# 技术文章

## 说明

1. 文字语言随意化，口语风格，易理解，程序猿亲切的语言，必须融入专业知识。
2. 能过紧跟时事热点最好，其中融入专业名词。
3. 填入参考或者相关的文献资料
4. 有啥说啥，不必为了写文章而矫情
5. 鼓励学习，进步，快乐和幸福。
6. 评论中专业性或有用的则引入到正文中。

单独申请微博账号？

## 程序猿的吐槽一——序言

介绍

账号

说明（开始原因）

免责声明

希望

以前上班的地方不能上网，所以这么多年白天上班的时候几乎没有上过网。又加之自己喜欢看书，所以书倒是看了很多。憋了这么久之后，现在终于可以在有互联网的情况下一吐为快。吐槽不分对错，不分内容，不分对象，吐槽仅仅是为了吐槽而已。可以较真也可以不较真。真可谓是“假作真时真亦假”。

既然是程序猿吐槽，那么就合成了一个用来吐槽的名字。使用了我最崇拜的两个人的名字合成了一个Martin Stallman（Martin Flower和Richard Stallman）。

正如开头所说的，吐槽不分对错，没有理由。都是随心所想而已。当然吐槽也是有引用的，我都会尽量标明。这算我的免责声明吧，如有雷同，纯属巧合。

因为个人关注方向的不同，所以吐槽中的背景或者依赖的一些知识或者侧重点就会不同。我个人的方向有必要说一下，免得大家的侧重点或者关注点不同而产生沟通的深沟。个人喜欢或者关注面向对象，重构，模式，组件化，交互理论。除掉计算机技术方面，还喜欢看经济学，历史，心理学，文学方面的东西。当然空闲时间喜欢羽毛球还有台球，书法也是我的最爱，毛笔和硬笔书法都是。

最后想要吐槽的可以一起来吐槽。大家吐槽才是真的吐槽。

## 程序猿的吐槽二

## 程序猿的吐槽三

## 程序猿的吐槽四

## 面向对象模式（OOP-Pattern）

**动态加载**（set关联对象），继承变成组合。

桥接模式核心思想。

状态模式

命令模式(针对指定命令使用专一的调用者Invoker,Invoker和命令是同生死的(这里是聚合,应该不是同生死,即调用者可以调用不同命令),代码实现中有瑕疵,使用了构造方法)

备忘录模式——CareTaker类对拥有的Memento进行动态的加载。前者管理后者。

**代理控制**（当前类方法的核心使用关联对象的方法，即当前方法核心业务处理实际是一个代理方法，用于解耦或者分离控制）

代理模式，

命令模式。

装饰模式

外观模式（关联对象方法的组合，一组方法）

适配器模式

策略模式

状态模式

命令模式(调用者的Action执行命令的Excute),命令执行的核心(Excute)是命令接收的执行(receiver.Action(), )

中介者模式——使用中介者类将同事类之间的关系分立开来，在中介者类中进行控制，如果执行则在中介者使用同事类方法，也是一种代理。

**同生同死**（关联对象和当前对象同样生命周期，构造方法中设置关联对象）

策略模式()

命令模式(每一个命令的生存都依托一个命令接受者,同生同死)

