由一个简单的程序谈起－－之三（精华）

江苏 无锡 缪小东

本篇继续和大家探讨上面的程序。在此篇我将和大家共同构造一个 GUI 与业务逻辑分开的程序，同时对程序的模块进行一些抽象，在较高的层次构建这个简单的系统。同时我们还自己创建一个底层的数据结构，此数据结构不仅仅保存数据，而且具有 Cache 的特性……

一、GUI 设计

一谈到 GUI 其实是相当繁杂的！我构建这个 GUI 也化了小半天！不过我们的程序目的不是怎么构建 GUI，而是如何对 GUI 进行抽象，建立一个独立于 GUI 的抽象模板！以下是我们的 GUI。

图中将整个 GUI 分为两部分：1.显式部分（蓝色框框）；2.控制部分（红色框框）。显式部分主要是显式持续层（Psrsistence Layer）中的记录。控制部分主要是控制整个程序的逻辑运行，它有一些按钮组成。它主要负责：根据用户的需要显式记录，同时完成和持续层的交互。一般的 GUI 程序中都可以分为显式(View)和控制(Controller)这两部分，这看起来似乎和 MVC 架构很象，本例中我们没有使用 MVC 架构，而是使用类似调停者模式的思路，构建这个程序。控制部分就是一个调停者，协调视图（View）和 Cache 的工作。

二、GUI 的抽象

在做程序前我发现程序的显式部分只完成几样工作：1.显式某个学生记录（Student 类的实例）；2.获取用户输入的数据；3.控制界面的可编辑状态。

可编辑状态就是指，在我们单击界面上的“上一个”、“下一个”、“删除”按钮以及程序运行的绝大部分时间都是处于不可编辑的状态，只可以显示。在我们单击界面的“插入”、“更新”按钮时，程序的界面处于可编辑状态。因此我们将界面的需求抽象为以下接口：

```
//View.java
```

```
public interface View {
    public Student Collect();
    public void display(Student stu);
    public void setAllEnable(boolean b);
}
```

这样整个程序的显式就是完全抽象的了！以后不管你如何变换显式部分，只要实现此接口就可以插入本程序中。下面是本程序中的实体对象 Student 的类：

```
import java.io.Serializable ;

public class Student implements Serializable{
        private String name = "Unknow";
        private String grade = "高一";
        private String clas= "01 班";
        private int age = 18 ;
        private boolean male = true ;
        private String comment = "";

        public Student(){}

        public void setName(String name){
            this.name = name ;
        }
        public String getName(){
            return this.name;
        }
        public void setGrade(String grade){
            this.grade = grade ;
        }
        public String getGrade(){
            return this.grade;
        }
        public void setClas(String clas){
            this.clas= clas ;
        }
        public String getClas(){
            return this.clas;
        }
        public void setAge(int age ){
            this.age= age ;
        }
        public int getAge(){
            return this.age;
        }
        public void setAgender(boolean male){
            this.male = male;
```

```
    }
    public boolean isMale(){
        return this.male;
    }
    public void setComment(String comment){
        this.comment= comment;
    }
    public String getComment(){
        return this.comment;
    }

    static final Student BlankStudent = new Student();

    public static Student createBlankStudent(){
        return BlankStudent    ;
    }
}
```

Student 类也很简单吧！就是一些基本的 Get、Set 方法。***（其实不管多大的程序都是有一些很小很小的类组合而成的！当程序很小时如何设计这些类，以及设计他们的协作关系相对比较简单！但是如何写一个具有几千个甚至几万个类，以及设计他们的交互行为，就需要专门的知识、需要你对设计模式等具有极其深厚的理解！）***最后给出了一个特殊方法：静态的创建本类的一个实例的方法（尽管在本程序使用很少）。但是这种方式使用很多，主要是为了避免在系统中创建多个相同的对象。在某些情况下我们要在很多地方使用某个对象，是否每个地方都创建该对象呢！其实是没有必要的，我们可以在系统中仅仅创建一个，在需要的时候永远引用这个对象，从而节约内存，还可以保证所有地方的这些对象是完全一致的，可以在一处修改，达到全部改变的目的，所谓“牵一发而动全身”。例如在 Swing 库中的 BorderFactory 类，其中包含一些 static final 的 Border 的实例，有兴趣的读者可以查阅 BorderFactory 的源代码！（闲聊）以上是对于视图的抽象！下面就应该是对控制器的抽象。

在一般的程序中这些由按钮组成的控制器，它们一般完成一些对应的动作，不过这些动作是没有返回值的。对应于我们的程序，程序需要：1.向前遍列；2.向后遍列；3.删除；4.插入；5.更新等几个操作，但是在本程序中，一旦您需要插入或者更新时，你会在 View 中输入数据，输入完成后你要确认这些数据将此数据存入 Persistence Layer。因此我们需要下面的功能：6.确认。根据上面的分析我们定义以下的接口：

```
//Controller.java
public interface Controller /**extends DOA**/{
    public void next()throws Exception;
    public void previous()throws Exception;
    public void delete()throws Exception;
    public void update()throws Exception;
    public void insert()throws Exception;
    public void ack()throws Exception;
}
```

