目录

[1 simple-base 1](#_Toc457580762)

[1.1 配置文件读取 1](#_Toc457580763)

[1.2 Simplexml用法 3](#_Toc457580764)

[1.3 BeanParser 4](#_Toc457580765)

[1.4 工具类 4](#_Toc457580766)

[2 simple-dao 5](#_Toc457580767)

[2.1 注解说明 7](#_Toc457580768)

[2.2 配置文件 7](#_Toc457580769)

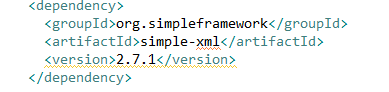
[3 simple-kv 9](#_Toc457580770)

# simple-base

## 配置文件读取

通过使用simplexml来实现对xml文件的读取，并反序列化为java类

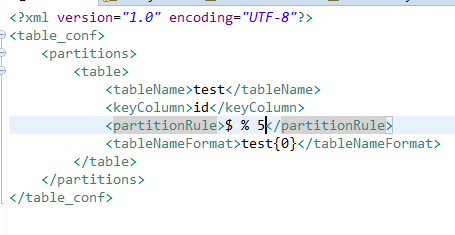
Pom引用



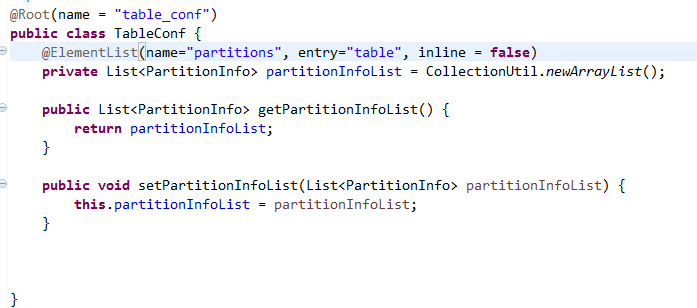
核心类：com.simple.base.config.XmlConfig

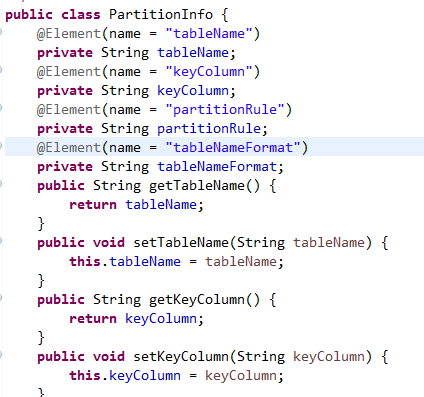
使用方法

* 定义xml文件



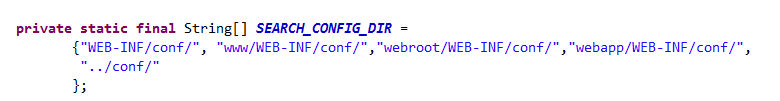
* 定义java bean对像





请xml文件放入项目约定目录下

默认支持如下目录：



如果需要放置在其它目录，

可以通过修改源代码或者设置自定义目录位置来实现



* 读取配置文件



* Logger名字 com.simple.base 如果需要查看log4j日志，添加这个logger

## Simplexml用法

**1. 属性名和XML元素名称必须一致**  
   从copy-config层向下依次类推, 假设解析类名为"CopyConfig", 则此类需要具有copyList的属性和其相应注解规则,  如果想另起别名可以使用注解"name"标注对应的XML元素.

|  |
| --- |
| <copy-config version="15">     <copyList>        <copy copyId="1" copyName="copy1"/>        <copy copyId="2" copyName="copy2"/>     </copyList>  </copy-config> |

**2. xml元素类型**

  xml元素的基本类型和Java类型必须一一对应. 如果是字符串的类型则在解析类内也必须是字符串类型, 否则会解析失败.  普遍只有两种基本类型, 字符串/整型.

