第7讲
OO程序的分析与设计原则

OO2018课程组

北京航空航天大学计算机学院

提纲

- 面向对象之“思想”什么
- 面向对象之“分析思想”
- 实现之前的设计
- 实现之后的设计检查
- 作业

面向对象之“思想”什么

- 面向对象思想(object-oriented thinking)是经常见到的词汇
- 什么是面向对象思想
  - 一种思维方式，以对象为视角的思维方式
  - 有哪些对象？
  - 对象做什么？
  - 对象之间有什么连接？
- 在不同阶段的面向对象思想
  - 分析阶段：理解和识别需求中的“对象”
  - 设计阶段：构造“对象”来实现需求
  - 实现阶段：使用程序语言来实现“对象”
  - 测试阶段：逐个检查“对象”的功能和性能，然后对“对象集成”进行测试

面向对象之“分析思想”

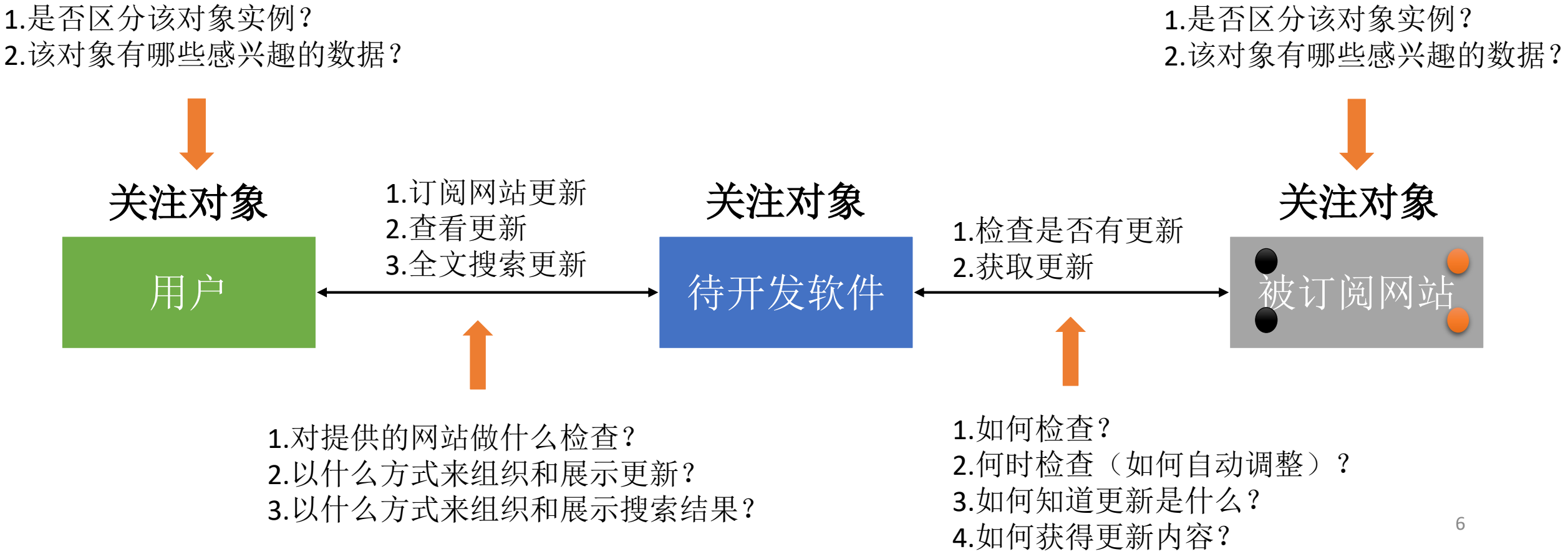
- 对象化思维来理解软件需求
  - 识别类-----数据、控制、设备...
  - 识别类的职责-----管理数据、实施控制/控制策略、输入输出处理
  - 基于类来分析软件功能-----多个类协同的方式来阐述功能
- 需求一般都是从用户视角来规定软件的功能和性能
  - 输入、输出
  - 平台
  - 数据
- 我们目前的作业要求在层次上相当于软件需求
  - 实际上比一般的软件需求要细致，介于软件需求和设计之间

面向对象之“分析思想”

