09下 2003.9.15

Windows 开发 Java
.Net 数据库 算法

程序员

实战应用 纯粹技术

Csdn 开发高手

普及开发知识 提高开发技术 交流开发经验 享受开发乐趣

著名游戏修改软件整人专家作者李果兆 《游戏修改历史》
外挂开发元老的现身之作 《外挂开发大揭密》
多位专业游戏研发者联合推出 《反外挂战略体系浅析》
中国大陆反外挂第一人林阜民专访 《游戏反外挂的法律维权和未来发展》

联合打造

游戏外挂之前世今生

高手的风范—— Jeffrey Richter 源码赏析
用 C# 实现弹出广告杀手—— Popup Killer
都是“单线程”惹的祸—— 一次惊心动魄的调试
举重若轻的 ASP.NET 开发工具—— Web Matrix

关于学习 C++ 和编程的 50 个观点（2003 修订版）
从 VB 到 VB.net—— VB6 程序员如何转向 .NET
Java Learning Path—— 带你走上 Java 开发之路
数据库设计经验谈系列之 给你的数据对象起个好名儿！

本期优惠价 8.8元

生于下半月

——致读者的一封公开信(代发刊词)

《CSDN开发高手》消息发布后,引起了很大的反响,众多读者和网友纷纷来信或发表评论,对新杂志的定位、名称及两本杂志的发展方向提出了很多中肯的意见和改进建议。论坛上长长的帖子、言简意赅而语意深长的网友发言,都让我们深受感动,也更感自身责任重大。

首先感谢大家对我们始终一致的关注、支持与厚爱。《程序员》经过三年的发展,已经得到了广大读者的认可,大家也对我们提出了更多更高的希望,但是限于篇幅和定位,一本杂志很难满足广大读者的需求,我们经过精心的准备,从9月份开始,对《程序员》杂志进行扩容和提速。

对今后的杂志发展,我们有如下规划:

一、两本杂志内容定位各有侧重,形成互补

《程序员》将继续保持其综合性、产业性和技术性的特点,重视产业、人文、管理和技术四大内容。在原基础上,《程序员》将做一些重大内容调整,加强文章的深度,提高实用性和指导性。

人物&报道:从技术视角,关注中国软件的发展与创新,报道产业中最具活力、最新的人物和事件

这部分内容是我们杂志的传统强项。我们将继续强化这个版块,捕捉产业界发展的最新动态,及时响应重大事件,追踪技术名人,进行系统化的纵深报导。另外,我们会逐渐强化评论功能,提高交互性,鼓励广大读者参与其中。

管理:加强实践方法的探讨与应用,强调企业和项目实战案例的剖析与挖掘

随着软件产业的逐渐成熟,管理和软件工程越来越重要。为了顺应这一潮流,伴随我们的读者一起成长,我们将逐渐加大在管理和软件工程方面的力度,突出中国特色,注重调查研究,加强实践方法的探讨与应用,重视国内企业和项目实战案例的挖掘与剖析,与读者一起探索一条实实在在的软件企业管理与软件工程之路。

技术:关注技术趋势,企业开发,深度应用。

程序以技术为本,《程序员》杂志深深认识到这一点。我们的技术板块一向走高端路线,偏重技术前瞻,关注新技术,探讨技术趋势。未来我们将继续坚持这条路线,但是会更加注意结合实际,关注实践,帮助读者掌握第一手的趋势性技术资讯,开拓技术视野,融入全球性技术文化。

《CSDN开发高手》将是一本注重实战的纯技术杂志,她的宗旨是:普及开发知识,提高开发技术,交流开发经验,享受开发乐趣。她以帮助读者更迅速、更有效地提高开发水平、提升个人价值为已任。

杂志内容将分为特别专题、技术专区、特色专栏以及学习&文化四大部分。致力于实用技术与技术实践。

特别专题:每期杂志将推出一个特别专题,围绕一个热点开发技术,分层次、多角度地深入探讨。

技术专区:针对目前最重要的几个技术方向,汇集国内外知名技术专家和CSDN技术高手们的智慧和经验,伴随程序员的学习成长。

特色专栏:作为《CSDN开发高手》特别奉献给读者的栏目,“特色专栏”倾注了编辑部和大量“外援”、“顾问”的心血。我们力图以新鲜、精彩、交互性强的新形式、新风格,做出实用,好看的栏目。

我们也深知要做好这样的栏目,难度很大,因此我们特别期待有更多的专家能参与,把您的经验、研究所得与读者一起分享。

源码赏析:有经验的程序员都知道,提高编程技术的最好办法,就是通过阅读大量的经典源码,就象好的作家必须阅读前辈的著作,吸取写作的营养一样。

经典名著《莱昂氏UNIX源码分析》造就了整整一代黑客高手,成为编程历史中的传奇。今天各种高水平的开源软件随处可见,但是又有多少人去仔细读过这些代码?去体会高手的心血之作呢?

我们相信,源码学习不在多而在精,与其贪多,不如细细品味,反复赏析,实力就会在不知不觉中增长!

软件克隆:优秀的软件,都至少拥有一项超越一般软件的功能,只有不断追求卓越的软件,才能获得最终的成功。你一定会在使用软件过程中被某项特色功能所震撼,或者为某个特性而着迷。程序员之不同于普通用户,就在于我们不仅仅停留在感叹上,而是会考虑How和Why,在解析和实现中体味成功的快乐。“软件克隆”专栏就是要为大家揭密软件精彩的内部奥秘,通过实际动手提高你的实践能力。

大话Design:编程能力有三要素:算法、技术与设计。国内程序员普遍重视技术,疏于算法,而荒于设计。软件的实现,通常有多条道路可选,高手设计之道,在于简洁,在于深刻。“大话Design”通过轻松活泼的对话,以纪实风格向大家展示真实的设计过程,生动而富有启发性。

二、两本杂志的上市时间

调整后,两本杂志上市时间分别为每月的月初和月中。

《程序员》将固定为每月1日发行;

《CSDN开发高手》固定为每月15日发行。

三、技术专刊合集出版

除上述两本综合性杂志外,很多读者希望看到专业技术领域的专刊(如Java、.NET),为此,编辑部特别引进了国外6本著名技术期刊,即《Visual Studio Magazine》、《Java Pro》、《asp.net PRO》、《Delphi INFORMANT MAGAZINE》、《XML & WEB SERVICES》等。

这些一流的国际技术刊物,在其专业技术领域内,都有着很好的影响和口碑。它们内容精彩,技术新颖,极受国内外读者的青睐。

我们将在10月一共推出多本汇集国外期刊2002-2003精彩文章的技术专刊中文合集,同时,我们也非常期待国内技术专家在这些专业方向给我们撰文投稿。

一份杂志的创刊、发展与提升,就如同一个新生命的诞生和成长一样,离不开所有喜欢她、帮助她的朋友,我们期待着大家的支持、帮助与参与!

本刊编辑部

2003.9.1

目 录

CONTENTS

csdn

目 录

CONTENTS

技术专区

特色专栏

目 录

CONTENTS

Csdn 开发高手 Programmer 程序员

（总第71期）下半月

主管单位 ： 中国社会科学院
主办单位 ： 中国社会科学院文献信息中心
协办单位 ： 北京百联美达美数码科技有限公司
www.csdn.net
出版单位 ：《程序员》杂志社

总 编 ： 黄长著
社务副总编 ： 张悦校
副 社 长 ： 蒋 涛
执行副总编 ： 熊池舟
编 委 会 ： 黄长著 张悦校 陈洋彬 蒋 涛
熊池舟 唐 琦 曾登高

编 辑 部 ： 孟迎霞（主任）
责任编辑 ： 汤 韬 闫 辉 韩 磊 郭红俊 宋志翔
华东记者站 ： 熊 节(xiongjie@csdn.net)
华南记者站 ： 张 里(zhangli@csdn.net)
美术编辑 ： 关峻峰 孔祥利 吴志民
电 话 ： 010-84540265
投稿信箱 ： editor@csdn.net

发行总监 ： 俞 波
电 话 ： 010-84540235
信 箱 ： sales@csdn.net

广告总监 ： 唐 琦
电 话 ： 010-84540265/36
信 箱 ： adv@csdn.net market@csdn.net

华南联络站 ： 刘 羽
电 话 ： 020-61213341/42/43/44/45/46转918
信 箱 ： hnoffice@csdn.net

读者服务部(邮购)
读者信箱 ： reader@csdn.net
地 址 ： 北京市朝阳区北三环东路8号
静安中心26层（100028）
电 话 ： 010-84540262
传 真 ： 010-84540263
邮 购 ： 读者服务部

国际刊号 ： ISSN 1672-3252
国内刊号 ： CN11-5038/G2

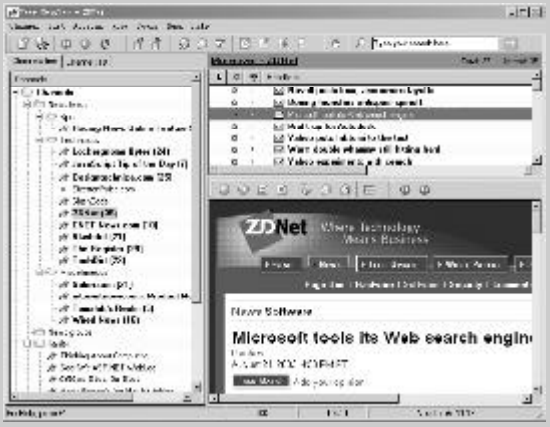
邮发代号 ： 2-423
印 刷 ： 北京乾洋印刷有限公司
广告经营许可证号 ： 京东工商广字0188号
网 址 ： http://www.csdn.net/magazine
出版日期 ： 每月1日
零 售 价 ： 人民币10.00元

固定栏目

本期大嘴 CSDN网站内容总监 韩磊

1

WEB是一张网,当你试图用它捕获信息时会发现网眼太大了。幸好有RSS,我得以借助新闻聚合编织一张小些的网——不过,要覆盖“天下”之大,恐怕就力不从心了。在读者抱怨本栏目内容贫乏之前,我想推荐功能强大的RSS新闻聚合工具——NewzCrawler。该工具的1.5新版刚刚推出,不过,官方网站(http://www.newzcrawler.com)只提供1.4共享版的下载。



2

好吧,我的网里出现的第一条鱼是随SQL Server Yukon Beta1发行的.NET Framework 1.2 SDK开发包。虽然微软没有大张旗鼓地宣传,好事者已经将它分离出来供人下载。如果你热衷于追踪新技术,也许会想一试,但绝对不要在正式运行的服务器上安装,由此导致的问题不会有人负责(在2.0整数版本发布之前,或许我还要无数次重复这个忠告),已经有人吃过苦头了。另外一条跟微软数据库产品有关的消息是MDAC(Microsoft Data Access Component)2.8发布,这东西以前是包括在Windows Server 2003里面的,现在可以作为单独组件下载安装。

3

2002年引起轩然大波的.NET/Java平台生产力和运行性能评估报告已经有了更新版本(http://www.theserverside.com/home/thread.jsp?thread_id=20655&article_count=39)。Middleware Company再次甘冒天下之大不韪发布不利于Java平台的评估报告。报告中宣称最好的J2EE Web Service实现速度也只能达到.NET Web Service的29%。Java的拥趸和.NET的反对者岂肯善罢甘休?看来一场论战已是在所难免,不过显然Sun公司的注意力并不在此——在JavaOne 2003研讨会上,该公司副总裁和CTO均阐述了J2SE1.5(Tiger)/J2EE发展计划。据称新版本Java平台将集中于改善稳定性和易用性,对于开发人员而言可能“泛型、autoboxing、元数据工具”等新特性更具吸引力。遗憾的是我们得继续等待一年以上时间才能尝试这些令人心动的特性。.NET/Java平台之争不是一天两天了,这事儿没法从技术上去判断。市场,只有市场才是最终裁决者。真正的开发者不会一再询问“到底.NET和Java哪个好”之类的问题。只要精通其中一种已足够让你得到一份高薪工作。

Middleware的评估报告(部分)

4

在Port80 Software关于WEB服务器占有率的调查中，微软IIS占到53.7%的份额。这个数字令人吃惊，不过该调查是基于用户填表的，未经由权威机构认证，可信度到底如何很难说。比较可靠的消息是Office2003发布日期和价格已经确定；此前的测试版本流传甚广（显然有人违反了测试用户授权条款），据说一些新的功能值得一用。不过微软也有不开心的时候，最近的MSBlast系列病毒应该很令MS烦心。先是“冲击波”，后是打着“冲击波杀手”招摇撞骗的变种病毒；Windows，你的名字是弱者。不知道准备大举进军安全软件市场的MS，是否会有些郁闷？另一个郁闷的是Novell，由于营收持续下降，该公司不得不裁员10%。Netscape停止Mozilla开发，转行在线教育——现在轮到我们郁闷了。正是：滚滚长江东逝水，浪花淘尽英雄。在以火箭速度向前冲（有时是向后退）的IT业，“明天”永远是未知数。

5

Nokia购买Sega游戏部门；AMD购买VIA前Cyrix Media-GX部门；Oracle宣布延长对PeopleSoft的收购时限；被IBM成功收购的Rational宣称将执行改进产品的宏伟战略（事实上该战略是两年前IBM投入4千万巨资豪赌Eclipse开源项目的自然延续，Eclipse旨在集成多个开发工具）。在并购的叫嚷声甚嚣尘上之时，AOL却要求母公司AOL时代华纳把名字中的“AOL”去掉——看起来颇有些反讽的意味。天下大势，分久必合，是否又会合久必分呢？IBM收购Rational，Borland收购Together，Oracle欲图收购PeopleSoft，这还不只是简单的合并。对于程序员，也许更该看到并购背后的东西。“建模”已经成为开发界讨论的热点。如李维先生所言，集成化开发应该会独领风骚。

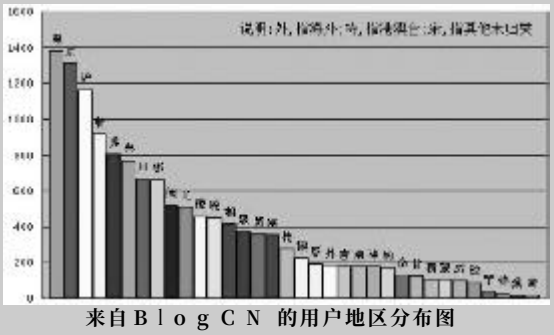
6

好了，让我们关心一下纯技术的新闻吧。SourceForge.net8月的Project of the Month是Boa Constructor，一个Python开发IDE工具，据作者称初衷是做一个Delphi for Python。Python越来越受欢迎，开源社区中的人们趋之若鹜，似乎将成为编程语言的另一王者。SourceForge是著名的开源社区，近七万个开源项目在它提供的平台上维护。在SourceForge上面，你甚至可以找到与Janeva功能类似的Corba/.NET集成中间件。与SourceForge的无私相比，SCO简直就是贪得无厌的山洞怪人。在争夺Linux所有权的拉锯战中，SCO似乎有恃无恐。该公司CEO Darl McBride在接受采访时，宣称SCO诉IBM案若不在2005年之前开始审理，将对IBM不利。自从SCO跳出来宣布“拥有”Linux后，股价已经从1美元/股涨到10美元/股。Borland Delphi 8很快会推出，神秘的Borland C++ Builder X渐渐开始显露庐山真面目。它们的共同特点就是跨平台。前者可用来开发Win32 Native应用和.NET应用，后者更是跨越Win32、Linux和Solaris。

