object[]类型，条件占位符的值（数组式） 4. isDesc：boolean类型，是否降序（因为可能有多个） |
| 返回值： | <T **extends** Table> T |
| 说明： | 返回tableClass指定：类、条件的对象 |
| 示例： | String tjSQL=”name like ? and age=?”;  Object tjList[]={“%王五%”, 27};  Student obj =Table.findOne(Student.class,tjSQL,tjList,true); |

#### 3.2.2.8 findOne(Class,String,Object[],boolean,String…)

查询：1个对象，根据指定的继承Table的Model类，指定的条件，指定显示的字段

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsArr：Object[]类型，条件占位符的值（数组式） 4. isDesc：boolean类型，是否降序（因为可能有多个） 5. specifiedFieldNameArr：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要显示的列（属性、变量）映射的字段名称。**可不填，主键是必定会返回的** |
| 返回值： | <T **extends** Table> T |
| 说明： | 返回tableClass指定：类、条件的对象 |
| 示例： | String tjSQL=”name like ? and age=?”;  Object tjList[]={“%王五%”, 27};  Student obj =Table.findOne(Student.class,tjSQL,tjList,true,”name”,”age”); |

#### 3.2.2.9 findOne(Class,String,List,boolean,String…)

查询：1个对象，根据指定的继承Table的Model类，指定的条件，指定显示的字段

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符的值 4. isDesc：boolean类型，是否降序（因为可能有多个） 5. specifiedFieldNameArr：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要显示的列（属性、变量）映射的字段名称。**可不填，主键是必定会返回的** |
| 返回值： | <T **extends** Table> T |
| 说明： | 返回tableClass指定：类、条件的对象 |
| 示例： | String tjSQL=”name like ? and age=?”;  List<Object> tjList=new ArrayList<Object>();  tjList.add(“%王五%”);  tjList.add(27);  Student obj=Table.findOne(Student.class,tjSQL,tjList,true,”name”,”age”); |

#### 3.2.2.10 findByIdList(Class,List, String…)

查询：根据指定的idList主键列表值，可指定显示的字段

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. idList：List<Object> 类型，主键列表值 3. specifiedFieldNameArr：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要显示的列（属性、变量）映射的字段名称。**可不填，主键是必定会返回的** |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回tableClass指定：类、主键列表值的对象 |
| 示例： | List<Object> idList=new ArrayList<Object>();  idList.add(27);  idList.add(18);  idList.add(153);  List<Student> objList=Table.findIdList(Student.class,idList,”name”,”height”,”weight”); |

#### 3.2.2.11 findByNotIdList(Class,List, String...)

查询：根据tableClass所属对象映射的表的，除指定主键列表之外的其他数据，可指定显示的字段

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. idList：List<Object> 类型，主键列表值 3. specifiedFieldNameArr：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要显示的列（属性、变量）映射的字段名称。**可不填，主键是必定会返回的** |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回tableClass指定：类、主键列表值的对象 |
| 示例： | List<Object> idList=new ArrayList<Object>();  idList.add(27);  idList.add(18);  idList.add(153);  List<Student> objList=Table.findByNotIdList(Student.class,idList, ”height”,”weight”); |

#### 3.2.2.12 findAllSize(Class)

查询 : 根据tableClass所属对象映射的表的数据总数

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 |
| 返回值： | **long** |
| 说明： | 返回tableClass指定：类的数据总数 |
| 示例： | long size=Table.findAllSize(Student.class); |

#### 3.2.2.13 findDate(Class)

查询 : 当前数据库日期时间

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 |
| 返回值： | String |
| 说明： | 返回tableClass指定类绑定数据源的当前数据库时间，  格式（yyyy-MM-dd HH:ss:mm） |
| 示例： | String dateTime=Table.findDate(Student.class); |

#### 3.2.2.14 findIdList(Class,String,List)

查询 : 主键列表（条件为空则不查询）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<Object>类型，条件值 |
| 返回值： | <T> List<T> |
| 说明： | 返回tableClass指定类，根据条件的**主键**列表 |
| 示例： | String tjSQL=”name like ? and age=?”;  List<Object> tjList=new ArrayList<Object>();  tjList.add(“%王五%”);  tjList.add(27);  List<Long> idList=Table.findIdList(Student.class,tjSQL,tjList); |

#### 3.2.2.15 findAllIdList(Class)

查询 : 所有主键——List式

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 |
| 返回值： | <T> List<T> |
| 说明： | 返回tableClass指定类所有**主键**列表 |
| 示例： | List<Long> idList=Table.findAllIdList(Student.class); |

#### 3.2.2.16 findSpecifiedList(Class,String,List,String)

查询 : 指定列的值列表（条件为空则不查询）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件值 4. specifiedFieldName：String类型，指定要显示的列映射的字段名称 |
| 返回值： | <T> List<T> |
| 说明： | 返回tableClass指定类，指定显示的字段映射的列的值，默认为：主键 |
| 示例： | String tjSQL=”age=?”;  List<Object> tjList=new ArrayList<Object>();  tjList.add(27);  List<String> idList=Table.findSpecifiedList(Student.class,tjSQL,tjList,”name”); |

#### 3.2.2.17 find(Class,String,List)

查询 : 根据传来的SQL（返回的最顶层的List为行，Map为每一行所有的列名【key】和值【value】）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类，用来取数据源 2. sql：String类型，全SQL 3. placeholderList：List<Object>类型，占位符的值 |
| 返回值： | List<Map<String,Object>> |
| 说明： | 返回tableClass指定的数据源，根据全sql返回的的行列。  最外层的List为行，Map为每行的显示的列名（key，可别名）与值（value）  **注**：value类型是数据库与JDBC对应的类型，为了更好的匹配对应的类型。建议调用【toString()】方法，然后转成需要的类型 |
| 示例： | String sql=”select name,age,weight from t\_student where name like ? and age=?”;  List<Object> placeholderList=new ArrayList<Object>();  placeholderList.add(“%朱六%”);  placeholderList.add(27);  List<Map<String,Object>> mapList=Table.findSpecifiedList(Student.class,sql,placeholderList);  for(int i=0;i<mapList.size();i++){  Map<String,Object> t\_map=mapList.get(i);  String t\_name=map.get(“name”).toString();  Integer t\_name= Integer.parseInt(map.get(“age”).toString());  } |

#### 3.2.2.18 findSize(Class,String,List)

查询 : 总数，conditionsSQL=条件SQL；conditionsList=条件值（List）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<Object>类型，条件值，占位符的值 |
| 返回值： | **long** |
| 说明： | 返回tableClass指定类，指定条件的总数 |
| 示例： | String tjSQL=”name like ? and age=?”;  List<Object> conditionsList =new ArrayList<Object>();  conditionsList.add(“%朱六%”);  conditionsList.add(27);  long size=Table.findSize(Student.class,tjSQL,conditionsList); |

#### 3.2.2.19 findSizeMap(Class,String,List,String)

查询 : 分组总数，conditionsSQL=条件SQL；conditionsList=条件值；groupFieldName=分组列（ORM类字段名映射的列名）——Map返回式

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件值，占位符的值 4. groupFieldName：String类型，指定分组的字段名称（映射的列） |
| 返回值： | <T> Map<T,Long> |
| 说明： | 返回tableClass指定类，指定条件，指定分组列的总数 |
| 示例： | String tjSQL=”name like ? and age=?”;  List<Object> conditionsList =new ArrayList<Object>();  conditionsList.add(“%朱六%”);  conditionsList.add(27);  Map<String,Long> sizeMap=Table.findSizeMap(Student.class,tjSQL,conditionsList,”name”);  Long t\_size=sizeMap.get(“张三”); //取出姓名为“张三”的总数 |

#### 3.2.2.20 findAllForeign(Table,boolean,Page)

查询 : 所有外键数据

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. obj：Table类型，继承Table的Model对象 2. isDesc：boolean类型，是否降序，针对的是【一对多】的数据 3. page：Page类型，分页对象，针对的是【一对多】的数据 |
| 返回值： | void |
| 说明： | 根据传址逻辑，会动态的把查询到的外键设置到obj对应的外键（@ForeignKey）对象 |
| 示例： | Student obj=Table.findById(Student.class,157);  Page page=new Page(2,10);  Table.findAllForeign(obj,false,page); //查询所有外键，包含一对多 |

#### 3.2.2.21 findForeign(Table,boolean,Page,String...)

查询 : 外键数据（外键里的外键对象只留下id），根据foreign指定的外键字段（成员变量）名——单对象式

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. obj：Table类型，继承Table的Model对象 2. isDesc：boolean类型，是否降序，针对的是【一对多】的数据 3. page：Page类型，分页对象，针对的是【一对多】的数据 4. foreign：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要查询的外键的字段名称 |
| 返回值： | void |
| 说明： | 根据传址逻辑，会动态的把查询到的外键设置到obj对应的外键（@ForeignKey）对象 |
| 示例： | Student obj=Table.findById(Student.class,157);  Page page=new Page(2,10);  Table.findForeign(obj,false,page,”sex”,”class”,”courseList”); //查询指定的外键。包含courseList（一对多），是个List类型 |

#### 3.2.2.22 findForeign(Table,String...)

查询 : 外键数据（外键里的外键对象只留下id），根据foreign指定的外键字段（成员变量）名——单对象式

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. obj：Table类型，继承Table的Model对象 2. foreign：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要查询的外键的字段名称，一对多的外键则全查出来。 |
| 返回值： | void |
| 说明： | 根据传址逻辑，会动态的把查询到的外键设置到obj对应的外键（@ForeignKey）对象 |
| 示例： | Student obj=Table.findById(Student.class,157);  Table.findForeign(obj,”sex”,”class”,”courseList”); //查询指定的外键。包含courseList（一对多），是个List类型 |

