## 1.部门层级树接口开发

### 1.1定义DeptLevelDto

需要继承SysDept

*/\*\*  
 \* 用于展示的dto对象  
 \*/*@Getter  
@Setter  
@ToString  
**public class** DeptLevelDto **extends** SysDept {  
  
 */\*\*  
 \* 新增的属性  
 \*/* **private** List<DeptLevelDto> **deptList** = Lists.*newArrayList*();  
  
 */\*\*  
 \* SysDept转换DeptLevelDto  
 \*  
 \** ***@param:*** *[dept]  
 \** ***@return:*** *com.mmall.dto.DeptLevelDto  
 \** ***@author:*** *nino  
 \** ***@date:*** *2019/5/28 9:11  
 \*/* **public static** DeptLevelDto adapt(SysDept dept) {  
 DeptLevelDto dto = **new** DeptLevelDto();  
 BeanUtils.*copyProperties*(dept, dto);  
 **return** dto;  
 }  
}

### 1.2SysTreeService

在SysDeptMapper里添加获得部门列表的方法：

*/\*\*  
 \* 获得所有部门  
 \*/*List<SysDept> getAllDept();

编写mapper映射文件中的sql语句：

<**select id="getAllDept" resultMap="BaseResultMap"**>  
 select  
 <**include refid="Base\_Column\_List"** />  
 from sys\_dept  
</**select**>

*/\*\*  
 \* 部门树  
 \*  
 \** ***@param:*** *[]  
 \** ***@return:*** *java.util.List*<*com.mmall.dto.DeptLevelDto*>  
 *\** ***@author:*** *nino  
 \** ***@date:*** *2019/5/28 9:46  
 \*/***public** List<DeptLevelDto> deptTree() {  
 *//1.获取部门表下的所有数据* List<SysDept> deptList = **sysDeptMapper**.getAllDept();  
 *//2.将数据转化成Dto集合* List<DeptLevelDto> dtoList = Lists.*newArrayList*();  
 **for** (SysDept dept : deptList) {  
 DeptLevelDto dto = DeptLevelDto.*adapt*(dept);  
 dtoList.add(dto);  
 }  
 **return** deptListToTree(dtoList);  
}  
  
*/\*\*  
 \* 转换部门树算法  
 \*  
 \** ***@param:*** *[deptLevelList]  
 \** ***@return:*** *java.util.List*<*com.mmall.dto.DeptLevelDto*>  
 *\** ***@author:*** *nino  
 \** ***@date:*** *2019/5/28 13:21  
 \*/***public** List<DeptLevelDto> deptListToTree(List<DeptLevelDto> deptLevelList) {  
 **if** (CollectionUtils.*isEmpty*(deptLevelList)) {  
 **return** Lists.*newArrayList*();  
 }  
 *// 实际数据结构level -> [dept1, dept2, ...]  
 // MultiMap，一个key对应多个value* Multimap<String, DeptLevelDto> levelDeptMap = ArrayListMultimap.*create*();  
 *//3.收集所有根节点* List<DeptLevelDto> rootList = Lists.*newArrayList*();  
  
 *//归类，根据level分组* **for** (DeptLevelDto dto : deptLevelList) {  
 levelDeptMap.put(dto.getLevel(), dto);  
 **if** (LevelUtil.***ROOT***.equals(dto.getLevel())) {  
 rootList.add(dto);  
 }  
 }  
 *// 按照seq从小到大排序* Collections.*sort*(rootList, **new** Comparator<DeptLevelDto>() {  
 @Override  
 **public int** compare(DeptLevelDto o1, DeptLevelDto o2) {  
 **return** o1.getSeq() - o2.getSeq();  
 }  
 });  
 *// 递归生成树* transformDeptTree(rootList, LevelUtil.***ROOT***, levelDeptMap);  
 **return** rootList;  
}  
  
