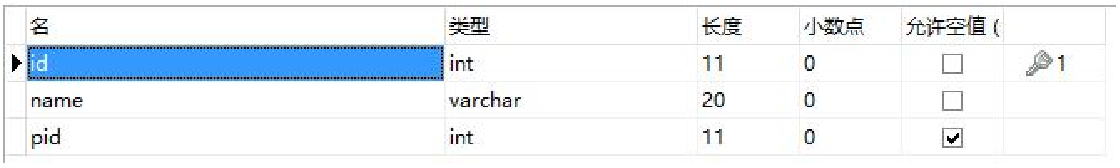
在实际开发中经常会遇到例如多级菜单这样的需求，最先考虑到的肯定是继续建表呗，但是这样只会增加表间复杂度，最好的做法就是单表操作，自己和自己连接，单表存储多级关系，然后从中获取多级关系。

我们这里以三级菜单为例，以商品目录为实际对比
先上数据表以及测试数据



名	类型	长度	小数点	允许空值 (	
id	int	11	0	☐	🔑1
name	varchar	20	0	☐	
pid	int	11	0	✓	

```
#一级
INSERT INTO t_product_type(id,name,pid) VALUES(1,'家电',null);
INSERT INTO t_product_type(id,name,pid) VALUES(2,'服装',null);

#二级
INSERT INTO t_product_type(id,name,pid) VALUES(3,'电视',1);
INSERT INTO t_product_type(id,name,pid) VALUES(4,'电脑',1);
INSERT INTO t_product_type(id,name,pid) VALUES(5,'男装',2);
INSERT INTO t_product_type(id,name,pid) VALUES(6,'女装',2);

#三级
INSERT INTO t_product_type(id,name,pid) VALUES(7,'液晶电视',3);
INSERT INTO t_product_type(id,name,pid) VALUES(8,'曲面电脑',3);
INSERT INTO t_product_type(id,name,pid) VALUES(9,'台式电脑',4);
INSERT INTO t_product_type(id,name,pid) VALUES(10,'笔记本电脑',4);
INSERT INTO t_product_type(id,name,pid) VALUES(11,'风衣',5);
INSERT INTO t_product_type(id,name,pid) VALUES(12,'哈伦裤',5);
INSERT INTO t_product_type(id,name,pid) VALUES(13,'连衣裙',6);
INSERT INTO t_product_type(id,name,pid) VALUES(14,'吊带衫',6);
```

可以非常明确其结构，三级菜单嘛。

我们这里第一个需求就是从一级开始查询出所有分类并按一级二级三级层次排列

先上实体类

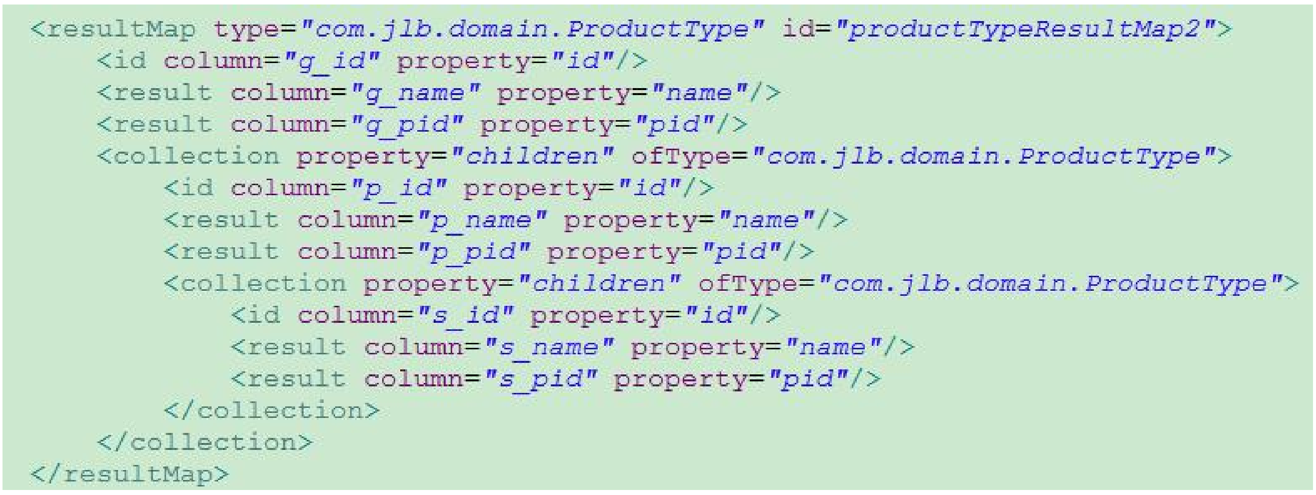
```
public class ProductType implements Serializable {

    private Integer id;
    private String name;
    private Integer pid;

    /* 视图属性 */
    private ProductType parent;
    private Set<ProductType> children;

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
        return result;
```

```
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		ProductType other = (ProductType) obj;
		if (id == null) {
			if (other.id != null)
				return false;
		} else if (!id.equals(other.id))
			return false;
		return true;
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Integer getPid() {
		return pid;
	}

	public void setPid(Integer pid) {
		this.pid = pid;
	}

	public ProductType getParent() {
		return parent;
	}

	public void setParent(ProductType parent) {
		this.parent = parent;
	}

	public Set<ProductType> getChildren() {
		return children;
	}

	public void setChildren(Set<ProductType> children) {
		this.children = children;
	}

	@Override
	public String toString() {
		return "ProductType [id=" + id + ", name=" + name + ", pid=" + pid + ",
parent=" + parent + ", children="
				+ children + "]";
	}

}
```