#### 3.2.2.23 findForeign(List,boolean,Page,String[],String...)

查询 : 外键数据（外键里的外键对象只留下id），根据foreign指定的外键字段（成员变量）名，指定显示的字段——列表式

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<? extends Table>类型，继承Table的Model对象列表 2. isDesc：boolean类型，是否降序，针对的是【一对多】的数据 3. page：Page类型，分页对象，针对的是【一对多】的数据 4. specifiedFieldNameArr[]：String数组类型，指定外键要显示的字段列表（请传成员变量名，建议对取单个外键的方式使用），数组式，为NULL，则显示所有字段 5. foreign：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要查询的外键的字段名称 |
| 返回值： | void |
| 说明： | 根据传址逻辑，会动态的把查询到的外键设置到list列表对应的外键（@ForeignKey）对象 |
| 示例： | List<Student> objList=Table.findAll(Student.class);  Page page=new Page(2,10);  //查询指定的外键。包含courseList（一对多），是个List类型  Table.findForeign(objList,false,page,new String[]{“name”},”sex”,”courseList”); |

#### 3.2.2.24 findForeign (List,String…)

查询 : 所有外键数据（外键里的外键对象只留下id）——列表式

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<? extends Table>类型，继承Table的Model对象列表 2. foreign：String可变式数组类型，无限参数，可往后面持续累增字符串。指定要查询的外键的字段名称，一对多的外键则全查出来。 |
| 返回值： | void |
| 说明： | 根据传址逻辑，会动态的把查询到的外键设置到list列表对应的外键（@ForeignKey）对象 |
| 示例： | // objList=Table对象列表  Table.findForeign(objList,”sex”,”class”,”courseList”); //查询指定的外键。包含courseList（一对多），是个List类型 |

#### 3.2.2.25 findAllForeign(List,boolean,Page)

查询 : 所有外键数据（外键里的外键对象只留下id）——列表式

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<? extends Table>类型，继承Table的Model对象列表 2. isDesc：boolean类型，是否降序，针对的是【一对多】的数据 3. page：Page类型，分页对象，针对的是【一对多】的数据 |
| 返回值： | void |
| 说明： | 根据传址逻辑，会动态的把查询到的外键设置到list列表对应的外键（@ForeignKey）对象 |
| 示例： | List<Student> objList=Table.findAll(Student.class);  Page page=new Page(2,10);  Table.findAllForeign(objList,false,page); //查询所有的外键。包含【一对多】，是个List类型 |

#### 3.2.2.26 findOneForeign(Table,String,String...)

查询：单个外键（指定foreign单个外键字段，specifiedFieldNameArr=外键要显示的字段列表，可变式数组；单对象式）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. obj：Table类型，继承Table的Model对象 2. foreign：String类型，指定要查询的外键的字段名称，一对多的外键则全查出来。 3. specifiedFieldNameArr：String可变式数组类型，无限参数，可往后面持续累增字符串。指定外键要显示的字段列表（请传成员变量名，建议对取单个外键的方式使用），为NULL，则显示所有字段 |
| 返回值： | void |
| 说明： | 根据传址逻辑，会动态的把查询到的外键设置到obj对应的外键（@ForeignKey）对象 |
| 示例： | Table.findForeign(obj,“sex”,”name”,”code”); |

#### 3.2.2.27 findOneForeign(List,String,String...)

查询：单个外键（指定foreign单个外键字段，specifiedFieldNameArr=外键要显示的字段列表，可变式数组；列表式）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<? extends Table>类型，继承Table的Model对象列表 2. foreign：String类型，指定要查询的外键的字段名称，一对多的外键则全查出来。 3. specifiedFieldNameArr：String可变式数组类型，无限参数，可往后面持续累增字符串。指定外键要显示的字段列表（请传成员变量名，建议对取单个外键的方式使用），为NULL，则显示所有字段 |
| 返回值： | void |
| 说明： | 根据传址逻辑，会动态的把查询到的外键设置到list列表对应的外键（@ForeignKey）对象 |
| 示例： | Table.findForeign(objList,“sex”,”name”,”code”); |

### 3.2.3保存（插入、更新）

#### 3.2.3.1 save(List)

【自动选择】批量保存，根据

【注】：保存的所有对象是属于同一个类，以List第一个为基础

根据第1个对象的主键是否有值，判断是插入还是更新；

**插入**：主键没值（为NULL对象）

**更新**：主键有值（字符串为空””算有值）

插入成功后，会把主键值设置到对象里（标注了@Id注解的属性字段）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<? extends Table>类型，继承Table的Model对象列表 |
| 返回值： | **int** |
| 说明： | 返回影响的行数，类似于操作数据库的**插入更新**影响的行数，一般返回值大于0就是执行成功了。 |
| 示例： | List<Student> list=new ArrayList<Student>();  //第1条数据  Student t\_obj\_1=new Student();  t\_obj\_1.setName(“李1”);  t\_obj\_1.setAge(19);  //第2条数据  Student t\_obj\_2=new Student();  t\_obj\_2.setName(“孙2”);  t\_obj\_2.setAge(24);  //  list.add(t\_obj\_1);  list.add(t\_obj\_2);  int result=Table.save(list); |

#### 3.2.3.2 save(List,boolean,boolean)

【手动选择】批量保存

【注】：保存的所有对象是属于同一个类，以List第一个为基础

插入成功后，会把主键值设置到对象里（标注了@Id注解的属性字段）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<? extends Table>类型，继承Table的Model对象列表 2. isAdd：boolean类型，是否插入。为false则为更新 3. isInsertAll：boolean类型，是否插入全部值（主要用于主键有值的情况下，做插入操作） |
| 返回值： | **int** |
| 说明： | 返回影响的行数，类似于操作数据库的**插入更新**影响的行数，一般返回值大于0就是执行成功了。 |
| 示例： | List<Student> list=new ArrayList<Student>();  //第1条数据  Student t\_obj\_1=new Student();  t\_obj\_1.setId(259);  t\_obj\_1.setName(“李1”);  t\_obj\_1.setAge(19);  //第2条数据  Student t\_obj\_2=new Student();  t\_obj\_1.setId(328);  t\_obj\_2.setName(“孙2”);  t\_obj\_2.setAge(24);  //  list.add(t\_obj\_1);  list.add(t\_obj\_2);  int result=Table.save(list,true,true); |

#### 3.2.3.3 save(Class,String,List)

【自定义】保存

自己写插入、更新的全SQL

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<? extends Table>类型，继承Table的Model对象列表 2. sql：String类型，自写的插入、更新全SQL 3. placeholderList：List<Object>类型，占位符的值 |
| 返回值： | **int** |
| 说明： | 返回影响的行数，类似于操作数据库的**插入更新**影响的行数，一般返回值大于0就是执行成功了。 |
| 示例： | String saveSQL=”insert into t\_student(name,age) values(?,?)”;  List<Object> placeholderList=new ArrayList<Object>();  list.add(“刘5”);  list.add(21);  int result=Table.save(Student.class,saveSQL,placeholderList); |

#### 3.2.3.4 update(Class,String,String,List)

更新 : 根据传来的set和条件SQL语句

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. setSQL：String类型，set语句，设置哪些列要改变的值 3. conditionsSQL：String类型，条件SQL 4. placeholderList：List<Object>类型，占位符值 |
| 返回值： | **int** |
| 说明： | 返回影响的行数，类似于操作数据库的**更新**影响的行数，一般返回值大于0就是执行成功了。 |
| 示例： | String setSQL=”name=?,age=?”;  String tjSQL=”class=?”;  List<Object> placeholderList=new ArrayList<Object>();  placeholderList.add(“张三2”);  placeholderList.add(17);  placeholderList.add(“应用0931”);  int result=Table.update(Student.class,setSQL,tjSQL,tjList); |

### 3.2.4自增/自减

#### 3.2.4.1 increaseOrDecrease(Class,String,double,Object)

自增减（根据主键ID）

**自增**：step为正数

**自减**：step为负数

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. fieldName：String类型，指定要自增减的字段名称 3. step：double类型，自增减的步长值 4. id：Object类型，指定要自增减的主键值 |
| 返回值： | **int** |
| 说明： | 返回影响的行数，类似于操作数据库的**更新**影响的行数，一般返回值大于0就是执行成功了。 |
| 示例： | //自减年龄2岁  int result=Table.increaseOrDecrease(Student.class,”age”,-2,159);  //自增身高10.5  Table.increaseOrDecrease(Student.class,”height”,10.5,159); |

#### 3.2.4.2 increaseOrDecrease(Class,String,double,String,List)

自增减（带条件）

**自增**：step为正数

**自减**：step为负数

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. fieldName：String类型，指定要自增减的字段名称 3. step：double类型，自增减的步长值 4. conditionsSQL：String类型，条件SQL 5. conditionsList：List<Object>类型，条件占位符值 |
| 返回值： | **int** |
| 说明： | 返回影响的行数，类似于操作数据库的**更新**影响的行数，一般返回值大于0就是执行成功了。 |
| 示例： | String tjSQL=”name=?”;  List<Object> tjList=new ArrayList<Object>();  tjList.add(“张三”);  //自增身高10.5  Table.increaseOrDecrease(Student.class,”height”,10.5,tjSQL,tjList); |