SourceForge.net：最大的开源社区

7

然而，现在最热门的，既不是病毒，也不是并购，甚至不是厂商策划的、发布的、跳票的新版本产品，而是Blog和RSS。我没有给出RSS的全称，因为它有好几种解释。前几年的“频道”、“推”概念借尸还魂，真正开始为Internet读者提供有效的内容定制服务。不用怀疑他会不会持久，连MSDN都有了自己的各栏目RSS，还有什么好犹豫的呢？如你所愿，很快CSDN.NET也会推出RSS客户端定制。与此相关的新闻肯定会令方兴东博士兴奋——Blog一词被收入牛津字典。从BlogCN的用户地区分布图上可以看出这东西的火爆程度。由Blog引申出来的新词Moblog在美加大停电期间大显神威，许多blogger通过手机把停电现场情况抓拍下来发到自己的blog上。



来自BlogCN的用户地区分布图

本期“大嘴说事”到此结束，谢谢关注。如果读者有更多热点新闻或流行话题，欢迎联系大嘴栏目：bigmouth@csdn.net

游戏外挂之前世今生

序

小时候,拿着掌上黑白游戏机,与其说是我在玩游戏,不如说是游戏在玩我。大一些了,面对ROM卡带游戏,我很快就习惯去询问种种秘笈。后来,部分卡带游戏有了密码续玩功能,给了玩家以通关的捷径,而就在那时,《GAME集中营》上一篇名为《怎样破译游戏节目中的密码》的文章,不仅在玩家中引起巨大反响,也直接促使国内第一个专门研究游戏破译的工作室成立。那是什么时候?一九九五年。

接下来的几年,就是FC单机游戏逐步征服大陆玩家的日子,而修改游戏也成为玩家的一种习惯,GB4和FPE成就了这样一道上抛线。

其实,一切都是发自人的本能,这么多年我们做着什么?只有一件事情:打破规则,寻求刺激。找秘笈如此、破译密码如此、使用FPE也是如此,即便你只想快一点完成游戏。

接下来,在我大学生涯的后期,郁闷许久的中国玩家终于等到了为之疯狂的电玩新宠,网络游戏。网络游戏让真人成为游戏的对象,但已经习惯去打破规则的玩家们并不会因此有所改变,那是由本能进化而来的习惯,也正是那道上抛线的动力。

然而,意外的是,这一次玩家们对规则的破坏,却给这道曲线带来了新的外力,外力或许会让曲线变得更加优美,却也让本不平静的社会又多了一个麻烦,而这,就是今天的主题:外挂问题。

本期外挂专题将从技术、社会、法律多个侧面,把关于外挂最真实的情况展示给读者,让你自己去做这个简单而又复杂的受力分析,至于最后结果如何,我想每个人心中都会有自己的答案。

本刊特邀技术编辑王轶男

漫谈游戏修改历史

▶撰文/FPE之父 李果兆

所谓修改游戏,就是借由一些工具程序去修改游戏存盘文件或内存,达成无敌或者打不死的最高境界。原本游戏工作小组都会设定一些剧情及参数,让玩家一步一步参与,不过我个人是比较喜欢当超人的,现实环境中无法实现的事,由游戏来达成不是也挺好的吗?

既然要修改游戏存盘文件或内存,就必须先了解PC储存资料的基本方式(相信写过程序的人都知道):

- 内存上的地址观念:在IBM PC上的设计是低位对应低地址的,所以比如十六进制的"12AB",就会被存成"AB 12"这样。大部分内存和磁盘文件都是这样去储存资料,不过有些工具会自行帮你转换这些排列方式。
- 游戏中使用的数据结构:一般来说屏幕上如果看见10,内存中存放的方式是十六进制位的0A,不过如果是BCD码就会存成十六进制的10或是0100。此外还有其它数种编码方式。
- 游戏中连续数值的排列:如果你在RPG游戏中看到500/500这样的数字,那么在内存中它一般就是500,500连续排在一起的。

DOS时代

在以前的DOS时代,依据修改的方式我们可以把游戏修改分成下面四类:

- 直接修改游戏存盘
- 使用常驻修改程序(TSR)
- 使用某游戏专用修改程序
- 使用游戏内建的欺骗码(CHEAT CODE)

前面两类是通过搜寻要修改的目标数值,进而判断出游戏储存这些目标数值的位置,然后对其加以改变就行了,所以前两类工具的重点放在"搜寻"功能上。后面两类工具的重点在于游戏程序设计师或者有闲心的人针对特定游戏帮我们设计出方便的修改方法,这种设计可能是基于预先分析出的修改地址自动完成修改,也可能是直接向游戏实现嵌入修改服务,各种游戏都有不同的专用修改程序或欺骗码。

一、直接修改游戏存盘

这是笔者看过的最早的游戏修改方式,当时游戏一片要卖60元哦,不过当时游戏只是由简单的线条组成,声光效果当然没有办法可以和现在比。但在那时候就已经有杂志刊登 xxx 游戏要用PCTOOLS找ooo字符串改成zzz这类的修改方法了,甚至有人去追踪游戏程序代码,目的只是让游戏主角无敌而已。不过,现在除了要对游戏进行破解外,已经没有人做这种事了,因为如今游戏的规模通常很大,追踪起来会很累。如今进行程序追踪使用的工具大都是SoftIce。

修改游戏存盘除了要研究存盘各位置的含义外,还必须小心检查码。许多游戏程序设计师为了防止玩家乱改游戏存盘,会把待存储的数据先经过一番逻辑运算之后再存盘,这样就从很大程度提升了修改存盘的难度。

二、常驻型游戏修改程序(TSR)

话说当年在笔者还边看杂志边修改游戏存盘的时候,一位笔者的死党带来了一份惊天地、泣鬼神的礼物....陈伟谷先生所撰写的"电动克星5.19版",它的随呼随到以及多功能修改真是令只会使用PCTOOLS的笔者眼睛一亮,差点中风瘫痪。这就是笔者见过的第一个常驻型游戏修改程序。不过,当时"电动克星"只支持单色屏幕,而且只有高级扫瞄功能。但在那时"电动克星"已经能够连续扫瞄01 02 03这样的字符串了,而且还能进行如01 $1A 02 $1B这样的十进制、十六进制混合输入,这比后来的一些游戏修改程序还要先进,笔者的FPE(游戏修改至尊就受到了它相当大的影响。

FPE2001

这类游戏修改工具常驻于内存中,借由拦截键盘中断(INT 9)抢得系统控制权,在需要的时候随传随到。常驻修改工具可以记住所有符合当前修改条件的地址,我们只要不断告诉常驻修改程序我们在屏幕上看到的数字变化,它就能凭借这些信息,过滤掉部分地址,直到找到完全正确的地址。比如笔者的FPE就是其中一种,其最重要的功能就是搜寻。FPE的搜寻接口采用汇编语言语法,如1,2,ah,'jaw',所以FPE中包含一个分析器去分析玩家输入的资料格式,然后把它转换为一连串二进制,再基于这串二进制对游戏内存进行核对。FPE还支持低阶分析,也就是未知数分析,例如1,2,?,'jaw',+,1:100,其中第一个问号表示那一格未知,加号表示相对之前情况数值有增加,1:100表示该格数值范围在1到100之间。为了能够实现这些特别控制,FPE会由分析器产生出一个句柄串行,在进行游戏内存核对的时候,这些句柄会被同时进行考虑。

继"电动克星"后,接下来就是这类常驻型游戏修改工具的战国时期了,一大堆常驻修改工具纷纷出笼,其中比较有名的有精讯出版且具有低阶分析能力的GB4、小弟我的FPE 2.5、雄中的GBP、可以改DOS/V的DGB、加拿大的GW、兼具debug功能的GAMETOOLS等等。不过,后来由于内存技术的翻新,出现了可以切入保护模式的DOSEXTENDER,许多修改工具无法克服技术问题而没有再出新版,最后只剩下GW32和FPE4.1a可以修改保护模式的游戏了。

其实DOS Extender主要目的就是突破DOS 640k的内存限制,但由于当时真实模式的CS:IP无法索引1MB以外的地址,这就造成常驻游戏修改程序相继阵亡。FPE为了能够修改XMS/EMS内存,使用INT 16h对内存进行搬移,把XMS/EMS一块一块搬回DOS 640k内区域进行分析,计算对应地址,再通过INT 16h进行锁定,这种技术在当时算是一项创举了。

后来,操作系统迈入Win95时代,FPE 5.0通过上

GB4的十六色主界面,好难才截取到,若你是二十三岁以上的PC玩家,或许这幅画面会勾起你的一段燃情往事吧

述技巧,加上一些兼容性调整,便成功修改了Win95下的游戏,成为第一个能够修改Win95游戏的修改软件。FPE除了能够修改游戏外,还加有一些额外功能进去,比如抓图、阅览中文档案、听CD音乐、随时修改游戏存盘等。

撰写常驻程序有许多问题要注意,因为常驻修改程序在游戏中需要随呼随到,所以其拦截键盘的能力、兼容性以及稳定性都必须很好,此外还有DOS重入(INT 21h)、重复调用、常驻内存大小、重设8259 IC以及XMS/EMS支持等多方面问题。

PC设计中,CPU能够存取的东西都必须被存放在内存内,包括指令及资料,所以能够修改整个内存的常驻型修改程序,理论上就可以修改游戏中的所有东西了。不过,这毕竟是理论,若游戏资料都以特异的格式进行存放,而你又不是游戏的设计师,那又有什么办法?幸好大部分游戏设计师都不会无聊到设计种种增加游戏开发难度的东西去害自己。所以只要具备一定的经验,是可以改出一些新奇又好玩的东西来的。下面笔者就举出一些修改技巧:

- 一般来说,像RPG这种数值参数很多的游戏,游戏程序设计师都喜欢把各种参数放在一起,比如对于HP 200/200、MP 100/100,那么通常输入"200,200,100,100"就可以一次找到正确的存放地址,否则,可以试着改变一下资料型态。
- 正如游戏存盘会被编码,屏幕上你所看到的资料也可能会欺骗你哦,比如它在实际数值后面多加一两个零,本来应该是10的就变成了100或1000,这样在屏幕上会比较好看,但你分析的时候就需要猜出原本的数值,这就是经验加运气的事情了。

三、某游戏专用修改程序

专用修改程序的历史就无从考据了,最早应该是发

生在国外吧。这种程序大多是免费流传的,其工作原理非常类似于PCTOOLS,先由某位热心的朋友帮你分析出某个游戏的修改规则,然后他根据这个规则编写一个自动完成修改的小程序，这样就不需要玩家再拿PCTOOLS去改存盘,有部分游戏修改软件也可以自动产生这类程序。正是这个原因,这类修改程序也没法用在其它游戏中,"专用"就是这么来的。至于这些专用程序具体修改的地方,从游戏存盘到内存甚至于游戏程序代码都是可能的。

四、游戏内建的欺骗码(CHEAT CODE)

最早内建密技的游戏发生在电视游戏机上,相信玩过任天堂的玩家对于"上上下下左右左右BABA"一定非常熟悉,这也算是一种欺骗码。早期游戏机由于没有常驻修改软件,游戏本身又被存放在不能修改的ROM中,而又没有磁盘驱动器可以把它转录下来,使用欺骗码就成为唯一修改游戏的方法了。而在PC上最早的欺骗码应该算是"毁灭战士DOOM"的"IDDQD",按下去主角就立即成为战神。那么,为什么输入IDDQD就会让主角无敌呢？这是由于游戏程序设计师故意加了这样一段程序代码在游戏中,通常是为了辅助游戏的开发和除错,后来人们慢慢感觉到这种欺骗码的存在可以增加游戏的多样性,所以现在游戏大都被加入了各种欺骗码去实现穿墙、瞬间移动、武器物品全满、换服装等各种奇特的功能,这些都让游戏增加了乐趣。早期游戏杂志里用来刊登如何用PCTOOLS修改游戏的板块到后来就都变成了刊登游戏欺骗码和怎样利用游戏BUG达到变相修改游戏的专栏了。

窗口单机时代

接下来该谈谈现在的情况了。在Windows XP盛行的今天,DOS下的常驻修改程序还能如过去一般帮助我们在游戏中如鱼得水、虎虎生风吗？很抱歉,DOS下的常驻修改软件到了Windows下便全挂了,因为Windows的内存策略、资源管理方式和DOS截然不同。在Windows下已经没有"常驻"这个概念,因为Windows系统具有多任务特性,所以大多数游戏修改程序都把调用及切换工作交给Windows去做,而自己则只做最核心的分析扫瞄工作。

目前大多数游戏修改软件都是利用WIN32的debugging APIs,该APIs组合让程序员能够对另一个进程内的逻辑内存进行读写。除了读写内存外,拦截键盘消息也是一个重点,此时,DOS下那套拦截INT 9的方法已经不能用了(除非你要写的是如SoftIce这种在Windows加载前就得到执行的程序)。在Win32平台上,程序设计师需要编写一个动态链接库(DLL),然后利用Win32的Hook API去把这个间谍函数库加载到游戏的逻辑空间,让它去偷偷拦截键盘消息,当拦截到调用热键时就触发修改程序内核。这里尤其要注意间谍函数库与修改程序的沟通,解决办法有数种,我在FPE中采用的是共享内存。此外,分析内存时的效率也非常值得注意,因为低阶分析要同时扫描游戏的内存和硬盘档案,此处你可以使用Win32提供的内存映像文件(Memory Mapping File)去加快存取速度。

网络时代

前面谈的都是单人单机游戏,但是目前最热门、最抢手的游戏类型非网络游戏莫属,网络游戏把游戏对象由原来笨笨死死的计算机改成尔虞我诈的人类,他可能是你室友、邻居甚至是远在地球另一端的美国玩家,这种游戏方式很快风靡大多数玩家。早期最受欢迎的网络游戏,首推MUD(Multiple User Dimension或Multiple User Dungeon),简单来讲,它是一种多人文字角色扮演游戏。它只有文字接口,你必须用想象力去画出画面来,不过在其中你能够和真实的人物对谈、合作或PK(Player Killer即玩家间的对战)。由于早期CPU和内存太贵,所以信息处理都集中在一台"主计算机"中,并执行多人多任务的操作系统,像IBM大型主机操作系统Vax9k就是笔者接触的第一个大型集中式系统。"那如果很多人要同时操作怎么办？",因为这个问题,就有了"终端机"这种东西,它负责把主计算机的屏幕讯号显示在使用者的屏幕上,并把使用者的操作讯号传回主计算机。终端机本身并不执行指令,所有的运算都在主计算机内执行,MUD以及BBS就是这样的架构。现在的"Telnet"就是利用你的个人计算机(PC)来模拟终端机，当你用Netterm连接上MUD时,看到的只是主计算机的执行画面。

所以,如果我们想修改MUD,通过修改PC本身的内存,我们只能改到一些键盘命令,例如向左走改成向右走,却没有办法改到真正关键的资料,除非侵入主计算机修改,而侵入别人的计算机除了需要程序功力外,还是违法的行为唷！