*/\*\*  
 \* 递归生成树  
 \* level:0, 0, all 0->0.1,0.2  
 \* level:0.1  
 \* level:0.2  
 \*  
 \** ***@param:*** *[deptLevelList, level, levelDeptMap]  
 \** ***@return:*** *void  
 \** ***@author:*** *nino  
 \** ***@date:*** *2019/5/28 16:04  
 \*/***public void** transformDeptTree(List<DeptLevelDto> deptLevelList, String level, Multimap<String, DeptLevelDto> levelDeptMap) {  
 **for** (**int** i = 0; i < deptLevelList.size(); i++) {  
 *// 遍历该层的每个元素* DeptLevelDto deptLevelDto = deptLevelList.get(i);  
 *// 处理当前层级的数据* String nextLevel = LevelUtil.*calculateLevel*(level, deptLevelDto.getId());  
 *// 处理下一层* List<DeptLevelDto> tempDeptList = (List<DeptLevelDto>) levelDeptMap.get(nextLevel);  
 **if** (CollectionUtils.*isNotEmpty*(tempDeptList)) {  
 *// 排序* Collections.*sort*(tempDeptList, **deptSeqComparator**);  
 *// 设置下一层部门* deptLevelDto.setDeptList(tempDeptList);  
 *// 进入到下一层处理* transformDeptTree(tempDeptList, nextLevel, levelDeptMap);  
 }  
 }  
}  
  
*/\*\*  
 \* 自定义部门排序比较器  
 \*/***public** Comparator<DeptLevelDto> **deptSeqComparator** = **new** Comparator<DeptLevelDto>() {  
 @Override  
 **public int** compare(DeptLevelDto o1, DeptLevelDto o2) {  
 **return** o1.getSeq() - o2.getSeq();  
 }  
};

思路：

* 首先建一个DeptLevelDto类继承自SysDept类，private List<DeptLevelDto> deptList = Lists.newArrayList();为扩展字段，可以根据自己的业务具体扩展开发；
* 从数据库查询出所有部门数据的集合List<SysDept>
* 将从数据库查询出来的集合赋值给DeptLevelDto，转变为List<DeptLevelDto>，每条数据库记录都是一个新的DeptLevelDto对象，在下一步用到的是DeptLevelDto的集合，不是数据库查询出的list集合。
* 格式化DeptLevelDto集合：

创建以level为key，DeptLevelDto对象为value的map

// 实际数据结构level -> [dept1, dept2, ...]

// MultiMap，一个key对应多个value

Multimap<String, DeptLevelDto> levelDeptMap = ArrayListMultimap.create();

遍历DeptLevelDto集合取出每个DeptLevelDto对象，获取它的level属性，将level作为key，DeptLevelDto对象作为value添加到MultiMap<String, DeptLevelDto>中。

然后处理第一层，root

如果DeptLevelDto对象的level属性值等于与0，则将这个对象添加到根节点集合。

* 首层处理完，让其递归，不断地处理下一层。

transformDeptTree(List<DeptLevelDto> deptLevelList, String level, Multimap<String, DeptLevelDto> levelDeptMap)

deptLevelList——当前层级部门列表

level——每次会被更新

levelDeptMap——MultiMap<String, DeptLevelDto>层级与DeptLevelDto对象的一种映射。辅助计算

*/\*\*  
 \* 对rootList的下级进行递归排序  
 \* level:0, 0, all 0->0.1,0.2  
 \* level:0.1  
 \* level:0.2  
 \*  
 \** ***@param:*** *[deptLevelList, level, levelDeptMap]  
 \** ***@return:*** *void  
 \** ***@author:*** *nino  
 \** ***@date:*** *2019/5/28 16:04  
 \*/***public void** transformDeptTree(List<DeptLevelDto> deptLevelList, String level, Multimap<String, DeptLevelDto> levelDeptMap) {  
 **for** (**int** i = 0; i < deptLevelList.size(); i++) {  
 *// 遍历该层的每个元素 例如：根元素的第一个deptLevelDto 部门0.0* DeptLevelDto deptLevelDto = deptLevelList.get(i);  
 *// 处理当前层级的数据 例如：nextLevel = 0.0* String nextLevel = LevelUtil.*calculateLevel*(level, deptLevelDto.getId());  
 *// 处理下一层* List<DeptLevelDto> tempDeptList = (List<DeptLevelDto>) levelDeptMap.get(nextLevel);  
 *// 如果下一层有数据 例如：0.0 -> [0.0.1, 0.0.2, ...]* **if** (CollectionUtils.*isNotEmpty*(tempDeptList)) {  
 *// 排序* Collections.*sort*(tempDeptList, **deptSeqComparator**);  
 *// 设置下一层部门* deptLevelDto.setDeptList(tempDeptList);  
 *// 进入到下一层处理* transformDeptTree(tempDeptList, nextLevel, levelDeptMap);  
 }  
 }  
}

相当于

level:0 level等于0的所有部门传进来 0—>研发部、产品部、客服部

通过自定义的部门层级算法LevelUtil，计算0级的时候的部门层级

然后处理下一层，得到level:0.1、level:0.2并排序

最后设置下一层部门，进入到下一层部门，形成递归调用。