让技术人员看得懂的流程

——面向对象设计全流程概述

谈到流程，大家都会想到熟悉的瀑布模型、螺旋模型、迭代开发、敏捷、RUP 等一堆软件工程相关的软件开发流程，但是请不要误会，本文的流程和这些管理流程完全不同，为了以示区分，我把瀑布模型、敏捷、RUP 等流程成为项目流程，也就是说这是给项目管理用的；而本文的流程是技术流程，是给技术人员（主要是设计人员）看的流程。

在开始讲解之前，看看如下问题你是否能够回答？

1、客户的需求是描述性的，例如“我们需要一个 POS 机”，而代码是一个一个具体的类和函数，那么如何从描述性的语言最后转化到具体的类和函数呢？

2、具体语言的特性，例如 Java 和 C++的 private、protected、public 这些属性是从哪里来的？什么时候设计的？

3、不管什么代码，最后都要运行在具体的平台上，如 Windows、Linux、UNIX 等，那么这些平台相关的进程、线程什么时候设计、如何设计？（不要说你所有的产品都是单线程或者单进程哈：-P）

4、如果是稍微大一点的产品，需要运行在多台机器上，那么如何确定需要多少机器？如何分工？

。。。

怎么样？以上这些问题是否似曾相识，或者自己是否考虑过？

如果你的第一反应是去翻开《软件工程》、RUP、敏捷开发等相关著作，那么我可以很肯定的告诉你你会失望的，就像前面提到的，这些东西不是管理流程的事情，而是技术流程的事情。

如果你心有疑问，或者不敢肯定自己的答案，那么“不懂的人有福了”，因为我将通过几篇短的博文和一个实例来简明概要的讲述这个流程，概要的讲，主流程如下：

用例模型->领域模型->设计模型->实现模型->进程模型->部署模型

实例就用一个简单的 POS 机系统来讲解，欲知详情如何，且听下回分解！

——用例模型

一般的管理流程都将软件项目划分为“需求->分析->设计->实现->维护”，对应的技术流程

中首先也肯定是要将需求明确，而“用例模型”就是用于获得和分析需求的。

简单来说，用例模型就是要将客户的需求写下来。“需求”不是很好理解，更加通俗的讲法是“故事(story)”。我觉得“故事”这个词非常好，非常形象，非常容易理解！我们看看为什么“故事”更加容易适合说明客户要求：

（1）故事中有很多角色，而需求中也有很多角色；

（2）故事有具体的经过，有开始、进行、结束；而需求也是一个完整的流程，有输入、处理、输出。

（3）故事必须有意义，而需求对客户来说，也是必须有意义的；

具体的用例格式等这里就不详细描述了，大家上网搜索一下就知道了，如何更好的把握需求可以参考我的博文《需求分析的故事——如何练就需求分析的火眼金睛？》，下面我把用例模型阶段容易犯的错误重点说明一下。

1）“需求”和“功能”混淆

很多朋友在分析需求的时候，将“功能”和“需求”混淆起来，导致了需求的粒度不合理，或者把握不住重点的需求，我总结出一个很简单的区分办法：“需求是对客户有价值的东西，功能是为了实现需求而作的东西”。

以 POS 机为例：对于客户来说，“买单”是一个有价值的东西，因为客户买完单就可以把商品拿走了；而买单过程中的“读商品条形码”、“计算商品总额”、“打印购物清单”等都是功能，因为它们当中任何一个独立的功能对客户没有价值（你不会跑到超市“读商品条形码”玩吧:-P），只是为了实现“买单”而做的。

2）在用例模型阶段进行系统分解

除了容易将“功能”和“需求”混淆外，用例模型阶段还有一个常见的错误就是在用例模型阶段对系统进行分解。

以 POS 机为例，有的人将需求描写为：扫描机扫描商品条形码信息，然后将信息发给库存系统，库存系统返回详细的商品信息给交易系统……在用例模型阶段这样做是不对的，因为这样转移了我们的注意力：从关注用户需求转到了系统分析了。

