▪ **Javabean中和DAO类配合使用的类**

VO

播报 编辑 讨论 上传视频

Javabean中和DAO类配合使用的类

全称为Value Object,就是一个普通的JavaBean。一般配合DAO来使用，用于实例化对象。

中文名	数据对象	外文名	Value Object

术语介绍

全称为Value Object,其实就是一个普通的JavaBean。一般配合DAO来使用，用于实例化对象。

一般将数据库的操作封装在DAO内，把从数据库查询到的信息实例化为VO，然后再进行各种操作。

四种实体类类型概念：VO、DTO、DO、PO

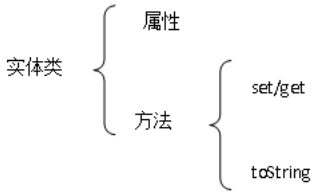
在项目应用中，VO对应于页面上需要显示的数据（表单），DO对应于数据库中存储的数据（数据表），DTO对应于除二者之外需要进行传递的数据。

一、实体类

百度百科中对于实体类的定义如下：

实体类的主要职责是存储和管理系统内部的信息，它也可以有行为，甚至很复杂的行为，但这些行为必须与它所代表的实体对象密切相关。

根据以上定义，我们可以了解到，实体类有两方面内容，存储数据和执行数据本身相关的操作。这两方面内容对应到实现上，最简单的实体类是POJO类，含有属性及属性对应的set和get方法，实体类常见的方法还有用于输出自身数据的toString方法。

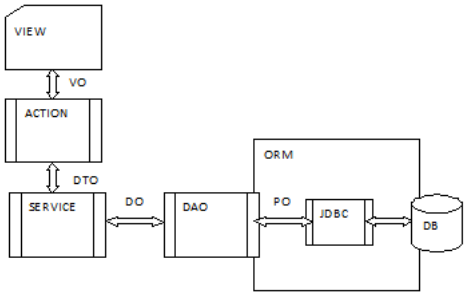


二、领域模型中的实体类

领域模型中的实体类分为四种类型：VO、DTO、DO、PO，各种实体类用于不同业务层次间的交互，并会在层次内实现实体类之间的转化。

业务分层为：视图层（VIEW+ACTION），服务层（SERVICE），持久层（DAO）

相应各层间实体的传递如下图：



项目中我们并没有严格遵循这种传递关系，但这种和业务层次的关联对我们理解各实体类的作用是有帮助的。（我们没有接触到PO的原因，我理解为ORM对PO进行了封装）

以下是资料的原文，上图是基于此绘制的：

概念：

VO（View Object）：视图对象，用于展示层，它的作用是把某个指定页面（或组件）的所有数据封装起来。

DTO（Data Transfer Object）：数据传输对象，这个概念来源于J2EE的设计模式，原来的目的是为了EJB的分布式应用提供粗粒度的数据实体，以减少分布式调用的次数，从而提高分布式调用的性能和降低网络负载，但在这里，我泛指用于展示层与服务层之间的数据传输对象。

DO（Domain Object）：领域对象，就是从现实世界中抽象出来的有形或无形的业务实体。

PO（PersistentObject）：持久化对象，它跟持久层（通常是关系型数据库）的数据结构形成一一对应的映射关系，如果持久层是关系型数据库，那么，数据表中的每个字段（或若干个）就对应PO的一个（或若干个）属性。

模型：

下面以一个时序图建立简单模型来描述上述对象在三层架构应用中的位置

l 用户发出请求（可能是填写表单），表单的数据在展示层被匹配为VO。

l 展示层把VO转换为服务层对应方法所要求的DTO，传送给服务层。

l 服务层首先根据DTO的数据构造（或重建）一个DO，调用DO的业务方法完成具体业务。

l服务层把DO转换为持久层对应的PO（可以使用ORM工具，也可以不用），调用持久层的持久化方法，把PO传递给它，完成持久化操作。

l 对于一个逆向操作，如读取数据，也是用类似的方式转换和传递，略。

三、项目中的实体类

项目中常见的实体类有VO，DO和DTO，命名规则也常是以相应字符串结尾，如*VO.Java。但是DTO不总是遵循这个规则，而通常与他的用途有关，如写成*Query.java，表示存储了一个查询条件。项目中实体类出现的业务层次也没有这么严格，例如我们可以在视图层就组装一个DO，也可以将一个VO从持久层传出来，所以与业务分层相关联的划分方法显得有些冗余。从项目代码中抽象出的理解是：VO对应于页面上需要显示的数据，DO对应于数据库中存储的数据，DTO对应于除二者之外需要进行传递的数据。