- 例子
  - 新闻类、博客类网站每天会以不确定的频率发布新的消息
  - 用户难以知道什么时候有信息更新
  - 不能要求用户使用浏览器刷新网站页面来获得信息更新
  - 开发一个网站内容更新订阅系统，功能要求如下：
    - 能够根据用户需要订阅一个网站的内容更新
    - 能够自动获得网站内容的更新，包括主题、日期和信息片段(snippet)
    - *能够自适应网站内容更新的频度*
    - 能够对更新的主题、日期、信息片段进行有效管理
    - 提供对更新的全文搜索
    - 能够把来自不同网站的相同更新进行合并

面向对象之“分析思想”

- 1.交互关系分析：待开发的**软件**与**用户**、**外部环境**有哪些交互？

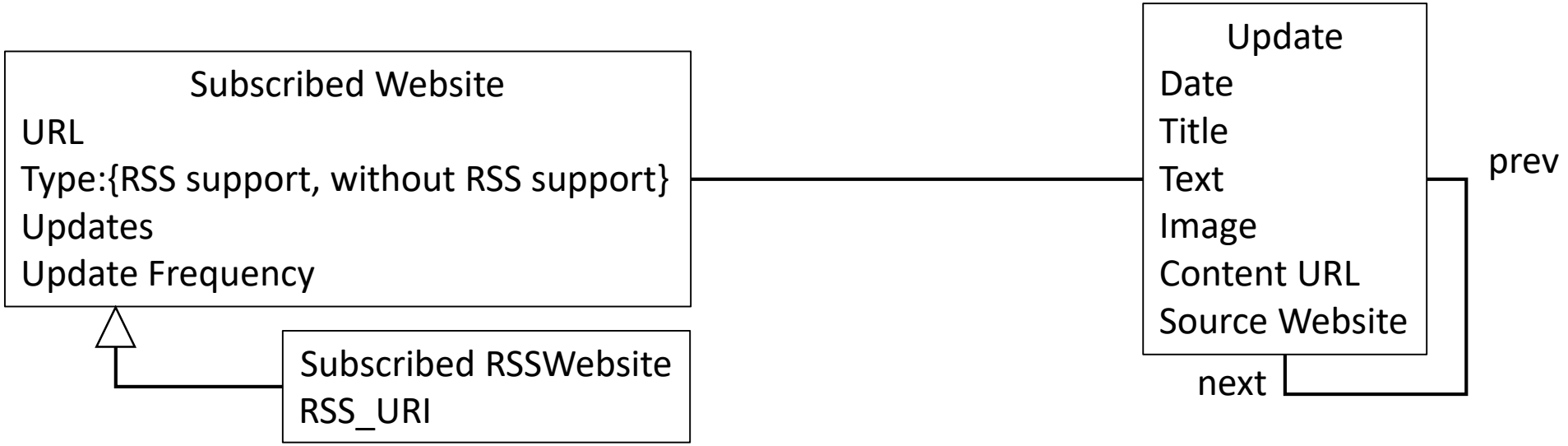


面向对象之“分析思想”

- 通过这种对象分析，确定了待开发软件的边界、对象之间的交互关系、对象交互的数据与时间特征
  - 用户交互的**数据特征**：提供待订阅网站、提供更新搜索词
  - 用户交互的**时间特征**：交互频度？交互持续时间？是否有多用户并发？
  - 被订阅网站交互的数据特征：网页内容(html)、更新列表文件(xml)、网页层次结构(html)
  - 被订阅网站交互的时间特征：更新频度、通讯速度
- 在对象分析过程中不仅理解了软件需求，更是细化了需求，并识别出潜在的问题
  - 如何找到网页中有更新？
    - 有更新列表文件的情况：按照更新列表来识别和提取更新
    - 无更新列表文件的情况：按照给定的页面来识别和提取更新

面向对象之“分析思想”

- 2.进入“待开发系统”对象内部进行分析
  - 运行平台：单机/Web服务器？
  - 管理哪些数据：被订阅网站、网站内容更新
  - 对数据的管理手段：维护订阅网站列表、管理更新、管理订阅网站与更新之间的关系、维护订阅网站的更新检查频度、检查不同订阅网站的更新是否相同、检查订阅网站是否有更新、提取订阅网站的更新