记住：用例模型关注的是用户需求，不需要进行系统分解，把系统当做一个黑盒看待就可以了。

下面我们给出 POS 机一个简单的用例片段，有兴趣的可以自己写一个完整的（这个完整的可不简单哦，至少可以写 3 页）：

1）顾客携带商品到收银台；

2）收银员扫描商品条形码；

3）系统根据条形码获取并显示商品信息；

4）收银员重复 2~3 步，直到所有商品扫描完毕；

5）系统计算商品总额；

。。。。。。。。。。。。。。。。。。。

n）系统打出商品清单，完成交易。

==========2010.03.20 补充==========

（经过 garrodran99 的提醒，补充 SSD（系统顺序图）的内容，在此感谢 garrodran99）

SSD 即系统顺序图，目的就是站在系统的角度，以 UML 的顺序图来描述系统与“外部实体”（包括人、其它系统、输入输出设备）的交互过程，简单来说就是“图形化的用例”。

那为什么有了文字化的用例文档，还要图形化的 SSD 呢？其实很简单：字不如表，表不如图，因为图可以一目了然的看出交互过程。

可能有的朋友又要问了：既然图那么好，还要文字描述的用例做什么呢？有两个原因：第一个是客户不会用 UML 给你描述需求，客户只会用自然语言描述需求；第二个就是图形的优点是简洁、一目了然，但图形的缺点就是不能承载太多信息，详细和丰富的信息还是要文字来承载。

用一句话总结就是：交互用图形，说明用文字！

那么，是否一个用例就要写一个 SSD 呢？基本上不需要，因为 SSD 重点在描述系统的交互过程，如果用例本身没有什么交互、或者交互很少或者很简单，就不需要 SSD。只有主要的、复杂的用例才需要 SSD。

——领域模型

按照一般的项目管理过程，“需求”之后是“分析”，那么在分析阶段对应的技术流程又是哪个？如何将需求阶段和分析阶段联系起来呢？答案就是“领域模型”

什么是“领域模型”呢？只要抓住“领域（Domain）”二字就可以理解，也就是说领域模型是帮助我们理解相关领域知识的模型。

进一步来问：为什么需要领域模型？前面不是有“用例模型”吗，看起来用例模型好像就是描述相关领域知识的，是否完成用例模型就可以进行设计了？

如果你曾经写过用例文档，或者仔细看过用例文档，那么你肯定知道“用例”本身和“面向对象”、“类”这些都没有关系，用例就是纯自然的语言（比如英语、汉语）写的，因为用例实际上由客户口述给我们、然后由我们形成文档化的用例文档，客户才不管我们是什么面向对象还是面向过程还是阿猫阿狗的。

因此，单纯从用例来看，是没有类的概念的，无法完成从自然语言到面向对象语言的转换。但是，既然我们已经完成了用例模型，那么就必须基于用例模型进行分析(否则用例模型岂不是白做了)，而“领域模型”就是自然语言向面向对象语言的一座桥梁。

让我们来看看“领域模型”阶段我们要做什么、该怎么做。其实很简单，简单概括就是“找名词”，分为四个步骤：（1）找出用例模型中的名词，（2）然后识别这些名词本身的相关信息，（3）以及名词之间的相互关联关系，（4）用 UML 画出领域模型。

我们来看在“用例模型”章节中的 POST 用例如何得出领域模型。

第一步：找出用例模型中的名词

原有用例如下，用蓝色加黑标出名词（重复的就不标了）：

1）顾客携带商品到收银台；

2）收银员扫描商品条形码；

3）系统根据条形码获取并显示商品信息；

4）收银员重复 2~3 步，直到所有商品扫描完毕；

5）系统计算商品总额；

。。。。。。。。。。。。。。。。。。。。

n）系统打出商品清单，完成交易。

这个用例中的名词有“顾客”、“商品”、“收银台”、“收银员”、“商品条形码”、“系统”、“商品信息”、“商品总额”、“商品清单”、“交易”。稍加整理：

1）“顾客”、“收银员”是系统的外部对象，不需要我们进行设计，但这些对象要和系统进行

交互；

2）“商品”、“商品条形码”、“商品信息”、“商品总额”、“商品清单”、“交易”是领域对象，但“商品条形码”、“商品信息”可以算作“商品”的属性、“商品总额”可以算作“交易”的属性，最后从这个用例总结出来的领域对象有“商品”、“商品清单”、“交易”三个。

第二步：识别这些名词本身的相关信息

一个对象的属性可能分布在多个用例中，因此可以通过迭代不断的完善一个对象的属性，大家可以看到，我们在第一步中的样例就已经分析了一部分了：“商品条形码”、“商品信息”可以算作“商品”的属性。

对象除了属性外，还有一些约束或者限制，这些在用例中可能有，也可能没有，这就需要分析人员来发现了。比如说交易金额必须大于 0.1 元小于 99999 元这种约束，用例中不一定会体现，可能需要分析人员向客户咨询。