在CPU和内存价格大幅下降之后,具备运算能力的

PC日渐普遍,PC游戏也多了起来,于是有人想到了以PC通过资料机和另一台PC联机,这样两部PC不就可以对战了吗?这就是"点对点"方式的联机,两台PC具备同等地位,如"魔兽争霸"及"终极动员令"等通过IPX进行联机的游戏,就使用了点对点架构,由于没有使用运算能力强大的主机及高速网络,这种构架只适合于少数人的小规模对战。

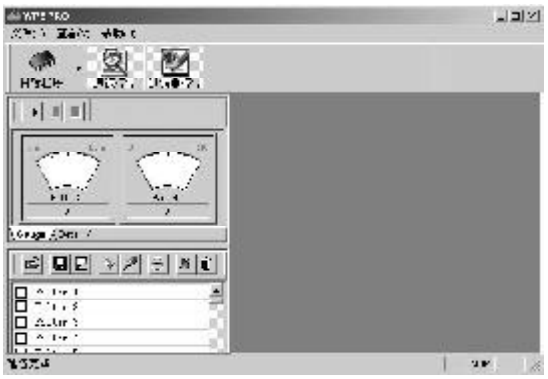
如果想修改基于点对点架构的游戏资料,就得看这个游戏是如何设计的了。大部分点对点游戏都会运用一些资料检查技巧,来验证对方传过来的资料正不正确,不正确的话就发送一个"同步错误"消息,然后结束程序。所以,如何避开检查是修改这类游戏的重点,而这就要求我们先了解游戏的检查机制,像有的游戏是用校验和(checksum)的方式来对比传过来的资料,例如保存游戏金钱数的内存内容为"02,01"(十进制为258),两个数值加起来就是3,所以游戏可能会先把这个3存下来,这时如果你把这两个数值改成"03,04",加起来就不等于3了,于是游戏就会发现错误。避开这个检查的方法是把原来的数值对调为"01,02",这样加起来还是3,不过数值却变成了513(十进制),也就达到了修改游戏的目的。

后来Internet逐渐发达,虽然点对点游戏也搬上Internet,但小规模对战无法满足那些喜欢一大群人混在一起、杀来杀去的玩家们,于是有人想到了可以利用大型主机强大的运算能力,通过快速网络专线去服务更多的玩家。而这,就是当今线上游戏所采用的分布式"客户端-服务器端"构架,如第一个国产线上游戏"万王之王"就采用了这种架构。所谓的"客户端-服务器端"架构和之前所说的"终端机-主计算机"相当类似,然而,在今天游戏接口都是全图形显示情况下,若还照以前那样把全部运算都交给主计算机去做,游戏速度便铁定跟乌龟爬一般,所以有人就很聪明地把原来终端机的地位换成PC,把部分运算交给PC去处理,主计算机则负责协调及交换资料,这种架构下的PC就被称为"客户端",而负担减轻后的主计算机就被叫做"服务器"。为什么是"服务器"这个名称呢?因为它代表着"一台随时等着为客户服务的机器",所以游戏公司的服务器都是24小时全天候等待玩家联机过去,而联机路径则由原来的内部网络变成了Internet。

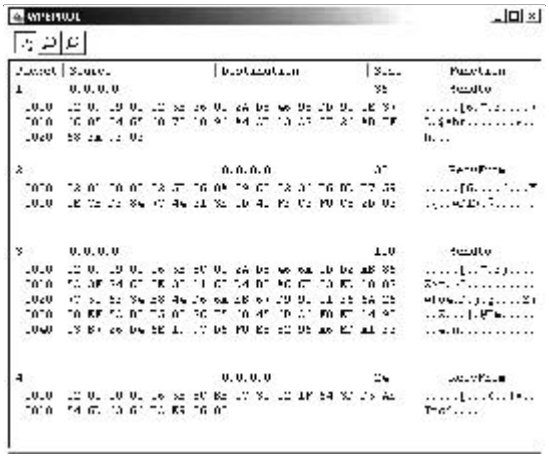
目前来说,若想修改线上游戏主要有三种方式:

第一种就是老方法"侵入服务器偷改资料",但如同前面所说,这除了需要你具备如"黑客帝国"中基努.李维斯一般的侵入能力外,还要小心被警察局逮捕!所以笔者建议大家还是不要轻易尝试。

第二种方法就是用如WPE这种工具去拦截客户端传出的资料,然后照我们的希望对它进行修改,再把这份假的资料发送给服务器,不过如同前面"点对点"的问题一样,服务器也会用种种检查方式来验证传送过来的资料正不正确,而且其中还有许多不确定性,所以同样困难重重。此外,发送假封包给别人的服务器还存在法律问题,像前阵子美国就有一个人通过发送大量假封包去干扰网站服务器,结果就被FBI逮捕了。



WPE PRO的主界面



WPE PRO的主界面

第三种方法以破解游戏程序中的游戏规则,或是找出游戏保护机制的漏洞为主,如前一阵子的"加速器",就是以破坏发送资料的速度来达到加速的目的。但是每一个线上游戏的客户端程序都不同,而且游戏公司又非常勤于改版,所以这类破解程序一般用很短时间之后就没法用了。

此外,直接以"逆向工程"追踪破解别人的程序也是违法的行为,目前很多游戏公司都在极积的取缔外挂

程序，例如很多挂在线上的 GM(Game Master)一旦看到有疑似外挂的行为就会直接踢人。所以，身为正人君子的笔者，是从来不会考虑采用这种方式去修改游戏的。

“讲来讲去都是破解、侵入的方法，难道没有其它比较合法的方法吗？”有的！所谓山不转水转，这里先介绍一个名词“宏(Macro)”。所谓的宏就是把一连串功能集合成一个功能，例如要在线上游戏“龙族”中进行练功的话，需要“把鼠标移到木桩上”、然后“按下鼠标左键”，这样你的角色就会打一下木桩来增加功力，把这两个动作集合在一起，就成了“练功宏”了。

或许你会问：“一直重复执行练功这个宏，不就可以一直练功了吗？”没错，但是我们却面临其它问题，例如“刀子钝了怎么办？”、“有人偷袭我怎么办？”、“力气用尽了怎么办？”。目前市面上已经有了几个提供宏功能的程序，虽然它们可以让主角重复执行一个宏，但同样都无法处理上述问题，此外，因为它们都是针对标准窗口应用软件设计的，对于DirectX全屏幕游戏也存在很大问题。

接下来重头戏就登场了，游戏修改软件“FPE 2001”的“机器人”就是改良自宏的解决方案，利用机器人可以实现各种全自动功能。FPE采用的方式为“录制”和“播放”，我们先操作一遍所需要执行的动作，如前面练功的步骤“把鼠标移到木桩上”、“按下鼠标左键”，FPE就会把这些步骤录制成一组机器人，然后就可以在任何时间、任何地点，通过FPE播放出来，当然也可以循环播放而达到一直练功的效果。FPE最厉害的地方是，你可以另外录制两组“补充法力”以及“离线”的机器人，这样我们可以用一组“练功”的机器人帮我们打木桩练功，我们自己跑去睡觉，然后当FPE发现你的法力用完，就会启动第二组机器人“补充法力”来帮你进行补充，但就在此时若有个坏蛋跑来偷袭你，那么FPE就会发现你的体力在下降，便启动第三组机器人“离线”去帮助你关闭程序。在你呼呼大睡的同时，FPE已经帮你做了这么多事情，是不是觉得很神奇？

其实FPE机器人的工作原理就是拦截各种键盘鼠标消息，把它们储存下来并在需要的时候进行回放。所以前面所说的间谍函数库就非常重要了，它必须能够精确控制各种消息并且接受修改程序的指令，包括播放时机、播放方式、启动及停止时机这些都必须控制得宜。

未完成的任务

有人说玩家永远是老大，玩家总是希望用最省力的方法去做更多的事情。由于机器人必须配合各种网络速度，所以发放速度无法加快，这显然无法满足玩家快速成功的愿望。另一个问题就是很多线上游戏练功所打的对象并不是静止的木桩，许多都需要攻击怪兽或是会移动的NPC，要完成这项需求就需要用影像辨识或是其它能够分析出移动坐标的方法，这又是另一项大课题了。笔者目前正在研究解决这类问题，希望不久以后可以造福更多“苦海无边”的玩家！

李国兆简介

台湾著名的FPE系列游戏修改工具的作者。FPE具有“悠久”的历史，早在DOS年代，FPE就以其强大的内存扫描功能以及对键盘和鼠标有效的拦截能力，而得到众多游戏爱好者的青睐。进入到了Windows时代，众多DOS下的修改工具已无力延续其辉煌。而FPE却经过几次版本更新后，又重新回到了焕发了青春活力。

(接27页)问：目前中国的外挂情况比较严重，请问你对国内网游产业的前景是否看好？

林：个人来说，前景我是看好的。关于盗版这个问题，光盘都掌握在盗版商那里，并没有其他机构介入，只要他们破掉了软件，接下来工作就太好做了。而线上游戏就不一样，服务器是放在游戏公司这边，而且游戏公司本身又有源代码，只要这两个方面不削弱，基本就会保持某种优势，这样产业就会慢慢起来。你知道为什么那些程序员去做外挂？因为他们没有出路，因为他们买的产品跟青菜萝卜一样的价，怎样去获取暴利，只能去做外挂。或许你觉得那样不错，的确有钱赚，但实际是错的，那样叫做短多长空，利益是短暂的。反之，若游戏产业起来了，需要更多的优秀程序员参与，这些程序员就可以进来，让产业更加壮大。就像我这里有些程序员，他们过去是怀才不遇，因为没有环境去给他们很好从事这个行业，或者他努力很久，念到清华，出来搞程序，工资只有三千，他会感到很不平衡。那怎么办，只好去做外挂。像在国外，特别是游戏开发程序员，他们在公司里的地位和待遇都非常高，现在就需要我们去把这个观念扶正。

网络游戏的协议分析

▶云网

记得刚到大学时,和大家一起玩的第一件事就是打牌,相信很多人都会有这个经历;我也是在那时候学会“升级”(“拖拉机”)的。

大二时曾经疯狂玩过,所以对“升级”一直情有独衷;但工作后,一方面找不到足够的人,另一方面就算找到人了也不能像以前一样通宵的去玩。不过,一次看到别人在网上玩联众的“升级”,呵呵,此后不怕找不到人了,工作之余可以随时和别人玩一下,过一把瘾。

现在转入正题,对于做外挂,以前没有做过,因为没有很多时间,所以也没有在网上去详细的找相关的资料,如果哪一位有这方面资料,不妨与大家共享一下。

一、数据的获取

1、对于动作游戏类:可以通过API发命令给窗口或API控制鼠标、键盘等,使游戏里的人物进行移动或者攻击,这个是对于本地游戏而言的,网上有很多这方面的介绍,在这里就不再写了。

2、截获消息:通过hook技术,获得与游戏相关的数据,之后就看你自己怎么处理这些数据了。

3、拦截socket包:要替换winSock.dll或者winsock32.dll,我们写的替换函数要和原来的函数一致才行,就是说它的函数输出什么样的,我们也要输出什么样子的函数,而且参数、参数顺序都要一致,然后在我们的函数里面调用原始的winSock32.dll里面的函数就可以了。当游戏进行的时候它会调用我们的动态库,然后从我们的动态库中处理完毕后才跳转到真正动态库的函数地址,这样我们就可以在里面处理自己的数据了;不过这种方法要自己重新去写。如下例子所示:

[redacted]

winSock32.dll里有很多函数,要自己一个一个的替换,岂不是很累,况且这个动态库还是公用的,万一哪一个地方写错了,就会造成很严重的问题,所以建议不要采用这种方法。

4、直接监听网络数据包:这个方法与sniffer和comview所用的技术差不多,不过我们并不要直接去监听到网络层或网络层以下几层的数据包,只要到IP层的就可以了。

5、利用 Raw Socket(原始套接字):它来发送和接收IP层以上的原始数据包,如ICMP、TCP、UDP等,一般的游戏数据量不大的话多采用TCP协议传输数据,如某某网络游戏的“升级”就是这样,而泡泡堂采用的是UDP发送的,数据量很大。关于这种方法介绍大家看一篇文章(http://web.nyist.net/~shadowstar/essay/security/sniffer1.html),会有很大的帮助,同时要感谢作者(shadowstar)提供了这么好的文章,我也从中得到不少帮助。

下面是我写的一段数据接收和分离出来的IP包的程序:

```
.....
try
{
nIpRevLen = recv( pCtlSocket->m_socket, cIpRevBuff,
       MAX_COMMAND_SIZE - 24, 0 ) ;
}
catch( ... )
{
}
if( nIpRevLen == SOCKET_ERROR ) continue ;
IP * p_ip  = ( IP * )cIpRevBuff ;
TCP * p_tcp = ( TCP * )(cIpRevBuff + IP_HdrLen( p_ip)) ;
if( p_ip->DstAddr != pCtlSocket->addr_in.
         sin_addr.S_un.S_addr ) continue ;
file://这句和下一句过滤其它不要的包
if( p_ip->Protocol != IPPROTO_TCP ) continue;

int nSrcPort = ntohs( p_tcp->SrcPort ) ;
if( nSrcPort != PORT_DODZ ) continue ;

char * pPackConten = ( char * )p_tcp + TCP_HdrLen(
           p_tcp ) ;
file://这里就是得到了包的内容了
nPackLen = ntohs( p_ip->TotalLen ) - IP_HdrLen(
           p_ip ) - TCP_HdrLen( p_tcp ) ;
file://这个是包的长度
.....
```

二、数据的协议分析

下面来分析一下“升级”游戏的协议，本来自己接收到数据，就可以把数据写下来，不过还是用别人的现成工具会更好一点，在这里我用的是comview3.3来接收数据的，这个工具还是很好用的，建议大家可以下载一个去用一下试一试；下面是一些接收到的数据：

```
0x0000   05 02 00 00 22 00 00 00-00 20 00 00 C1 00 00 00 ....".... ..?..
0x0010   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0x0020   06 00 00 00 67 67 2E 68-74 6D 05 02 00 00 23 00 ....gg.htm...#.
0x0030   00 00 01 20 00 00 46 00-00 00 00 00 00 00 00 00 ... ..F.........
0x0040   00 00 00 00 00 00 00 00-00 00 07 00 00 00 67 67 ..............gg
0x0050   2E 68 74 6D 6C                                  .html
```

像这样的数据，相信谁都会搞不清楚是什么东西，不过不怀疑有一些天才可以异想天开，不过要从原始数据来分析别人的协议确实要一些灵感的。好像是在为自己吹嘘，不好意思了.....不过呢，协议的分析要与实际的操作结合起来，如果你不会玩“升级”，很难想像你会很容易分析出别人的协议来。还好我自己在这方面还不错，在游戏“升级”中是一个小官员的级别了。

我们可以想一下游戏整个流程，登录、进入游戏室、找到玩家、发牌、扣底、出牌和结束。有了这样一个流程在脑子中可以帮助想象每一个数据的信息。

我用的是一个比较笨的方法，让这个过程和接收到的数据合在一起来分析：

1、启动comview或其它的接收数据包的工具，设置好过滤条件，只接收IP/TCP包，目标IP是自己的机器，如果不设置这些条件，网络的大量数据包就全部被你接受到了，如果要从几万个包中找出你所要的数据，太难了吧……

2、接着就启动网络游戏，进入“升级”（为了分析这些协议，我总被人家说慢得像乌龟，还失去了很多分），这时你就可要注意收集数据了，记下当前状态，什么时候叫牌、反牌、得分，很多数据都要记录下来，同时还要想着如何出牌才能赢，要不真有些对不起和你一起玩的对家了。