**3. inline注解特性**  
     inline注解代表取当前类"工作ROOT"内的第一层条目列表***,*** 确认inline是否为TRUE可参考此方法: 第一确定当前解析类的"工作ROOT"(root);  第二确认要读取的"列表名称"(name). 第三确认要读取的"列表条目名称"(entry). (***工作ROOT=列表名称)  或者说  (列表条目名称是工作ROOT第一层)***时则inline为true.

     实例: 工作ROOT为copy-config,读取的列表名称为copy-config, 读取的列表条目名称为copy.   即 工作ROOT=列表名称, 同时列表条目名称是工作ROOT第一层, 这种情况下inline=true;

|  |
| --- |
| <copy-config version="15">    <copy copyId="1" copyName="copy1"/>    <copy copyId="2" copyName="copy2"/>  </copy-config>  @ElementList(entry = "copy", inline = true) |

  2) inline注解具有欺骗性, 即使解析失败他也会正常返回对象,  但对象会没有属性.

**4. entry注解特性**  
  此注解的真实意思为"条目", 不是入口.  一般作用于***列表***属性, 主要根据当前entry指定的值确认***列表的条目***的名称;  如下面的例子

|  |
| --- |
| xml:  <copy-config version="15">     <copyList>        <copy copyId="1" copyName="copy1"/>        <copy copyId="2" copyName="copy2"/>     </copyList>  </copy-config>  解析类:  @ElementList(required = false, inline = false, entry = "copy")  private List<copyItem> copyList |

## BeanParser

通过反射操作类的属性，反射结果会缓存

伪代码：

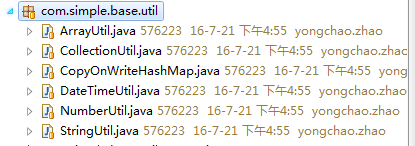
BeanParser parser = BeanParser.getBeanParser(beanClass);

BeanParser.FieldStruct bfi = parser.getFieldStruct(fieldName);

bfi.set(destObject,value);

## 工具类

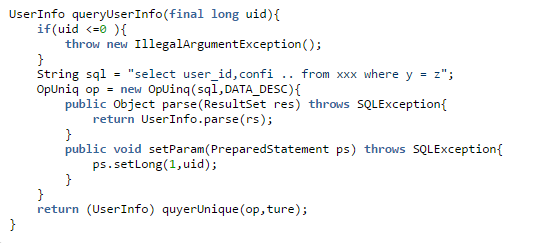
提供了一些常用的工具类包



# simple-dao

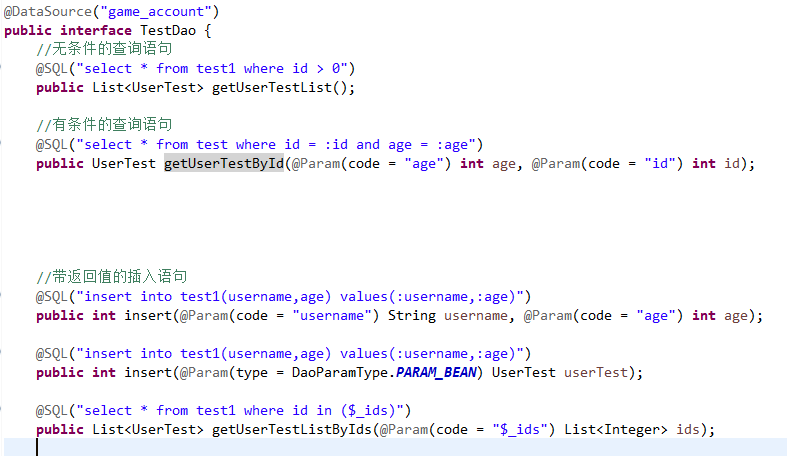
通过java注解，动态代理实现在sql解析，resultSet处理，支持散表散库

