# ChildrenBuilder使用说明

## 原理介绍

数据库表经常有自关联的树形结构的数据，但是从数据库中查寻是查询不了树形结构的数据。要达到树形结构的数据，有两种方式。第一种就是分层逐级多次查询数据库，效率比较低。第二种是把符合要求的数据先查询出来，再通过一定的方法转换为树形结构。

ChildrenBuilder就是采用第二种做法，其效率更高。在Oracle中可以通过相关的查询一次找到对应的数据。其它的数据也可以先通过相关的方案找到合适的数据。比如采用层级编码，通过层次编码的前缀相同的方式取到所需要的数据。

ChildrenBuilder是一个比较好的解决方案。

## ChildrenBuilder接口说明

|  |
| --- |
| /\*\*  \* **@author** 彭仁夔  \* **@param** <PK>  \* 主键类型，如String ,Integer \*/  **public** **class** ChildrenBuilder<PK> {  /\*\*  \* 把列表数据转换为数据形结构的数据。当然列表数据有数据结构的逻辑 即其有父节点的id和存放子节点的集合  \*  \* **@author** 彭仁夔于2015年10月19日下午7:23:17创建  \* **@param** data  \* list的列表数据  \* **@param** idName  \* id的字段名称 ,一般为id  \* **@param** pidName  \* 父字段的名称, 一般为pid,parentid  \* **@param** childrenName  \* 子集合字段的名称，一般为children  \* **@return**  \*  \*/  **public** <T> List<T> build(List<T> data, String idName, String pidName, String childrenName)  /\*\*  \* 把列表数据转换为数据形结构的数据。当然列表数据有数据结构的逻辑 即其有父节点的id和存放子节点的集合  \*  \* **@author** 彭仁夔于2015年10月19日下午7:23:17创建  \* **@param** data  \* list的列表数据  \* **@param** idName  \* id的字段名称 ,一般为id  \* **@param** pidName  \* 父字段的名称, 一般为pid,parentid  \* **@param** childrenName  \* 子集合字段的名称，一般为children  \* **@param** pidValue  \* 父节点的值  \* **@return**  \*  \*/  **public** <T> List<T> build(List<T> data, String idName, String pidName, String childrenName, String pidValue)  /\*\*  \* 把列表数据转换为数据形结构的数据。当然列表数据有数据结构的逻辑 即其有父节点的id和存放子节点的集合  \*  \* **@author** 彭仁夔于2015年10月19日下午7:23:17创建  \* **@param** data  \* list的列表数据  \* **@param** idName  \* id的字段名称 ,一般为id  \* **@param** pidName  \* 父字段的名称, 一般为pid,parentid  \* **@param** childrenName  \* 子集合字段的名称，一般为children  \* **@param** pidValue  \* 父节点的值  \* **@param** leafName  \* 节点的名称，节点为叶子节点，其值为1，不是为0，向该字段写入1，或0  \* **@return**  \*  \*/  **public** <T> List<T> build(List<T> data, String idName, String pidName,String childrenName, String pidValue, String leafName)  /\*\*  \* 把列表数据转换为数据形结构的数据。当然列表数据有数据结构的逻辑 即其有父节点的id和存放子节点的集合  \*  \* **@author** 彭仁夔于2015年10月19日下午7:23:17创建  \* **@param** data  \* list的列表数据  \* **@param** idName  \* id的字段名称 ,一般为id  \* **@param** pidName  \* 父字段的名称, 一般为pid,parentid  \* **@param** childrenName  \* 子集合字段的名称，一般为children  \* **@param** pidValue  \* 父节点的值  \* **@param** leafName  \* 节点的名称，节点为叶子节点，其值为1，不是为0  \* **@param** levelName  \* 节点登级的名称，向该字段写入其所在层级数  \* **@return**  \*  \*/  **public** <T> List<T> build(List<T> data, String idName, String pidName, String childrenName, String parentIdvalue, String leafName, String levelName) |

## BaseService.findNestedEnities说明和示例

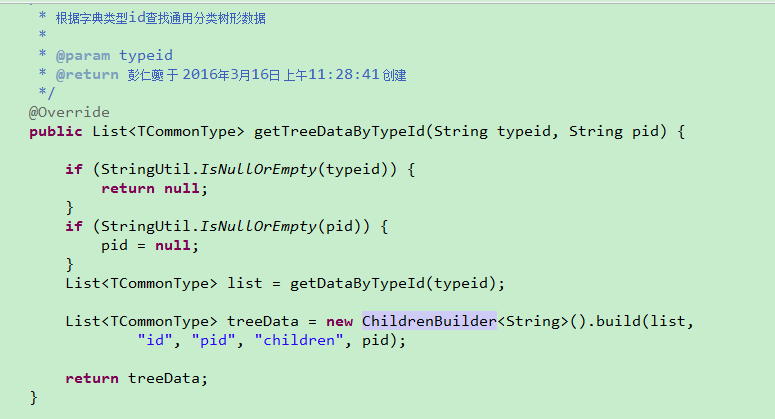
//todo:把该方法修改成能应用其它数据库

|  |
| --- |
| /\*\*  \* 从指定节点的某行开始，找到所有子节点<br>  \* // --包括本行<br>  \* // select \* from XT\_COMMON\_TYPE<br>  \* // start with id = '402881ff5397c87c015397ebd7820001'<br>  \* // connect by pid= prior id ;<br>  \* // --不包括本行<br>  \* // select \* from XT\_COMMON\_TYPE<br>  \* // start with pid = '402881ff5397c87c015397ebd7820001'<br>  \* // connect by pid= prior id ;<br>  \*  \* **@param** tableName 表名  \* **@param** id id字段的名称 一般为id ,数据库中的字段  \* **@param** pid 父节点的名称，一般为pid,parent\_id，数据库中的字段  \* **@param** clz 转换为类型  \* **@param** containSelf 是否包括自身行（查询的起始的节点值）  \* **@param** pidvalue 查询的起始的节点值  \*/  <T1> List<T1> findNestedEnities(String tableName, String id, String pid,  Class<T1> clz, **boolean** containSelf, String pidvalue); |



List<TCommonType> list = this.findNestedEnities("xt\_common\_type", "ID","PID", TCommonType.class, containself, pid);

## 使用示例



|  |
| --- |
| **public** List<TCommonType> getTreeDataByTypeId(String typeid, String pid) {  **if** (StringUtil.*IsNullOrEmpty*(typeid)) {**return** **null**; }  **if** (StringUtil.*IsNullOrEmpty*(pid)) {pid = **null**; }  List<TCommonType> list = getDataByTypeId(typeid);  List<TCommonType> treeData = **new** ChildrenBuilder  <String>().build(list,"id", "pid", "children", pid);  **return** treeData;  } |