3、有了数据，分析协议就有了着手的地方，我们先看下面的一段数据：

```
0x0000   00 02 00 80 F0 00 00 00-01 00 00 00 04 00 02 00 ...€?.........
0x0010   00 00 00 00 63 61 6F 77-65 69 5F 30 30 31 30 00 ....dddd_0010.
0x0020   11 20 00 00 18 00 00 00-67 79 75 67 68 00 00 00 .......dddd...
```

从这里可以看到，ddddd_0010（原始数据不是这个，我修改了）与一起玩的名称是不是一样呢？如果是一样的就好了，这个可是服务器返回来的与你一起玩的玩家信息。

如果你做过SCOKET方面的编程，你会知道一个包的大体结构。包的大致结构是：应用包头标志；包序列号；整个包的长度；CRC校验（或累加和校验）；命令字；数据体长度；数据体。

网络上的数据并不像我们想象的那样，发一个包出去，就会在对方接下整个包，很多时候会分几次才能接收下来整个完整的包。我们再看一个包：

```
0x0000   10 20 00 80 24 00 00 00-D9 54 DF 77 01 00 00 00 ..€$...ÙTßw....
0x0010   01 00 00 00 00 00 00 00-03 00 00 00 00 00 00 00 ................
0x0020   03 00 00 00 01 00 00 00-01 00 00 00 17 71 40 00 .............q@.
```

这两个包有什么相同之处，就是包头开始的8个字节，是不是可以看出来一个包的整体结构了，注意网络数据格式转换为机器数据格式的话要反转一下，MSDN上可以找到类似这样的函数ntohl、ntohs、htonl和htons，所以我们在看数据时也要把字倒转一下。

如果你了解的话，0x80是作为一个回应包的标识，这样我们就可以看到两个命令字：

```
80 00 00 02  十六进制表示：0x80000002
80 00 20 10                 0x80002010
```

这两个命令都不同，可以看到，协议上并没有统计的应用包头标志，直接从命令字开始了。有了上面的概念，接下来的自然就是长度，因为第一个包不完整，我们看第二个包，一般长度用两个字节就可以了，但会有用四个字节的，也就是一个int类型的值，第二个包的长度不难看到就是0x24，假设长度使用四个字节，那么内容就应该是D9 54....71 40 00，0x24=36，我们计算一下，刚好就是这个长度，呵呵，到现在你和我一样，了解包的结构形式了，我们正在一步步的往成功的方向去了……

到了这一步，大家也差不多可以动手自己去分析一下协议了，因为某些原因，我在这里就不再往下分析各个命令字的具体作用了，不过可以告诉大家0x80000002是用户登录后服务器返回来的所有玩家信息，注意一下，我们在注册用户时，最长的登录名称不能超过19个字节，所以包就加上一个结束符，成为20个字节。

上面的分析只是一个引子，由于本人水平有限，有什么错误的地方请大家指出，让我们共同学习。

外挂开发大揭密

▶撰文 / 蓝色天蝎

如果我没记错的话,第一个图形MUD应该就是UO(网络创世纪)了,那时候就有同学寄光盘过来,问我能不能开发个UO的外挂,那也是我第一次听到“外挂”这个说法。到了去年,网络游戏的赚钱效应开始广为人知,于是便造成了目前网络游戏异常火爆的局面,有人甚至把网络游戏称为中国IT业的救命稻草,各大门户网站也开始进军网络游戏市场,同时,一直伴随着网络游戏成长起来的产物,也就是本文的主角－外挂,便也浮出水面。当然,主流网络媒体对外挂都是一片喊打声,但外挂到底是好是坏?这个问题我觉得应当由玩家来裁决,而不是游戏运营商,也不是外挂开发者,因为他们都有利益在其中,自然无法说出客观的话来。当然,本文不是要去讨论这个容易引起争论的问题,本文的目的只是去探讨外挂的技术原理,试着从技术角度去揭开外挂开发的神秘面纱。

外挂的定义

外挂到底该如何去定义,什么样的软件叫做外挂?目前,这个问题还没有哪位IT权威人士给我们一个确切的说法,若按某网络游戏运营商在打击外挂时所给的定义,就是未经授权的、第三方开发的、对某游戏软件的功能进行修改、扩充的程序。其实,即使你不玩网络游戏,你也会经常接触到外挂程序,比如以前Win95刚出来的时候,市面上出现的中文之星这类中文平台,就是通过拦截Windows操作系统的API去实现自动翻译、内码转换等功能,他们也属于外挂,不过是Windows的外挂。类似还有金山词霸、各种病毒防火墙、网络防火墙,他们也都采用类似技术,时时刻刻监控着用户计算机内的内存和网络封包,去实现各种功能,他们同样没有得到被监控程序开发方的授权,按前面的定义,这些软件也算外挂,至于像FPE这类游戏修改工具就更不用说了。所以,我觉得外挂本身并不是一个贬义词,外挂就是在没有特定软件的源码的情况下,通过高超的技术手段,实现对此软件进行功能扩展的一种程序。至于外挂是否非法,应该看程序是否对使用者造成损害(比如木马、病毒、盗取帐号这些就是明显的破坏行为),而不应该针对某种技术手段去定义。本文所讲的外挂,专指那些针对网络游戏进行功能扩展的程序,因为我实在想不到其他名词来形容这类程序,这里只好沿用旧习,大家也都习惯那样去称呼。

外挂的历史

从第一款图形MUD网络创世纪诞生以来,网游外挂也就随之而生,之后出来的每款游戏,只要有玩家基础,都会有一些玩家兼程序高手开发出相应的外挂来,因为游戏总是没有办法设计到尽善尽美,不同的网络游戏运营商对游戏不合理之处的修改往往又有一定的滞后性,甚至一些游戏运营商为了延长玩家的在线时间,人为制造一些繁琐的操作放在游戏中,因此,外挂程序也就有了它出现的理由。存在即是合理的,有了需求,自然就会出现供应者。不过在刚开始,外挂大都是以免费形式传播的,当网络三国出来后,部分外挂开发者开始以共享软件的形式,通过收取注册费去卖外挂,于是,中国的程序员从此又发现了一个新的赚钱领域。网络三国之后,金庸、石器、传奇等游戏又创造了好几个开发外挂而一夜爆富的故事,遗憾的是在众多外挂开发者中,大部分人都没有把自己赚到的第一桶金用在正确的地方,这又是题外话了,暂且不表。

外挂开发的境界

我个人把外挂按技术实现手段,分为了4种。为什么标题称其为“境界”而不是“种类”呢?那是因为每种外挂的实现难度是递进的,对开发者功力的要求也同样呈正比例上升.

第一层境界:模拟键盘、鼠标操作,“蜻蜓点水”

这种可以说是最简单的外挂了,最初出现的外挂很多都属于这种境界,要写这种程序无非就是摆弄几个

Windows的键盘鼠标相关APIs。由于以前网络游戏的界面按钮位置大都是固定的,所以这些外挂用起来还是满有效的,不过现在很多新的网络游戏为了防止这种外挂,界面位置会随机出现,所以这种外挂的用武之地也越来越小。此类软件的杰出代表:按键精灵。

第二层境界:强行修改游戏数据及代码,“霸王硬上弓”

这种外挂目前最为普遍,一般都是通过将自己的程序注入到游戏进程,然后修改游戏的数据和代码来实现各种功能。现在网上有很多人说这招对网络游戏已经没用了,其实不然,原因只是他们的功力不够。在将自己的程序注入目标程序之后,几乎就可以为所欲为,通过逆向分析游戏程序,甚至可以直接在游戏中调用游戏相关的子程序段来实现任何想要实现的功能(逆向分析出相关函数的功能、参数以及地址之后,定义函数指针指向该函数的地址,接下来的调用就和调用普通函数没任何差别了)。不过,这种方式的缺点就是工作量太大,游戏一旦更新,就得重新分析程序,找到大量相关地址(函数地址、数据地址),这些都对外挂作者的逆向分析以及汇编功力要求很高,而且外挂功能越多,外挂作者更新一次就越吐血,要更新此类程序需要有极大的动力才行(金钱?)。此类软件的代表:传奇的大部分外挂、金庸早期的外挂(极差、神行)以及奇迹的大部分外挂。

第三层境界:拦截游戏封包,“四两拨千斤”

目前的网络游戏都是瘦客户端,玩家发送相关操作命令给服务器运算,服务器运算完后把结果返回给客户端显示,之间都是把数据打包通过TCP/IP协议来传输,只要拦截到这些数据,进行解密分析之后,通过修改数据包,再进行发送,就可以实现大部分功能。拦截数据包的方法有很多种,下面列举常用的几种:

1. 代理服务器形式:把游戏连接服务器的地址修改为自己写的代理服务器程序的地址以及监听端口,代理服务器会对封包进行过滤和修改,之后再转发给真正的服务器端,服务器端返回的信息也是以同样方式处理。
2. API HOOK:基于Windows的游戏都要通过Winsock APIs来发送和接收数据,所以只要把相关函数(Send、Recv)拦截,就可以实现封包的修改和伪造。
3. SPI:使用SPI可以在比API HOOK更低一层的层次上拦截相关Winsock APIs。不过由于其特性,它更适合用在防火墙类的软件中,用来做网游外挂反而不大适合,因为它需要对注册表进行修改,而且对系统的影响是全局的,稍有不慎,就可能造成用户网络访问出错,而且外挂使用者一般都喜欢绿色软件,不喜欢那些对注册表动手动脚的程序。

拦截数据包只是最基本的一步,做到这步是不够的,因为游戏的封包一般都有经过加密,而且你还需要解开游戏的一些秘密(相关的数据结构、物品地图格式什么的),此时逆向分析技术又派上用场了。这部分和第二层境界的要求是一样的,所以做外挂最关键的还是看汇编功力。不过,采用这种方式,软件更新起来就要比第二种境界轻松许多,只要游戏的通信协议格式不变,封包加密算法不变,基本上都不用什么更新,要添加功能也更容易。此类软件代表:金庸代理等。

第四层境界:脱离游戏,“任我驰骋”

在达到了第三境界,在你对游戏的通信协议了如指掌以后,自然会有一个念头:为什么我不自己开发一个客户端呢?只要封包格式兼容游戏,不用游戏客户端照样也能和服务器通讯,实现想要的功能,于是,脱离游戏独立运行的机器人程序就诞生了。这层境界又是建立在第三层之上,第三层境界要求你熟悉Winsock编程,有深厚的汇编功力,到了第四层,就要加上深厚的编程功力了,你不但要对数据结构和算法很熟悉,还要对面向对象编程、软件工程都有了解,因为写一个取代游戏客户端的程序代码量很大,一般都在两万行以上,其中涉及到的知识已经和开发一个游戏所需的知识有很多重合之处(除了图形编程方面不要求),像数据结构和算法(图、树、深度优先广度优先这些)、脚本、寻路以及其他许多AI技术都要用到,同时还需要利用上面第三层境界的技术写一个自己的封包查看器,这样才能开始分析利用封包。编写此类软件的工作量很大,一个人完成已经有些吃力,所以得开始采用建模分析和团队开发。此类软件代表:网三工作狂、网金也疯狂、魔力挂机MM等。

第五层境界:??????

不是说只有四层,怎么又多出来一个?其实这是我前一阵忽然冒出来的想法,在写完机器人程序之后,忽然想到既然对游戏的通讯协议了如指掌,自己也已经写

了一个客户端,为何我不自己开发一个服务器端,封包格式兼容游戏,这样不就可以开设私服了吗？于是,我便兴冲冲动手搞了一个端口服务器模型(纯粹一时冲动,所以也没做任何可行性分析),设计了一个服务器集群架构草图,完成之后继续下一步时却发现这可不是普通的工程,以个人之力几乎是不可完成的任务,因为游戏的逻辑太复杂,要实现和游戏一摸一样的逻辑的话,再加上地图解密、游戏物品、图形ID解密,还有服务器负载均衡、架构设计……给我一个五个人的团队,或许我可以做到……我有五个人的团队吗？没有,所以,我只好把它当作一个设想摆在一边了。如果哪位看官对此有兴趣,可以来找我,我们也来搞个公开源码的服务器模拟器协同开发小组…☺

直面外挂开发

在大概介绍完外挂的种类之后,我们终于可以转入正题,具体探讨各种外挂的实现了。由于篇幅的关系,这里我将集中讨论第二层和第三层这两种最常见的外挂实现方式,现在市面上的外挂有90%属于这两种。第一种太容易,程序员翻翻平台SDK与鼠标键盘相关的APIs就大概能做出来了,而第四种又太复杂,说起来恐怕几天都说不完。其实市面上很多游戏的外挂都是第二和第三两种方法的并用,有些功能若单单拦截封包是没办法实现的,你不得不修改游戏代码才可能(比如传奇的显示血条、雷达功能)

外挂开发初步，注入目标进程

不管是第二种还是第三种方法,这一步都是必须的,你先得注入目标进程,才可以为所欲为。注入其他进程有好几种方法，各自的优缺点在Jeffrey Richter写的《Windows核心编程》一书里已经有过经典权威的论述,这本书相信是每个Windows程序员案头必备的,所以我就不多废话了。在下面的示例中,我采用的是最常用的SetWindowsHookEx这种方法,也就是消息钩子。和其他地方介绍不一样的是,为了实现底层逻辑和应用逻辑分开,在本例子中我使用了两个DLL文件,一个专门负责SetWindowsHookEx,挂勾成功后,再在运行于目标线程中的GetMsgProc回调函数中加载真正的应用DLL,然后用UnhookWindowsHookEx卸载自己。真正对游戏进行功能扩充的是在GetMsgProc回调函数中被加载的应用DLL。这样做有什么好处呢？因为对游戏功能扩充的应用DLL一般都很复杂,而用钩子注入的DLL如果直接处理这些逻辑的话,一旦出错就容易影响整个系统的稳定(主要针对Win9X/ME),所以专门弄一个DLL负责注入,注入成功之后再卸载挂钩,就有助于程序的稳定性。下面是负责注入的DLL的核心代码:

```
// 这个单元必须用VC编译
// vcin.cpp:负责注入应用DLL的DLL文件

#include "stdafx.h"
#include "vcin.h"
const char DllPath[]="d:\\myprog\\vcin\\call\\test.dll";
// 要注入的应用DLL模块位置

#pragma data_seg("Shared")
HHOOK g_hhook = NULL;
HINSTANCE g_hinstDll = NULL;
#pragma data_seg()

#pragma comment(linker, "/section:Shared,rws")
HINSTANCE g_shinstDll = NULL;

//////////////////////////////////////////////////////
BOOL APIENTRY DllMain(HINSTANCE hinstDll,DWORD fdwReason,PVOID fImpLoad)
{
    switch (fdwReason)
    {
            case DLL_PROCESS_ATTACH:
                    g_shinstDll = hinstDll;
                    break;
            case DLL_THREAD_ATTACH:
                    break;
            case DLL_THREAD_DETACH:
                    break;
            case DLL_PROCESS_DETACH:
                    break;
     }
    return TRUE;
}

//////////////////////////////////////////////////////
void LoadUserDll()
{
    if (!g_hinstDll)
    {
            g_hinstDll = LoadLibrary(DllPath);
            //加载成功,卸载钩子
            if (g_hinstDll)
            {
                    SetMsgHook(0);
            }
    }
    return;
}
//////////////////////////////////////////////////////
LRESULT WINAPI GetMsgProc(int nCode, WPARAM wParam,LPARAM lParam)
{
   static BOOL fFirstTime = TRUE;
   if (fFirstTime)  // 第一次运行,加载主功能DLL
    {
```

```
        fFirstTime = FALSE;
        LoadUserDll();
      }
    return(CallNextHookEx(g_hhook,nCode,wParam, lParam));
  }

  ////////////////////////////////////////////////////
  // 设置和取消钩子的函数,dwThreadId=0表示要取消钩子
  BOOL WINAPI  SetMsgHook(DWORD dwThreadId)
  {
    BOOL fOk = FALSE;
    if (dwThreadId != 0)
    {
       if (g_hhook == NULL)
       {
              // 安装钩子
              g_hhook = SetWindowsHookEx(WH_GETMESSAGE,
GetMsgProc, g_shinstDll, dwThreadId);
              fOk = (g_hhook != NULL);
              if (fOk)
              {
              // 向目标程序发送一条消息,激活GetMsgProc
                fOk=PostThreadMessage(dwThreadId,WM_NULL,0,0);
              }
       }
       else
       {
              fOk = true;
       }
    }
    else
    {
       // 钩子是否已经安装好
       if (g_hhook != NULL)
       {
              fOk = UnhookWindowsHookEx(g_hhook);
              g_hhook = NULL;
       }
       else
       {
              fOk = false;
       }
    }
    return fOk;
  }
  // vcin.h
  #define VCIN_API extern "C" __declspec(dllexport)
  VCIN_API BOOL WINAPI SetMsgHook(DWORD dwThreadId);
```