传统的sql处理方式



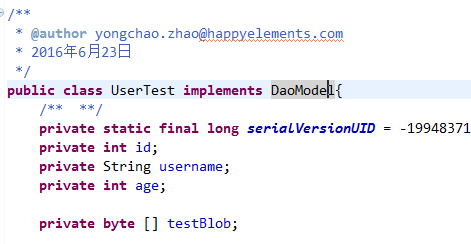
simpleDao的处理方式

仅仅需要一个接口



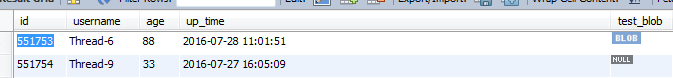
特别重要：

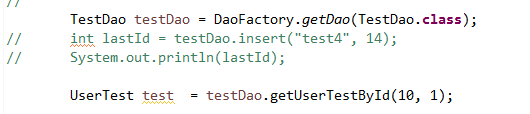
Bean对像必须实现DaoModel接口，如：



属性名字与DB表字段名字需要对应

如testBlob这个属性，在DB里需要将B变为小写，并在B字母之前加入下划线





## 注解说明

@Datesource 指定数据源 数据源在db\_conf进行配置

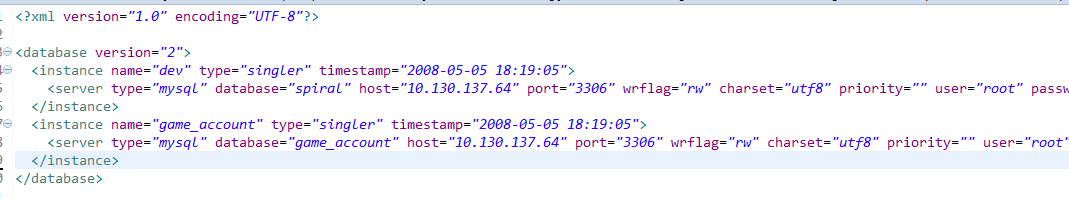
@SQL 指定sql语句，其实变量用占位符” : “ in操作 $\_

@Param 参数的注解

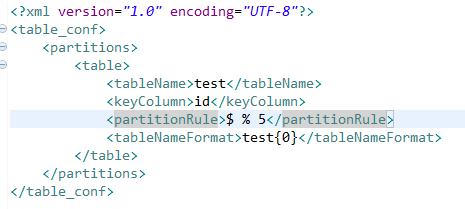
## 配置文件

db\_conf.xml

示例



table\_conf.xml



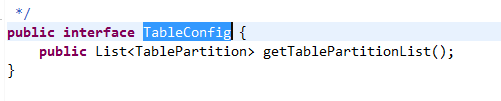
示例： 新test这个表按id这个字段的值 取模%5 后的生成新表名 如 id为14 散表后为test4