Java中Vo类的理解

发布于 2022-11-21 12:12:04 1.9K 0

先来解释两个概念

实体类（持久对象）=PO

值对象（新的对象）=VO

```
一、PO:persistant?object?持久对象
可以看成是与数据库中的表相映射的java对象。使用Hibernate来生成PO是不错的选择。

二、VO:value object值对象。
通常用于业务层之间的数据传递，和PO一样也是仅仅包含数据而已。但应是抽象出的业务对象
可以和表对应,也可以不,这根据业务的需要.

有一种观点就是：PO只能用在数据层，VO用在商业逻辑层和表示层。各层操作属于该层自己的数据对象
这样就可以降低各层之间的耦合，便于以后系统的维护和扩展。
如果将PO用在各个层中就相当于我们使用全局变量，我们知道在OO设计非常不赞成使用全局变量。
但是每次都得进行VO-PO的转换，也确实很烦。我觉得有时候也可以在某个商业逻辑或者表示层使用PO
此时在这个商业逻辑的过程中PO的状态是不发生变化的，比如显示一条商品详细信息的商业逻辑。
在开发过的项目中，规模都很小，我一直都把PO当VO用，因为PO确实很方便，结合Hibernate的DAO
我使用JAVA的集合对象作为值传递的载体，当然Struts也是我的不二之选。
我认为：在一些直观的，简单的，不易发生变化的，不需要涉及多个PO时，传递值还是使用PO好
这样可以减少大量的工作量（也就意味着减少bug，减少风险），也不需要担心未来的维护工作!

vo:value object,值对象
一般在java中用的多的是pojo:plain oriented java object
原始java对象，pojo一般和数据库中的表是一一对应的。
vo一般是来做值的存储与传递。
```

既然有了实体类与数据库中的字段一一对应了 那为什么还要VO呢

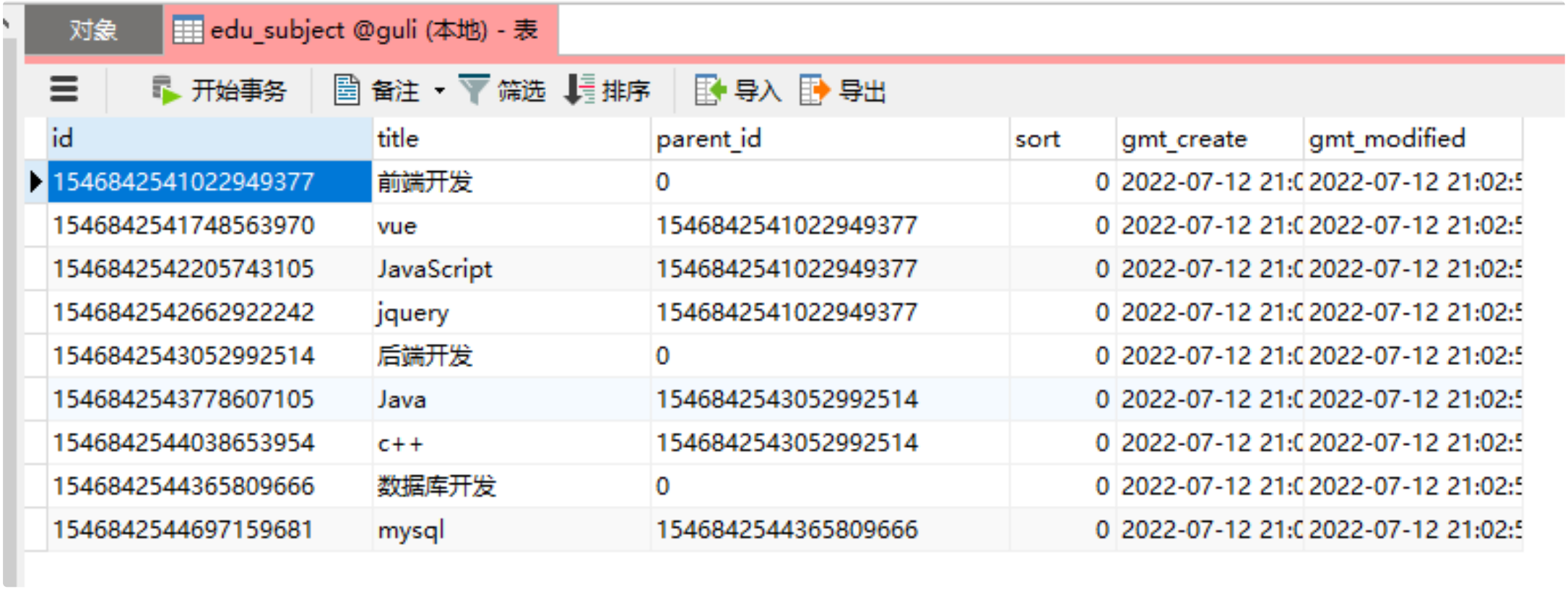
答案是因为在复杂的业务逻辑中，往往单一实体类无法满足我们的需求，就举个简单的例子，一个课程系统中有一级分类和二级分类，那么一个一级分类应该会对应多个二级分类，如果我们使用二级菜单的话，就应该可以实现这种联动的效果，一旦一级菜单确定下来了，那么二级菜单的下拉项中的选项只能是一级菜单下的二级菜单，也许有点绕 发证大意就是需要做到 一对