三、持久层的抽象

上一篇提出了持久层抽象的原因及其如何抽象！这里我们仅仅给出该接口。这类似与 Data Access Object 模式，具体可以查阅相关书籍。（上篇文章中笔误！不是 Data Object Access）这是一个数据存取对象，主要完成数据的存取！在我们的程序中，我们的持久层提供：1.向前遍列；2.向后遍历；3.删除当前对象；4.向持久层增加某个对象；5.保存操作。

```
// DOA.java
import java.io.IOException;

public interface DOA {
    //public void init()throws Exception;
    public void save()throws Exception ;
    public Student next()throws Exception;
    public Student previous()throws Exception;
    public void delete()throws Exception;
    public void add(Student stu)throws Exception ;
}
```

不是所有的持久层都提供以上的功能，这只是对于与我们程序的抽象！（我经常讲的设计模式、设计理念不是一成不变而是变化的，是一种思维方式就是这个意思！设计模式提供一种理念、一种思维方式，而不是一个模板。不是你对 23 种经典设计模式了如指掌，倒背如流就说明你真正懂设计模式、设计理念！关键在合适的场景运用）

上面的接口中提供一个 sava 方法，这在其它程序中很少见的，那我为什么有此方法呢！主要是我们的持久层非常、非常特殊，它类似于一个 Cache－－总是在内存中保存一定数量的对象，同时在需要相应元素的时候会从本地文件系统得到这些记录！具体请关注下篇文章。

四、业务流的抽象

讲到此处我们关于程序的各个部分都有了相应的抽象，是否就该结束了，该讲代码如何设计呢！不过我在做此程序的时候发现一个现象：在我们具有 View、Controller 和 DOA 后好像整个程序的交互的逻辑就已经初步显现了！

想想吧！程序中 Controller 要做的事，不就是对 View 和 DOA 进行协调吗！我们有了 View 接口和 DOA 不就行了吗!!一个 Controller 能完成的事务流的抽象，顺其自然我将此类命名为 ControllerXXXX，此类肯定要实现或者继承 Controller 接口吧!考虑考虑该类只可能为一个类，不可能为实现 Controller 接口的接口！为什么啊？很简单因为它包含了方法的实现，尽管这些方法是接口的未实现的方法。以下就是这个类的代码：

```
// ControllerTemplate.java
public /**abstract*/ class ControllerTemplate implements Controller {
    DOA model = null ;
    View view = null ;

    public ControllerTemplate(DOA doa , View view ){
```

```
        this.model = doa ;
        this.view = view;
    }

    public void next()throws Exception{          //控制器上的“下一个”按钮的操作
        Student stu = model.next();              //得到模型中的下一个 Student 记录
        view.display(stu);                       //在视图中显式此 Student 记录
        view.setAllEnable(false);                //一定要记住此时界面不可编辑
    }

    public void previous()throws Exception{      //同上
        Student stu = model.previous();
        view.display(stu);
        view.setAllEnable(false);
    }

    public void delete()throws Exception{
        model.delete();                          //在模型中删除当前记录
        Student stu = model.next();              //返回模型中下一个记录
        //if(stu == null)
        view.display(stu);                       //在视图中显式下一个记录
        view.setAllEnable(false);                //别忘了界面不可编辑
    }

    public void update()throws Exception{        //和 ack 完成一个逻辑
        Student stu = view.Collect();            //搜集视图中的数据，变为一个记录
        model.delete();                          //删除模型中的当前记录
        view.display(stu);                       //显式的可是刚才的对象哦
        view.setAllEnable(true);                 //只是此时的界面可以编辑了
    }

    public void insert()throws Exception{        //很有意思的方法，与 Update 类似
        view.display(new Student());             //显式一个空白的 Student 记录，
                                                 //应该用 Student 类的那个静态方法哦!!
        view.setAllEnable(true);                     //可以编辑哦
    }

    public void ack()throws Exception{               //确认。和 insert 、update 完成独立的逻辑
        Student stu = view.Collect();                //搜集视图中的数组
        model.add(stu);                              //将此对象加入模型中
        view.display(stu);                           //同时视图还显式此对象哦
    }
}
```

简单吧！就是协调 View 和 DOA 的关系！大家可以看到在构造这个类的时候，我本想这个类为一个 abstract 的抽象类。但是，在后来我们可以看到具体控制器的实现 ControllerPanel 是一个从 JPanel 继承的子类，大家肯定知道：在 java 中不支持多重继承，因此我们将其改为一个一般的非抽象类。这样就不可

以通过继承重用此业务流的抽象了！可惜!! 可惜!! 不过很幸运 java 中可以通过包含或者成为组合来达到重用，不过 Java 高手都推荐通过组合达到重用的目的！因祸得福啊!!!

关于这个类的注释，请仔细研究！限于篇幅的限制我将其放在代码中！作为注释了！

关于 insert 和 update 他们有一些共同点：1.点击这两个按钮后界面都处于可编辑状态；2.都没有完成真实数据的输入；3.都要显式要一个记录.....

仔细研究吧！想想为什么这么做！不这么做又该如何做！

下篇将讲述控制器、视图以及主程序的实现！再下篇将讲述底层数据结构的设计！最后会讲述系统设计的意图！可能还有其它！

更多精彩请关注：

http://blog.163.com/miaoxiaodong78/