上边2个文件放入约定的配置文件后即可自动读取，配置文件的默认目录可以参照第1章simple\_base

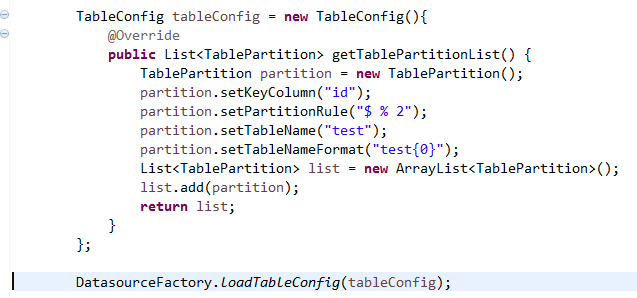
如果不想采用xml文件的方式

可以通过实现DBConfig， TableConfig的接口





以下图所示的方式进行DB的配置



# simple-kv

原理基本和simpleDao类似，将对像以kv的方式存在memcache 可CMEM

使用步聚：

先创建一个抽像类，实现storageConfig接口，可以查看storageConfig源代码查看更多注释

示例：

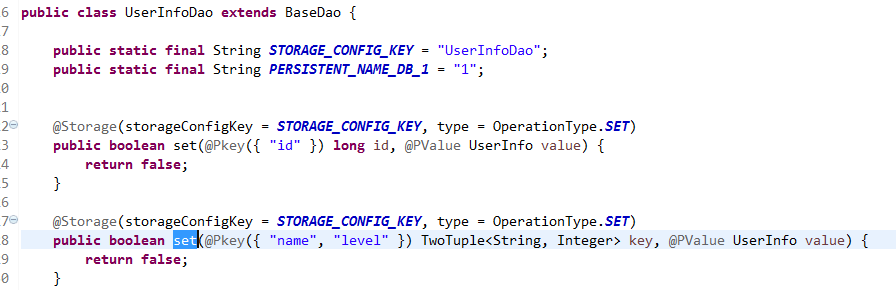
UserInfoDao继续BaseDao



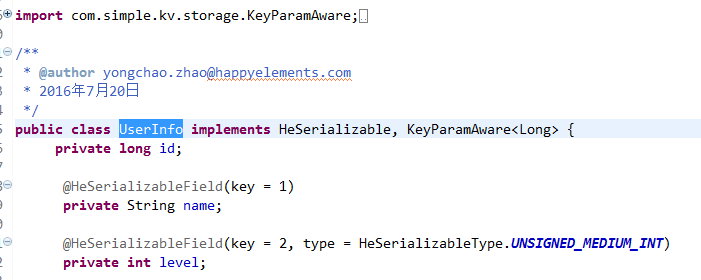
* 

这个注解指明了要用之前在BaseDao中注册的storageConfig，然后定义了本次操作的操作类型

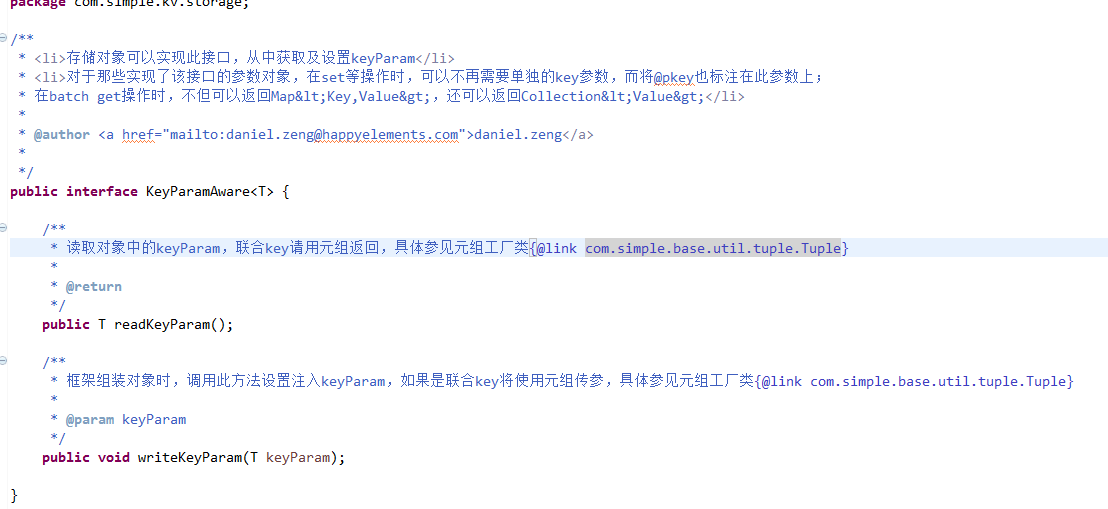
* 如果要使用字段组合做为key，则可以使用TwoTuple或ThreeTuple来实现
* 则为实际存储的内容