中介者模式（Colleague类出生的时候就需要拥有一个Mediator对象）

**对象缓存**（缓存和延迟加载，关联的对象不立即创建，需要时创建，也不立即销毁，用于缓存）

单例模式

享元模式

**分离无关操作**（将容易变化的部分分离出来）

访问者模式，（将Element结果中的访问操作分离出来，形成Visitor类）

## 稳定性和效率关系Done

2012-8-8

线上运行系统有一个矛盾的点，就是如果要对其进行修改，就会引入潜在的问题，进而影响系统的稳定性，当然这只不过是一种潜在的风险。而这种潜在风险的高低有一些影响因素。

1. 对现有系统的熟悉程度
2. 对修改技术的掌握，比如重构技术等（重构技术对于修改软件来说非常重要，如果有非常高超的重构技术，那么在对系统不是很熟悉的情况下依然能修改出来高质量的系统，也就是说高的重构技术可以弥补对系统的不熟悉）。
3. 现有系统的架构，如果现有系统的架构比较好，耦合性很低。那么新修改就大大降低了影响范围。
4. 流程的规范程度（开发和测试）。
5. 开发人员的责任心和细心程度（在所有的工作中，一定首先考虑“人是不可靠”的这一前提，然后在这一前提之下做好其他相关工作。这样人如果是可靠的，那么事情会处理的非常顺利。当然事情还是人做出来的，所以最后一条我引入了人的责任心和细心这一因素。还是要相信有一些人可以干好非常细致和繁琐的活而不粗心）

某天听到两个人在吵这件事情，一个激进派觉得现有系统的一些缺陷影响系统的性能，需要进行修改。而另一个属于保守派的人则认为，你的修改可能会引起系统的不稳定，导致系统出现故障。为修改这一个问题不划算。两个人争的脸红鼻子粗。

对于任何一个系统来说，如果要修改，则制定一个稳定性和高效率之间的一个可接受比例，这个比例的制定根据对系统的了解情况和系统的质量来决定，根据以往的经验可以做一个估值，比如我一个修改能提高了50%的效率或者性能，但是这一块的修改会影响到30%的核心，那么就应该考虑修改。经过规范的流程和优秀的技术可以将影响降低很多。

所以，结论就是，在保守（稳定性）和激进（效率）之间根据自己系统的情况设定一个估值比例（改还是不改的估值）。当超过这个比例之后以稳定性为重而考虑放弃高效率，如果在这个可允许的比例之内，则不用害怕影响稳定性而显得非常保守，以至于不敢对系统进行效率上的优化和改进，这样日积月累终将会导致系统虽然很稳定，但是已经没法使用了。

其实在上面的这个估值过程中，还有三个至关重要的因素，一个是领导是否属于保守者，另一个是用户是否属于保守者（固步自封者），还有一个就是真出问题了责任能否担的起。

## 重构时机Done

2012-8-9

在对项目的业务和代码不是很熟悉的情况下，在添加功能或者修改代码的时候，尽量遵守和之前代码一样的风格和逻辑，即使以前的代码风格和逻辑很不好，此时不管你有多么大的重构冲动，还是应该忍耐一下。但是你可以做一个重构标记，作为一个任务待定。切忌装13而贸然标新立异的改动，记住即使风格混乱，只要混乱的非常一致，那也不错，因为风格没有对错，只有是否一致。如果你需要修改的代码你觉得很不爽，但是人家全部的代码都是这种风格，那么你就需要好好考虑一下。你的改进也许使得代码变得真正的混乱和不一致了。

正如一排走方阵的队伍，如果大家都迈脚迈错了（或者大家都顺拐了），那么对于整个队伍来说还是整齐的。当然观众可以通过这种“错误”的逻辑规则来理解这种方式。但是突然队伍里面来了一个人，他觉得大家都走错了，或者这种走方阵的方式觉得太变态，太out，太没有技术，自己用这种方式都觉得丢人。这时候他自己修改了之前的这种出脚的方式（比如之间都是先出左脚，而他修改为先出右键以显得更为先进和牛逼或者“正确”），这样就出现了有迈左脚有迈右脚的情况。这种不一样的情况出现之后，观众看到的就是混乱和无法理解，或者规则就会变得更加负责和说明文档变得更加厚实。

那么对代码的熟悉如何量化？1，不熟悉此段的业务逻辑，2，不了解此段代码的全局调用关系。

如果此段代码相对独立，调用关系单一，或者你掌握了全部的调用关系，那么即使业务逻辑你不是很熟悉，那么也可以利用重构理论在不改变现有代码逻辑和实现的情况下进行代码结构和风格的重构。