### 3.2.5删除

#### 3.2.5.1 delete(Class,Object)

删除：根据tableClass所属对象映射的表，删除数据库存在的主键值集的数据。**动态式删除法**

可主键类型：String、Integer、Long、List<【主键类型】> 等等

若为List类型则是**批量删除**

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. ids：Object类型，可主键类型：String、Integer、Long、List<【主键类型】> 等等 |
| 返回值： | **int** |
| 说明： | 返回影响的行数，类似于操作数据库的**删除**影响的行数，一般返回值大于0就是执行成功了。没有就返回0 |
| 示例： | List<Object> idList=new ArrayList<Object>();  idList.add(159);  idList.add(362);  Table.delete(Student.class,idList); //根据主键列表，批量删除  Table.delete(Student.class,143); //根据单个主键，删除单个 |

#### 3.2.5.2 delete(Class)

删除全部：根据当前tableClass所映射的表，删除该映射表所有的数据

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 |
| 返回值： | **int** |
| 说明： | 返回影响的行数，类似于操作数据库的**删除**影响的行数，一般返回值大于0就是执行成功了。 |
| 示例： | int result=Table.delete(Student.class); |

#### 3.2.5.3 delete(Class,String,List)

删除：根据传来的条件语句

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<Object>，条件占位符值 |
| 返回值： | **int** |
| 说明： | 返回影响的行数，类似于操作数据库的**删除**影响的行数，一般返回值大于0就是执行成功了。 |
| 示例： | String tjSQL=”name=?”;  List<Object> tjList=new ArrayList<Object>();  tjList.add(“李四”);  int result=Table.delete(Student.class,tjSQL,tjList); |

### 3.2.6 DDL（数据库模式定义语言）

#### 3.2.6.1 call(Class,String,Object[],Integer)

调用 : 存储过程（returnType不为空返回一个值）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. proc：String类型，存储过程名称 3. paramArr：Object[]类型，参数数组（可为空） 4. returnType：Integer类型，返回值类型（可为空，请用java的java.sql.Types包下的类型，如：java.sql.Types.BIGINT） |
| 返回值： | <T> T |
| 说明： | 返回：调用存储过程返回的数据，类型为泛型，根据实际返回的值的类型接受，如：调用orderId()过程，返回个整型订单号，那么接受值类型为：Integer或Long |
| 示例： | Object paramArr=new Object[2];  paramArr[0]=”张三”;  paramArr[1]=25;  int studentId=Table.call(Student.class,”generateStudentId”,paramArr,java.sql.Types.INTEGER); //生成该学生的学号 |

#### 3.2.6.2 callList(Class,String,Object[],Integer[])

调用 : 存储过程并返回所有值（根据returnTypeArr数组）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. proc：String类型，存储过程名称 3. paramArr：Object[]类型，参数数组（可为空） 4. returnType：Integer[]类型，返回值类型数组（可为空，请用java的java.sql.Types包下的类型，如：java.sql.Types.BIGINT） |
| 返回值： | <T> T |
| 说明： | 返回：调用存储过程返回的数据，类型为泛型，根据实际返回的值的类型接受，如：调用orderIdAndSerialNumber()过程，返回个整型订单号和流水号，那么接受值类型为：Integer[]或Long[] |
| 示例： | Object paramArr=new Object[2];  paramArr[0]=”张三”;  paramArr[1]=25;  Object resultTypeArr=new Object[2];  resultTypeArr[0]=java.sql.Types.BIGINT;  resultTypeArr[1]=java.sql.Types.INTEGER;  List<Object> noList =Table.call(Student.class,”generateStudentIdAndClassNo”,paramArr,resultTypeArr); //生成该学生的学号和班号 |

#### 3.2.6.3 create(Class,String)

执行create语句

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类，这里主要是代表数据源 2. proc：String类型，create全SQL语句 |
| 返回值： | **boolean** |
| 说明： | 返回：执行结果，true=执行成功；false=失败 |
| 示例： |  |

#### 3.2.6.4 copy(Class,String,boolean)

复制**原表**到**新表**（含：表结构，数据**[可选]**）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类，这里既是代表数据源，也是代表原表（主表） 2. newTable：String类型，新的表名（**oracle**会转成大写表明） 3. isCopyData：boolean类型，是否复制数据 |
| 返回值： | **boolean** |
| 说明： | 返回：执行结果，true=执行成功；false=失败 |
| 示例： | **boolean** isSuccess\_copy=Table.copy(Sex.**class**, "sex3", **true**); |

#### 3.2.6.5 drop(Class,String)

执行drop语句（需要传drop全SQL语句，这里为了安全误操作而这样设计）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类，这里主要代表数据源 2. sql：String类型，drop全SQL语句 |
| 返回值： | **boolean** |
| 说明： | 返回：执行结果，true=执行成功；false=失败 |
| 示例： | **boolean** isSuccess\_drop=Table.drop(Sex.**class**, "drop table sex2"); |

#### 3.2.6.6 getTableSQL(Class)

获取 : 表SQL语句（不支持：SQL Server）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类，这里既代表数据源，也代表要获取的表SQL |
| 返回值： | String |
| 说明： | 返回：指定Model的建表DDL的SQL语句 |
| 示例： | String t\_tableSQL=Table.getTableSQL(Sex.**class**); |
|  |  |

#### 3.2.6.7 isTableExist (Class)

判断 : 当前tableClass对应的表是否存在

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类，这里既代表数据源，也代表要判断的表 |
| 返回值： | **boolean** |
| 说明： | 返回：执行结果，true=存在；false=不存在 |
| 示例： | **boolean** isExist =Table.isTableExist(Sex.**class**); |

#### 3.2.6.8 isTableExist (Class,String)

判断 : 指定的table表是否存在

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类，这里既代表数据源，也代表要判断的表 2. table：String类型，要判断的表名 |
| 返回值： | **boolean** |
| 说明： | 返回：执行结果，true=存在；false=不存在 |
| 示例： | **boolean** isExist =Table.isTableExist(Sex.**class**,”sex123”); |

#### 3.2.6.9 getColumn(Class)

获取列名集（真实的数据库列名），以Set<String>接收

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类，这里既代表数据源，也代表要获取的表 |
| 返回值： | Set<String> |
| 说明： | 如：id, createtime, updatetime, sexid, majorid, date, test2id, sexida |
| 示例： | Set<String> colSet=Table.*getColumn*(Student.**class**); |

### 3.2.7数学函数（sum、avg、max、min等）

#### 3.2.7.1 sum(Class,String)

求和：根据指定fieldName映射字段的所有数据【总数求和】，不带条件

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. fieldName：String类型，指定要求和的字段名称 |
| 返回值： | Number |
| 说明： | 返回：求和的总数，用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | Number t\_sum=Table.*sum*(Student.**class**,"age"); |

#### 3.2.7.2 sum(Class,String,List,String)

求和：带条件数据的总和，根据指定fieldName字段名称映射的列的数据

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求和的字段名称 |
| 返回值： | Number |
| 说明： | 返回：求和的总数，用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  Number t\_sum=Table.*sum*(Student.**class**,"sex=?",tjList,"age"); |

#### 3.2.7.3 sum(Class,String,List,String,String)

求和：根据指定的fieldName字段名称做【总数求和】，指定的groupFieldName字段的值做返回Map的key，value为每组的总数

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求和的字段名称 5. groupFieldName：String类型，分组字段名称，返回的Map的key是为该字段映射的列名的值（及类型） |
| 返回值： | <T> Map<T,Number> |
| 说明： | 返回：求和的总数Map，key=指定groupFieldName字段名称对应列的值，value用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  Map<Object,Number> t\_sumMap=Table.*sum*(Student.**class**,"sex=?", tjList, "age","sex"); |

#### 3.2.7.4 avg(Class,String)

求平均

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. fieldName：String类型，指定要求平均的字段名称 |
| 返回值： | Number |
| 说明： | 返回：求平均的数，用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | Number t\_avg=Table.avg(Student.**class**,"age"); |

#### 3.2.7.5 avg(Class,String,List,String)

求平均：带条件

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求平均的字段名称 |
| 返回值： | Number |
| 说明： | 返回：求平均的数，用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  Number t\_avg=Table.avg(Student.**class**,"sex=?",tjList,"age"); |

#### 3.2.7.6 avg(Class,String,List,String,String)

求平均：根据指定的fieldName字段名称做【总数求平均】，指定的groupFieldName字段的值做返回Map的key，value为每组的平均

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求平均的字段名称 5. groupFieldName：String类型，分组字段名称，返回的Map的key是为该字段映射的列名的值（及类型） |
| 返回值： | <T> Map<T,Number> |
| 说明： | 返回：求平均的数Map，key=指定groupFieldName字段名称对应列的值，value用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  Map<Object,Number> t\_avgMap=Table.avg(Student.**class**,"sex=?", tjList, "age","sex"); |

#### 3.2.7.7 max(Class,String)

求最大值

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. fieldName：String类型，指定要求最大值的字段名称 |
| 返回值： | Number |
| 说明： | 返回：求最大的数，用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | Number t\_max=Table.max(Student.**class**,"age"); //求最大年龄数 |

#### 3.2.7.8 max(Class,String,List,String)

求最大值：带条件

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求最大值的字段名称 |
| 返回值： | Number |
| 说明： | 返回：求最大的数，用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  Number t\_max=Table.max(Student.**class**,"sex=?",tjList,"age"); |

#### 3.2.7.9 max(Class,String,List,String,String)

求最大值：指定的groupFieldName字段的值做返回Map的key，value为每组的值

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求最大值的字段名称 5. groupFieldName：String类型，分组字段名称，返回的Map的key是为该字段映射的列名的值（及类型） |
| 返回值： | <T> Map<T,Number> |
| 说明： | 返回：求最大的数Map，key=指定groupFieldName字段名称对应列的值，value用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  Map<Object,Number> t\_maxMap=Table.max(Student.**class**,"sex=?", tjList, "age","name"); //指定的性别，每组姓名里，年龄最大的数 |