第三步：识别对象间的关系

面向对象设计就是依靠对象间的互相协作来配合完成相应的功能，因此识别出对象和对象本身的属性外，还要识别对象间的关系，例如 1 对多、1 对 1、依赖等，详细的各种关系可以参考 UML 的标准定义。

我们以第一步识别的三个对象为例：“商品清单”包含多个“商品”、一次“交易”对应一个“商品清单”、一个“商品”只能属于一个“交易”等。

第四步：画出领域模型 UML 图

UML 这里就不啰嗦了，我们结合前三步的分析，画出这个样例的 UML 领域模型图：

（由于 CSDN 关闭了图片上传功能，因此无法贴出，大家可以根据前三步自己画一个）

大家可以看到，经过“领域模型”的分析后，已经完成了自然语言到面向对象语言的初步转化了，领域对象已经识别出来，面向对象已经初具雏形。

需要提醒的是：领域模型过程中识别出来的对象和具体的语言无关，也没有方法。换句话说，public、private、函数这些面向对象的属性在领域模型阶段不需要分析出来。

——设计模型

完成了“领域模型”阶段后，面向对象已经初具雏形，我们已经看到了那熟悉的“对象”了，例如“商品”、“交易”、“商品清单”等，看起来已经进入了面向对象的世界了，你是否已经

摩拳擦掌，跃跃欲试，准备开始编码了呢？

且慢，“领域模型”只是万里长征的第一步，通过领域模型分析得到的类还不能指导编码，还需要经过“设计模型”这个阶段的处理，才能基本上指导编码。

前面我们提过，领域模型的对象是没有方法的，但最终的实现肯定是有方法的，因此设计模型的第一个任务就是“为对象添加方法”。

那么是否给领域模型中的对象添加完方法就算是完成了设计模型呢？没有这么简单，给领域模型中的对象添加方法只是设计模型中最简单的一部分工作，设计模型阶段第二个任务是“围绕领域对象设计出非领域对象”。

这句话看起来比较难拗口，为什么要设计出非领域对象呢？道理很简单：领域模型中的对象是静态的，要让这些静态的对象动起来，才能最终完成客户需求。因此必须添加非领域对象，让这些非领域对象来完成让领域对象动起来的事情。

例如：“商品”本身是一个领域对象，但是这个对象是谁创建、谁使用、谁管理呢？领域模型中并没有相关的对象来完成这些职责，因此需要我们设计额外的对象来完成这些职责。

经过前两步之后，看起来设计模型的对象都已经出来了，但是我们如何知道设计得好还是不好，以什么标准来判断我们的设计是否正确呢？相信基础扎实的朋友们已经想到了，这就是万人期待、万众瞩目的，大家都耳熟能详的一个东东：设计模式。设计模型第三个任务就是“应用设计模式、设计原则”。

通过应用设计模式、设计原则，又会添加一批新的对象，接口、父子类、继承、依赖等面向对象的相关概念也逐步清晰，这样就为最终的编码打下了坚实的基础。

到这里为止，“设计模型”阶段的任务基本讲述完了，下面我们看一个简单的样例，还是以 POST 机为例：

在“领域模型”阶段我们已经分析出了“商品”、“商品清单”、“交易”三个领域对象，我们按照前面所讲的三个步骤一步一步来看“设计模型”阶段如何做（都以“商品”对象为例）。

第一步：“为对象添加方法”

商品对象的方法有：“获取名称”、“获取条形码”、“获取价格”等（有兴趣的朋友可以自己完善），这样对象的几个方法就出来了：getName（）、getBarCode（）、getPrice（）。

第二步：“围绕领域对象设计出非领域对象”

“商品”对象的生命周期包括：创建、修改、使用、销毁，这些任务对象本身是无法完成的，必须添加新的对象来完成，这里我们添加一个新的对象“商品管理”来完成这些任务。这样“商品管理”对象就出来了，而这个对象在用例模型和领域模型中都是没有的。

第三步：“应用设计模式、设计原则”

我们简单的应用 DIP 原则（可以参考我的 Blog《软件设计漫谈之三：30 分钟掌握面向对象类的设计原则》）就可以发现，“商品”本身应该作为一个接口，因为不同的商品之间是有很大差异，“商品”又可以分为“食品类商品”、“饮料类商品”、“服装类商品”等等。这样应用了设计原则后，在领域模型中作为对象的“商品”，在设计模型中不再是具体对象，而是接口，然后这个接口派生出“食品类商品”、“饮料类商品”、“服装类商品”等具体对象。