软开心法一||声明

经过考虑，准备将两年前开始写的一本给自己看的书连载出来。这本书主体已经完成，但是会不断的扩展。暂时的名字如题目所示《软开心法》——即软件开发心法。面向软件开发人员，内容涵盖比较广泛，当初名字也是改过多次，在这里暂时以此为名，如有改动再行说明。

本系列的内容最初来源于在一个技术论坛上的连载，当时受版主所邀，开始在论坛写设计模式，所以最初的公开文章应该是在求实bbs论坛。不过在那期间部分文章我也在自己的baidu博客上公开过一些。但不管怎么样，这几个地方的文章所有权都归本博主，本博主保留对此文章的一切权利。

由于本系列都属于技术文章，而技术不是我发明的，所以文章所述都是自己对已经存在技术的理解和心得，还有一些是从事技术过程中的经验体会。当然有些观点可能是出自某个技术书籍或者文章，这些我都会在下一篇说明。

文章中涉及到的代码已经公开在google code和github，你可以通过搜索dprun来得到，C#的代码已经完成，所以基本停止开发。我希望能有对应的java版本的代码，虽然差距不大，也不影响阅读和理解，有了总是有好处的。但是由于时间有限没有完成，当然如果你希望加入开发，那么你可以在github上fork此项目，修改之后可以告诉我，如果我觉得可以会合并你的版本。或者在google code上加入开发组，也可以直接联系博主。此代码通过GPL协议发布，当然也遵守google code和github的相关协议和约定。

关于本系列的传播与发布说明，本系列文章全部是原创文章，你可以转载，但是必须注明，如果有其他用途可以通过联系本博客的主人来说明情况，禁止以营利为目的的公开发表和侵犯本博主对文章的所有权。

最后如果你有什么问题可以和我讨论，或者直接留言，或者通过邮件方式（email可以在本博客的左侧“个人简介”或者顶部的“关于我”中看到）。我非常希望批评和讨论，因为这是推动技术和进步的最好方式。

所有已经发表的，都算公开了，这样如果你想要pdf版本的，可以通过邮件免费索取。当然我也会不定期的将公开部分以pdf文档的形式上传提供下载。

软开心法二||参考文献及推荐阅读

我的任何表达都不会离开自己的技术背景，而文献列表在很大的程度上能表现出来写作的技术背景。所以我在声明之后首先将参考文献列示出来，因为有些表达或者思想我也不能明确来自哪本文献的哪个部分，但是坦白的说，这些文献确实影响了我，有些甚至深入了骨髓，以至于一些时候我所写的都不过是对文献原文转述或者类似表达而已。

下面我将列出本系列文章的主要参考文献，因为这些都是在写作的时候手头会翻的，或者遇到问题会查阅的。我不得不说，这些列表中的书籍都是很经典的，基本都是我认真阅读过至少一遍以上的。你即使不读本书也希望能读一读这个参考文献列表中的书籍，因为更多的时候我觉得这个列表是我的推荐阅读列表。我希望每一个计算机相关专业或者软件开发的人员都能读一读。

下面参考文献列表，名称和版本我是以我书桌上的为准，是否有最新的你可以去查。

[1] Michael Blaha,James Rumbaug.UML面向对象建模与设计.北京：机械工业出版社，2006.

如果你想了解UML及面向对象方面的知识，那么这本书非常不错，前面是UML的基本知识，后面对于软件工程和面向对象的讲解非常不错，由于看真本书之前已经了解了UML的知识，所以这本书对我最大的影响是其中关于面向对象和软件工程思想的一些见解。

[2] Erich Gamma,Richard Helm,Ralph Johnson,John Vlissides.设计模式——可复用面向对象软件设计.北京：机械工业出版社，2000.

真本书是业界所称的GOF的宝典，设计模式的源头书籍，使用C++讲解，是一本博士论文，所以还是比较晦涩的。

[3] 阎宏.Java与模式.北京：电子工业出版社，2002.