#### 3.2.7.10 maxString(Class,String)

求最大值（返回类型：字符串）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. fieldName：String类型，指定要求最大值的字段名称 |
| 返回值： | String |
| 说明： | 返回：求最大的数，String类型 |
| 示例： | String t\_max=Table.max(Student.**class**,"age"); //求最大年龄数 |

#### 3.2.7.11 maxString(Class,String,List,String)

求最大值：带条件（返回类型：字符串）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求最大值的字段名称 |
| 返回值： | String |
| 说明： | 返回：求最大的数，String类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  String t\_max=Table.max(Student.**class**,"sex=?",tjList,"age"); |

#### 3.2.7.12 maxString(Class,String,List,String,String)

求最大值：指定的groupFieldName字段的值做返回Map的key，value为每组的值（返回类型：字符串）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求最大值的字段名称 5. groupFieldName：String类型，分组字段名称，返回的Map的key是为该字段映射的列名的值（及类型） |
| 返回值： | <T> Map<T,String> |
| 说明： | 返回：求最大的数Map，key=指定groupFieldName字段名称对应列的值，value为String类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  Map<Object,String> t\_maxMap=Table.max(Student.**class**,"sex=?", tjList, "age","name"); //指定的性别，每组姓名里，年龄最大的数 |

#### 3.2.7.13 min(Class,String)

求最小值

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. fieldName：String类型，指定要求最小值的字段名称 |
| 返回值： | Number |
| 说明： | 返回：求最小的数，用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | Number t\_max=Table.max(Student.**class**,"age"); //求最小年龄数 |

#### 3.2.7.14 min(Class,String,List,String)

求最小值：带条件

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求最小值的字段名称 |
| 返回值： | Number |
| 说明： | 返回：求最小的数，用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  Number t\_max=Table.max(Student.**class**,"sex=?",tjList,"age"); |

#### 3.2.7.15 min(Class,String,List,String,String)

求最小值：指定的groupFieldName字段的值做返回Map的key，value为每组的值

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求最小值的字段名称 5. groupFieldName：String类型，分组字段名称，返回的Map的key是为该字段映射的列名的值（及类型） |
| 返回值： | <T> Map<T,Number> |
| 说明： | 返回：求最小的数Map，key=指定groupFieldName字段名称对应列的值，value用Number对象包装，可以根据需求调用不同的方法，返回不同的基本数类型，如：intValue()返回的是int类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  Map<Object,Number> t\_maxMap=Table.max(Student.**class**,"sex=?", tjList, "age","name"); //指定的性别，每组姓名里，年龄最小的数 |

#### 3.2.7.16 minString(Class,String)

求最小值（返回类型：字符串）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. fieldName：String类型，指定要求最小值的字段名称 |
| 返回值： | String |
| 说明： | 返回：求最小的数，String类型 |
| 示例： | String t\_max=Table.max(Student.**class**,"age"); //求最小年龄数 |

#### 3.2.7.17 minString(Class,String,List,String)

求最小值：带条件（返回类型：字符串）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求最小值的字段名称 |
| 返回值： | String |
| 说明： | 返回：求最小的数，String类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  String t\_max=Table.max(Student.**class**,"sex=?",tjList,"age"); |

#### 3.2.7.18 minString(Class,String,List,String,String)

求最小值：指定的groupFieldName字段的值做返回Map的key，value为每组的值（返回类型：字符串）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T> 类型，继承Table的Model类 2. conditionsSQL：String类型，条件SQL 3. conditionsList：List<? extends Object>类型，条件占位符值 4. fieldName：String类型，指定要求最小值的字段名称 5. groupFieldName：String类型，分组字段名称，返回的Map的key是为该字段映射的列名的值（及类型） |
| 返回值： | <T> Map<T,String> |
| 说明： | 返回：求最小的数Map，key=指定groupFieldName字段名称对应列的值，value为String类型 |
| 示例： | List<Object> tjList=**new** ArrayList<Object>();  tjList.add(2);  Map<Object,String> t\_maxMap=Table.max(Student.**class**,"sex=?", tjList, "age","name"); //指定的性别，每组姓名里，年龄最小的数 |

### 3.2.8其他

#### 3.2.8.1 setAutoCommit(Class,boolean)

设置 : 自动提交（会话）状态

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，继承Table的类 2. isAutoCommit：boolean类型，是否自定提交，true=是；false=不是 |
| 返回值： | void |
| 说明： | 跟JDBC的setAutoCommit()方法一样,设置为false则是开启了事务  true=自动提交；false=手动提交 |
| 示例： | Table.setAutoCommit(Student.class,false); |

#### 3.2.8.2 isAutoCommit(Class)

获取：指定继承Table的类绑定的数据源的自动提交状态。

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，继承Table的类 |
| 返回值： | boolean |
| 说明： | 返回：指定继承Table的类绑定的数据源的自动提交状态。  true=自动提交；false=手动提交 |
| 示例： | boolean isAutoCommit=Table.isAutoCommit(Student.class); |

#### 3.2.8.3 commit(Class)

提交事务（指定的数据源的事务）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，继承Table的类 |
| 返回值： | void |
| 说明： | 跟JDBC的commit()方法一样,手动提交事务  这里是手动提交指定的数据源（指定的tableClass会绑定数据源） |
| 示例： | Table.commit(Student.class); |

#### 3.2.8.4 setSavepoint(Class,String)

设置 : 事务保存点，并返回。若抛出{@link SQLFeatureNotSupportedException}，则表示驱动程序不支持此操作。

在当前事务中创建一个具有给定名称的保存点，并返回表示它的新 Savepoint 对象。

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，继承Table的类 2. name：String类型，设置的指定tableClass事务保存点的名称 |
| 返回值： | Savepoint（java.sql.Savepoint） |
| 说明： | 跟JDBC的Savepoint()方法一样,保存事务点，然后返回JDBC的保存事务点的对象，用来日后回滚到该事务点 |
| 示例： | Table.setSavepoint(Student.class,”transaction\_1”); |

#### 3.2.8.5 releaseSavepoint(Class,Savepoint)

移除指定的 Savepoint 和后续 Savepoint 对象。在已移除保存点之后，对该保存点的任何引用都会导致抛出 SQLException。

若抛出{@link SQLFeatureNotSupportedException}，则表示驱动程序不支持此操作。

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，继承Table的类，绑定的数据源 2. savepoint：Savepoint类型，之前保存的事务点返回的事务点对象 |
| 返回值： | void |
| 说明： | 跟JDBC的releaseSavepoint()方法一样,释放事务点。  从当前事务中移除指定的 Savepoint 和后续 Savepoint 对象。在已移除保存点之后，对该保存点的任何引用都会导致抛出 SQLException。 |
| 示例： | ………….  Table.releaseSavepoint(Student.class,savepoint); |

#### 3.2.8.6 rollback(Class)和rollback(Class,Savepoint)

回滚事务（{@link TableParent2#setSavepoint}，t\_savepoint=回滚至该事务保存点，为null则回滚所有）

不带Savepoint事务点对象的，则跟一般的JDBC回滚一样，回滚所有的插入、更新、删除点等

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，继承Table的类，绑定的数据源 2. savepoint：Savepoint类型，之前保存的事务点返回的事务点对象 |
| 返回值： | void |
| 说明： | 取消在当前事务中进行的所有更改，并释放此 Connection 对象当前持有的所有数据库锁。此方法只应该在已禁用自动提交模式时使用。  若带savepoint，则只取消到该事务点及以下的所有事务更改 |
| 示例： | ………….  Table.rollback(Student.class,savepoint);  Table.rollback(Student.class); |

#### 3.2.8.7 close(Class)

释放Connection连接到指定的tableClass绑定的数据源的连接池。

若超过配置文件的max最大保持连接数，则关闭该Connection

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，继承Table的类，绑定的数据源 |
| 返回值： | void |
| 说明： | 释放Connection连接到指定的tableClass绑定的数据源的连接池。  若超过配置文件的max最大保持连接数，则关闭该Connection |
| 示例： | Table.close(Student.class); |

#### 3.2.8.8 setTimeout(Class,int)

设置 : 执行单个SQL时的超时数（单位：秒），此为设置【单会话】的超时配置，并非全局。

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，继承Table的类，绑定的数据源 2. timeout：int类型，执行SQL超时的（单位：秒） |
| 返回值： | void |
| 说明： | 执行单个SQL时的超时数（单位：秒）。  因为wind-orm.xml有配置setTimeout的超时，因此，只对当前线程（会话）有效。若不调用该方法。则取wind-orm.xml的 |
| 示例： | Table.setTimeout(Student.class,30); |

#### 3.2.8.9 getTimeout(Class)

获取 : 执行单个SQL时的超时数（单位：秒），此为获取【单会话】的超时配置，并非全局，未设置则返回null

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，继承Table的类，绑定的数据源 |
| 返回值： | Integer |
| 说明： | 获取：执行单个SQL时的超时数（单位：秒）。  因为wind-orm.xml有配置setTimeout的超时，因此，只取当前线程（会话）的。若没有设置，则返回null |
| 示例： | Integer timeout\_current=Table.getTimeout(Student.class); |

# 工具方法

## 4.1 TableUtil（Table常用工具类）

### 4.1.1 getMethodMap\_set(Class)

获取 : 前缀为"set"的Method（缓存）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?>类型，指定要获取的类 |
| 返回值： | Map<String,Method> |
| 说明： | 返回tableClass所有（含：深层次父类）的Method（Map式），格式：【小写】方法名+所有参数类型名（按顺序）。  若缓存没有，将会执行寻找，并放入缓存 |
| 示例： | //获取Student所有的前缀为为“set”的方法Method  Map<String,Method> methodMap=TableUtil.getMethodMap\_set(Student.class); |