以上的步骤不是瀑布式的，而是迭代式的，例如：第三步识别出“饮料类商品”这个对象后，也需要为它添加方法、设计出相关的类。

——实现模型

经过前面的“用例模型”、“领域模型”、“设计模型”的讲解，面向对象分析设计都完了，面向对象已经基本成型，接下来就是要具体实现了，对应的就是“实现模型”。

“实现模型”是整个技术流程中大家接触最多的阶段，只要是做开发的，基本上都是先参与这个阶段的工作。顾名思义：实现模型就是使用具体的技术来实现设计，也就是通常意义上的编码。

但要注意的是，编码不等于敲键盘，之所以称为“实现模型”，因为这里还是有设计的，只不过这个设计和具体的实现技术有关。

例如：Interface 在 C++中没有，而 Java 中就有，具体编码的时候如果要实现设计图中的 interface，那么就只能分别如下实现：

1）C++：声明没有成员变量、所有成员函数都是纯虚函数的 Class；

2）Java：直接声明为 interface。

由于具体的实现技术差别很大，因此没有什么通用的方法，“实现模型”阶段需要大家积累具体的技术知识和经验。

——处理模型

看完“实现模型”，你是否长吁一声，准备拿起咖啡，惬意的喝上一杯？毕竟我们已经完成了从用例到编码的全过程了，确实是值得庆祝的一件事情，但“革命尚未成功、同志还需努

力”，现在还不是享受的时候，接下来我们需要进入“处理模型”阶段。

l “处理模型”阶段的任务

“处理模型”英文是“Process Model”，Process 在 IT 里面又叫“进程”，虽然和进程相关，但直接叫“进程模型”会误导大家，所以我叫它“处理模型”，也就是和处理相关的设计。我们来看看“处理模型”阶段的任务：

1）进程、线程设计；

2）子系统设计；

为什么需要“处理模型”呢？相信看到上面的任务后，聪明的你应该已经知道了原因：

1）随着系统规模增大，处理性能要求增加，必须采用多进程多线程处理方式；

2）随着系统规模增大，复杂度增加，加上需要考虑可扩展性、可测试性、可靠性等质量属性，必须采用“分而治之”的方式划分子系统（注意此时还不是架构设计，欲知详情，请关注下一篇博文）；

l 为什么现在才开始进行进程、线程、架构设计？

讲到这里，估计很多朋友都有疑惑了：按照一般的经验，都是最开始就要进行子系统设计、进程线程设计的，怎么你的这个流程到现在才开始进行进程、线程、子系统设计呢？

我们知道：进程、线程、子系统设计都必须有基础，而不是凭空创造或者想象出来的。那种所谓的先画一个圈表示系统、然后再在这个圈下面画几个圈表示子系统、子系统下面再画几个圈表示进程或者线程的自顶向下的设计方式就像“浮沙筑高台”，其实是完全行不通的，为什么呢？

因为这个时候划分子系统没有任何可靠的依据。架构设计、子系统设计、进程线程设计主要是为了解决性能、可靠性、可扩展性、可测试性、安全性等质量属性，而不是客户主要关注的功能属性。性能、可靠性、安全性可以从客户获得，但无法像功能属性那样一步一步的映射到代码（客户说要“每秒支持 10000 个交易”，然后你画了三个圈，说“这样就可以达到每秒 10000 个交易”，谁会相信呢？），而可扩展性、可测试性在客户需求阶段根本不会体现。所以我们必须等到“实现模型”完了之后再进行进程、线程、子系统设计、架构设计。

可能大家还有疑问：按照你这个说法，岂不是要等到系统全部编码完成后再来进行进程、线程、架构设计？

回答这个问题的关键词就是“迭代”，第一个迭代（一般都叫做 Demo）把最主要、最核心、最关键的需求按照“用例模型”->“领域模型”-> “设计模型”-> “实现模型”->“处理模型”走一遍流程，这样第一个迭代就可以把架构、子系统、进程、线程初步设计完毕，后续的迭代基本上只要走“用例模型”-> “设计模型”-> “实现模型”就可以了，即使有调