相比上面的一本设计模式书籍，这本非常厚的书籍很详细，代码用java实现，例子很生动。

[4] Bruce Eckel.Java编程思想.北京：机械工业出版社，2007.

这本书是java方面不错的一本书，我接触java的时候用java大全入门，然后就看的这本书，书籍很不错，坐着也是C++编程思想的作者，对于面向对象的理解比较透彻。

[5] Christian Nagel,bill Evjen,Jay Glynn.C#高级编程.北京：清华大学出版社，2006.

因为我的书籍中代码实现使用C#，所以有一些语言语法上的东西，我会从这本书中查找，对于C#的学习者来说，刚开始通读一下本书是一个不错的开始。

[6] Pressman，R.S.软件工程——实践者的研究方法.北京：机械工业出版社，2007.

这本软件工程书籍很不错，我们当初使用的教材就是这本，在工作中我也不断的看这本书，从实践者的角度来解析的软件工程的实用性。书籍中的很多工具和建议都非常不错。

[7] Joshua Kerievsky.重构与模式.北京：人民邮电出版社，2010.

这本书英文是refactoring to patterns，从英文上可以看出来，是重构到模式，汉语的翻译会影响对本书的理解。本书是通过对现有的代码重构，可以引入设计模式来重构已有的代码。对于理解设计模式来说，非常有帮助的一本书。但是里面只有部分的模式。

[8] Matin Flow.企业应用架构模式.北京：机械工业出版社，2004.

Martin Flow是我的偶像，如果想从宏观上来理解软件架构，针对企业级的软件架构发法，这本书写的很不错，强烈建议读一下，不同时间读就会有不同感受的一本书。

[9] Matin Flow.重构——改善既有代码的设计.北京：中国电力出版社，2006.

这本书我一直认为是做软件开发必读的，他会教你如何写代码，如何写高质量的代码，看了之后你就觉得，写代码也是一门艺术。

[10] Dino Esposito, Anrea Saltarello.Microsoft .NET企业级应用架构设计.北京：人民邮电出版社，2010.

对于想了解.Net技术架构的人来说，这本书应该读一读。

[11] Jeffrey Richter.CLR Via C#.北京：清华大学出版社，2010.

这本书不用说了，如果你是搞C#或者.Net开发的，你应该多次仔细的看一下本书，不然你会好长时间也不能进入.Net美好的大门。

[12] Donald Norman，梅琼译.设计心理学.北京：中信出版社，2011.

这是Norman的书，感觉和计算机没有说明关系，但是我要说的是，每个人都应该看一下这本书，并且能理解其中的精髓。理解了这本书之后你的脾气就不会那么大了，并且能够在各种地方用到，包括写代码，设计，接口调用等。

[13] Alan Cooper,Robert Reimann,David Cronin,刘松涛译. About Face 3 交互设计精髓.北京：电子工业出版社，2008.

针对计算机的交互设计，作者是cooper的创始人，交互设计之父，vb之父。他对于交互设计和计算机的理解比较深入，做程序应该看一下这本书，不要做一个“不解风情”的开发人员。

[14] 张亮.细节决定交互设计成败.北京：电子工业出版社，2009.

这本书简单易读，如果觉得上面的两本不好读，那么这本也非常不错。我非常强调用户交互，并且倡导将交互理论引入代码设计中。并且用户交互不仅针对计算机，对于用户和工业设计来说，都可以看一下。

[14] Scott W.Ambler,Pramodkumar J.Sadalage著,王海鹏译.数据库重构.北京：机械工业出版社，2007.

因为我专门针对重构技术进行了研究，而本系列文章中也有对此的描述，所以传统的认为重构只能对于应用程序，但是这本书教你如何对数据库进行重构，虽然看着像雷池，但是技术是可以减少风向以达到可行性的，如果你是重构的粉丝，那么你一定要看此书，不然你对重构的理解就比较狭隘。

[16] Eric Freeman,ElElisabeth Freeman等.Head First设计模式.北京：中国电力出版社，2007.