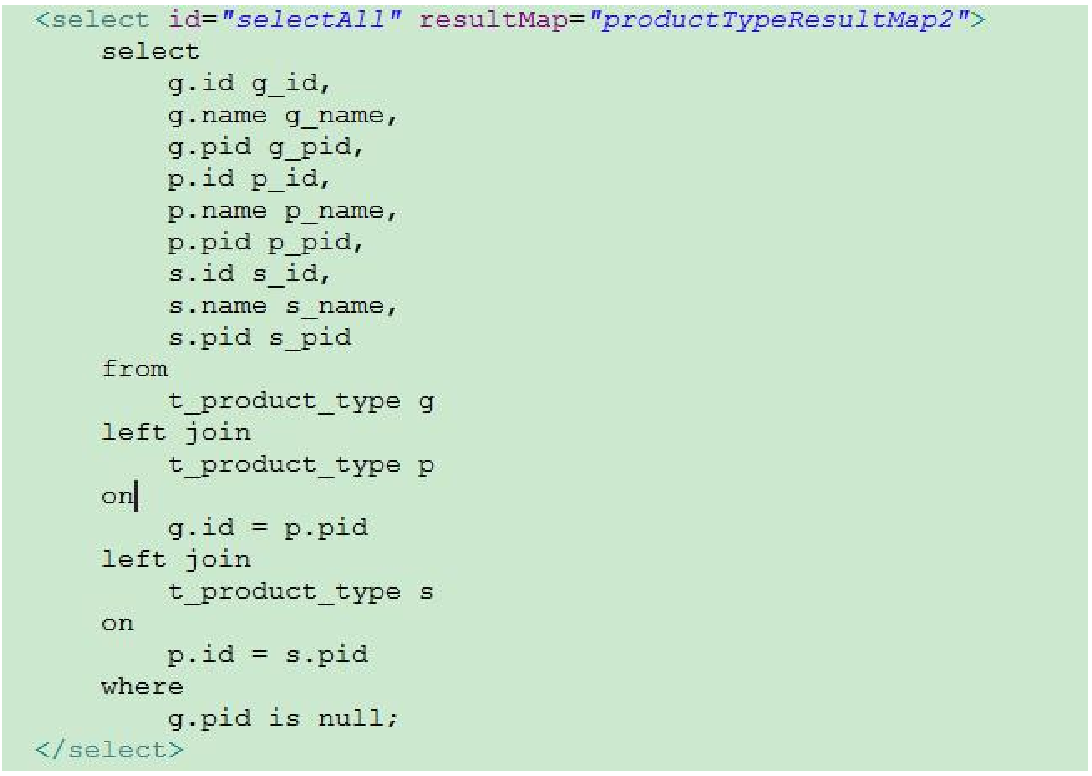
我们第二个视图属性children便是我们第一个需求的主角了，怎么处理呢，这里面无非就是要三次用到本表并进行外连接，直接上代码

```
<resultMap type="com.jlb.domain.ProductType" id="productTypeResultMap2">
		<id column="g_id" property="id"/>
		<result column="g_name" property="name"/>
		<result column="g_pid" property="pid"/>
		<collection property="children" ofType="com.jlb.domain.ProductType">
			<id column="p_id" property="id"/>
			<result column="p_name" property="name"/>
			<result column="p_pid" property="pid"/>
			<collection property="children" ofType="com.jlb.domain.ProductType">
				<id column="s_id" property="id"/>
				<result column="s_name" property="name"/>
				<result column="s_pid" property="pid"/>
			</collection>
		</collection>
	</resultMap>

	<select id="selectAll" resultMap="productTypeResultMap2">
		select
			g.id g_id,
			g.name g_name,
			g.pid g_pid,
			p.id p_id,
			p.name p_name,
			p.pid p_pid,
			s.id s_id,
			s.name s_name,
			s.pid s_pid
		from
			t_product_type g
		left join
			t_product_type p
		on
			g.id = p.pid
		left join
			t_product_type s
		on
			p.id = s.pid
		where
			g.pid is null;
	</select>
```