多

一个一级菜单对应多个二级菜单

下面就来一个例子吧，首先有一个在线课堂的项目中有一个分类表

我们先看一下表结构



id	title	parent_id	sort	gmt_create	gmt_modified
1546842541022949377	前端开发	0	0	2022-07-12 21:0	2022-07-12 21:02:5
1546842541748563970	vue	1546842541022949377	0	2022-07-12 21:0	2022-07-12 21:02:5
1546842542205743105	JavaScript	1546842541022949377	0	2022-07-12 21:0	2022-07-12 21:02:5
1546842542662922242	jquery	1546842541022949377	0	2022-07-12 21:0	2022-07-12 21:02:5
1546842543052992514	后端开发	0	0	2022-07-12 21:0	2022-07-12 21:02:5
1546842543778607105	Java	1546842543052992514	0	2022-07-12 21:0	2022-07-12 21:02:5
1546842544038653954	c++	1546842543052992514	0	2022-07-12 21:0	2022-07-12 21:02:5
1546842544365809666	数据库开发	0	0	2022-07-12 21:0	2022-07-12 21:02:5
1546842544697159681	mysql	1546842544365809666	0	2022-07-12 21:0	2022-07-12 21:02:5

id就是主键

titie是名称

那么parentId就是上级分类的ID 如果值为0则为一级分类 如果为其他

Springboot中的实体类PO：

```
1  package cn.tompro.eduservice.entity;
2
3  import com.baomidou.mybatisplus.annotation.FieldFill;
4  import com.baomidou.mybatisplus.annotation.IdType;
5  import java.util.Date;
```

```java
import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import java.io.Serializable;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.Accessors;

/**
 * <p>
 * 课程科目
 * </p>
 *
 * @author testjava
 * @since 2022-07-12
 */
@Data
@EqualsAndHashCode(callSuper = false)
@Accessors(chain = true)
@ApiModel(value="EduSubject对象", description="课程科目")
public class EduSubject implements Serializable {

    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value = "课程类别ID")
    @TableId(value = "id", type = IdType.ID_WORKER_STR)
    private String id;

    @ApiModelProperty(value = "类别名称")
    private String title;

    @ApiModelProperty(value = "父ID")
    private String parentId;

    @ApiModelProperty(value = "排序字段")
    private Integer sort;

    @ApiModelProperty(value = "创建时间")
```

```
    @TableField(fill = FieldFill.INSERT)
    private Date gmtCreate;

    @ApiModelProperty(value = "更新时间")
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Date gmtModified;


}
```

但是 正如我在前面说到过的 这种单一实体类 是满足不了我们的需求的

因为例如下图中的需求

我们选中一个一级分类之后 那么下一个下拉列表中的选项只能是 已选中的一节分类下的对应的二级分类

是会有一个联动的反应，所以我们的单一实体类就不能做到了。

这时候，引入我们的Vo类

首先新建一级分类Vo类

```
package cn.tompro.eduservice.entity.subject;

import lombok.Data;

import java.util.ArrayList;
import java.util.List;