此外,还要有个主程序调用上面编译成的DLL中的SetMsgHook函数来实现功能的激活,一般这个主程序就是商业外挂中常见的登录窗口,登陆成功后就执行SetMsgHook, 加载运行SetMsgHook的函数实现:

```
  typedef BOOL WINAPI (*MYHOOKFUNC)(DWORD dwThreadId);
  ////////////////////////////////////////////////////
  void __fastcall TForm1::Button1Click(TObject *Sender)
  {
    HINSTANCE hDLL;
    String DllPath;
    HWND TagWindow;
    DWORD TagThreadId;
    DllPath = ExtractFilePath(Application->ExeName)+"vcin.dll";
    hDLL = LoadLibrary(DllPath.c_str());
    if (hDLL)
    {
       Hookfunc=(MYHOOKFUNC)GetProcAddress(hDLL,"_SetMsgHook@4");
      if (Hookfunc)
      {
        TagWindow=FindWindow("游戏的窗口类名",NULL);
  // 也可以查找游戏的窗口标题
        TagThreadId = GetWindowThreadProcessId(TagWindow,NULL);
        if (TagThreadId) Hookfunc(TagThreadId);
      }
    }
  }
```

解释一下上面的代码,vcin.cpp主要是用来完成设置钩子、加载应用DLL这些任务。在SetMsgHook中,程序根据传入的线程ID,对此线程进行挂钩,挂钩完成后,该线程第一次调用GetMessage时,就会先运行GetMsgProc这个回调函数,GetMsgProc里面的代码已经是在目标线程里面运行的了,所以这里我们可以完成加载真正的应用DLL的动作。在确认应用DLL被加载成功之后,任务完成, 就可以把自己卸载掉了。SetMsgHook按extern"C"方式导出,这样可以供Delphi或BCB等编译器调用。此处要注意的是,g_hhook因为需要跨进程访问,所以利用#pragma data_seg("Shared")的VC编译器功能进行支持,如果要将vcin.cpp在BCB下面进行编译,得用内存映射文件来实现全局共享变量g_hhook(主要针对Win9x/Me,在Win2000下系统会自己维护钩子的句柄)。在调用的主程序示例中,通过查找游戏窗口,从而获得游戏主线程的线程ID,然后就可以调用负责注入DLL的挂钩函数,是不是很简单呢?此处还有一个问题,就是游戏窗口的类名如何去获得?这个问题更简单,我们只需要反编译游戏的窗口进行分析,具体的分析技巧我会在后面讲到,此外,你也可以用SPY++等工具进行查看。

好了,做完了第一步,我们还得提供一个应用DLL的示例,应用DLL就是个你可以自由发挥的海阔天空的地方,因为是在游戏的主线程中运行,所以你可以像主人一样访问游戏进程中的任何资源,执行游戏的任何一个函数(当然你要知道函数的调用方式以及参数)。同时,你也可以用SetWindowLong函数去子类化游戏的主消息函数处理过程,这样就可以掌握整个程序的消息控制权。下面的示例代码很简单,就是在应用DLL加载的时候,创建一个以游戏主窗口为父窗口的子窗口。过去经常在论坛上看到有人问如何在游戏中显示自己的窗口,其实

说破不值钱，就这么简单：

```
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason,void*lpReserved)
{
HWND TagWindow;
switch (reason)
{
case DLL_PROCESS_ATTACH:
        //DLL第一次映射到进程的地址空间的时候
   TagWindow= FindWindow("游戏的窗口类名",NULL);
        //也可以查找游戏的窗口标题
    if (TagWindow)
    {
        Application->Handle= TagWindow;
   Application->CreateForm(__classid(TForm2), &Form2);
        //VCL创建窗口的代码
//显示窗口,由于窗口在游戏进程中,父窗口又是游戏主窗口,所以可以在游戏中显示与隐藏
    Form2->Show();
}
    break;
     case DLL_THREAD_ATTACH:
    break;
     case DLL_THREAD_DETACH:
         break;
case DLL_PROCESS_DETACH:
         break;
}
return TRUE;
}
```

上面的程序虽然简单，但基本就是一个外挂程序的主要框架，同时也是第二和第三种境界外挂共同的基础。两种外挂的区分就在于应用DLL的编写方法，第二种境界把ReadProcessMemory、WriteProcessMemory和函数指针发挥到极至，随意修改内存中的数据和代码，调用游戏程序的相关函数，其基本方法在《程序员》2001年2期的《跨越内存禁区，修改游戏数据》中有比较详细的论述，这里就省略了。那么第三种境界呢？则是需要你了解Winsock编程，还必须把游戏的Send和Recv函数挂上，从而紧紧把握住游戏数据通讯的七寸咽喉，挂钩API的方法在程序员2001年2期的《编写Win32 API钩子》一文有详细讲述，这里再次略过。下面的重点是放在少有人提及，如何对游戏进行逆向分析这部分内容上，没有逆向分析这一步，即使你搭成了外挂程序的架子也全然没用，外挂具体功能实现全部要靠分析游戏程序来实现。

外挂开发的主活，分析程序和封包

分析程序和封包，需要借助一些静态反编译工具，比如W32Dasm、IDA，还有动态调试软件SoftIce或者TRW2000等。静态反编译工具主要用来分析代码和静态数据，动态调试软件则可以用来获得一些动态的数据，观察程序的运行流程，这两类工具在逆向分析中都是必须的。下面，我就是简单介绍几个常见的分析方法，同时谈谈一个著名网络游戏(匿名)的解密封包程序的分析和模拟过程，这些内容都假设读者对静态反编译工具以及动态调试软件有一定使用经验。

分析示例一：寻找游戏窗口的类名、窗口标题名以及窗口消息处理函数

意义：有了窗口类名和标题，我们可以轻易找到窗口句柄，才可以用各种Windows API对它进行操作。找到了消息处理函数，我们就可以用SetWindowLong函数去子类化该窗口，在游戏代码高一级的地方，优先处理我们感兴趣的消息。

思路：VC写的游戏比较容易分析，因为VC程序都是标准的SDK风格，创建窗口的步骤简单固定，当然被MFC封装的程序要麻烦些，不过有哪个游戏会用MFC来写呢？所以，我们若要分析VC写的游戏，先找到CreateWindowEx函数的调用，就可以知道开头那两个我们感兴趣的东西了：

```
HWND CreateWindowEx(
  DWORD dwExStyle,          //extended window style
  LPCTSTR lpClassName,      //pointer to registered class name
  LPCTSTR lpWindowName,     //pointer to window name
  DWORD dwStyle,            //window style
   int x,                   //horizontal position of window
   int y,                   //vertical position of window
   int nWidth,              //window width
   int nHeight,             //window height
  HWND hWndParent,          //handle to parent or owner window
  HMENU hMenu,              //handle to menu, or child-
                              window identifier
  HINSTANCE hInstance,      //handle to application instance
  LPVOID lpParam            //pointer to window-creation data
);
```

相应汇编代码一般是这样的：

```
push    0               ; lpParam
push    edx             ; hInstance
mov     edx, [esp+20h+Rect.top]
push    0               ; hMenu
sub     eax, edx
mov     edx, ds:Y
push    0               ; hWndParent
push    eax             ; nHeight
mov     eax, [esp+2Ch+Rect.left]
sub     ecx, eax
mov     eax, ds:X
push    ecx             ; nWidth
```

```
push    edx             ; Y
push    eax             ; X
loc_4372F0:
push    esi             ; dwStyle
push    offset aWindowsName ; lpWindowName,这里和下面就
                                是我们要的东西
push    offset aWindowsName ; lpClassName
push    0               ; dwExStyle
call    CreateWindowExA
mov     ds:hWnd, eax
```

要查找窗口处理消息回调函数，要在ATOM RegisterClass(CONST WNDCLASS *lpWndClass)上着手：

```
typedef struct _WNDCLASS {    // wc
    UINT    style;
    WNDPROC lpfnWndProc;
    int     cbClsExtra;
    int     cbWndExtra;
    HANDLE  hInstance;
    HICON   hIcon;
    HCURSOR hCursor;
    HBRUSH  hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
} WNDCLASS;
```

汇编代码一般是这样：

```
push    67h             ; lpIconName
push    esi             ; hInstance
mov     ds:stru_807E60.style, 1008h
; 我们要的东西在下面, loc_437590就是窗口消息函数的实际地址
mov     ds:stru_807E60.lpfnWndProc, offset loc_437590
mov     ds:stru_807E60.cbClsExtra, 0
mov     ds:stru_807E60.cbWndExtra, 0
mov     ds:stru_807E60.hInstance, esi
call    LoadIconA
push    65h             ; lpCursorName
push    esi             ; hInstance
mov     ds:stru_807E60.hIcon, eax
call    LoadCursorA
push    4               ; int
mov     ds:stru_807E60.hCursor, eax
call    GetStockObject
push    offset stru_807E60      ; lpWndClass
mov     ds:stru_807E60.hbrBackground, eax
mov     ds:stru_807E60.lpszMenuName, 0
mov     ds:stru_807E60.lpszClassName,offset aDo_ii ;"窗口类名"
call    RegisterClassA
```

如果是用Delphi或BCB写程序，情况会麻烦一点，因为VCL层层封装，但如果利DeDe这个工具的话，情况就完全改观。DeDe可以让你获得不亚于Delphi或BCB集成开发环境的可读性(当然对汇编要熟)，结果反而是分析用Delphi或BCB写的游戏程序来得更轻松，正应了“工欲善其事，必先利其器”这句老话，合理的应用工具，会让你事半功倍！

分析示例二：现在，我们真枪实弹地拿一个网络游戏的解密封包函数来做分析。此处分析的解密函数只针对该游戏某一个版本，不一定适用在最新版本上，据说此游戏不久就会停止运营，这也是选择它的一个原因。

意义：我们知道，在做第三种境界的外挂时，封包的加密解密是我们必须面对的一个问题，只要搞定了这个问题，一个外挂就呼之欲出。什么！？你说游戏没有加密封包？呵呵，这种美事应该不会经常发生，据我所知，目前只有RO(仙境传说)采用的明码包，结果当然是外挂满天飞。

思路：首先，我们要找到游戏相应的加解密函数。按照经验，一般都是在Send数据之前进行加密，Recv数据之后进行解密，所以第一步要先定位Send和Recv函数，这个和定位其他Windows API函数一样容易。完成定位之后，就要观察一下它们附近的代码，如果是直接进行加解密处理(最原始方式)，那你可要谢天谢地，因为目前大部分游戏都不是这么简单，很多游戏都采用了队列方式，也就是在Recv接收到数据之后，不直接进行处理，而是根据通讯协议拆好封包后，将单个封包(未解密)扔到队列中，然后用一个定时器处理封包队列(当然时间间隔很短)。发送同理，一般是接收到用户操作指令后，将命令打包后扔到发送队列，然后用定时器进行发送，基于这种情况，查找就会麻烦些，此时要重点关注timeGetTime或者GetTickCount函数附近的代码，此时需要的是经验、耐心以及SoftIce的辅助。下面就是我靠经验和耐心，找到的一段解密代码(当然，为了便于查看，函数名已经被我改过)：

```
decrypt     proc near       ; CODE XREF: sub_45B22C+48 p
                            ; CODE:0045B328 p ...
var_28      = dword ptr -28h        ; 局部变量
var_23      = byte ptr -23h
var_22      = byte ptr -22h
var_21      = byte ptr -21h
var_20      = dword ptr -20h
var_1C      = dword ptr -1Ch
var_18      = dword ptr -18h
var_14      = dword ptr -14h
var_10      = dword ptr -10h
var_C       = dword ptr -0Ch
var_8       = dword ptr -8
var_4       = dword ptr -4

            push    ebp
            mov     ebp, esp
            add     esp, 0FFFFFFD8h
            push    ebx
            push    esi
```

```
            push    edi
            mov     [ebp+var_C], ecx        ; MaxLen
            mov     [ebp+var_8],edx ;解密数据存放地址指针
            mov     [ebp+var_4], eax ;待解密数据存放地址指针
            mov     eax, [ebp+var_4]
            call    @@LStrAddRef    ;__linkproc__ LStrAddRef
            xor     eax, eax
            push    ebp
            push    offset aSVI_LxN@ ;
            push    dword ptr fs:[eax]
            mov     fs:[eax], esp
            xor     eax, eax
            push    ebp
            push    offset loc_45B1FB
            push    dword ptr fs:[eax] ; 异常保护
            mov     fs:[eax], esp
            mov     eax, [ebp+var_4]
            call    unknown_libname_7;取得数据长度(Delphi String形式)
            mov     [ebp+var_14], eax ; 初始化各个局部变量
            mov     [ebp+var_18], 2
            xor     eax, eax
            mov     [ebp+var_1C], eax
            xor     eax, eax
            mov     [ebp+var_20], eax
            mov     [ebp+var_22], 0
            mov     eax, [ebp+var_14]
            test    eax, eax
            jle     loc_45B1E7
            mov     [ebp+var_28], eax
            mov     [ebp+var_10], 1
loc_45B13B:                             ; 解密过程
            mov     eax, [ebp+var_4]
            mov     edx, [ebp+var_10]
            movzx   eax, byte ptr [eax+edx-1]
            sub     eax, 3Ch
            js      short loc_45B15C
            mov     eax, [ebp+var_4]
            mov     edx, [ebp+var_10]
            mov     al, [eax+edx-1]
            sub     al, 3Ch
            mov     [ebp+var_21], al
            jmp     short loc_45B166
;---------------------------------
loc_45B15C:                 ; CODE XREF: decrypt+79 j
            xor     eax, eax
            mov     [ebp+var_20], eax
            jmp     loc_45B1E7
;---------------------------------
loc_45B166:                 ; CODE XREF: decrypt+8A j
            mov     eax, [ebp+var_20]
            cmp     eax, [ebp+var_C]
            jge     short loc_45B1E7
            mov     eax, [ebp+var_1C]
            add     eax, 6
            cmp     eax, 8
            jl      short loc_45B1BC
            mov     ecx, 6
            sub     ecx, [ebp+var_18]
            mov     al, [ebp+var_21]
            and     al, 3Fh
            and     eax, 0FFh
            shr     eax, cl
            or      al, [ebp+var_22]
            mov     [ebp+var_23], al
            mov     eax, [ebp+var_8]
            mov     edx, [ebp+var_20]
            mov     cl, [ebp+var_23]
            mov     [eax+edx], cl
            inc     [ebp+var_20]
            xor     eax, eax
            mov     [ebp+var_1C], eax
            cmp     [ebp+var_18], 6
            jge     short loc_45B1B3
            add     [ebp+var_18], 2
            jmp     short loc_45B1BC
;---------------------------------
loc_45B1B3:                 ; CODE XREF: decrypt+DB j
            mov     [ebp+var_18], 2
            jmp     short loc_45B1DB
;---------------------------------
loc_45B1BC:         ; CODE XREF: decrypt+A7 j decrypt+E1 j
            mov     ecx, [ebp+var_18]
            mov     al, [ebp+var_21]
            shl     al, cl
            mov     edx, [ebp+var_18]
            and     al, ds:byte_4A2CBE[edx]
            mov     [ebp+var_22], al
            mov     eax, 8
            sub     eax, [ebp+var_18]
            add     [ebp+var_1C], eax
loc_45B1DB:             ; CODE XREF: decrypt+EA j
            inc     [ebp+var_10]
            dec     [ebp+var_28]
            jnz     loc_45B13B
loc_45B1E7:   ; CODE XREF: decrypt+5B j decrypt+91 j ...
            mov     eax, [ebp+var_8]
            mov     edx, [ebp+var_20]
            mov     byte ptr [eax+edx], 0
            xor     eax, eax
            pop     edx
            pop     ecx
            pop     ecx
            mov     fs:[eax], edx
            jmp     short loc_45B205
;---------------------------------
loc_45B1FB:                 ; DATA XREF: decrypt+2B o
            jmp     @@HandleAnyException
;---------------------------------
loc_45B205:                 ; CODE XREF: decrypt+129 j
            xor     eax, eax
            pop     edx
            pop     ecx
            pop     ecx
            mov     fs:[eax], edx
            push    45B222h
            lea     eax, [ebp+var_4]
            call    @@LStrClr  ; __linkproc__ LStrClr
            retn
decrypt     endp ; sp = -3Ch
```