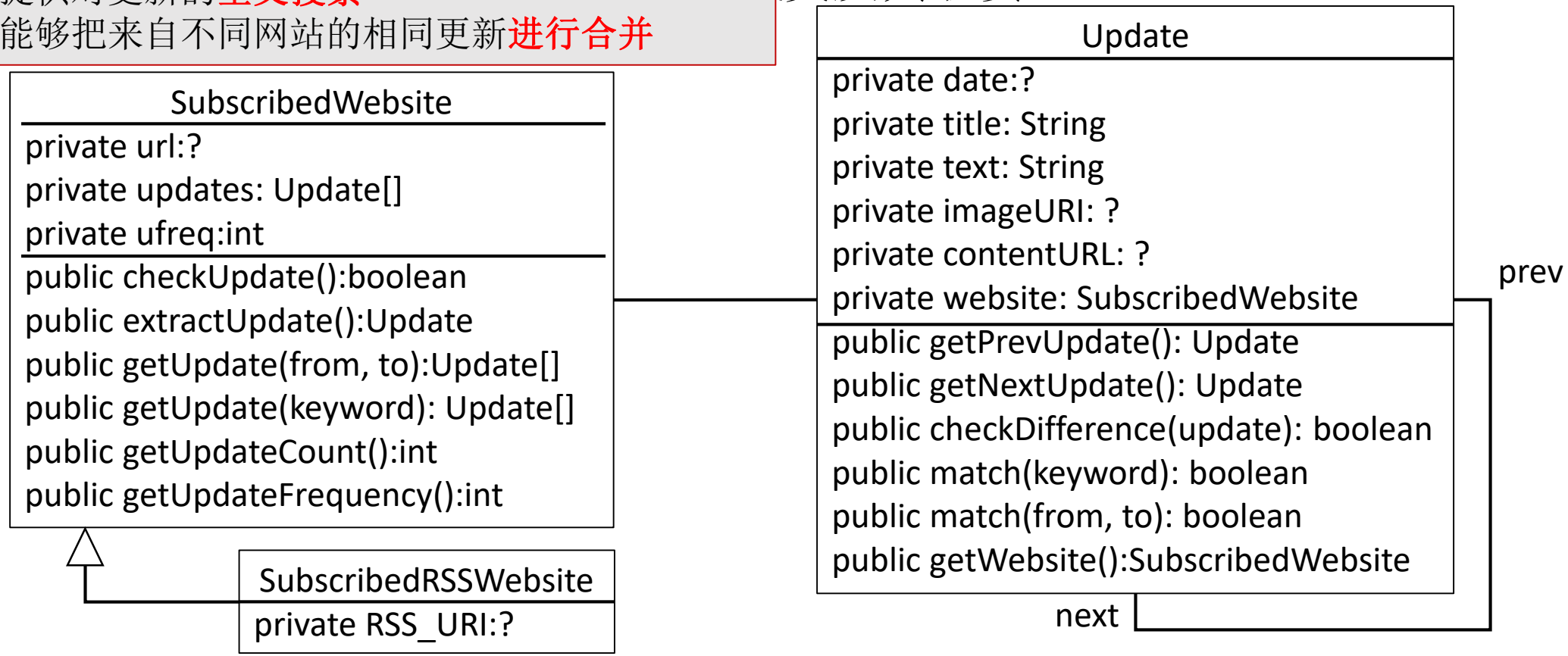


面向对象之“分析思想”

检查是不是所有的需求都能够得到满足(如何满足是设计问题)

- 能够根据用户需要**订阅一个网站内容更新**
  能够**自适应**网站内容更新的频度
  能够**获得网站内容的更新**
  能够**管理更新**的主题、日期、信息片段(snippet)
  提供对更新的**全文搜索**
  能够把来自不同网站的相同更新**进行合并**