```

```
/**
 * 2 * @Author: AkaTom
 * 3 * @Date: 2022/7/12 21:43
 * 4 一级分类
 */
@Data
public  class OneSubject {

    private String id;
    private String title;

    //一个一级分类有多个二级分类
    private List<twoSubject> children= new ArrayList<>();

}
```

然后是二级分类Vo类

```
package cn.tompro.eduservice.entity.subject;

import lombok.Data;

/**
 * 2 * @Author: AkaTom
 * 3 * @Date: 2022/7/12 21:43
 * 4 二级分类
 */
@Data
public  class twoSubject {
    private String id;
    private String title;
}
```

然后在前端的时候我们则需要获取我们的vo类

vue中的methods：

```
    //点击某个一级分类，触发change，显示对应二级分类
    subjectLevelOneChanged(value) {
      //value就是一级分类id值

```

```
      //遍历所有的分类，包含一级和二级
      for ( var  i=0;i< this .subjectOneList.length;i++) {
        //每个一级分类
        var  oneSubject =  this .subjectOneList[i]
        //判断：所有一级分类id 和 点击一级分类id是否一样
        if (value === oneSubject.id) {
          //从一级分类获取里面所有的二级分类
          this .subjectTwoList = oneSubject.children
          //把二级分类id值清空
          this .courseInfo.subjectId = ''
        }
      }
    },
    //查询所有的一级分类
    getOneSubject() {
      subject.getSubjectList()
        .then(response => {
          this .subjectOneList = response.data.list
        })
    },
```

data：

```
subjectOneList:[],//一级分类
subjectTwoList:[]//二级分类
```

页面：

```
<!-- 所属分类 TODO -->
      <el-form-item label="课程分类">
        <el-select
          v-model="courseInfo.subjectParentId"
          placeholder="一级分类" @change="subjectLevelOneChanged">

          <el-option
            v- for ="subject in subjectOneList"
            :key="subject.id"
            :label="subject.title"
            :value="subject.id"/>

```

```
        </el-select>

        <!-- 二级分类 -->
        <el-select v-model="courseInfo.subjectId" placeholder="二级分类">
          <el-option
            v-for="subject in subjectTwoList"
            :key="subject.id"
            :label="subject.title"
            :value="subject.id"/>
        </el-select>
      </el-form-item>
```

后端service层中的主要方法：

```
public List<OneSubject> getAllOneTwoSubject() {
        //1 查询所有1级分类
        QueryWrapper<EduSubject> wrapperOne= new QueryWrapper<>();
        wrapperOne.eq("parent_id","0");
        List<EduSubject> oneSubjectList = baseMapper.selectList(wrapperOne);

        //2 查询所有二级分类
        QueryWrapper<EduSubject> wrapperTwo= new QueryWrapper<>();
        wrapperTwo.ne("parent_id","0");
        List<EduSubject> twoSubjectList = baseMapper.selectList(wrapperTwo);
        //创建list集合 存储最终封装数据
        List<OneSubject> finalSubjectList= new ArrayList<>();
        //封装一级分类
        //查询出来所有的一级分类list集合遍历，得到每个一级分类对象，获取每个以及分类对象值
        //封装到要求的list集合里面
         for (int i = 0; i < oneSubjectList.size(); i++) {
            //得到oneSubject每个对象
            EduSubject eduSubject = oneSubjectList.get(i);
            //把edusubject的值获取出来 放到onesubject中
            OneSubject oneSubject= new OneSubject();
//              oneSubject.setId(eduSubject.getId());
//              oneSubject.setTitle(eduSubject.getTitle());
            BeanUtils.copyProperties(eduSubject,oneSubject);
            finalSubjectList.add(oneSubject);
            // 在一级分类循环遍历查询所有的二级分类
            // 创建list集合封装每一个一级分类的二级分类

```

```
            List<twoSubject> twoFinalSubjectList= new ArrayList<>();
            for (int m = 0; m < twoSubjectList.size(); m++) {
                EduSubject eduSubject1=twoSubjectList.get(m);
                if (eduSubject1.getParentId().equals(oneSubject.getId())){
                    //把eduSubject1复制到twosubject中 放到twofinalsubjectlist中
                    twoSubject twosubject= new twoSubject();
                    BeanUtils.copyProperties(eduSubject1,twosubject);
                    twoFinalSubjectList.add(twosubject);
                }
            }
            oneSubject.setChildren(twoFinalSubjectList);
        }
        //封装二级分类
        return finalSubjectList;
    }
```