由于这个游戏是用Delphi写的,所以用到了一些Object Pascal库函数和一些VCL的东西,传入数据也是String类型(把String当作动态数组用),不过这些都没关系,我们只

要当中的解码部分就可以了。现在首先来弄清楚它的入口参数，用IDA往外跳一层(X键)，我们可以看到：

```
mov    ecx, 400h
mov    edx, ds:dword_4F7D6C
mo                                  v    eax, [ebp+var_4]
call    decrypt
lea    edx, [ebp+var_14]
```

有了上面的调用代码，我们大概就可以知道解密函数是采用fastcall方式，用寄存器直接传递3个参数，但是其中两个参数的含义我们无法知道，因为是动态内存地址，这时候就得请出SoftIce，在mov ecx，400h处下个断点，进入游戏，连接服务器，等待数据从服务器返回的时候，断点中断，观察一番后发现EDX是解密数据存放地址指针，EAX是待解密数据存放地址指针，ECX固定，用途我们得在函数内部进行分析才能知道。在decrypt函数的cmp eax，[ebp+var_C]这里我们可以知道，400h是解密后数据最大长度限制。OK，搞清楚了这些参数，现在就用高级语言把算法部分模拟出来，当然，我们也可以直接复制上面的汇编代码，修改一下嵌入自己的外挂程序，而且这样更省事。不过，为了弄清楚整个函数的流程，就多花点时间，其实也才100多行汇编代码，并不是很麻烦，除了理清循环和跳转逻辑关系，就是一句一句进行简单直译。翻译结果如下，一个C++的解密函数：

```
    int TMirProcessor::DecryptData(const String &DataBuf ,
LPBYTE DesBuf,DWORD MaxDesLen)
    {
       const char temp[10] = {0x40,0x0,0xFC,0xF8,0xF0,0xE0,
0xC0,0x8D,0x40,0x0};
        DWORD var_28;
        BYTE  var_23;
        BYTE  var_22;
        BYTE  var_21;
        DWORD var_20;
        DWORD var_1C;
        DWORD var_18;
        DWORD var_14;
    DWORD var_10;
    DWORD DataLen;
    DataLen  = DataBuf.Length();
        var_14 = DataLen;
        var_18 = 2;
        var_1C = 0;
        var_20 = 0;
        var_22 = 0;
        var_28 = DataLen;
        for (var_10 =1;var_10<DataLen;var_10++)
         {
            if (DataBuf[var_10-1]>=0x3C)
             {
                var_21 = DataBuf[var_10-1]-0x3C;
                if (var_20< MaxDesLen)
                  {
                     if ((var_1C+6)>=8)
                       {
                        var_23=(var_21 &0x3F & 0xFF)>>(6-var_18)| var_22;
                         DesBuf[var_20] = var_23;
                var_20++;
                var_1C = 0;
                if (var_18<6)
                {
                var_18+=2;
                var_22 = var_21<< var_18 & temp[var_18];
                var_1C += 8-var_18;
                }
                else
                {
                var_18=2;
                }
                }
                else
                {
                var_22 = var_21<< var_18 & temp[var_18];
                var_1C += 8-var_18;
                }
        }
        else
        {
                DesBuf[0] = 0;
                break;
        }
             }
            else
             {
               DesBuf[0] = 0;
               break;
             }
         }
        DesBuf[var_20] = 0;
         return var_20;
    }
```

因为是直译，所以代码有点臃肿累赘，不过经过N次调试之后，终于能正确进行封包解密了(用前面的知识，自己写一个封包截取程序吧，不要用什么WPE，那个太不好用了)，算下来从分析解密函数到翻译成高级语言并调试通过也才花了个把钟头时间。接下来就依样画葫芦，把加密函数也搞定，就能伪造封包发给服务器，一个外挂便基本成型，剩下要做的就是玩游戏，倾听玩家的功能要求，然后去分析封包，编写相应实现了……

尾 声

做外挂，说穿了，苦力活而已，在大部分时间里，你需要面对的只是枯燥的封包数据和汇编代码，根本无暇去领略编程所带来的那种快感，中国的程序员日子不好过呀！最后，如果哪位对本文内容有所异议，或者有什么新奇的想法和思路要和我共享、讨论，都欢迎来信到bluely@vip.163.com！(注明，本文有配套工程源码下载)

反外挂战略体系 浅析

▶撰文 / 冰·蝎

反外挂是一个很大的话题,涉及的层面也很广,单从技术细节上很难孤立去讲,由于前面蓝蝎兄已经剖析了外挂开发技术,在本文里,我就和大家简单谈谈国内反外挂的一些基本情况,旨在让大家对这个特殊领域有所了解。这里先要声明,由于阅历所限,我个人对外挂和反外挂除了技术兴趣外,没有任何倾向,本文也只是写给大家开开眼界。此外,我要特别感谢沈大哥、玉泉国际的研发李千照、网星副总林先生以及研发孙先生为我提供的信息。

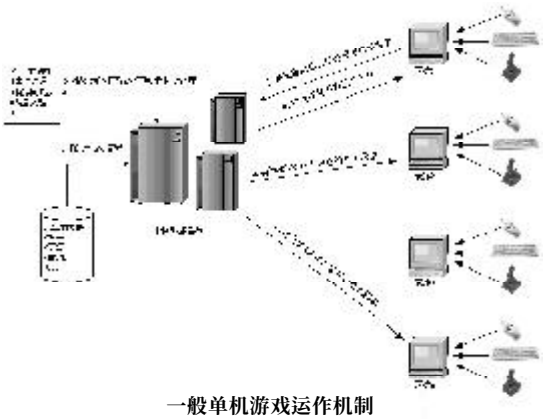
好了,回归正题,从策略上讲,反外挂也就三种方式,一是依靠立法,二是依靠技术,最后是依靠玩家反馈。(注明:本文所讲的"外挂"不包括游戏机器人技术,反外挂不会也无法涉及这类情况)

就目前国情来看,立法和普法应该是杜绝外挂的根本,技术上讲,没有做不出来的外挂,也没有破不了的外挂,所以,想单从技术上去防外挂根本不可能,这就像病毒程式和杀病毒程式一样,这些东西总是成对出现,哪一方都不可能长时间占得绝对优势。同理,有了新的网游就会有新的外挂,有了新的加密算法和通信协议就会有新的破解方法,这一过程永远都不会停止,原因只有一个,这些东西都有着很强的目标和针对性。还记得什么样的人最难对付吗?是无欲无求的人,很简单的逆反逻辑。

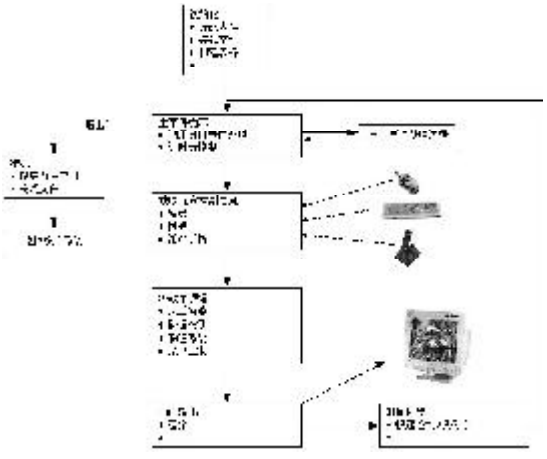
关于外挂国内还没有专门的立法,只知道在今年七月,中国出版工作者协会在新闻出版总署有关部门的直接提议下,由几个著名游戏公司积极倡议,成立了一个游戏工作委员会。由于对这个游戏工作委员会的具体情况我不太清楚,这里就不进一步讨论,下面来看看技术方面的反外挂策略。

反外挂技术策略包括正反两方面,所谓"正"就是做好游戏本身的防御工作,这一方面涉及游戏开发本身采用的设计和架构,另一方面则是使用第三方防外挂软件。

由于每个游戏的具体情况不同,比如有些游戏是即时的,有些游戏是回合的,这些都会影响游戏的架构,而对不同架构的游戏,开发外挂或者反外挂的具体方式就会有很大差异。很多时候网游发生问题并不是封包设计的不足,而是游戏本身的缺陷所导致。分析目前的一些外挂,可以看到产生外挂最大的原因是服务器直接接受了客户端的资料。例如一些修改本地数据的外挂,在游戏本地数据被修改后,由于游戏设计的问题,这些数据被上传至服务器并存盘,便造成了永久性的修改。反之,如果本地数据只做显示用,真正的数据保存在服务器上,这种本地修改外挂自然就会无效。从绝对角度讲,只要把所有计算都放在服务器上进行,客户端只做图形显示,就不会有外挂的问题,除非你直接去攻击服务器。但实际上为了保证游戏的流畅性,节省服务器端资源,设计者会将一部分计算放到客户端进行,而正是这些客户端计算成了外挂的突破口。所以,当有些计算不得不在客



一般单机游戏运作机制



一般单机游戏运作机制

户端上进行时,就要采取一些屏蔽措施,比如网络数据加密,在网络连接建立时,双方约定一种加密方法,数据经过加密后再传输,这样即使被外挂截获到数据包,要对其进行分析也会有相当的困难。此外就是内存数据加密,一种方法是编写一个加密模块,游戏中数据的存取都通过这个模块进行,另一种方法是对重要数据产生一条校验码,每次修改数据后同时更新校验码,这样在某个时刻若发现校验码与源数据不符,就可以认为有第三方软件修改了游戏数据。

当然,上述屏蔽措施仍然存在度的问题。都知道,越高级的加密技术,相应的解密机制自然也就越复杂,于是,服务器端的负担也就越繁重,这就像我们在进行软件设计中,获取灵活性和可扩展性的同时,往往就要以性能为代价。所以,具体设计也还是要看服务器情况,否则上线人数一多,服务器就会垮掉。此外,这类问题也没法简单通过添加CPU去解决,因为服务器端的数据库瓶颈是基本固定的,无论你怎么添加CPU,数据库也只能跑那么快,毕竟硬盘是以机械速度运作。当然,你可以启用超级电脑,像以前就有人用工作站做游戏,一个房间只能放一台服务器,号称绝对没有外挂,所有运算数据都放在服务器端,由于设备非常高档,服务器就要一百万美金一台,而且没有维护,只保固半年,最多承载八千人,从运营成本考虑就完全不现实了。所以,从设计角度讲,好的游戏需要开发者花费大量时间去对客户端与服务器端的架构进行优化,避免留下漏洞被外挂利用。像韩国的<天堂>系列在这方面做得就比较出色,尽管运行了数年,有着大量的用户群,但只出现了一些简化操作的外挂,对游戏规则本身并没有很大的破坏性。

然而,开发者的精力也是有限的,若花在反外挂上面的时间太多,必然会影响游戏层的开发,甚至得不偿失,所以,一些游戏公司会直接购买第三方的反外挂软件,例如<奇迹>和<天堂>系列都使用了nProtect保护系统,<精灵>则采用了金山的防火墙。这些反外挂软件并不能100%确保屏蔽所有外挂,但它的好处是可以节约开发人员的时间,可以屏蔽大部分通用型外挂,可以增加外挂的开发难度。外挂开发者在做外挂之前,都需要用工具对游戏程式进行分析,这些反外挂软件增大了分析的难度,外挂自然就不容易开发出来。从这个趋势来看,开发反外挂软件在未来可能会类似开发杀毒软件一般成为一个产业,像现在韩国就已经有了专门研发反外挂软件的公司。

接下来就是反外挂技术策略中的"反"了。所谓"反"就是被动出击,比如现在网星就有六七位研发人员专门从事这方面工作,他们都是先拿到外挂程序,进行分析,分析出来之后,再针对它的漏洞或特点去进行反外挂。这是一个相互的过程,外挂是找游戏的漏洞,而他们则是找外挂的漏洞,外挂也是程序,自然也有漏洞。当然,在这场攻防战中,游戏运营商处于一个不利的位置,因为他们的服务器是透明公开的,都放在固定的IP段,所以一个游戏可能有七八个人在写保护,却有几千人在进行破解。

第三个策略,并且是当前国内游戏运营商比较依赖的反外挂途径,就是通过玩家反馈了。许多被骗玩家都会把问题公告出来,比如遇到帐号被别人用什么手段骗了这些不平的事情。此外,使用外挂的角色也会有其特别的地方,比如你发现一个角色一直在狂奔,和他对话又没有任何反应,其属性值又不断往上升,这种举动自然就有问题,被举报上去之后游戏公司就会对这个角色记录进行监控处理。

最后,我谈谈国外的情况。国外网游可以分为两个部分,欧美与亚洲。欧美的游戏产业已经比较成熟,开发人员经验丰富,从早期的《UO》到现在的《魔剑》等游戏,尽管结构非常复杂,但设计上并没有大的漏洞,一直没有恶性外挂出现,所以他们对外挂问题远不如国内这样敏感。而亚洲网游则以韩国游戏为代表,韩国游戏这几年突然崛起,抢占了大部分的网游市场,但由于发展时间较短,所以游戏是良莠不齐,有些游戏甚至可以直接用FPE修改,这样假设一个游戏有100级,设计者预计玩家需要花6个月才能满级,但由于恶性外挂的出现,可能玩家只要一个月就可以满级了,这样整个游戏的生命周期就被大大缩短,而代理那些游戏的国内商家自然便对外挂非常重视。

好了,到了本文该结尾的时候,但怎样去结尾?看着通篇的反字,不禁让我想起一个被遗忘却很关键的问题,那就是游戏的可玩性。关于这个话题我不想深究,因为我不想自找麻烦,但我相信读者你已经知道了我的意思,这里我真心希望国内在线游戏的品质能够提高,就如我认识的一个开发外挂的朋友所言:"要写外挂,先要对目标程序有所了解,而在体验其优缺点的过程中,如果开发商注重游戏的可玩性和交互性,为玩家想得很周到,在你玩了一两周后发觉这游戏还不错的话,那么建议你还是打消靠它做外挂赚钱的想法……"或许,游戏开发商们能够从中悟到些什么吧……

投稿信箱:tougao@csdn.net

网星高层外挂专访

▶记者 / 冰·蝎

网星史克威尔艾尼克斯网络科技有限公司(简称网星)是一家国内知名的网络游戏营运商,旗下有著名的网络游戏:魔力宝贝。今年五月,网星举办了以“绿色网游新世界”为主题的系列反外挂活动,随即掀起了国内反外挂高潮。《CSDN开发高手》也对网星负责人 林阜民 进行了一次采访。

问:最初你们为何想着要举办这个活动呢?

