计算机学院软件工程与实践课程实验报告

|  |  |  |
| --- | --- | --- |
| 实验题目：实验十一 软件体系结构设计（三），设计模块（一） | | 学号：201900130126 |
| 日期：2022.5.29 | 班级： 计科19级1班 | 姓名： 史子涵 |
| Email：2567626732@qq.com | | |
| 实验目的：  1.练习故障树分析  2.培养设计原则实践的能力  3.学习依赖注入 (dependency injection)  4.学习设计模式，能在项目设计中运用设计模式进行面向对象设计 | | |
| 实验软件和硬件环境：  Windows 10操作系统 | | |
| 实验步骤与内容：  团队名称：没有bug软工实践团队  团队成员（排名不分先后）：史子涵 201900130126，刘诗婷 201918130217，陈文盼 201920130238，赵一帆 201905130197，郑晓旭 201900130107。  首先，我们先给组员划分了所负责的模块。史子涵同学负责学生选课模块，郑晓旭同学负责管理员管理课程信息模块、陈文盼同学负责用户信息管理模块、刘诗婷同学负责批量信息管理模块、赵一帆同学负责信息查询模块。所有组员都需要从设计原则的几个方面，对负责设计的模块进行评估，思考存在的问题和解决方案思考。  1. 第五章课后习题14，故障树转割集树练习。（此部分为刘诗婷同学负责）  针对自己项目分析、描绘故障树，分解为割集树 （附到最终提交的SAD）。  故障树：    分解为割集树：    G1：违反安全性  G2：不合理的使用  G3: 密码问题  G4：不合法访问  A1：密码泄露  A2：密码未改变  A3：密码多次错误  A4：重复登陆  A5：异常退出  A6：用户查看信息  A7：用户修改信息  2. 参考教材6.2，结合项目的进程和开发历程，从设计原则的几个方面，组员对负责设计的模块进行评估，思考存在的问题和解决方案。  史子涵同学：负责设计的模块：学生选课模块。学生端可以通过登录该模块，选择课程进行选课。  问题1：如果一个学生在查看课程信息，刚好某个管理员登录系统修改了课程信息，那么学生是否会看到错误的课程信息？  解决方案：可以设置专门的时间段，时间段内由管理员对课程信息进行维护，学生不可查看；该时间段外学生可自由登陆。  问题2：如果两个学生同时选课，课容量信息如何更改？  解决方案：使用锁来控制每次只有一个学生选课访问并更改课容量信息。  郑晓旭同学：我设计的模块是管理员管理课程信息模块  具体功能如下：管理员可以修改课程信息，进行课程评分  评估：  模块化：我在本部分使用了自顶而下和自下而上相结合的方法，各个模块之间耦合度较小。  接口：本部分使用的接口较少，主要是管理员和学生两部分，管理员有3个接口，分别对应  其三个功能，学生只需要一个接口。我定义的接口能够完整的实现相应功能。  信息隐藏：由于我设计的这一部分所涉及的方法较少，因此对于各个方法没有使用信息隐藏，只是对对象进行了隐藏。  增量式开发：我在设计的时候应用了增量式开发，对各个模块的依赖关系进行分析后作出了使用图，然后设计。  抽象：我通过对模块进行分解，实现了层次结构，从而进行抽象  通用性：我设计的这个模块通用性并不好，基本只能在本项目中使用，主要是因为我是根据本项目进行的设计，未考虑通用性。  对于我的模块，我认为存在的问题是通用性与抽象做的不好。我准备通过将原来的特定上下文环境进行参数化来让它满足各种环境的工作。  陈文盼同学：我设计的模块是用户信息管理模块  模块化角度：  接口：区分管理员登录与学生登录的方法。需不需要将身份作为参数提供。  解决方案：作为参数提供。或不作为参数提供都可以。  作为参数提供是否会提示身份选择错误。  解决方案：若是作为参数提供，则会提示。  信息隐藏：  增量式开发：  抽象：  通用性：管理员登录与学生登录，学生注册和管理员注册，管理员信息修改和学生信息修改设计方面具有通用性。  刘诗婷同学：我负责的模块主要是批量信息管理模块，主要内容是批量课程信息管理和批量学生信息管理模块。  首先，数据库在后台，且遇到错误时会自动断开，所以具有可靠性和安全性；由于批量处理的命令比较简明，基本不会出现命令错误，所以鲁棒性也基本保证；PHP语言编写的代码也容易进行完善和增加内容，所以也有一定的可修改性和可扩展性；而且利用mysql数据库操作效率也比较高。  目前存在的问题为功能相对比较简单，只限于简单的上传损坏情况和输出信息更改后的结果。  赵一帆同学：我负责的模块是信息查询模块。具体功能就是查询学生信息和教师信息，以及通过课程名称、课程编号或教室姓名查询课程信息。  下面将从六个方面对这个模块进行评估：  1.模块化  对于这个功能，我们如果从与其他模块的耦合度分析，会发现其实耦合度很小，几乎可以看成是一个独立的个体。其次是内聚赌，报修模块功能集中，就围绕保修内容，报修提交，保修结果返回几个方面进行实现，主题集中。所以模块化这方面还是做的足够完善的。  2.接口  我们这个项目并不是那种一般的通过调用就可以的。我们因为是一个web程序，所以对于接口，一般都是设置一个新连接导向一个新页面，然后实现具体操作。但是在实现的时候我们对于数据库的操作没有进行过一个封装，如果查看网页的源代码，甚至可以看到直接队数据库的操作对于信息隐藏这方面确实不是做得很好。