进行分析
类及其职责



实现之前的设计

- 识别并发行为
- 增加额外的数据管理类
- 更好的刻画数据中的结构
- 简化类方法的职责
- 设计类之间的协同
- 空间与时间的平衡

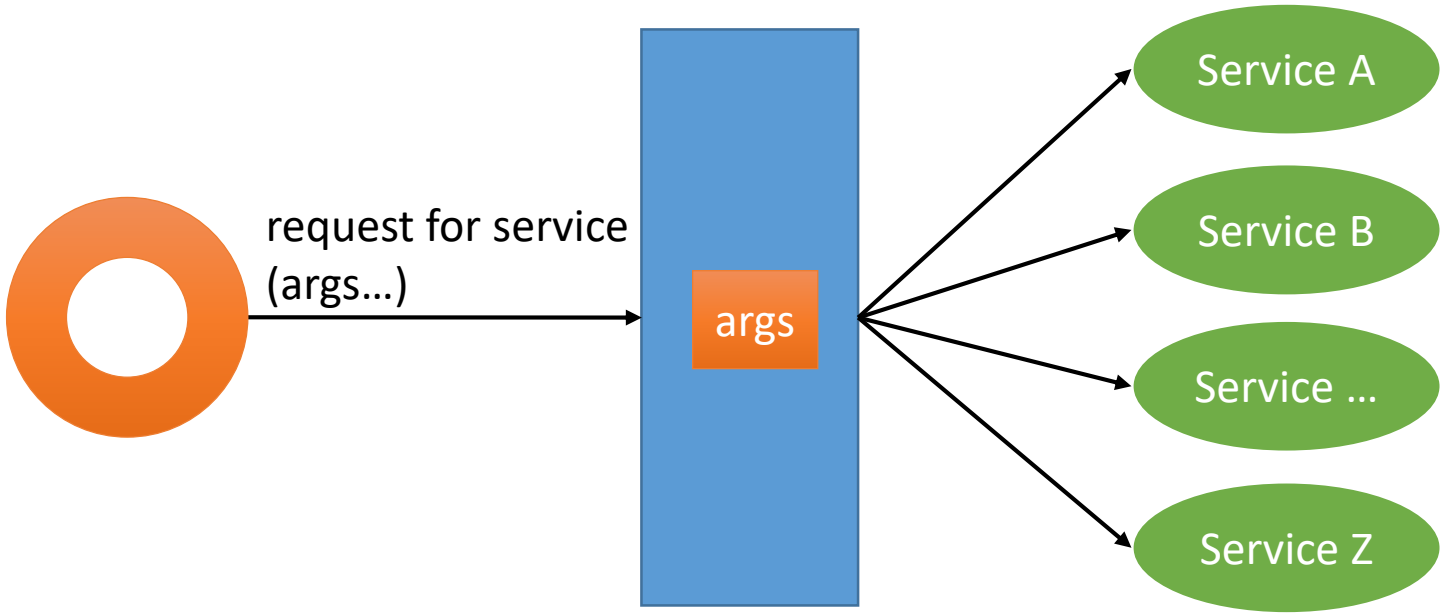
实现之前的设计

多线程与多种线程

- 识别并发行为
  - 待实现软件与外部多个对象进行相对独立的交互
    - 按照交互特征进行分类，一个类别对应一个线程设计
    - 订阅网站数量可以很多，一般相互独立
      - 有RSS支持的网站和无RSS支持的网页
      - 对应两类不同的线程设计：RSSWebSiteChecker、WebPageChecker
  - 待实现软件要处理的数据有显著的重复模式，且处理相对独立
    - Web系统的日志处理：日志记录着用户访问Web系统的行为，具有典型的模式；处理目标是提取用户行为、出现的问题、系统响应时间等，相对独立
    - 扫描一个规模较大的字符串数组看是否出现某个关键词：重复模式是不断在一个局部的数组中查找关键词的出现情况
    - 计算两个大矩阵的乘积：重复模式即为一个矩阵的行向量乘以另一矩阵的列向量

实现之前的设计

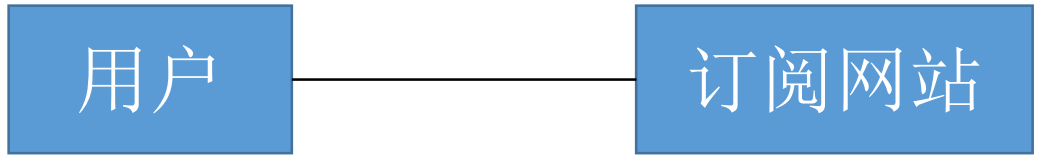
- 识别并发行为
  - 设想一个软件持续监听某网络端口以获得一种特定请求“request for service”，根据服务参数的不同，该软件需要进行不同的处理。