林:线上游戏,是一种社群的观念,在游戏企划所定的规则中游玩,每个玩家按照这种规则去公平地玩游戏。而使用外挂,就如在奥运马拉松比赛中,有人为了好成绩,使用激素一样,他个人的确可以拿到好的成绩,但却破坏了公平竞争,会影响别人。外挂让游戏的生命缩短,就如同激素让运动员的运动生涯受到影响一样。由于外挂存在,一个游戏好不好玩不再是企划和研发所决定,游戏怎么玩?是外挂帮你决定。外挂也使游戏存在欺骗性,比如一个物品值十万,外挂利用封包欺骗玩家,实际只给玩家五毛钱,但假封包却告诉玩家那就是十万,这样坏人拿到钱就跑掉,被害玩家却打电话到公司来:我的东西在游戏中被骗了,你要帮我找回来,你说我怎么处理?此外就是木马盗号问题。

所以,外挂是两方面的问题,一是程序上的外挂,二个是相关的网络电脑犯罪,就如病毒和木马,这种情况非常多见。像国外的玩家,对游戏规则都非常遵守,但放在亚洲情况就不一样了,像在国内,可能由于玩家过去玩单机版游戏改惯了,网游一进来,第一就是破坏数据包,然后分析其结构,接着就做外挂。所以,这次活动和新浪联手,就是希望把影响做大。你知道,国内游戏业一直不起来的最大原因就是盗版。现在,好不容易线上游戏比较不会有盗版,这样有了市场获益价值,人才就好进来,游戏质量进一步提高,产业自然会慢慢兴旺。然而,现在外挂来了,破坏了这个平衡,产业被扭曲,一切就可能如过去单机版游戏一般,还未成长就夭折。

这次搞绿色网游活动主要目的有三个:第一,让业界、玩家、社会能够重视这个产业。第二,让人们对游戏玩法有正当认识,因为线上游戏很容易反映国家、社会、人民的价值观,而现在看来已经有些不对了。第三,希望游戏要求一个高品质,许多经营线上游戏的公司进来的时候都是做代理,他们没有源码,防外挂很无力,而给韩国方面说也没用,因为他们不用外挂,他们不理解,最后结果就是姑息。这次活动就想促使业界不仅要做好自己的游戏,更要做好防外挂技术。

问:当你们对使用外挂的用户进行大规模封查的时候,是否想过这样会影响你们的运营收益?而又是什么让你们坚持去做的呢?

林:许多玩家在被删帐号之后,认为公司会把帐号里面没用的点数都收归公有,其实事实不是这样的,在我们公司查到有外挂帐号的时候,只会把有犯罪记录的外挂角色清除掉,其他没用外挂的角色以及物品、点数都是全数归还给玩家,只不过这个处理时间会长点,一般是七到十五天,但处理完之后帐号是一定会还给玩家的。当然,对于被查封帐号的玩家来说,他们是很难过的,他们会有莫大的压力,甚至来公司抗争,像这次就来了好几百人,但这是公司的原则啊!所以,这次查封活动的确给我们的营收带来了影响,在公司砍外挂那段时间,在线人数明显有掉,那么,为什么我们还坚持去做?因为我宁愿不要少数人,那些坏分子,也要照顾大多数玩家的利益和他们玩游戏的乐趣与公平,没有公平玩游戏就没什么意思,比如你玩半年的角色,还不如别人开外挂练一个晚上,这样玩家便会对游戏失去信心,因为游戏怎么玩已经不是游戏本身和游戏公司说了算,而是外挂说了算。

此外,外挂也是一山还比一山高,功能越来越强,已经侵入游戏核心,加上许多游戏公司技术力量有限,最后只能姑息,这样一个游戏很快就会完蛋,像在国外EQ、UO这些游戏的生命力长达十年,而放在国内可能两年就死掉,这样企划也很难继续做下去。所以,只有没有外挂,游戏才可以安稳运作,玩家才可以在游戏世界里安然徜徉,如果突然出来几个江洋大盗,奸淫掳掠,那么游戏就进入了一个乱世,严重影响玩家和运营商。所

以，对我们来讲，原则很简单，信念很坚定，就是遇神杀神，遇鬼杀鬼，管你什么外挂，管你多牛！为什么？因为他们对游戏的破坏，结果还是我来承担，我还不如用我亏损的钱去对付他们，就像美国这次杀了乌代，就从他房间里查到很多钱，他们等于用这些钱去买他的命，我宁愿这样。

问：在过去单机游戏中，玩家都喜欢用FPE、金山游侠这些工具去修改游戏，好像当时大家不觉得有什么？

林：那当然了，那时候的修改操作是发生在你自己一台电脑上，修改是你自己的事情，玩高兴了想怎么改都没人有权管你。但现在不一样了，在网游中你的游戏对象是人，别人搞走你的帐号，弄掉你的钱，然后再也不用原来那个犯罪帐号，这种现象经常出现，但对被害玩家来说这是一件很痛苦的事情。帐号是玩家的劳动所得，可能是练半年、练一年的成果，就如你有一本写了许多年的日记被别人偷走，你一定会去告他，因为这种行为对你造成了严重的侵害，或许你的日记在别人眼中是垃圾，但对你来说却是无价之宝啊！

问：从外挂出现历史这方面讲，你们是从什么时候发现使用外挂这种现象的呢？

林：一开始运营就有外挂了，最初是按键精灵，后来用的人多了，那些做外挂的人就开始把它们做成共享软件，最初一份一块钱，一个月下来，哇，五六万，比工资还高，好了，他们就认为这是一个可以投入的产业，又不用缴税，一两台电脑一个小房间就可以做，这样慢慢外挂行业就起来了。

问：在台湾及国外地区，针对外挂有明确的立法吗？

林：有的，台湾针对外挂有明确的立法，使用外挂判有期徒刑三年，罚金二十万，事实上他们当局也抓过好些人关起来。欧美许多地方也都对外挂进行了立法，把制作和使用外挂定义为网络犯罪，而不简单是侵犯制作权。这些国家大都由政府牵头，比如你帐号被外挂盗了，就可以去警察局进行立案。所以，这里也希望人们认识到做外挂和使用外挂是犯罪行为，对产业成长非常有害，长此以往，市场就会成为幻影，而这正是程序员毁掉了程序员。

问：对于用户在自己的计算机上使用的软件品种，网游运营商是否有权干预呢？

林：外挂虽然是软件，但它发送的封包是到我们的服务器，并在服务器这边产生结果，换句话说，其客户端不是个独立的程序，而只是一个发射端，结果都在我们服务器这边产生，这和单机版游戏情况是完全不同的。网络游戏是一个整体的架构，是一个服务社群，你不能为了自己个人的利益去影响别人的权益。其实像在游戏服务过程中，我们都有清楚告诉玩家：希望你不要使用不合法的外挂……因为做外挂事实上是违法的，首先，我的封包是我的制作权，外挂凭什么对我的封包进行拦截，而且使用外挂的客户端还可以发送我的封包，这样来说他们也可以模拟别人的信用卡帐号封包，在线进行购物哦？而且从他们客户端发送的封包，结果是在服务器端产生，这样带来的客户问题还要我们去处理，这明显就是违法嘛。此外，那些做外挂的公司他们也买点卡，但一方面他们并没有互联网经营许可，也不缴纳一分钱给工商，某方面来讲这就是一种经济犯罪。最后就是侵犯制作权，像我们的封包都有自己的制作权，是经过新闻出版署的人审批的。

问：用户由于使用了外挂而被删除ID是否合法，或者有无此方面条款依据？

林：这部分是比较模糊的了，但就像前面说的，我们只是对其使用外挂的角色进行永久扣留，把它关起来，而帐号中其他角色、物品、金钱都会保留。对我们来说，扣留这些角色也是要花成本的，因为也要在服务器上处理和存储这些信息，但我们这样做也就是为了除掉那些害群之马，就像以前有个外挂很霸道，发一个封包过来，屏幕上其他所有人都会死掉，然后外挂角色就去捡钱，这和抢劫有什么差别？

问：可否预言一下国内网游的外挂反外挂战争在未来的发展趋势？主要有哪些特点？

林：技术上讲，我们一定是可以破掉它们的，但短期来看会有一些问题，也会出现死灰复燃的情况。而最重要的就是国家的立法和一般人的态度，就像一个人随地吐痰，你问他为什么随地吐痰，他可能说他们那里人人都这样，于是，吐痰成为理所当然。然而，一旦吐痰成为人人鄙视的行为，国家立法对吐痰进行罚款，好了，人们就会知道吐痰是违法的，是一种错误的行为。这是一种态度了，也是法律面前每个人的义务。技术就像一把刀，你可以用来切菜，也可以用来杀人。（转13页）

2003修订版

关于学习C++和编程的50个观点

▶撰文 / 方舟

前言

很久很久以前,kingofark无心之中写了一篇关于学习C++和编程的发牢骚的文章。未曾料到的是,这篇文章被N个好心的网络同胞流传至今,以至于作者得意忘形的将文章简称为Ks50PV,在浅薄和浮躁中麻醉自己。

某个醉生梦死的深夜,kingofark关上电脑,屋内一片墨黑,突然想到了七夕的传说--牛郎织女早已灰飞烟灭,只有故事被流传下来,因为被流传而流传。

但是,一篇文章不应该因为被流传而流传,而应该因其价值而被流传。带着这样的冲动,kingofark又打开电脑.....

眼前给大家展现的,是经过kingofark重新审视,结合作者最新感受的修订版。其中有些条款彻底的更换或者修改了。这一方面反映了国内图书市场近年来发生的变化,另一方面也提出一个命题:其实有些时候,无知的人说得最多。欢迎大家批评讨论甚至予以唾弃。

在这个修订版中,作者没有删掉旧的条款,仅在新旧条款前加了"新"、"旧"二字以示区别。新旧条款下面伴有作者的评论。虽然个别条款与C++或者编程甚至都没有关系了,但仍保留原标题。

1 把C++当成一门新的语言学习(和C没啥关系!真的。);

是的,我们仍然应该抱着这样一种心态来学习C++。前一段时间,C/C++ User's Journal上面有一连串关于C与C++关系的文章,Bjarne Stroustrup、Herb Sutter等几位C++领域的大师从不同方面讨论了C/C++今后发展的问题,颇为深刻,也很好展现了计算机语言发展的动力和阻力之所在。

看问题有深有浅,有高有低。我们不是大师,不要把C和C++说得好像自己的两个儿子。

2 看《Accelerated C++》,看《C++ Gochas》;(新)

2.看《Thinking In C++》,不要看《C++变成死相》;(旧)

Andrew Koenig的《Accelerated C++》是一本真正具有实践性的C++入门书。该书传承了《Ruminations On C++》的阐述方法,通过一个又一个具体的设计实例充分体现使用C++进行开发的真正优势之所在。事实上,笔者觉得Andy的书都是C++程序员必读(还好不算多)。

Stephen C. Dewhurst的《C++ Gochas》是一本"奇特"的书。说"奇特"不是因为Gotcha这个单词对于中国人难于理解,而是在于:Dewhurst用酸中带刺、笑里藏刀的口吻提醒程序员一些编写C++程序(本来就)应该注意到的问题。这些问题中,有编码细节问题,有编码习惯问题,甚至有个人修养问题。CUJ上有一篇对该书的评论,笔者认为还比较中肯。笔者看了该书的样章以后的感受是:忠言向来都逆耳;这本书中的观点你可以不认同,但其确实引出了一些应该注意(而通常没被注意)的问题,这对于一个程序员来说是非常值的关注的。

3 看《The C++ Programming Language》和《Inside The C++ Object Model》,不要因为它们很难而我们自己并不想搞"学术研究"所以就不看;(新)

3.读《The C++ Programming Language》和《Inside The C++ Object Model》,不要因为它们很难而我们自己是初学者所以就不看;(旧)

这两本书的确都不适合初学者看,旧条款说得有些过激,想必当初kingofark是希望刺激一下那些"浮躁的人(包括kingofark)"(见条款10-15)。

4 不要被VC、BCB、BC、MSC、TC等词汇所迷惑——他们都是集成开发环境,而

我们要学的是一门语言；

5 不要放过任何一个看上去很简单的编程小问题——它们往往并不那么简单，或者可以引伸出很多知识点；

侯捷先生译的《C++ Primer Answer Book》和裘宗燕老师译的《C++ Solutions - Companion to TCPPL》都已经出版了。作为在校学生或者C++自学者，通过这样两本书来操练自己的编码实践，实在是最合适不过了。

6 会用 Visual C++，并不说明你会C++；

会唱歌又会弹吉他，并不说明你会搞音乐创作——只不过有点“苗头”而已（且不能拔）。

7 学class并不难，template、STL、generic programming也不过如此——难的是长期坚持实践和不遗余力的博览群书；

博览群书不一定必要，这里强调的是学习新知识。To learn or not to learn...that is the question! 学习C++，这些内容不可或缺。

8 学编程与玩游戏，两者“相互作用，相互影响，辩证统一”，可以分割；不过的确有很多人因为爱玩游戏而学编程，也有不少人因为不爱学编程而玩游戏；其实这里面的道理再简单不过：如果你不是厨师——而自己会做一手好菜，自己当然有余地饱尝美味；如果你是厨师——拜托你不要光顾着吃，好好学厨艺行不行？回头吃肥了跌进猪圈人家认不出你。(新)

8.如果不是天才的话，想学编程就不要想玩游戏——你以为你做到了，其实你的C++水平并没有和你通关的能力一起变高——其实可以时刻记住：学C++是为了编游戏的；(旧)

旧条款是一句理由不充分的气话，源于当时笔者对沉溺游戏的“科班学生”的焦虑。

9 在工作环境中，请小声与同事说话。(新)