这可能会存在一些问题。  3.信息隐藏  对于B/S架构的项目来说，一般安全性会普遍低于C/S架构，因为他的信息数据等，都是通过网页代码传送的。所以如果解析了网页源代码，那就有可能会导致信息泄露。这是针对于在网页上的操作，但是如果是到了数据库这里，安全性还是有保障的。  4.从增量式开发上考虑  因为我们的模块功能划分明确，在外部各模块之间有相联系但是却又各自逻辑分明，在模块内部各子功能紧密结合。对于增量开发方式非常适合。  5.抽象  这个模块需求明确，整体系统之间也能协调工作，对这个模块进行抽象的话十分容易。就比如接口就是ui\_id(学生ID),ui\_password(学生密码)。当然，也可以增加图片描述功能。毕竟我们的功能是报修，如果有图片的花就更加方便了。  6.通用性  其实，只要是有关这类人员信息管理的模块，都能够通过修改部分代码打到复用的目的的，就算不能够复用，这个模式，代码结构，也是可以作为一个参考的。所以说，还是有一定的复用性。  存在的问题：  通过我们上面对模块的评估，我们可以发现该模块在接口和信息隐藏方面存在一定的问题。  对于前者，我们可以稍费一点功夫，把能够直接联系数据库的操作给封装起来，这是面向对象设计中常常用到的，不仅有助于模块化管理，在信息安全上有有一定用处。后者的花，对于我们在网页上流动的数据，我们有两个解决方案：第一个是在我们输入数据的时候，通过javascript对数据进行一定的加密，我们在服务器端在通过解码将其解析出来，这是第一重保险，第二重保险是，我们要避免从其他非本系统网页能够导入到本页面的情况，所以我们需要设置一个跳转检查机制，如果某次的跳转不是从登录界面正经转入的，那我们都应该拒绝其访问，并将其转到登陆界面，实现一个双重保险。  3. 阅读下面DI资料（或查阅其它相关资料），学习依赖注入技术（此部分由史子涵同学负责）  Dependency injection - Wikipedia  Dependency Injection-A Practical Introduction.pdf  依赖注入是解决一个对象如何获得它依赖的其他对象的技术。例如，一个对象A可以通过注入器(Injector)获得它所依赖的其他对象B、C。其中B、C对象是A对象的依赖(Dependencies)被传入到A对象中，一般被称为Service，A作为接收对象一般叫Client，而负责将依赖传递入接收对象的代码叫做Injector。  因此我们可以看到Client具体使用什么Service是由Injector去指定的，并传入到Client中，而不是Client自己去指定或创建的，这一点也是依赖注入的基本要求。  依赖注入是控制反转(Inversion of Control)这种技术的一种形式。目的是为了进一步解耦对象。Client需要某种服务，但是它仅仅知道这种服务(接口），而不知道具体实现服务的是哪个类。Client把找到这个实现类的工作交给Injector去完成。Injector会把找到的实现类的实例通过Client提供的注入方式注入到Client中。这样一来Client就和具体的服务实现解耦了，Client既无需要知道具体的服务实现，也能使用到服务了。  依赖注入将Client的依赖的创建和Client本般的行为分割开来。依赖注入促进了松耦合编程、依赖反转、单一责任原则等这些编程设计思想。  依赖注入这种技术中，一般包含四种角色，各司其职：  1.Service 对象，它包含了Client要使用的服务。  2.interface接口，通过它Client只知道有什么服务，而不需要知道具体实现。  3.Client对象，它的行为依赖于它所使用的服务  4.Injector注入器，它负责构造具体的Service并将其传入到Client对象中。  Client通过以下方式(不限于)来向Injector提供注入口：  1.构造器注入，即通过Client的构造器注入，简单来说就是new这个Client对象时，通过它的构造函数把依赖传进去。  2.set方法注入，即Client对象提供一些set方法，Injector就可以调用set方法把依赖传进去。  3.接口注入，即这些依赖的接口提供一个注入器方法  4. 阅读下面设计模式资料（或查阅其它相关资料），结合项目的进程和开发历程，分析项目采用了那些设计模式  Design Patterns-Elements of Reusable Object-Oriented Software.pdf  The GoF Design Patterns Reference.pdf  Design Patterns – Wikipedia  郑晓旭同学：根据我们的设计，我觉得我们应该采用伪合理的设计过程。先设计最难的最不明确的部分，因为隐蔽问题所引发的异常可能会迫使整个设计做出改变。然后设计我们熟知的拿手的部分。最后按照理想过程编写我们的文档。  刘诗婷同学：我们的项目是学生选课信息管理系统，在这个项目中，我们使用了自顶向下和自下而上相结合的方法。我们先使用自顶向下的方法，将该系统分成5个模块：学生选课模块，管理员管理课程信息模块、用户信息管理模块、批量信息管理模块、信息查询模块。这五个模块有5名队员分别进行，我负责的是批量信息管理模块，在设计各自的模块时，我们既使用了自顶而下又使用了自下而上的方法。