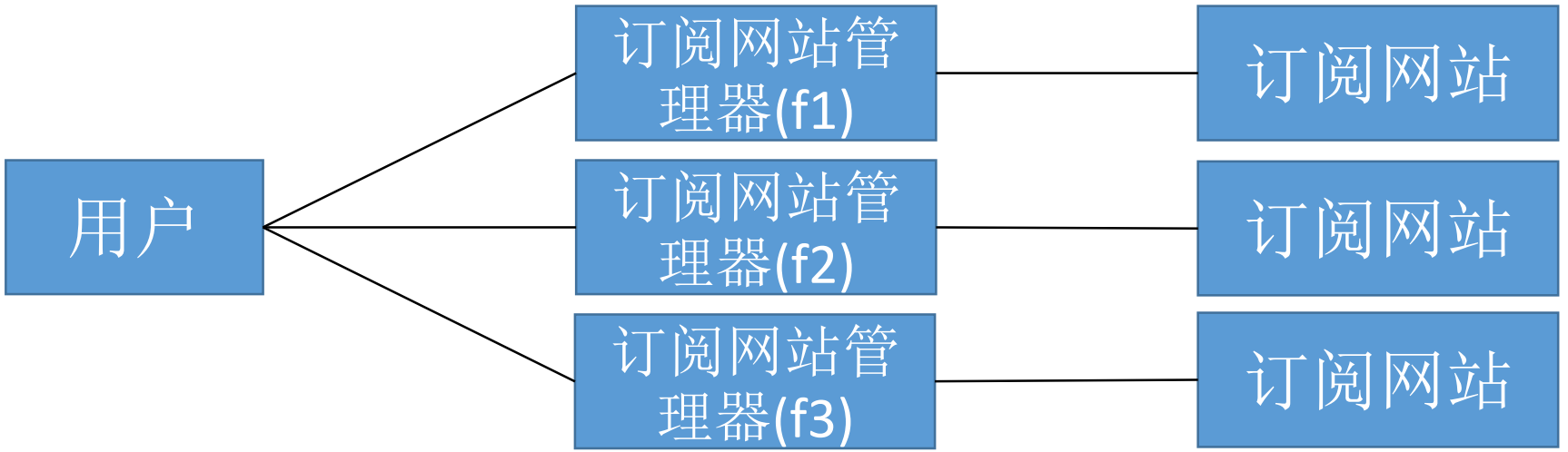
需要几类线程来支撑这个软件？

实现之前的设计

- 增加额外的数据管理类
  - 数据管理并不简单是增加一个数组或容器的问题
  - 网站内容更新订阅系统中，如何管理用户订阅的网站？



如何对订阅网站进行内容更新检查？
如果是多用户怎么办？



实现之前的设计

- 增加额外的数据管理类
  - 需要对被管理对象的特征和行为进行分类，按照类别进行管理==》简化相应的管理机制和行为逻辑
  - 出租车呼叫系统如何管理出租车和乘客？
    - 出租车：位置、信用、公司
    - 乘客：VIP、普通会员和普通乘客

实现之前的设计

- 更好的刻画数据中的结构
  - 很多数据都可以使用字符串来表示，但是字符串并不一定都能描述数据固有的逻辑结构
    - 电话号码
    - URL地址
  - 如果软件对相应数据的处理不涉及数据的逻辑结构
    - 字符串可以满足功能要求，但是不利于将来扩展和分析
  - 如果软件对相应数据的处理涉及数据的逻辑结构
    - 字符串不能满足功能要求
  - 结构化～非结构化

实现之前的设计

- 简化类方法的职责
  - 尽量保持每个方法只做一件事情
  - 不同方法不可避免存在交叉的行为，或者一个方法正好可以做两件事情
  - SubscribedWebsite中谁负责更新网站检查频率？
    - 方案A：单独增加一个方法updateFrequency(int freq)
    - 方案B：在checkUpdate中来更新频率
    - 方案C：在extractUpdate中来更新频率

SubscribedWebsite
private url:? private updates: Update[] private ufreq:int
public checkUpdate():boolean public extractUpdate():Update public getUpdate(from, to):Update[] public getUpdate(keywork): Update[] public getUpdateCount():int public getUpdateFrequency():int

实现之前的设计

- 简化类方法的职责
  - 建议采用方案B：在checkUpdate中来更新频率
    - 首先SubscribedWebsite的管理已经按照其更新频率进行了分类
    - checkUpdate的执行具有简单的周期性
    - 如果连续*[三次]*发现未更新，则调整更新频率，同时把相应对象调整到别的管理类别中
    - 即简化了方法职责，又保证了效率，同时减少了反复检查导致的CPU浪费
    - 如果连续三次发现都已经更新了呢？
  - 从方法职责单一性角度，把频率计算和更新单独设计成一个私有方法，被checkUpdate调用

Gmail大致就是按照这个思路来动态调整对POP3邮箱新邮件的检查

SubscribedWebsite
private url:? private updates: Update[] private lastNhits: int private ufreq:int
public checkUpdate():boolean public extractUpdate():Update public getUpdate(from, to):Update[] public getUpdate(keywork): Update[] public getUpdateCount():int public getUpdateFrequency():int private calcFrequency():int

实现之前的设计

- 设计类之间的协同
  - 从上面所说的多个角度会增加新的类、方法和属性
  - 这个过程就是设计
  - 仅仅使用这些单视角的方法进行设计会导致整体性的丧失
    - 做事情多的类倾向于做更多的事情
    - 管理数据多的类倾向于管理更多的数据
  - 需要从协同角度开展整体性设计
    - 在已有类的基础上，针对软件功能(和一些核心类的方法)来设计协同
      - Sit-together to work
      - Peer-to-peer
      - Client/server