### 4.1.2 getMethodMap\_get(Class)

获取 : 前缀为"get"的Method（缓存）。

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?>类型，指定要获取的类 |
| 返回值： | Map<String,Method> |
| 说明： | 返回tableClass所有（含：深层次父类）的Method（Map式），格式：【小写】方法名+所有参数类型名（按顺序）。  若缓存没有，将会执行寻找，并放入缓存 |
| 示例： | //获取Student所有的前缀为为“get”的方法Method  Map<String,Method> methodMap=TableUtil.getMethodMap\_get(Student.class); |

### 4.1.3 getMethodMap(Class,Class,String)

获取 : 前缀为prefix的Method（Map式），含：深层次的父类

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?>类型，指定要获取的类 2. parentClass：停止寻找的父类（不处理该类） 3. prefix：String类型，方法名前缀，可模糊取prefix匹配的方法,传null则不验证 |
| 返回值： | Map<String,Method> |
| 说明： | 返回tableClass所有（含：深层次父类）的Method（Map式），格式：【小写】方法名+所有参数类型名（按顺序） |
| 示例： | Map<String,Method> methodMap=TableUtil.getMethodMap(Student.class,"set"); //获取Student所有的前缀为为“set”的方法Method |

### 4.1.4 getParameterNull(Method)

获取：设置Method参数NULL值，Object数组式返回

假如m方法为setTest(String s1,Integer int2);则返回Object objArr[2]=[null,null];

其中：

objArr[0]等于String的null

objArr[1]等于Integer的null

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. m：Method 类型，Class下的1个Method方法对象 |
| 返回值： | Object[] |
| 说明： | 返回指定的Method对象的所有参数，组成数组参数 |
| 示例： |  |

### 4.1.5 get(Object,Map,Field)

【get方法】：反射执行对象get方法的Method后的返回值

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. obj：Object类型，Java对象 2. methodMap：Map<String,Method>类型， obj的所有Method对象，并且key是以"get"+【小写f字段名】 3. f：Field类型，obj的一个属性Field对象,指定调用的Filed的getXXX方法 |
| 返回值： | Object |
| 说明： | 返回反射执行getXXX()方法后的返回值 |
| 示例： | //该Class已有“getAge()”方法  Field field\_age=Student.class.getField(“age”);  //获取Class的所有带“get”的方法  Map<String,Method> getMap=TableUtil.getMethodMap(Student.class,”get”);  //生成1个Student对象  Student obj=Student();  obj.setAge(17);  Object t\_age=TableUtil.get(obj,getMap,field\_age); //获取age值 |

### 4.1.6 set(Object,Map,Field,Object)

【set方法】：反射执行对象set方法的Method后的返回值

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. obj：Object类型，Java对象 2. methodMap：Map<String,Method>类型， obj的所有Method对象，并且key是以"set"+【小写f字段名】 3. f：Field类型，obj的一个属性Field对象，指定调用的Filed的setXXX方法 4. paramValue：Object类型，参数值 |
| 返回值： | void |
| 说明： | 反射执行setXXX()方法 |
| 示例： | //该Class已有“setAge()”方法  Field field\_age=Student.class.getField(“age”);  //获取Class的所有带“set”的方法  Map<String,Method> setMap=TableUtil.getMethodMap(Student.class,”set”);  //生成1个Student对象  Student obj=Student();  TableUtil.set(obj,setMap,field\_age,18); //设置age值为18 |

### 4.1.7 setPrimaryKeyValue(Object,Object)

设置：obj主键值

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. obj：Object类型，Java对象 2. primarykeyValue：Object类型，主键值 |
| 返回值： | void |
| 说明： | 反射执行setXXX()主键值的方法 |
| 示例： | Student obj=Student();  TableUtil.setPrimarykeyValue(obj,126); //设置主键值为126 |

### 4.1.8 getPrimaryKeyValue(Object)

获取：obj主键值

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. obj：Object类型，Java对象 |
| 返回值： | Object |
| 说明： | 反射执行getXXX()主键值的方法，并返回主键值 |
| 示例： | Student obj=Student();  obj.setId(147);  TableUtil.getPrimarykeyValue(obj); //获取主键值，值为上面的147 |

### 4.1.9 getPrimaryKeyField(Class)

获取：tableClass所映射的主键@Id字段

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?>类型，继承Table的Model类，并且有本框架的@Id注解 |
| 返回值： | Field |
| 说明： | 返回主键Field字段对象 |
| 示例： | Field idField=TableUtil.getPrimaryKeyField(Student.class); |

### 4.1.10 cast(Object,Class)

对象类型转换（含：基础数据类型、String、Boolean、cast强转）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. source：Object类型，源对象，主要用于基础类型的封装类（他们继承了Number类） ，字符串类型转换基础数据类型，父子类的转换 2. dstTypeClass：Class<?>类型，要转换的目标类型 |
| 返回值： | Object |
| 说明： | 返回转换后的对象 |
| 示例： | //Double转Integer  Integer t\_new\_1=TableUtil.cast(10584.5,Integer.class);  //Long转Double  Double t\_new\_2=TableUtil.converType(70584555454L,Double.class); |

### 4.1.11 getDataSource(Class)

获取 : Class类映射的数据源（数据库）的id值，如果没有@DataSource注解或value()为空，则返回null。

**注意**：若本类没有，则寻找父类；若父类没有，则继续寻找父类的父类.....直到有或父类为Table为止

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?>类型，带有@DataSource注解的Class，没有就找父类 |
| 返回值： | String |
| 说明： | 返回数据源的id值 |
| 示例： | String dataSourceId=TableUtil.getDataSource(Student.class); |

### 4.1.12 getClass(Class,Class)

获取 : 指定类指定注解所在的类。

**注意**：若该类没有则去寻找深层次的父类，直到Table类为止

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，要指定寻找的主类 2. anClass：Class<? extends Annotation>类型，指定的注解类 |
| 返回值： | Class<? **extends** Table> |
| 说明： | 返回：指定注解类在指定的主类自己或深层次父类中，所处在的位置的类。如：有A类、A1类、A2、A3类（A后面的数字越大，则深层次父类最深。如：A1是A的父类，A2是A1的父类……），其中@DataSource注解在A2类，A3类继承Table，那么执行以下代码的结果是：  Class<? extends Table> posClass=TableUtil.getClass(A.class,DataSource.class);  **posClass**为A2类 |
| 示例： | 看说明↑ |

### 4.1.13 getAnnotation(Class,Class)

获取 : 指定类指定的注解的对象，若该类没有则取寻找深层次的父类，直到Table类为止。

**注意**：若该类没有则去寻找深层次的父类，直到Table类为止

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<? extends Table>类型，要指定寻找的主类 2. anClass：Class<? extends Annotation>类型，指定的注解类 |
| 返回值： | Class<? **extends** Annotation> |
| 说明： | 返回：指定注解类在指定的主类自己或深层次父类中，所处在的位置的类的该注解对象。如：有A类、A1类、A2、A3类（A后面的数字越大，则深层次父类最深。如：A1是A的父类，A2是A1的父类……），其中@DataSource注解在A2类，A3类继承Table，那么执行以下代码的结果是：  Annotation an\_pos=TableUtil.getAnnotation(A.class,DataSource.class)  **an\_po**s为A2的DataSource注解对象; |
| 示例： | 看说明↑ |

### 4.1.14 getConfig(Table)

获取 : 数据源（连接池）Config配置对象，对象式

**注意**：若该对象的所属类没有则去寻找深层次的父类，直到Table类为止

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. obj：Table类型，带有@DataSource注解的对象的类(不带则取默认的”default”)，没有就找父类 |
| 返回值： | Config |
| 说明： | 返回数据源的配置对象 |
| 示例： | Student obj=new Student(); //Student继承了Table类  Config config=TableUtil.getConfig(obj); |

### 4.1.15 getConfig(Class)

获取 : 数据源（连接池）Config配置对象。

**注意**：若该类没有则去寻找深层次的父类，直到Table类为止

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?>类型，带有@DataSource注解的Class(不带则取默认的”default”)，没有就找父类 |
| 返回值： | Config |
| 说明： | 返回数据源的配置对象 |
| 示例： | Config config=TableUtil.getConfig(Student.class); |

### 4.1.16 getConfig(String)

获取 : 数据源（连接池）Config配置对象，根据数据库连接池id，默认‘default’配置

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. poolName：String类型，直接根据数据源（连接池）id获取 |
| 返回值： | Config |
| 说明： | 返回数据源的配置对象 |
| 示例： | Config config=TableUtil.getConfig(“mainWind”); |

### 4.1.17 getTable(Class)

获取：Class类映射的表名，如果没有Tables注解或value()为空，则返回表前缀+类名（注：不带表全路径，只单纯的返回表名。表前缀是由@TablePrefix设置）

**注意**：若该类的@TablePrefix没有则去寻找深层次的父类，直到Table类为止

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?> 类型，带有Tables注解的Class |
| 返回值： | String |
| 说明： | 返回继承Table的类映射的表名（默认取类名，拼接配置的表前缀为表名） |
| 示例： | String tableName=TableUtil.getTable(Student.class); |

### 4.1.18 getPrimaryKey(Class)

获取：tableClass所映射的主键名。如果没有Column注解或value()为空，则返回列前缀+字段名（注：列前缀是由@ColumnPrefix设置）

**注意**：若该类的@ColumnPrefix没有则去寻找深层次的父类，直到Table类为止

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?> 类型，继承Table的类 |
| 返回值： | String |
| 说明： | 返回主键名（默认取带@Id注解的字段名，拼接配置的列前缀为列名） |
| 示例： | String idName=TableUtil.getPrimaryKey(Student.class); |