我们先对最难的一部分进行设计，然后对其他部分自定而下分块设计。  赵一帆同学：我们的学生选课信息管理系统主要采用了面向对象的设计方法。首先我们这个项目的要求就是面向用户的包括学生与管理员，向他们提供便于操作的相应输入输出功能。我们在设计的过程中使用了很多UML用例图、对象图和类图用来表示相关数据和功能，并用这些作为设计目标，这些软件开发方法符合建立模型体现抽象思维过程和面向对象的方法。此外，我们的这一软件系统将相应计算机系统中的数据以及与这个数据相关的一切操作语言都封装了起来，提供的是便于理解和使用功能接口，符合面向对象方法中“封装”的特点。在我们的软件生存期的各个阶段所开发出来的“部件”都是相应的类，在面向对象生存期的各个阶段对各个类进行细化，使类成为分析设计和实现的基本单元，而且它们也有继承和对象组合，这些也是面向对象设计方法的特点。  陈文盼同学：我们所承接的项目是：学生选课信息管理系统。我们所采用的软件设计方法，是自顶而下的设计方法。也可以从我们的工作中，看到我们的设计方法。  自顶而下的设计方法主要是以下几个步骤：  1.编写问题：  实际上，比如我们最初的时候的需求分析，就是去发现我们需要解决的问题。  1.编写主要模块：  我们的设计初期，做了不少的准备工作，比如软件体系结构，模块划分之类的。我们主要都是在大的方向上进行一个划分，将项目分化成几个方面的问题。比如我们的项目就可以分成管理员模块，学生模块，选课模块等。  2.编写更小的模块。  这就是我们的更细小的划分了，比如管理员模块需要实现什么功能，学生模块需要实现什么功能等等。  3.模块组合  当我们每个模块都完成之后，就进行组合，形成一个完整的系统，进行测试。  史子涵同学：  采用了面向对象的设计方法。  学生选课信息管理系统基于数据库进行操作，有学生、管理员、课程信息、用户信息等多种对象实体，不同的实体有不同类型的数据，所有的操作都是针对这些对象展开的。  5. 给出4种设计模式的例子（语言不限，以组为单位），并总结其特点 （保存到每个小组选定的协作开发平台上）（此部分由郑晓旭和陈文盼同学负责）  1> 创建型模式—工厂方法模式（定义一个创建对象的接口，但由子类决定要实例化的类是哪一个，单产品系）  （1） 实例：  // 产品  public interface ICar {}  public class MidCar implements ICar {}  public class UpCar implements ICar {}  // 抽象工厂  public abstract class AbstractFactory {    public abstract ICar create();  }  // 实例工厂  public class MidCarFactory extends AbstractFactory {    @Override  public ICar create() {  return new MidCar();  }  }  public class UpCarFactory extends AbstractFactory {    @Override  public ICar create() {  return new UpCar();  }  }  （2） 特点：  把类的实例化延迟到子类；  一个工厂对应生产一个产品。  2> 创建型模式--抽象工厂模式（提供一个接口，用于创建相关或依赖对象的家族，而不需要明确指定具体类，多产品系）  （1） 实例：  // 车产品  public interface ICar {}  public class MidCar implements ICar {}  public class UpCar implements ICar {}  // 颜色产品  public interface IColor {}  public class RedColor implements IColor {}  public class BlueColor implements IColor {}  // 抽象工厂  public abstract class AbstractFactory {    public abstract ICar createMidCar() throws Exception;  public abstract ICar createUpCar() throws Exception;  public abstract IColor fillRed() throws Exception;  public abstract IColor fillBlue() throws Exception;  }  // 车工厂，生产除车外的产品，会抛出自定义异常  public class CarFactory extends AbstractFactory {    @Override  public ICar createMidCar() throws Exception {  return new MidCar();  }  @Override  public ICar createUpCar() throws Exception {  return new UpCar();  }  @Override  public IColor