实现之前的设计

- 空间与时间的平衡
  - 完成需求功能设计之后，其实还可以做的更好
  - 网站更新订阅系统
    - 网站不仅内容会发生变化，结构也可能会发生变化
    - 非RSS网站的内容变化建立在网页快照的对比基础之上
    - 这会带来什么问题？
  - 通过缓存一些历史状态，可以加速处理，提高程序执行效率
    - 几乎所有的服务器设计都会使用cache机制

SubscribedWebsite
private url:? private updates: Update[] private lastNhits: int private ufreq:int **private cachedSiteMap: Map**
public checkUpdate():boolean public extractUpdate():Update public getUpdate(from, to):Update[] public getUpdate(keywork): Update[] public getUpdateCount():int public getUpdateFrequency():int private calcFrequency():int

SubscribedRSSWebsite
private RSS_URI:? **private cachedSitePage: String**

实现之后的设计检查

- 经典的5个设计原则检查(SOLID)
  - SRP、OCP、LSP、ISP、DIP
- 我们还需要强调的几个设计原则检查

设计模式：用以解决特定（具有普遍性）问题的一种方案，如对象构造工厂、职责代理等

体系结构：软件模块被组织/集成为系统的结构，如MVC，Pipeline

设计原则：关于设计的整体要求和约束，通过满足设计原则来获得好的设计质量

SOLID之SRP

- Single Responsibility Principle
  - 每个类或方法都只有一个明确的职责
  - 类职责：使用多个方法，从多个方面来综合维护对象所管理的数据
  - 方法职责：从某个特定方面来维护对象的状态（更新、查询）

```
public class Elevator{
    //fields such as floor, status, …
    public void move(int dest_floor){
        //让电梯运动到目标楼层
    }

    public void Scan4TakingRequest (Queue q)
    {
        //扫描请求队列来寻找可以捎带的请求
    }
}
```

类/方法职责多，就意味着逻辑难以封闭，容易受到外部因素变化而变化，导致类/方法不稳定。

SOLID之OCP

Open/closed principle. In object-oriented programming, the **open/closed principle** states "software entities (classes, modules, functions, etc.) should be **open** for extension, but **closed** for modification"; that is, such an entity can allow its behaviour to be extended without modifying its source code.

Open/closed principle - Wikipedia
https://en.wikipedia.org/wiki/Open/closed_principle

Bertrand Meyer

- Open Close Principle
  - 无需修改已有实现(close)，而是通过扩展来增加新功能(open)
- 当扩展电梯系统支持ALS调度时
  - 改写Scheduler
  - 扩展Scheduler
- Employee类的display()方法
  - 普通员工：可以显示员工的基本信息和状态信息
  - 部门经理：把部门的其他所有员工也按照某种方式显示出来
  - 假设先实现了普通员工相应的功能，再来实现部门经理功能，怎么办？

SOLID之LSP

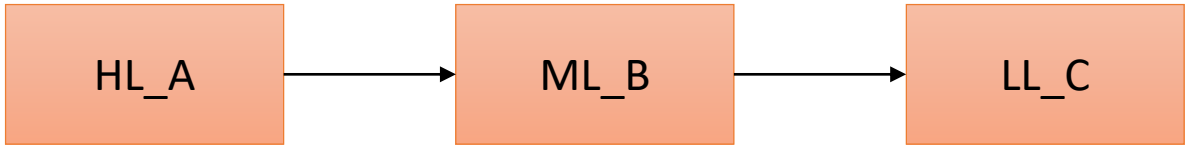
- Liskov Substitution Principle
  - 任何父类出现的地方都可以使用子类来代替，并不会导致使用相应类的程序出现错误。
    - BaseClass b = new BaseClass(…) → BaseClass b = new DerivedClass(…)
  - 子类虽然继承了父类的属性和方法，但往往也会增加一些属性和方法，可能会破坏父类的相关约束
  - 例：Queue和SortedQueue类
    - Queue类提供了一个getLastInElement()方法，即返回最近一次入队列的元素，其实现是返回队列尾部的元素
    - SortedQueue类则对队列中的元素进行排序，每次有新元素加入队列时，按照元素之间的大小关系插入到特定的位置
    - 此时调用SortedQueue的getLastInElement会怎么样？如何解决这个问题？