这本书我是最初了解设计模式的时候看过的，第一次看的是GOF的设计模式，看过之后稀里糊涂的，因为是学生并且GOF的书本来不适合入门，但是这本书对我印象很深刻，如果你想了解设计模式，并且觉得GOF的书籍太晦涩，那么这本你可以看看。

[17] 梁爱虎.SOA思想，技术与系统集成.北京：电子工业出版社，2007.

这是一本关于SOA方面的实战性书籍，讲述了SOA的应用。

[18] Connolly,T.M,何玉洁等译.数据库设计教程.北京：机械工业出版社，2005.

脱离了SQL的数据库，如何设计，面向对象和对现实的理解如何转化成高效的数据库，这本书你应该看一下，否则你永远不明白数据库设计的奥秘，并且会阻碍你的进步。

[19] Andrew S.Tanenbaum,陈向群等译.现代操作系统.北京: 机械工业出版社，2005.

操作系统是计算机的基础，这本书对于操作系统的讲解很不错，并且操作系统中的一些理论和原理对于普通程序的设计来说非常都是有借鉴意义的。比如我就常常使用“鸵鸟算法”，如果你不知道什么是“鸵鸟算法”，那么你该看看这本书。

[20] David LeBlanc, 程永敬等译.编写安全的代码.北京: 机械工业出版社，2005.

这是一本教你如何写代码的书籍，如果说《重构》教你写好看的代码，那么这本书就教你如何写安全的代码，如果你想提高自己的代码功夫，那么这本书还是得看一下。不然你不会知道代码中的白名单和黑名单是啥意思。

这些书籍是我在写这本书的时候直接参考过的，当然还有很多没有直接参考，但是凭借自己的记忆和理解，在本书中也都有体现，不一一列举了，但是我看过的一些书籍都在豆瓣的读书列表中有维护，你可以在豆瓣上查找用户Stallman（没错，这是我从偶像Richard Stallman的名字里面摘出来的）的读书列表。当然这个列表也不全，因为开始维护的时候之前读的一些也没有全部维护进去。

所以这篇作为声明的补充部分，作为对相关作者作品的尊重。

三 说明

三 目录

四

## 勿用则不可见Done

2012-11-19

勿用则不可见是什么意思呢？先不要着急，来看两个例子。

第一个例子是在高速路上的一次经历。有次在一条新开的高速公路上坐车，走了一段时间觉得应该从出口下高速了，但是发现前面还有路，于是大家觉得既然这里路没有封，那么说明前面还是有路，于是就继续往前开，打算在没有路的地方下高速。结果开着开着发现前面的路还没有修好，立着一个很大的牌子“前方不通”。于是大家在大骂坑爹的同时掉头回来。

第二个例子是北京地铁的例子。北京地铁中，新开的地铁一般都有屏蔽门，而早一点的都没有，这个屏蔽门给人一种很安全的感觉。你不会掉到铁轨上，也不会在列车到站的时候因为你站在了黄线之后而导致什么后果，因为不允许你站的地方你根本就接触不到。

这两个例子一正一反，正是我需要说的“勿用则不可见”，即不让我使用的则不要让我知道，如果让我知道，让我看到了，那么就让我使用。这归结为一个用户交互的原则。我将利用这一原则来说明软件开发中如何应用。