### 4.1.19 getGeneric(Field)

获取：List字段的泛型Class

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. genericField：Field类型，类型为List的Field字段 |
| 返回值： | Class<?> |
| 说明： | 返回List字段的泛型Class |
| 示例： | Field courseListField=Student.class.getField(“courseList”);  Class<CourseL> courseClass=TableUtil.getGeneric(courseListField); |

### 4.1.20 copyProperties(Object,Object)

bean对象属性浅拷贝，忽略null对象

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. soure：Object类型，源对象，不能为空 2. target：Object类型，目标对象，不能为空 |
| 返回值： | void |
| 说明： | 将soure对象的所有属性浅度拷贝到target目标对象里（需要Field字段匹配），基础类型不会因为源对象的改变而改变目标对象，自定义对象则会 |
| 示例： | Student obj\_1=new Student();  obj\_1.setName(“张三”);  Student obj\_2=new Student();  TableUtil.copyProperties(obj\_1,obj\_2); |

### 4.1.21 getField(Class)

获取：指定Class类的所有Field字段（默认），直到父类为Table为止

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?> 类型，继承Table的类 |
| 返回值： | Field[] |
| 说明： | 返回指定Class类的所有Field字段，直到父类为Table为止，便不在继续往更深层次的父类去获取 |
| 示例： | Field allFieldArr[]=TableUtil.getField(Student.class); |

### 4.1.22 getField(Class,Class)

获取：指定类的所有Field字段，包含父类，直到指定父类停止（不在获取）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?> 类型，继承Table的类 2. parentClass：Class<?> 类型，继承Table的类，停止继续往深层次父类获取的Class类（为空则一直获取到底） |
| 返回值： | Field[] |
| 说明： | 返回指定Class类的所有Field字段，直到父类为指定的parentClass类为止，便不在继续往更深层次的父类去获取 |
| 示例： | Field allFieldArr[]=TableUtil.getField(Student.class,School.class); |

### 4.1.23 getField(Class,String)

获取：指定ORM类指定字段名称的Field字段对象

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?> 类型，继承Table的类 2. fieldName：String类型，完全匹配的字段名称 |
| 返回值： | Field |
| 说明： | 返回指定Class类的指定字段名称的Field对象， |
| 示例： | Field ageField=TableUtil.getField(Student.class,”age”); |

### 4.1.24 getMethod(Class)

获取指定类的所有Method方法（默认）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?> 类型，继承Table的类 |
| 返回值： | Method[] |
| 说明： | 返回指定Class类的所有Method方法对象 |
| 示例： | Method allMethod[]=TableUtil.getMethod(Student.class); |

### 4.1.25 getMethod(Class,Class)

获取指定类的所有Method方法，包含父类，直到指定父类停止（不在获取）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?> 类型，继承Table的类 2. parentClass：Class<?> 类型，继承Table的类，停止继续往深层次父类获取的Class类（为空则一直获取到底） |
| 返回值： | Field[] |
| 说明： | 返回指定Class类的所有Mehod方法，直到父类为指定的parentClass类为止，便不在继续往更深层次的父类去获取 |
| 示例： | Method allMethodArr[]=TableUtil.getMethod(Student.class,School.class); |

### 4.1.26 getForeignKeyField(Class,Class)

获取 : 指定ORM类指定外键ORM类的Field字段（不推荐用，因为会重复）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?> 类型，继承Table的类 2. foreignClass：Class<?> 类型，继承Table的类，外键Class |
| 返回值： | Field |
| 说明： | 返回指定Class类的指定外键Class的Field字段。（不推荐用，因为会重复） |
| 示例： | Field t\_field=TableUtil.getForeignKeyField(Student.class,School.class); |

### 4.1.27 getIdList(List)

获取 : 指定ORM类对象列表的主键id列表

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list : List<? extends Table>类型，继承Table类的对象列表，泛型请指定是同一个类 |
| 返回值： | List<Object> |
| 说明： | 返回指定list的所有主键，List<Object>接收 |
| 示例： | List<Object> t\_idList=TableUtil.getIdList(list); |

### 4.1.28 getSpecifiedList(List,String)

获取 : 指定ORM类对象列表指定字段名称的值列表

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list : List<? extends Table>类型，继承Table类的对象列表，泛型请指定是同一个类 2. specifiedFieldName：String类型，指定字段的名称（若该字段是外键，则取外键的id） |
| 返回值： | <T> List<T> |
| 说明： | 返回指定list的指定specifiedFieldName字段名称的所有值，List<Object>接收 |
| 示例： | List<String> t\_nameList=TableUtil.getSpecifiedList(list,”name”);  List<Long> t\_ courseIdList=TableUtil.getSpecifiedList(list,”courseList”); |

### 4.1.29 getForeignKey(Field)

获取：field所映射的外键名。优先取@Column得

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. field : Field类型，带有@ForeignKey注解的字段 |
| 返回值： | String |
| 说明： | 返回映射的外键名，如果没有，则取@Column得，在没有就取：  【一对一】=取映射的主键名  【一对多】=取该字段所属的Class的主键名 |
| 示例： | Field oneField=Student.class.getField(“school”);  Field manyField=Student.class.getField(“courseList”);  String t\_oneName=TableUtil.getForeignKey(oneField);  String t\_manyName=TableUtil.getForeignKey(manyField); |

### 4.1.30 getColumn(Field)

获取：Field类映射的列名，如果没有Column注解或value()为空，则返回Field字段名

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. colField: Field类型，带有@Column注解的Field |
| 返回值： | String |
| 说明： | 返回映射的列名，没有则返回Field字段名 |
| 示例： |  |

### 4.1.31 getColumnValueStr(Field)

获取：Field类映射的列的值字符形式。需要有@!Value注解，如果没有则返回【?】

**注**：若拥有@Id注解，则不会取@Value注解的值

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. colField: Field类型，带有@Value注解的Field |
| 返回值： | String |
| 说明： | 返回指定Field字段的@Value注解值，该值所有占位符（【?】）的一种替换。  如：Student类的create\_time字段有@Value注解，且value值为:  ” DATE\_FORMAT(?,’%Y-%m-%d %H:%i:%s’)”  那么该字段在内部做条件查询时，该条件的语句为：  create\_time>= DATE\_FORMAT(?,’%Y-%m-%d %H:%i:%s’) |
| 示例： | 看说明↑ |

### 4.1.32 arrayConverList(Object[])

Object数组转List

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. objArr: Object[]类型，对象数组 |
| 返回值： | List<Object> |
| 说明： | 返回转换成List的对象 |
| 示例： | Object arr[]=new Object[2];  List<Object> list=TableUtil.arrayConverList(arr); |

### 4.1.33 sort(List,List)

对list数据重新排序，并返回新的列表——根据传来的idList（主键）列表顺序

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<T extends Table>类型，需要排序的继承Table的ORM对象列表 2. idList：List<? extends Object>类型，主键值列表，根据此参数值的顺序排序，需要类型和list的主键类型一致 |
| 返回值： | <T **extends** Table> List<T> |
| 说明： | 返回排序后新生成的List对象列表  抛出RuntimeException异常，一般是get方法反射错误，没有找到可排的字段 |
| 示例： | List<Student> list=new ArrayLIst<Student>();  Student obj1=new Student();  obj.setId(2);  Student obj2=new Student();  obj.setId(1);  //加入到list里  list.add(obj1);  list.add(obj2);  //id的排序顺序列表  List<Object> idList=new ArrayList<Object>();  idList.add(1);  idList.add(2);  //生成排序后的新列表  List<Student> list\_new=TableUtil.sort(list,idList); |

### 4.1.34 isMatch(List)

判断List里所有对象是否全部为同一个类，并且不为空

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<?>类型，需要验证的对象列表 |
| 返回值： | **boolean** |
| 说明： | 返回匹配结果，list为空或长度小于等于0，则返回false  抛出TableException异常，表明List里有类型不一致的对象 |
| 示例： |  |

### 4.1.35 getPrimaryKey\_Database(Connection,String)

获取：指定表名的主键列名（数据库真实字段名称）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. con：Connection类型，JDBC的Connection连接 2. table：String类型，数据库的表名 |
| 返回值： | **String** |
| 说明： | 返回指定Connection、表名的主键列名 |
| 示例： |  |

### 4.1.36 getConnection(String,String,String,Integer)

获取 : Connection连接

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. url：String类型，连接字符串 2. userName：String类型，数据库用户名 3. password：String类型，密码 4. timeout：Integer类型，连接超时数。为空则不设置 |
| 返回值： | Connection |
| 说明： | 获取Connection的连接对象 |
| 示例： |  |

### 4.1.37 closeConnection(Connection)

关闭 : Connection连接

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. con：Connection类型，JDBC的Connection连接 |
| 返回值： | void |
| 说明： | 关闭指定Connection的连接 |
| 示例： |  |

### 4.1.38 close(ResultSet,Statement)

关闭：ResultSet和Statement

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. rs：ResultSet类型，JDBC的ResultSet结果集 2. ps：Statement类型，JDBC的Statement执行SQL接口 |
| 返回值： | void |
| 说明： | 关闭指定的ResultSet和Statement |
| 示例： |  |

### 4.1.39 getColumn\_Database(Connection,String)

获取 : 指定ORM类映射的数据库表，真实列名列表

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. con：Connection类型，JDBC的Connection连接 2. table：String类型，真实表名 |
| 返回值： | Set<String> |
| 说明： | 返回指定ORM类映射的表的所有列，用Set<String>接收 |
| 示例： | Set<String> colSet=TableUtil.getColumn\_Database(con,”student”); |