fillRed() throws Exception {  throw new CqException("Please use ColorFactory to create!");  }  @Override  public IColor fillBlue() throws Exception {  throw new CqException("Please use ColorFactory to create!");  }  }  // 颜色工厂，生产除颜色外的产品，会抛出自定义异常  public class ColorFactory extends AbstractFactory {    @Override  public IColor fillRed() throws Exception {  return new RedColor();  }  @Override  public IColor fillBlue() throws Exception {  return new BlueColor();  }  @Override  public ICar createMidCar() throws Exception {  throw new CqException("Please use CarFactory to create!");  }  @Override  public ICar createUpCar() throws Exception {  throw new CqException("Please use CarFactory to create!");  }  }  （2） 特点：  当一个产品族中的多个对象被设计成一起工作时，它能保证客户端始终只使用同一个产品族中的对象；  产品族扩展非常困难，要增加一个系列的某一产品，既要在抽象工厂加代码，也要在工厂生产者里加代码。  3> 创建型模式—建造者模式（将复杂的构建与其表示相分离，使得相同的构建过程可以创建不同的表示）  （1） 实例：  /\*\*  \* 自带建造者（除了必须字段，其它字段是可选的，易于自己选择不同的表示）  \*  \* 发送邮件参数  \*/  public class SendEmailParams {  // 收件人邮箱，支持多个，用英文逗号隔开  private final String to;  // 发件人账号和密码  private final EmailAccount emailAccount;  // 抄送，支持多个，用英文逗号隔  private final String cc;  // 秘密抄送，支持多个，用英文逗号隔开  private final String bcc;  // 邮件标题  private final String subject;  // 邮件正文  private final String body;  // 邮件内容编码，默认utf-8  private final String encoding;  // 附件列表，每个元素包含fileName和content两个字段  private final List<Attachment> attachments;  private SendEmailParams(Builder builder) {  this.to = builder.to;  this.emailAccount = builder.emailAccount;  this.cc = builder.cc;  this.bcc = builder.bcc;  this.subject = builder.subject;  this.body = builder.body;  this.encoding = builder.encoding;  this.attachments = builder.attachments;  }  public static class Builder {  // 必须的参数  private final String to;  private final EmailAccount emailAccount;  // 可选参数  private String cc = "";  private String bcc = "";  private String subject = "";  private String body = "";  private String encoding = "";  private List<Attachment> attachments = new ArrayList<Attachment>();  public Builder(String to, EmailAccount emailAccount) {  this.to = to;  this.emailAccount = emailAccount;  }  public Builder cc(String cc) {  this.cc = cc;  return this;  }  public Builder bcc(String bcc) {  this.bcc = bcc;  return this;  }  public Builder subject(String subject) {  