首先我需要说明的是在界面设计和用户交互中如何来应用。我是做软件的，所以我对软件有时候很容忍，有时候又很挑剔，虽然看着很矛盾，实际上这就是真实的普遍情况，谁让人本身就是一个矛盾综合体呢。在软件中有一个很常见又很让人不爽并且大多人都遇到过的一幕：用户在软件中发现了一个功能，很可能是一个按钮或者一个链接。用户根据内容判断应该是自己需要的功能，然后很兴奋的点了一下，结果弹出来一个框，也许是一个很可爱的框及一段很“非主流”的提示（程序员好像很喜欢装可爱），总之意思就是你没有权限操作。然后用户嘟着嘴骂一句“我X”之类的话。这种情况还算属于容忍范围之内的，更可恶的是你点击了好多按钮，深入了更多层之后才会发现你没有权限操作。对于这种情况，是典型的从技术角度设计的操作，技术驱动的实现，在现在用户为王道的时候，必须用户驱动产品和功能。那么上面的这种情况就不能再出现。如果一个用户登录进入，那么他所看到的功能都是可以操作的，没有权限或者不能操作的功能（按钮，链接，输入等）必须隐藏掉，不要让用户看到，这便是“勿用则不可见”的界面实现。

其次来看看函数中的应用，这并不是我扯淡，交互理论我可以随便应用在程序的编写中，这不是瞎扯。研究交互是以人为本的行为，为了方便交互的主体——人。而程序的编写中不可能没有人的参与。面向过程程序设计的核心是函数。函数和其用户（注意程序的用户和产品的用户之间的区别，这个我曾经在相关的文章中说过）之间便存在交互。函数是用来被调用的，所以对于调用者来说，函数如果不想被调用，那么就必须避免对其可见。这里的可见表现为函数的命名和参数设计。对于函数的命名，避免名称与实现不一致而引发的用户“走叉路”，比如一个叫SetName的函数，里面不是设置名称或者不仅仅是设置名称，那么就不要让用户看见SetName。再看看参数，有一些方法会设计出默认参数，即这个方法存在一个参数对于调用者来说不用设置此参数或者随便设计都行，这个参数便属于“勿用”参数，那么保持对其“不可见”则算得上一种好的交互。这时候使用函数重载可以得到更友好的交互，或者是更好的可读性。虽然说函数的重载一直有争论，过度的重载很容易让用户糊涂，但是好是坏在于如何使用。

再次看看面向对象编程中这一原则的应用。可以说面向对象具有天然的属性来支持这一原则。面向对象技术的封装和可见性这两个属性，对于支持“勿用则不可见”是非常的方便。从语言的角度来说太抽象了，那么我就从开发的实现角度来说明，假设你有一个很强大的代码编辑器，vs或者eclipse都行，你使用这个编辑器来编写C#或者是JAVA的程序。你输入一个类之后，代码的智能提示功能就会提示出来此类的所有方法，但是请注意一下，私有的成员和属性这里是不会提示的，因为从语言语法上来说，不能在类外使用private修饰的方法和属性。所以智能提示是不会让你见到不能使用的方法。如果看到了，说明此编辑器的智能提示用户交互做的不好，违反了“勿用不可见”原则。然后来说说程序中的实现，通过对编辑器智能提示的说明，那么作为一名程序员，是不是在写程序的时候就考虑一下“勿用则不可见”，也就是在设计类和类的方法时候，不想让类的使用者使用时，则需要设置其为不可见，这样使用者可以完全不用知道不能使用的方法。依据“勿用不可见”和“最小范围知晓”原则，巧妙使用面向对象的可见性技术可以设计出用户交互非常好的接口和程序。

最后看看开发流程中如何使用此原则，交互理论可以使用在人类参与的任何活动之中。软件开发流程中最重要的两个环节——开发和测试。那么随之而来的两个环境——开发环境和测试环境。之前还看到了一些相关的文章，关于开发人员能否进入测试环境的争论。正好我的这个交互原则可以解释这个争论。能不能进入取决于你的测试环境对开发人员是否可用。在软件开发中，国内军用的软件过程体系是国军标即GJB5000和9000。在这个质量过程体系中，是不允许开发人员接触测试环境的。也就是测试环境对于开发人员来说是不可见的，这也是我基于的理论基础。在这种情况下，就不应该把测试环境的配置和地址及相关信息让开发人员知道。我见过很多地方，开发人员手中都有测试环境的地址，包括系统和数据库。只不过告诉开发人员不能修改，这无异于一个出差的妻子在自己老公房间里面安排了一个裸体美女，然后告诉老公不许出轨。当然你可以说我允许我的开发人员进入测试环境，那么软件出现质量问题你不就应该那么惊奇，你也不能怪开发人员的技术，也不能怪测试人员的技术。要记住，不出错最可靠的保证是流程和制度而非人的主动性。所以，“勿用则不可见”这一原则对于软件开发过程同样的可用，那么你还会把测试环境告诉开发人员吗？