整也不会太大，因为第一个迭代式把最主要、最核心、最关键的需求给实现了。

l 具体如何操作？

我们来看如何基于“实现模型”进行进程、线程、架构设计：

1）第一步：将已有的对象进行分组；

分组的原则其实就是大家常见的“高内聚、低耦合”，把最相关的、联系最紧密的对象划分到同一组；

2）第二步：将多个组划分为进程、线程；

将对象组再分组划分到具体的进程和线程，分组的原则主要看性能要求（响应时间、吞吐量），性能数据可以基于已有的“实现模型”进行评估或者测试。

3）第三步：设计进程的同步、通信；

既然是多进程、多线程，就必须设计出进程间同步和通信方式。

4）第四步：将进程划分到不同的子系统；

结合根据“高内聚、低耦合”的原则、以及性能、可靠性、可扩展性、可测试性等质量属性要求，将进程划分到不同的子系统；划分子系统也为下一个阶段（卖个关子，先不说:-P）打下了基础。

5）第五步：设计子系统间的同步、通信

和第三步类似，划分为不同的子系统后，必须设计子系统间的同步和通信方式。

看起来步骤比较多，不过每个步骤其实都不难，简单点说就是“排列组合”，将对象排列组合成进程线程，将进程排列组合成子系统。

千言万语不如一个用例，我们还是继续前面的 POST 机样例来看看“处理模型”阶段如何操作吧。

经过“实现模型”阶段后，我们的 POST 机可能是这样实现的：“商品”、“交易”、“商品管理”、“商品清单”、“付款方式”、“店铺”、“收银机”、“VIP 会员”、“供货商”等对象了，接下来我们就要进行“处理模型”设计了：

1）第一步：将已有的对象进行分组；

1.1）“商品”“商品管理”都是商品相关的对象，因此划为第一组，命名为“商品处理”；

1.2）“交易”“商品清单”“付款方式”都是和交易相关的对象，因此划为第二组，命名为“交易处理”；

1.3）“店铺”、“收银机”都是系统静态信息，因此划为第三组，命名为“系统信息管理”；

1.4）“供货商”、“VIP 会员”都是第三方的管理信息，因此划为第四组，命名为“第三方管理”

2）第二步：将多个组划分为进程、线程；

初步评估，“商品处理”和“交易处理”的性能要求会很高（因为商品很多，交易也在不断进行），而“系统信息”是一个基本静态的信息，性能要求会很低，而“第三方管理”性能要求相对也不高，因此可以设计出 3 个进程：“商品处理”进程、“交易处理”进程、“信息管理”进程，其中“信息管理”进程负责 “系统信息管理”和“第三方管理” 两组对象

3）第三步：设计进程的同步、通信；

进程间同步和通信和具体的实现有关，可以参考相关资料，例如 Windows 和 Linux 的可以参考我的博文《多核时代：并行程序设计探讨（4）——Windows 和 Linux 对决(进程间通信)》《多核时代：并行程序设计探讨（5）——Windows 和 Linux 对决(进程间同步)》。

4）第四步：将不同的进程划分到不同的子系统；

第二步已经设计出三个进程了（实际情况会更多）：“商品处理”进程、“交易处理”进程、“信息管理”进程，因此可以划分为三个子系统：商品管理系统、交易系统、信息系统。其中“信息系统”负责“系统信息管理”和“第三方管理”。

注意：由于此样例比较简单，所以出现了进程和子系统一一对应的情况，实际工作中应该是一个子系统包含 1 至多个进程。

5）第五步：设计子系统间的同步、通信

和第三步类似，划分为不同的子系统后，必须设计子系统间的同步和通信方式。

由于子系统可以是进程、线程，也可以是独立运行的程序，因此子系统间通信方式也随着实现方式不同而不同。例如程序间通信可以采用共享文件、共享内存、Socket 等方式。

——部署模型（完结篇）

在上一篇博文“处理模型”中已经提到：在“处理模型”阶段划分为子系统后，为下一阶段打下了基础。当时卖了个关子没说具体是什么，本博就来揭开它的面纱，这就是：“部署模型”。

l “部署模型”阶段的任务

“部署模型”英文是“Deployment Model”，正好对应 UML 中的“Deployment Diagrams”，有的文章或者书籍也叫“物理模型”。我之所以没有用“物理模型”，是因为“物理模型”的概念容易误解大家认为这个阶段只需要关注物理设备，而“部署模型”相对更加全面。我们来看部署模型的任务：