### 4.1.40 listInit(List)

List处理，返回新List（主要是调用ORM方法时：list是否为空、数据的拷贝等的处理；浅复制）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. sourceList：List<T>类型，源数据对象列表（一般用于基础数据类型的对象） |
| 返回值： | <T> List<T> |
| 说明： | 返回拷贝后，新生成的数据列表，浅拷贝，基础数据类型不会受相互使用的影响，自定义对象仍然会存在传址的问题而相互受影响 |
| 示例： | …………  List<Integer> sourceList\_new=TableUtil.listInit(sourceList); |

### 4.1.41 getPlaceholder(List)

获取（生成）in的占位符（如：list有5个数据，则返回“?,?,?,?,?”）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<? extends Object>类型，可以是任何类型的数据 |
| 返回值： | String |
| 说明： | 返回占位符，list有多少条数据。就返回多少个“?”  如：list有5个数据，则返回”?,?,?,?,?” |
| 示例： | …………  String pleace=TableUtil.getPlaceholder(list); |

### 4.1.42 getObject(Class,Object...)

获取（生成）新的对象（生成新对象，根据构造方法的参数对象，匹配相同的构造方法）；paramsArr=构造方法的参数对象（按顺序）,可变式数组

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<T>类型 2. paramsArr：Object…类型，可变式数组，根据构造方法的参数类型和顺序加 |
| 返回值： | <T> T |
| 说明： | 返回1个新的对象，若2个参数有1个为空（含：null），则直接返回null |
| 示例： | /\*  Student类有1个带参的构造方法  publii Student(String name,Integer age);  \*/  Student obj=TableUtil.getObject(Student.class,“刘琴”,16); |

### 4.1.43 getObjectList(List,String)

获取 : list指定字段（成员变量）名的值，列表式——可用于外键

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<? extends Table>类型，继承Table的ORM对象列表 2. fieldName：String类型，指定要取的字段（成员变量）名 |
| 返回值： | <T> List<T> |
| 说明： | 返回指定字段名称的所有对象值，一般用于取外键对象。  如果指定的字段是List，则调用addAll，放入返回的List里 |
| 示例： | …………  List<Course> courseList=TableUtil.getObjectList(list,”courseList”);  List<String> nameList=TableUtil.getObjectList(list,”name”); |

### 4.1.44 setObjectList(List,String,List)

设置 : list指定字段（成员变量）名的值——用于外键

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<T extends Table>类型，继承Table的ORM对象列表 2. foreignFieldName：String类型，指定要设置的字段（成员变量）名，外键字段名称 3. foreignList：List<? extends Table>类型，要设置到list指定外键字段名的列表，用外键的主键与该list的主键匹配，并设置 |
| 返回值： | void |
| 说明： | list的外键的主键与参数foreignList的主键匹配，匹配到才取对应的对象设置（覆盖）原有的对象 |
| 示例： | …………  List<CourseList> courseList=new ArrayList<CourseList>();  //courseList列表到设置list的courseList成员变量，  TableUtil.setObjectList(list,”courseList”,courseList); |

### 4.1.45 listToMap(List)

List数据转为Map（key=主键；value=对象），以主键做key，value做对象

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<S>类型，继承Table的ORM对象列表 |
| 返回值： | <T,S **extends** Table> Map<T,S> |
| 说明： | 注意 : 所有的异常均用【RuntimeException】抛出  @throws IllegalAccessException  @throws IllegalArgumentException |
| 示例： | …………  Map<Long,Student> map=TableUtil.listToMap(list); |

### 4.1.46 getPack(Config,String)

获取 : 包装真实名后的名（下同），Config式

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. config：Config类型，数据源对象，里面含有数据库关键字 2. column：String类型，下同 |
| 返回值： | 与下同 |
| 说明： | 与下同 |
| 示例： | 与下同，将第1个参数改为Config对象 |

### 4.1.47 getPack(String,String)

获取 : 包装真实名后的名（含：数据库名、表名、列名等。主要解决关键字的问题和匹配真实列名）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. db：String类型，数据库关键词（如：mysql、sql server、oracle） 2. column：String类型，数据库名、表名、列名等 |
| 返回值： | String |
| 说明： | 如：返回column包装后的列名。主流数据库的包装字符如下：  Mysql：``  Sql server：[]  Oracle：“”（小写的双引号） |
| 示例： | String pack\_mysql=TableUtil.getColumnPack(“mysql”,”id”); //返回：`id`  String pack\_sqlServer=TableUtil.getColumnPack(“sql server”,”id”); //返回：[id]  String pack\_oracle=TableUtil.getColumnPack(“oracle”,”id”); //返回：”id” |

### 4.1.48 findForeign(List,boolean,Page,String[],String…)

查询 : 外键数据（外键里的外键对象只留下id），根据foreign指定的外键字段（成员变量）名，指定显示的字段——列表式

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. list：List<? extends Table>，继承Table的ORM类对象列表 2. isDesc：boolean类型，是否倒序 3. page：Page类型，分页对象 4. specifiedFieldNameArr：String数组，指定外键要显示的字段列表（请传成员变量名，建议对取单个外键的方式使用），数组式，为NULL，则显示所有字段 5. foreign：String可变式数组，指定要取的外键（请传成员变量名），变长式，可数组式 |
| 返回值： | void |
| 说明： | 根据传址逻辑，会动态的把查询到的外键设置到list列表对应的外键（@ForeignKey）对象 |
| 示例： | //一批Student数据  List<Student> objList=new ArrayList<Student>();  ……  Page page=new Page(3,10);  //显示：id（主键，不填也会显示）、name（名称）、createTime（创建时间）  String specifiedFieldNameArr[]={“id”,”name”,”createTime”}  //后面的2个参数就是可变式数组，对应上述参数里的foreign  TableUtil.findForeign(objList,false,page, specifiedFieldNameArr,” majorList”,” sex”); |

### 4.1.49 isNull(Field,Object)

判断是否空值，并返回（为空返回null，不为空则返回第2个参数value值）

### 4.1.50 getInterfaces(Class<?>,Class<?>)

获取 : 指定类的所有接口类，包含父类，直到指定父类停止（不在获取）

|  |  |
| --- | --- |
| 参数：  (按顺序) | 1. tableClass：Class<?>，要获取所有接口的Class类 2. parentClass：指定要停止获取父接口的Class（为空则一直获取到底） |
| 返回值： | Class<?>[] |
| 说明： | 根据tableClass所属的接口列表，每个一直往父接口搜寻，知道为空位置 |
| 示例： | //接口继承：A—>A1—>A2，B—>B1—>B2  //类Z实现了A,B接口  // interfacesArr的数组顺序为：A,B,A1,A2,B1,B2  Class<?> interfacesArr[]=TableUtil. getInterfaces(Z.class,null); |

## 4.2 ConfigUtil

配置工具类，获取配置文件数据，以Map返回。

### 4.2.1 getDataSourceConfig()

获取wind-orm.xml下所有数据源的配置信息

|  |  |
| --- | --- |
| 返回值： | Map<String,Config> |
| 说明： | 获取wind-orm.xml下所有数据源的配置信息。  以key为数据源（连接池）id，value为配置类（Config） |
| 示例： | Map<String,Config> map= WindUtil.getDataSourceConfig(); |

# 连接池

本框架采用自研的连接池，采用**队列**的方式获取和释放。

队列：先进先出，类似于现实中的排队机制，先排在前面的先走。Java数据结构原理亦是如此。先取出来的连接管理器，在调用close()方法释放后，会加入到连接池队尾。如果达到了配置的最大保持连接数（wind-orm.xml的**max**参数），则会关闭连接

本框架**freeTime**参数，若连接管理器超过该数未被使用，则关闭connection连接（默认1800秒），即使连接管理器正在使用途中也会强行关闭，并被释放

连接池相关类的包路径：org.wind.orm.dao.service

## 5.1 ConnectionPoolFactory（连接池工厂类）

管理wind-orm.xml下所有的数据源，静态单例化（伪单例），不允许继承。

线程安全。**dataSourceWay**=读写标志（该标志可从org.wind.orm.cache.Cache2里获取）

|  |  |
| --- | --- |
| 类路径： | org.wind.orm.service.ConnectionPoolFactory |
| 静态方法： | 1. ConnectionPool getConnectionPool(String poolName, **int** dataSourceWay)： 获取连接池，poolName指定的数据库连接池别名（wind-orm.xml数据源的id值） 2. ConnectionPool createConnectionPool(String poolName,int dataSourceWay)： 创建 : 连接池，poolName指定的数据库连接池别名 |

## 5.2 ConnectionPool（连接池接口类）

连接池的接口类，该接口拥有如下方法可供调用：

|  |
| --- |
| 1. ConnectionManager getConnectionManager()：获取一个Connection管理对象 2. **void** releaseConnectionManager(ConnectionManager conManager)：释放一个ConnectionManager对象回到连接池里 3. String getDataSource()：获取 : 当前连接池的数据源名称 4. **int** getDataSourceWay()：获取 : 当前数据源方式（主、读、写） |

## 5.3 ConnectionPoolImpl（连接池接口实现类）

实现上面【连接池接口类】的所有方法。

线程安全。获取每个**连接管理器**都是唯一的，且获取时，是线程安全的

## 5.4 ConnectionManager（连接管理器接口类）

数据库Connection连接管理器接口，该接口拥有如下方法可供调用：

|  |
| --- |
| 1. **void** setConnection(Connection con)：设置Connection对象 2. Connection getConnection()：获取Connection对象 3. **void** commit() **throws** SQLException：提交事务 4. **void** rollbak()：回滚事务 5. **void** rollbak(Savepoint t\_savepoint)：回滚指定事务点 6. **void** close(ResultSet rs,Statement ps)：关闭操作数据库相关的类 7. **void** setLastOperationTime(**long** lastOperationTime)：设置 : 最后操作的时间戳 8. **long** getLastOperationTime()：获取 : 最后操作的时间戳 9. ConnectionPool getConnectionPool()：获取所属的连接池 |