SOLID之ISP

- Interface Segregation Principle
  - 当一个类实现一个接口类的时候，必须要实现接口类中的所有接口，否则就是抽象类(abstract class)，不能实例化出对象
  - 软件设计经常需要设计接口类，来规范一些行为。避免往一个接口类中放过多的接口
  - 例：Payment是一个接口类，用来规范电子商务中的付款方式
    - 信用卡付款: 一组接口
    - 储蓄卡付款: 一组接口
    - 支付宝付款: 一组接口
  - 假设Payment同时把这三类付款方式都纳入作为接口
    - 你开设了一个店铺，要纳入一个平台必须要使用Payment接口
    - 每个商品都要实现这三个接口
    - 有什么问题？如何解决？

SOLID之DIP

DIP: Dependency Inversion Principle



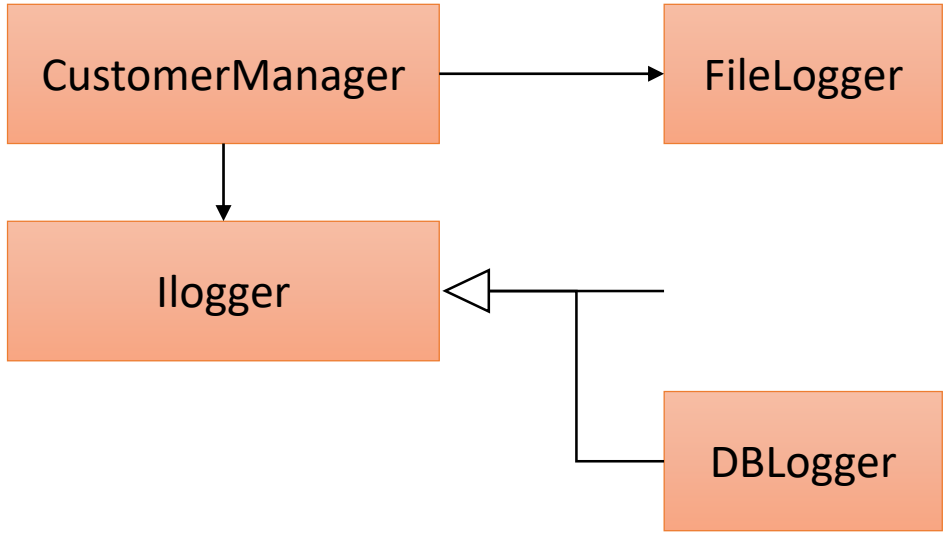

```
public class CustomerManager
{
     …field attributes here…
     public void Insert(Customer c)
     {
          try{
              //Insert logic
          }catch (Exception e){
              FileLogger f = new FileLogger();
              f.LogError(e);
          }
     }
}
public class FileLogger
{
    public void LogError(Exception e){//Log Error in a physical file }
}
```

CustomerManager依赖于Filelogger，即把异常信息记录到具体存储文件中

在数据库存储环境中想要重用CustomerManager类，怎么办？

SOLID之DIP

A. High-level modules should not depend on low-level modules. Both should depend on **abstractions**.
B. Abstractions should not depend on details. Details should depend on abstractions.




```
public class CustomerManager
{
    …field attributes here…
    public void Insert(Customer c)
    {
        try{
            //Insert logic
        }catch (Exception e){
            FileLogger f = new FileLogger();
            try{
                f.LogError(e);
            }catch (Exception e){
                mylogger.LogError(e);
            }
        }
    }
}
public class FileLogger
{
    public void LogError(Exception e){//Log Error in a physical file }
}
```

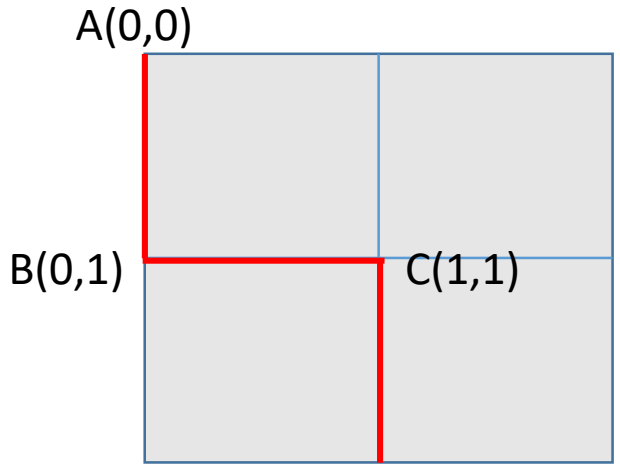
```
public class CustomerManager
{
    …field attributes here…
    private Ilogger mylogger;
    public CustomerManager(Ilogger logger){mylogger = logger;}
    public void Insert(Customer c)
    {
        //Insert logic
    }
}
interface Ilogger
{
    public void LogError(Exception e);
}
public class FileLogger implements Ilogger
{
    public void LogError(Exception e)
    {//Log Error in a physical file }
}

public class DBLogger implements Ilogger
{
    public void LogError(Exception e)
    {//Log Error in a DB }
}
```