这表格的自动左对齐也是贼麻烦，Java代码勉强清楚，xml的完全就不易读了，放两张图片方便阅读吧。



```xml
<resultMap type="com.jlb.domain.ProductType" id="productTypeResultMap2">
    <id column="g_id" property="id"/>
    <result column="g_name" property="name"/>
    <result column="g_pid" property="pid"/>
    <collection property="children" ofType="com.jlb.domain.ProductType">
        <id column="p_id" property="id"/>
        <result column="p_name" property="name"/>
        <result column="p_pid" property="pid"/>
        <collection property="children" ofType="com.jlb.domain.ProductType">
            <id column="s_id" property="id"/>
            <result column="s_name" property="name"/>
            <result column="s_pid" property="pid"/>
        </collection>
    </collection>
</resultMap>
```



```xml
<select id="selectAll" resultMap="productTypeResultMap2">
    select
        g.id g_id,
        g.name g_name,
        g.pid g_pid,
        p.id p_id,
        p.name p_name,
        p.pid p_pid,
        s.id s_id,
        s.name s_name,
        s.pid s_pid
    from
        t_product_type g
    left join
        t_product_type p
    on
        g.id = p.pid
    left join
        t_product_type s
    on
        p.id = s.pid
    where
        g.pid is null;
</select>
```

自然再放我们的测试类

```java
@Test
    public void testSelectAll() {
        List<ProductType> grands = productTypeMapper.selectAll();
        for (ProductType grand : grands) {
            System.out.println("爷爷： " + grand.getName());
            for (ProductType parent : grand.getChildren()) {
                System.out.println("爸爸： " + parent.getName());
                for (ProductType child : parent.getChildren()) {
                    System.out.println("儿子： " + child.getName());
                }
            }
            System.out.println("================================");
        }

        System.out.println("------------------------------------------------");
        Set<ProductType> gs = new HashSet<ProductType>(grands);
        testDiSelectAll(gs);
    }
    public void testDiSelectAll(Set<ProductType> set) {
        for (ProductType productType : set) {
            System.out.println(productType.getName());
            if (productType.getChildren() != null) {
                testDiSelectAll(productType.getChildren());
            }
        }
    }
```

学了一招，看代码，点开表格再看，就是原本的对齐方式，贼麻烦，汗・・・
这边其实获取到grands的list后，已经一级级排列好了，直接传给前端，页面哪边要什么拿什么就是了，但是为了直接在Java中显示吧，三级就用了三次for循环，后来自己又写了递归，减少掉过度的手撕for循环。

我们这里第二个需求就是查第三级，得到对应的一二级
这里第一个视图属性parent就是主角了

```
    <resultMap type="com.jlb.domain.ProductType" id="productTypeResultMap1">
        <id column="s_id" property="id"/>
        <result column="s_name" property="name"/>
        <result column="s_pid" property="pid"/>
        <association property="parent" javaType="com.jlb.domain.ProductType">
            <id column="p_id" property="id"/>
            <result column="p_name" property="name"/>
            <result column="p_pid" property="pid"/>
            <association property="parent" javaType="com.jlb.domain.ProductType">
                <id column="g_id" property="id"/>
                <result column="g_name" property="name"/>
                <result column="g_pid" property="pid"/>
            </association>
        </association>
    </resultMap>

    <select id="selectById" resultMap="productTypeResultMap1">
        select
            g.id g_id,
            g.name g_name,
            g.pid g_pid,
            p.id p_id,
            p.name p_name,
            p.pid p_pid,
            s.id s_id,
            s.name s_name,
            s.pid s_pid
        from
            t_product_type g
        left join
            t_product_type p
        on
            g.id = p.pid
        left join
            t_product_type s
        on
            p.id = s.pid
        where
```

```
                s.id = #{id}
    </select>
```

图就不上了，篇幅太大，看不明白，点开看吧

测试代码

```
    @Test
    public void testSelectById() {
        ProductType child = productTypeMapper.selectById(10);
        System.out.println(child);
    }
```