this.subject = subject;  return this;  }  public Builder body(String body) {  this.body = body;  return this;  }  public Builder encoding(String encoding) {  this.encoding = encoding;  return this;  }  public Builder attachments(List<Attachment> attachments) {  this.attachments = attachments;  return this;  }  public SendEmailParams build() {  return new SendEmailParams(this);  }  }  }  （2） 特点：  各个具体的建造者相互独立，有利于系统的扩展；  调用者不必知道产品内部构建的细节，有利于减少细节风险；  如果产品的内部变化复杂，会增加很多的建造者类。  4> 行为型模式--责任链模式（对请求的发送者和接收者进行解耦）  （1） 实例：  public class Request {}  public class Response {}  public interface Filter {    void doFilter(Request request, Response response, Chain chain);  }  /\*\*  \* 驱动责任链传递  \*/  public interface Chain {    void handle(Request request, Response response);  }  /\*\*  \* 责任链实例  \*  \* @author CQJames 2020-07-13  \*/  public class FilterChain implements Chain {    private Queue<Filter> filterQueue = new LinkedList<>();    public FilterChain addFilter(Filter filter) {    filterQueue.offer(filter);  return this;  }    @Override  public void handle(Request request, Response response) {    if (filterQueue.isEmpty()) {  return;  }  Filter filter = filterQueue.poll();  filter.doFilter(request, response, this);  }  }  public class OneFilter implements Filter {    @Override  public void doFilter(Request request, Response response, Chain chain) {    chain.handle(request, response);  }  }  （2） 特点：  降低模块之间的耦合度；  不能保证请求一定被接收。  6. 记录项目及小组每个人工作的进度、里程碑、工作量的跟踪图或表，将其保存到每个小组选定的协作开发平台上，每周更新。 | | |
| 结论分析与体会：  本周的实验重点在于结合书本第六章的内容，了解设计方法、设计原则，同时需要熟悉面向对象的设计。  设计方法主要有自顶向下、自下向上、自外到内的设计方法。基于模块化设计的基础上，强调模块设计的先后顺序，或者是先研究最难最不明确的部分，通过这个部分引出的问题调整整个设计以适应该部分，以此发散开来直到系统完成。  设计原则是指把系统功能和行为分解成模块的指导方针。它从两种角度明确了我们应该使用的标准：系统分解，以及确定在模块中将需要提供哪些信息，分为模块化、接口、信息隐藏、增量式开发、抽象和通用性。 | | |
| 就实验过程中遇到的问题及解决处理方法，自拟1－3道问答题：  Q：简要介绍一下设计原则的六个领域？  A：  模块化：也成为关注点分离，将系统中不相关的部分进行分离，以便于各部分能够独立研究。用两个概念度量模块的独立程度：耦合度（分为内容耦合、公共耦合、控制耦合、标记耦合、数据耦合和非耦合）和内聚度（分为巧合内聚、逻辑内聚、时态内聚、通信内聚、功能内聚、信息内聚）。  接口：为系统其余部分定义该软件单元提供的服务，以及如何获取这些服务。  信息隐藏：隐藏单元的设计决策，允许在接口和接口规格说明的帮助下描述各个软件单元。信息隐藏使软件单元具有低耦合度，使得每个软件单元易于理解和维护。  增量式开发：通过单元之间的依赖关系来设计出一个增量式设计开发进度表。指定单元之间的使用关系，为各个软件单元之间建立关联，形成使用图。扇入代表某个软件单元的软件单元数量，扇出代表某个软件单元使用其他软件单元的数量。我们的目标就是创建一个高扇入、低扇出的软件单元。  抽象：是一种忽略一些细节来关注其他细节的模型或表示。  通用性：在开发软件单元时，使它尽可能地能够成为通用的软件，来加强它在将来某个系统中能够被使用的可能性。 | | |