其他重要的设计原则

- 层次化抽象原则，按照问题域逻辑关系来识别类；
- 责任均衡分配原则，避免出现God类和Idiot类；
- 局部化原则，类之间不要冗余存储相同的数据，方法之间不能够出现控制耦合；
- 完整性原则，一个类需要提供针对相应数据进行处理的完整方法集。完整是个相对概念，一般来说是相对于问题域需求。
- 重用原则（共性抽取原则），把不同类之间具有的共性数据或处理抽象成继承关系，避免冗余；
- 显式表达原则，显式表达所有想要表达的数据或逻辑，不使用数组存储位置或者常量来隐含表示某个特定状态或数据；
- 信任原则，一个方法被调用时，调用者需要检查和确保方法的基本要求能够被满足，获得调用结果后需要按照约定的多种情况分别进行处理；
- 懂我原则，所有类、对象、变量、方法等的命名做到“顾名思义”。

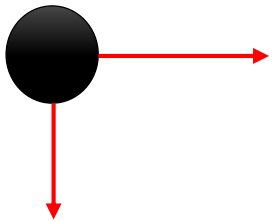
作业



- 模拟的出租车呼叫应答系统
  - 使用网格区域来模拟城市地图。所有的道路要么是水平方向，要么是垂直方向，如果两个点之间有道路，则这两个点之间存在一条连接。
  - 乘客在任意一点C发出呼叫请求后，系统只把呼叫请求发送给在以C为中心的4×4区域里行驶的出租车。请求包括目的地坐标信息，如果目的地无效，则出租车拒绝响应。
  - 出租车行驶一个格子边的时间为100ms。每辆出租车的状态包括：停止运行、服务（表明在运行且车内有乘客）、等待服务（表明在运行且车内无乘客）。出租车连续运行20s后且处于等待服务状态时需要停止(休息)1s，然后再次运行。在停止和服务状态下不能响应乘客请求。
  - 处于等待服务状态的出租车主要收到请求就会抢单。系统以3s为抢单窗口，如果3s内无应答则视为无车响应，告知乘客无可用出租车。否则，系统为乘客从当前窗口中抢单的出租车中自动选择信用度最高的。
    - 信用：每抢单一次，信用度+1；每服务顾客一次，信用度+3。

作业

- 城市地图不通过命令行输入，而是通过文件
  - 文件格式：文本文件
  - 文件内容：80×80邻接矩阵A，aij表示第i行和第j列的交叉点，记录地图中(i,j)到相邻点的连接。为简化表示，aij可以是4个值：
    - 0：表示(i,j)到(i,j+1)和(i+1,j)均无连接
    - 1：表示(i,j)到(i,j+1)有连接，到 (i+1,j)无连接
    - 2：表示(i,j)到(i,j+1)无连接，到 (i+1,j)有连接
    - 3：表示(i,j)到(i,j+1)有连接，到 (i+1,j)有连接
  - 不能根据aij取值来判断(i,j)到(i-1,j)和(i,j-1)是否有连接
    - “向右向下”原则来构造这个矩阵
    - 确保地图上的所有点都是连通的
- 出租车限定为100个，起始位置随机分布



作业

- 要求采用本讲介绍的方法进行分析和设计
  - 要求使用多线程和线程安全设计。提供线程安全的乘客请求队列，供测试使用。请求队列容量不小于300个。
  - 要求提供手段来让测试线程访问所有处于等待服务状态的出租车对象。
  - 要求提供分析文档，阐述需求分析。
- 测试
  - 对需求分析文档的详细程度给出综合评价
    - 很好(+3)：使用了所介绍的方法，分析完整
    - 好(+1)：基本使用了所介绍的方法，分析欠完整
    - 一般(0)：其他。
  - 按照SOLID设计原则对实现进行检查，如果发现设计原则被违背，则记为一个incomplete，但同一个原则不重复扣分
  - 编写乘客线程，向请求队列发送请求来模拟乘客呼叫出租车，并通过访问相关出租车对象的状态自动判断程序处理是否正确。