1） 确定部署实体，即采用什么样的物理设备，例如 PC 机、服务器、小型机；

2） 确定部署方式，例如局域网部署，企业网部署，因特网部署；

3） 确定部署连接，即组网方案，设计如何将这些物理设备连接起来；

l 具体操作步骤

1）确定部署实体

在“处理模型”阶段我们已经将子系统划分出来了，但如果你的系统性能、可靠性等质量属性要求很高，那么就需要进一步考虑将子系统再“排列组合”，分布到不同的机器上去。

“排列组合”的原则还是和划分子系统的第四步一样：根据“高内聚、低耦合”的原则、以及性能、可靠性、可扩展性、可测试性等质量属性要求，将子系统划分到不同的机器上去。

简单的说就是“一台不够用两台，两台不够用三台；PC 不够用工作站，工作站不够用小型机”。那么是不是机器数量越多越好，质量越高越好呢？也不尽然，因为还需要考虑成本的因素，这就需要架构师、设计师进行权衡了（可以参阅我的博文：《软件设计漫谈之一：什么是软件设计？》）。

2）确定部署方式

确定好“部署实体”后，就需要考虑部署方式，因为不同的部署方式决定了需要采用不同的网络设备和组网方式。

常见的部署方式有：局域网部署、企业网部署、因特网部署，下面简单的介绍一下：

（1） 局域网部署：俗称 LAN，即机器都在一个局域网内部，通过 Hub、Lanswitch、网桥等连接，这是范围最小的一种部署方式；

（2）企业网部署：金融、电信、物流等稍微大的公司都会有企业内部网，企业网内部还会划分 VPN 等，通过路由器、交换机等进行连接，这是中等范围的部署方式；

（3）因特网部署：相信大家对这个都很熟悉，最常见的就是网站，如 Sina、CSDN 等，需要向宽带服务商如电信等申请因特网 IP 地址，这是范围最广的一种部署方式；

具体采用哪种方式呢？其实很简单：客户会告诉你的：）

当然客户不会直接告诉你说“我们要采用企业网部署”，客户会在需求中隐含描述出来，比如说“分公司将数据发给总公司”，这句话就隐含了要采用“企业网部署”或者“因特网部署”这种方式。

3）确定部署连接

确定好部署实体和部署方式后，就需要考虑如何将这些机器连接起来了，即常说的组网，这时就是 TCP/IP 大显身手的时候，我们看看具体如何应用 TCP/IP：

（1）确定 IP 地址：包括网段、子网掩码、每台机器的 IP 地址；

（2）确定连接方式：单连接、双连接、四连接，至于八连接，好像还没有见过：）

（3）确定连接设备：连接设备和部署方式有关，同时也和性能、可靠性有关。例如：局域网要求不高的可以用 hub，要求再高点就要低端的 Lanswitch 了；企业网和因特网就需要路由器了。

千言万语不如一个样例，让我们来看看 POS 系统在“部署阶段”如何操作。在“处理模型”阶段我们已经划分出 3 个子系统了：商品管理系统、交易系统、信息系统，我们基于这 3 个系统进行简要分析：

1）确定部署实体

假设要买 POS 系统的超市的物流和库存是集中管理，而交易是由各个分店分散管理，那么“商品管理系统”和“信息系统”可以部署在同一台小型机上面（根据性能和可靠性等要求的不同，可能是两台、三台甚至更多，样例中我们假设一台就够了）。

“交易系统”由于是分散的，服务器配置应该就够了，每个分店一台。

2）确定部署方式

从需求可以很容易看出：POS 系统最好采用企业网部署的方式，因为有分店、有总店，这些分店和总店是分散的。

3）确定部署连接

IP 地址的分配根据具体情况分配即可。根据部署方式可以推断出需要采用路由器类型的设备进行连接，而为了保证交易系统的可靠性（总不能网线一断，超市就关门不做生意了吧），需要双连接甚至是四连接。

========================连载完毕==================================

到这里这一系列就结束了，各位看官是否有所收获呢？

如果以下问题能够正确答复，恭喜你，说明你已经基本上掌握了面向对象设计全流程的处理：

1、客户的需求是描述性的，例如“我们需要一个 POS 机”，而代码是一个一个具体的类和函数，那么如何从描述性的语言最后转化到具体的类和函数呢？

2、具体语言的特性，例如 Java 和 C++的 private、protected、public 这些属性是从哪里来的？什么时候设计的？

3、不管什么代码，最后都要运行在具体的平台上，如 Windows、Linux、UNIX 等，那么这些平台相关的进程、线程什么时候设计、如何设计？（不要说你所有的产品都是单线程或者单进程哈：-P）

4、如果是稍微大一点的产品，需要运行在多台机器上，那么如何确定需要多少机器？如何分工？

。。