## 5.5 ConnectionManagerImpl（连接管理器接口实现类）

实现上面【连接管理器接口类】的所有方法。

在整个会话（当前线程）期间用完后，要记得调用.close()方法释放回连接池。

如果连接池已达到最大保持连接数，则关闭新释放的connection

# 异常

## 6.1 ConnectionPoolException

ConnectionPool线程池Exception类

## 6.2 ConnectionPoolFactoryException

ConnectionPool线程池生产Exception类

## 6.3 SQLFactoryException

SQL工厂Exception类， 生成SQL具体实现类时，发生的错误

## 6.5 TableDDLException

【DDL】Table数据库操作Exception类

## 6.6 TableDeleteException

【删除】Table数据库操作Exception类

## 6.7 TableException

Table数据库操作Exception类，所有本框架异常类的超类

## 6.8 TableSaveException

【插入、更新】Table数据库操作Exception类

## 6.9 TableSelectException

【查询】Table数据库操作Exception类

# 缓存

**已删除**。

# 拦截器

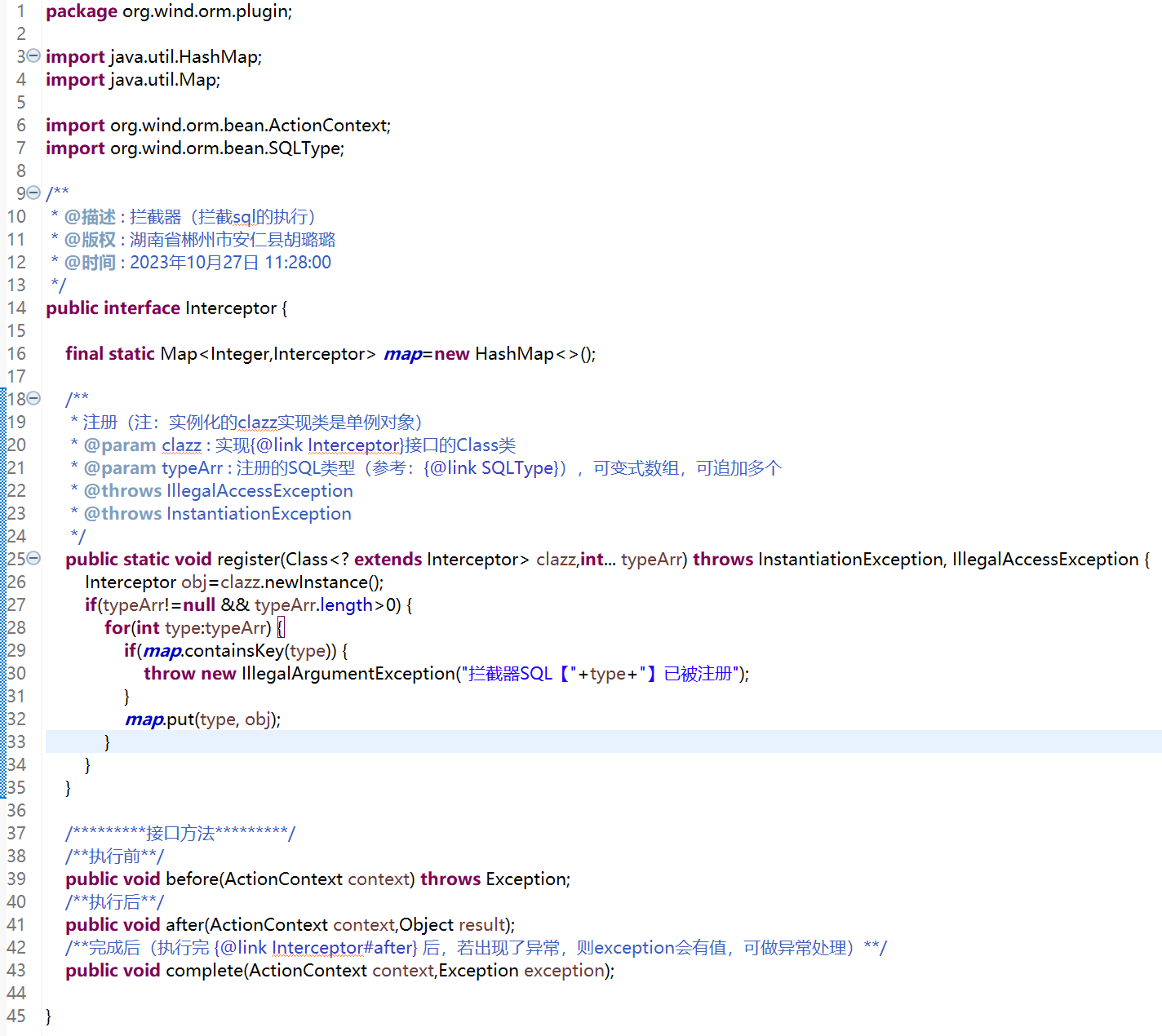
支持【SQL执行前】、【SQL执行后】、【处理完成后】（一般可做异常处理）的处理。

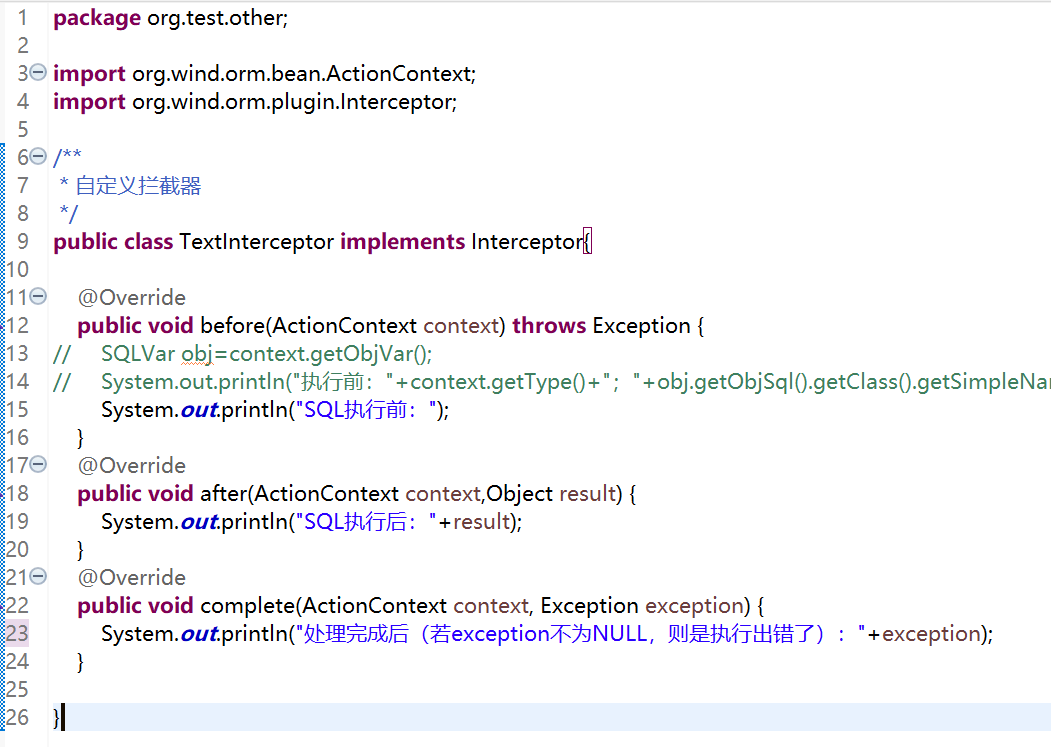
**支持的拦截类型**：



使用步骤：

1. 创建自定义拦截器实现类，实现【org.wind.orm.plugin.Interceptor】接口。如：

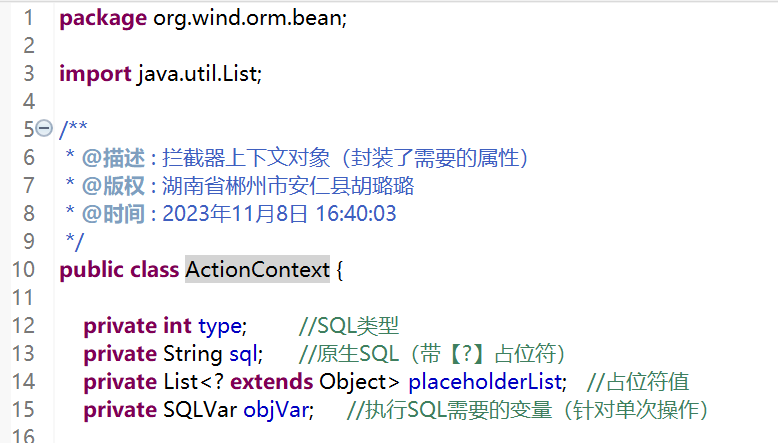




1. 注册：调用例子如下：

Interceptor.*register*(TextInterceptor.**class**, SQLType.***SELECT***,SQLType.***INSERT***,SQLType.***UPDATE***); //注册拦截器

1. 参数**ActionContext**的内容如下（可通过响应的getter方法获取）：



1. 属性**SQLVar**的内容如下：



可通过响应的getter方法获取，ConnectionManager的getter方法则需要根据情况使用