“勿用则不可见”不仅仅用于上面的这几个方面，你可以用在软件开发的所有过程中，也可以用在管理或者社会生活中。好的交互可以让人更舒服，并减少错误。使用工具的时候更称心，工作的时候流程更贴心，这样不就人们生活幸福了么。这也是xjp在十八大结束后的发言，想想，当全国的党员在学习他的讲话精神的时候，你早就贯彻实行了，这觉悟和超前性只有中国的程序员才有的。

## 短字串的压缩算法

## 什么是聪明人

## 自己技术工作的说明和勉励Done

2012-12-24

好长时间都没有写软开心法了，不是最近没有了软件开发的心里体会，也不是没有什么东西可以写了，关于技术方面的东西，于我而言永远有的说。却是最近有其他的事情占据了空闲时间的主要精力。

业务工作主要是两个方面的内容，一个仍然是软件开发的体会系列写作，第二个是个人开源项目的开发。

关于第一方面的内容，还是会接整体系列——软开心法——继续写下去，不过下面的一个小系列会是一个相对独立的部分，单也是在软开心法系列之前构思好的。属于心法7个系列中的一个，之前的部分包括了两个系列，分别是“软件实践工程”系列和“设计模式”系列，下面将要进行的是“面向对象模式”系列。这个系列不是一篇一篇写的，相对来说属于一个整体，我需要全部写完之后贴出来。具体内容方面，可以说是对设计模式的一个再抽象和提炼，一共有5-7篇。

第二方面的内容是一些个人的开源项目，当前主要进行的是两个，一个是TaskSubmitor(<https://github.com/chengn/TaskSubmitor> ),这个项目是一个自动化目前公司开发上线流程的一个东西，采用迭代方式，最小功能使用，不断完善功能。采用在最快时间使用，不断完善中的原则。这个项目中的需要采用外部API较多，需要有操作vss，excel，word，git等的接口，因为会集成这些工具及操作（不用重复造轮子的开发法则），所以时间耗费相对多。还有一个是ServiceTester(<https://github.com/chengn/ServiceTester> ),是提供一个可视化的界面，用最简单的工具和最小的功能来提供一个对服务的调用，用以确保服务的正常工作和使用一个服务。第一阶段首先要确保的是对webservice的服务调用。其他远程服务后续添加。

除过上述的两个方面，还要保证一定时间的阅读，而这种阅读包括专业技术方面，还必须保证人文社科方面的阅读。对我来说，自己有一个对自己的要求：就是如果要写一个小时的文章，那么必须保证最少有3个小时的相关阅读，2个小时的相关思考。

时间总是觉得不够，不过凡事总得一步一个脚印的来，过度着急容易心生浮躁，这是大忌。切记！自勉！。

## Java反编译class

ChengNing

2012-11-2

**工具**

工具使用jad(还有joda,jd等工具)

1、使用jad反编译class文件，jad可以配置到eclipse中当做插件，直接下载jad的eclipse插件jar包，放在eclipse的plugin目录下即可。把下载的jad.exe放在JAVA\_HOME的bin下。然后eclipse中点击class文件即可查看到源码。

2、使用其他工具，FrontEnd Plus集成了jad的java反编译工具，支持多个文件的反编译，也支持各种选项。

**矫正反编译代码错误和奇怪代码**

**1、异常错误**

反编译之后出入类似如下代码：

DocumentException e;

e;

e.printStackTrace();

break MISSING\_BLOCK\_LABEL\_67;

e;

e.printStackTrace();

break MISSING\_BLOCK\_LABEL\_67;

e;

e.printStackTrace();

break MISSING\_BLOCK\_LABEL\_67;

e;