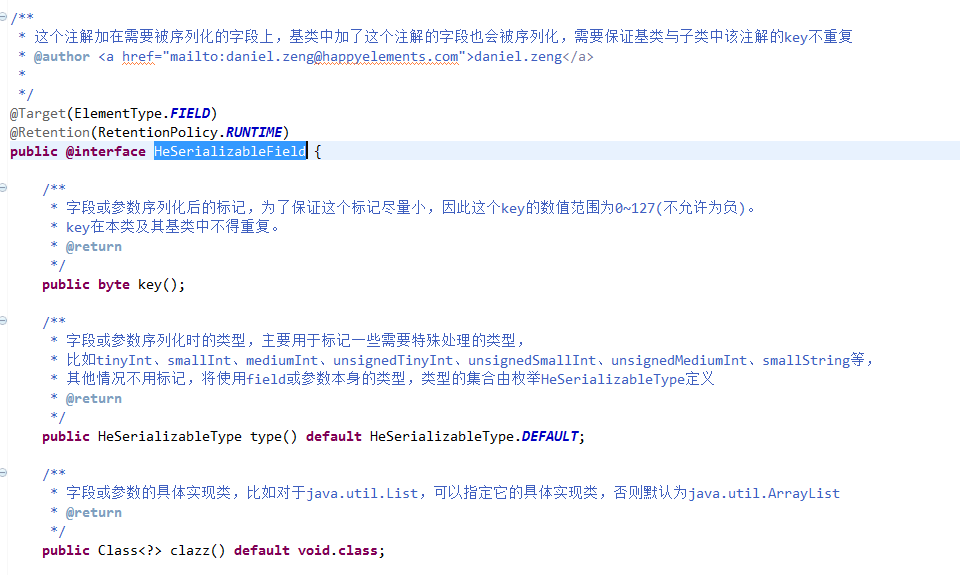


由于是kv存储数据也是以bytes value的形式被存储的。所以对象需要满足HeSerializable，以便被he-storage 序列化



实现KeyParamAward





he-storage的几个annotation

@Pkey:

* 该注解加在作为key的参数上，如果是多个字段作为联合key，请使用元组传参，具体参见元组工厂类@link com.happyelements.rdcenter.commons.util.tuple.Tuple

|  |
| --- |
| 实际存储为db时，PKey('xxx')的value对应db中table的primary key 'xxx'，当PKey({'xxx','yyy'})的value为多个键时，db中table需要对应的多个键位'xxx','yyy'联合唯一索引  实际存储为cache或cmem时，PKey('xxx') 的value对应cache中的key，为StorageConfig.keyPrefix\_'xxx';当Pkey({'xxx','yyy'})的value为多个键时，cache中key为StorageConfig.keyPrefxi\_'xxx'\_'yyy' |

* 批量get或批量delete时，加在用于传递多个key的集合上；
* 批量set时，加在用于传递key及value的Map上(Map的key作为key)，或是由实现了KeyParamAware接口的元素组成的value集合上
* 通常情况下该注解标记的参数同时也作为mysql分表计算的参数；如果是联合key，则以元组中第一个参数作为mysql分表计算的参数

@PValue:

* 当执行新增或修改操作时，该注解加在作为value的字段上
* 批量get或批量delete时， 加在用于传递多个value的集合上；
* 批量set时，加在用于传递key及value的Map上(Map的value作为value)，或是由实现了KeyParamAware接口的元素组成的value集合上
* value的存储使用HeSerialization机制
* 对于mysql，value所在的列名为"value"

@Storage

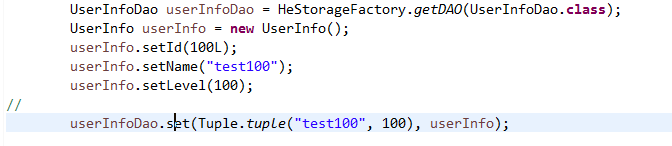
* 标记在方法上的注解，表明这是一个需要生成实现的持久化相关方法
* 参数-storageConfigKey：存储配置的key，根据这个key可以从注册的配置中找到具体的存储配置
* 参数-type：操作类型

@CasVersion

* 执行cas操作时，加在作为cas version的参数上

he-storage，目前支持set/get/delete及它们的批量操作，以及cas.

* 调用



Memcached的标准协议存在部分缺陷，其Get操作没有设计返回码，不能明确说明拉取数据失败的具体情况，所以Memcached API返回NO\_DATA时，有可能是网络原因造成的，不能完全信任。  
使用如下流程将是非常危险的，将造成用户数据初始化：

|  |
| --- |
| if（NO\_DATA） InitData(); |