这种处理方式就是异常处理，反编译之后出现的这种情况，修改的时候使用try catch即可，其实完全可以将这块删除之后，然后eclipse会自动提示需要添加异常处理的模块，自动try catch即可。

**2、语句标号（goto）**

这种情况常见于条件语句，即转化为if，else语句即可，这种比较容易判断。

i = 0;

goto \_L1

\_L3:

i++;

\_L1:

if(i < 4) goto \_L3; else goto \_L2

\_L2:

g.dispose();

上面的代码实际上就是如下的代码：

i=0;

if(i < 4){

i++;

}else{

g.dispose();

}

**3、jvm代码信息错误**

String rand = backStr[random.nextInt(backStr.length)];

this;

sRand;

JVM INSTR new #203 <Class StringBuilder>;

JVM INSTR dup\_x1 ;

JVM INSTR swap ;

String.valueOf();

StringBuilder();

rand;

append();

toString();

sRand;

这种错误代码，也是很常见的，比如上面的代码中，rand 是局部设定的变量，而存在一个类变量sRand，上面的代码可以使用如下的代码替换掉错误。

this.sRand += rand;

当然，对这段代码进行分析一下即可看出，分析的时候需要结合下面4中讲解的字符串拼接翻译器代码，从这段混乱的代码中可以看到，有this和sRand，这种可以看出是操作了this的sRand变量，下面的jvm instr是创建了一个stringBuilder的的变量，实际这个变量是保存了this的sRand字符串。最后是append了一个rand变量，然后把这个变量又赋值给了this的sRand。

**4、字符串拼接的代码**

(new StringBuilder(String.valueOf(prefix))).append(file).toString();

上面的这种代码，实际代码如下：

prefix + file

所有字符串拼接都是转为StringBuilder使用append拼接之后然后toString。

**5、continue关键字**

这种常常会隐含逻辑错误，但是编译并不报错，这种问题最难发现。这种的反编译一般伴随着大段的循环代码，然后反编译之后会将循环代码或者if代码转成goto语句（参见2），然后在某个情况之下不需要继续执行，便使用continue截断代码执行路径。代码出现如下情况。

continue; /\* Loop/switch isn't completed \*/

这样的代码出现的时候，并没有报语法的错误，我遇到这种情况之后，程序运行之后结果出错，最后将程序中的这个反编译问题找了出来，将continue修改成break语句正常了。这样的问题常常会出现在while循环中，可能反编译出来的程序丢失了while循环，变成了if语句，常常是迭代器循环的时候，非常容易出现这种情况。

**6、迭代器循环**

迭代器循环反编译之后常出险两种情况，一种容易引起逻辑上的错误，一种没有逻辑和语法上的错误，只不过不是原来代码的写法而已。

iter = datalist.iterator();

goto \_L1

if(iter.hasNext()) goto \_L3; else goto \_L2

上面的代码可以利用类型2来翻译出来，但是这里会有个地方需要修改，那就是if必须换成while，而else中是一些异常处理而已。这里常见的隐含问题在于，在代码段L3中，常常含有break，continue等跳转语句，一定要分析清楚，不然很容易出现难以排查的逻辑错误，使得反编译出来的代码运行结果不正确。

还原之后的代码如下：

try {

Iterator iter = datalist.iterator();

while(iter.hasNext()){

}

}catch (Exception e) {

e.printStackTrace();

}

还有一种没有错误的译法如下：

for(Iterator iterrisk = tRiskLIst.iterator(); iterrisk.hasNext();)

{

}

这种译法没有错误，只不过是看着不习惯而已，原程序程序通常如下：

Iterator iterrisk = tRiskLIst.iterator();

while(iterrisk.hasNext()){

}