9.看 Visual C++ 的书，是学不了C++语言的；(旧)

请你记住一点：你大声说话，同事只听得见你的声音；你小声说话，同事还听得见你的心情。交流需要的是心境。

10 浮躁的人容易说：XX语言不行了，应该学YY：——是你自己不行了吧！？

11 浮躁的人容易问：我到底该学什么：——别问，学就对了；

12 浮躁的人容易问：XX有钱途吗：——建议你去抢银行；

13 浮躁的人容易说：我要中文版！我英文不行！——希望这不是你不好好学的原因。(新)

13.浮躁的人容易说：我要中文版！我英文不行！——不行？学呀！(旧)

14 浮躁的人容易问：XX和YY哪个好；——告诉你吧，都好——只要你学好就行；

15 浮躁的人分两种：a)只观望而不学的人；b)只学而不坚持的人；

16 已时髦的技术挂在嘴边，还不如把过时的技术记在心里；

17 C++不仅仅是支持面向对象的程序设计语言；

C++是以multi-paradigm为目标的通用型语言，学习起来应该全面了解。标准库为你做了很多，不用都是浪费。而如果你是做嵌入式开发，不妨参考一下Embedded C++的规范。

笔者愿意相信，学生、年轻程序员都比较可能了解这些。但是，大学教授、项目经理、系统设计师、相信“人月不是神话”的领导者们——他们了解吗？我疑心重重。

技术跟进，工具更新，并不一定就是盲目浮躁--镰刀锄头怎比得过洋枪洋炮？笔者愿意相信，学生、年轻程序员、初学者都比较容易了解这些，同时也正需要被引导着去了解这些。但是在国内，大学教授、项目经理、

相信"人月不是神话"的领导者们--他们了解吗？我疑虑重重。

18 学习编程最好的方法之一就是阅读源代码;

19 书不在多,好书则灵;(新)

19.在任何时刻都不要认为自己手中的书已经足够了;(旧)

如今国内的图书市场较之过去几年,最大的改变的就是大量国外图书的引进。书少的时候,没有选择的余地;书多的时候,选择太多,无所适从。这里就有一个如何择书的问题。接受推荐是一个不错的选择——不,我不是说某些图书网站上的口舌对骂。

面对这种情况,读者需要的其实是引导。

20 请参考《TCPPL 3rd》《C++ Primer 3rd》《The Standard C++ Library》,掌握C++标准;(新)

20.请阅读《The Standard C++ Bible》(中文版:标准C++宝典),掌握C++标准;(旧)

21 看得懂的书,请仔细看;看不懂的书,请硬着头皮看;

要知道,很多时候其实不是你看不懂,而是你以为你看不懂。

22 别指望看第一遍书就能记住和掌握什么——请看第二遍、第三遍;

23 请看《Effective C++》《More Effective C++》《Effective STL》《Exceptional C++》《More Exceptional C++》《Exceptional C++ Styles》《C++ Templates》《Ruminations On C++》《C Traps and Pitfalls》《Expert C Programming》;(新)

23.请看《Effective C++》《More Effective C++》《Exceptional C++》;(旧)

24 不要停留在集成开发环境的摇篮上,要学会控制集成开发环境,还要学会用命令行方式处理程序;

25 和别人一起讨论有意义的C++知识点,而不是争吵XX行不行或者YY与ZZ哪个好;

26 请看《程序设计实践》,并严格的按照其要求去做;

27 注意C与C++之间的区别。(新)

27.不要因为C和C++中有一些语法和关键字看上去相同,就认为它们的意义和作用完全一样;(旧)

28 C++绝不是所谓的C的"扩充"——如果C++一开始就起名叫Z语言,你一定不会把C和Z语言联系得那么紧密;

关于C与C++的关系及C++的发展与演化,C++创造者Bjarne Stroustrup在《The Design And Evolution of C++》(中译本,裘宗燕老师译)里面已经说得很清楚了。学C++的朋友应该看看本书,就当是扫盲。

29 请不要认为学过XX语言再改学C++会有什么问题——你只不过又在学一门全新的语言而已;

30 读完了《Inside The C++ Object Model》以后再来认定自己是不是已经学会了C++;

事实上,这样的高阶书你不一定就非要去啃,关键是浮躁与不浮躁、自信与自满的问题。

31 学习编程的秘诀是:编程,编程,再编程;

这一点在Andrew Koenig的书《Ruminations On C++》《Accelerated C++》里面有很好的体现。随着Andy的书一起作思维上的编程,自己再动动手,将是非常美妙的学习经历。

32 请留意下列书籍:《Design by Contract,by Example》《Refactoring》《Design Patterns Explained》;(新)

32.请留意下列书籍:《C++面向对象高效编程(C++ Effective Object-Oriented Software Construction)》《面向对象软件构造(Object-Oriented Software Construction)》《设计模式(Design Patterns)》《The Art of Computer Programming》;(旧)

33 记住:面向对象技术不只是C++专有的;

34 善加利用配套光盘;(新)

34.请把书上的程序例子亲手输入到电脑上实践,即使配套光盘中有源代码;(旧)

35 把在书中看到的有意义的例子扩充;

36 请重视DbC(Design by Contract)以及异常处理技术,并将其切实的贯彻和运用到自己的程序中;

DbC绝不仅仅是assertion。DbC要求各个组件各尽其责,将交流和协作建立在非常明晰严格的条款之基础上。DbC是不容忽视的,其所涉及的层面和深度,或许比我们想象的要广,要深。(新)

36.请重视C++中的异常处理技术,并将其切实的运用到自己的程序中;

Exception Handling引发着无数的讨论。关于exception handing的各种议题,看Herb Sutter的文章的确让人有些“震惊”。Embedded C++来得干脆——STL,Exception Handling全部删光光——可见什么事物总有个适用范围。(旧)

37 经常回顾自己以前写过的程序,并尝试重写,把自己学到的新知识运用进去;

38 不要漏掉书中任何一道练习题——请全部做完并记录下解题思路;

39 C++语言和C++的集成开发环境要同时学习和掌握;

40 既然决定了学C++,就请坚持学下去,因为学习程序设计语言的目的是掌握程序设计技术,而程序设计技术是跨语言的;

41 就让C++语言的各种平台和开发环境去激烈的竞争吧,我们要以学习C++语言本身为主;

42 只有通过编码实践才能领会设计思维;(新)

42.当你写C++程序写到一半却发现自己用的方法很拙劣时,请不要马上停手;请尽快将余下的部分粗略的完成以保证这个设计的完整性,然后分析自己的错误并重新设计和编写(参见43);(旧)

43 别心急,设计C++的class确实不容易;自己程序中的class和自己的class设计水平是在不断的编程实践中完善和发展的;

44 决不要因为程序“很小”就不遵循某些你不熟练的规则——好习惯是培养出来的,而不是一次记住的;

45 每学到一个C++难点的时候,尝试着对别人讲解这个知识点并让他理解——你能讲清楚才说明你真的理解了;

46 记录下在和别人交流时发现的自己忽视或不理解的知识点;

47 请不断的对自己写的程序提出更高的要求,哪怕你的程序版本号会变成Version 100.XX;

郑重提醒:请学会使用版本控制工具!Visual Source Safe,CVS——使用它们作版本控制绝对比你新建一个名为“MyApp_New2”的目录要好得多!在实际开发中,不使用版本控制工具的结果可以用一句话概括:可能变得多糟糕,就一定会变得多糟糕。

48 保存好你写过的所有的程序——那是你最好的积累之一;

49 请不要做浮躁的人;

50 请热爱C++!

其实Eiffel、Java、C#也不错:)

关于 Ks50PV 修订版的对话

编者按 大约是在两年前,一篇名为《Kingofark关于学习C++和编程的50个观点》的文章先是在CSDN上出现,然后被不断转载,迅速传播到各个技术网站,引起了网友的热烈讨论,余波持续至今。这篇文章对C++和编程技术学习的诸多方面进行了品评,虽然不见得多么严密和深刻,但是观点鲜明,一针见血,读来确实畅快。这可能是文章备受网友欢迎的一个重要原因吧。

而今,作者在严酷的实际编程工作中又磨练了两年。因此,当他告诉编者,著名的Ks50PV已经进行了局部的改写,我们当然非常想知道两年的实践经验究竟能够给这篇有趣的文章带来什么有趣的变化,在这些变化的背后又有些什么。因此,我们在刊登这篇新版Ks50PV的同时,还对原作者kingofark做了一个小小的访问。

Myan:我们很想知道,两年以来,IT产业不景气,C++又似乎进展乏力,你也在职场中经历了许多酸甜苦辣,蓦然回首,你觉得应该怎样评价写这篇Ks50PV时的kingofark?

Kingofark:当时的kingofark,刚刚走出校门不久。面对社会、工作中的各种压力,回首大学生活的时候,有很多感触积压在心头。自己走过的许多弯路,都如实的反映到Ks50PV中去了。撰写Ks50PV的过程其实就是我自己反省的过程。这个过程中饱含着迟到的悔恨和自信的丧失。现在看来,那时候还是很天真,对国内的IT公司报有过多的幻想。一句话概括:当时的kingofark,是一个对国内的C++应用范围过于乐观的人。

Myan:会不会觉得当时把C++看得太重要了呢?算不算是一种盲目的狂热?

Kingofark:狂热可能倒算不上,不过当时确实把C++看得太重要了,对C++抱有过于乐观的看法。因为只学过C++一门语言,所以不知不觉中对它形成了一种依赖。当时C++方面的优秀书籍正逐渐多了起来,也不免增加了几分兴趣和狂热的氛围。

不过从另一方面说,当时写Ks50PV也只是想就事论事。说起来好笑:当时在一个论坛上看到有人就学习C++大发牢骚,我不知为什么一下被气着了,心想怎么这个人比我还浮躁还无知,于是就打开记事本一口气写了二十多条自己的想法——那就是Ks50PV的原型。

拘泥于一门语言,或许可以算作一种惰性,没有足够的勇气和自信来拓宽自己的技术广度,领悟不到更高层面的大局观。我认为这是一种可怕的惰性——尤其在IT这个日新月异的产业。

Myan:你的很多条款都是评价技术书籍的。你觉得好书对于中国程序员有多重要?

Kingofark:好书之于中国程序员,好比一部好车之于赛车手。赛车新手乘上一部好车,经过刻苦努力的训练便可与世界级的高手相抗衡;而对于赛车高手,有了一部好车便是如虎添翼,大可借以冲击比赛冠军。

特别重要的是:好书是一个环节,自己的刻苦努力付出也是一个环节,两个环节是互相支撑的。当时之所以提出很多书目,是因为我发现很多人对这方面的信息知之甚少,择书的时候比较盲目。这里面有市场宣传和引导不足的因素,也有择书方法的问题,很多人一看《XX大全》、《XX宝典》之类的"枕头"书,就以为会很好,自己又没有仔细考察书中的内容,买下来之后读一读又觉得没什么帮助,便扔到一边去了,浪费钱还没学到什么。

读者真正需要的,是一个具有宣传和引导性质的平台,能够以多样的方式为不同需求的读者提供这方面的信息和资源。而我写这些条款的原则就是,把我认为值得看的好书都说出来,至少让过目者"混个眼熟",站在书店里记得拿起来看一看。

Myan：近两年来国内作者也写出了一些好书，你好像一本都没有提到。你觉得什么时候国内读者的原创技术书籍可能会提高到令你满意的程度？

Kingofark：实际上我也知道不少原创的优秀作品，当然也达到了满意的程度，但从数量上来讲，确实还是太少。另外或许是我自己获取的信息有限，自己发现或者经人推荐的优秀作品一般都不是C++方面的或者与特定平台特定开发环境有关，因此也就没有在条款里面写出来。

国内缺乏一个读者与作者的互动体制，好作者产出的作品，往往得不到适当的宣传和反馈，导致了整个创作循环的滞后。另一个重要的问题就是现有的出版体制无法将好作者与差作者区分开来，有时候一个昏昏噩噩的差作者可以通过更少的付出在更短的时间拿到比好作者更丰厚的报酬——你说那谁还愿意全心全意去把书作好。

我相信国内有潜力的作者还是很多，就是需要被发掘出来，并有相应的机制支持这个群体的发展。其实看看国外，大部分优秀书籍的作者并非就都是世界第一的高手大师，他们在良好的体制下发挥了自己的独特优势，赢得了一片读者群，读者群反过来促进他们不断提高作品水平。

最近一个同事向我推荐一本优秀的原创作品《加密与解密（第二版）》（电子工业出版社，段钢编），里面介绍了必要的基本知识（比如PE文件格式），阐述了很多较底层的汇编、反编译、调试、加密解密技术，讲述如何通过各种工具来完成特定的任务，非常实用且通俗易懂。这本书对于这方面的爱好者，特别是在校学生，应该是非常好的一个选择。我特别愿意借此机会推荐一下。

Myan：你对“浮躁”讲了很多。你觉得中国程序员浮躁的本质原因何在？

Kingofark：说小了这是学习态度的问题，说大了这是计算机教育问题。

从大的角度说，国内的计算机教育明显落伍。等级考试的内容就特能体现中国计算机教育的落后程度——要我说等级考试本来就是针对非计算机专业人士的考核，更应该讲求技术实用性，应该是跟踪业界脚步最迅速的考核之一。

在大学里面，教师自身没有跟上技术发展的脚步，项目实践经验不足，当然也就没有能力向学生教授合适的内容，更谈不上引导学生作编码实践。很多教师把一本很老很落后的教材施用多年，并且反驳说“技术本质上没有变化，技术的精髓不会过时”——要我说这就是一种不负责的惰性：“我不会所以我不教”。

计算机教室里面的电脑配置其实都还算不错，可以装win98甚至win2000，但是学生们看到的总是预先装好的Turbo C 2.0/BC3.1/Masm 4.0，而不是 Borland C++ Compiler 5.5和Masm8.0。学生们一边（明知道BC3.1过时了）使用BC3.1交付C++语言的课程设计，（明知道DOS已经被淘汰了）编写DOS下的汇编程序交付汇编课程设计，一边又从各种信息源接触到C#/VC/VB/BCB/Dephi/.NET/J2EE/GP/XP/ CMM/SQLserver/Masm8.0...当然会产生很大的隔阂感。自己在学校所学的东西与现实中了解到的东西差距这么大，如何能够轻易的建立自信去面对？当然就容易形成浮躁的心态。

从小的角度说，端正的学习态度也是慢慢培养起来的。很多高中生进入大学后就开始醉生梦死，说这是为了释放从小学积累到高中的巨大学习压力也可以理解，但就是有人醉生梦死了四年、七年，实在不值得原谅。

习惯搞坏了，在实际工作中又要面对因在校积累不足而引起的巨大压力，从学生时代滋生起的浮躁就自然而然的被带到毕业之后的程序员生涯中，挥之不去。

Myan：第十六条很有意思，“把时髦的技术挂在嘴边，还不如把过时的技术记在心里”，过时的技术还值得学习吗？你会不会花很大精力去学习Borland C++ 3.1，去学习COM？

Kingofark：（笑）其实这一条我是做了很多斟酌才决定这样写的。

我觉得还是应该牢牢记住这一条，因为“过时的技术”有一定的相对性——我是想说“眼前看到的技术”。现在技术发展迅猛，日新月异，新技术、新思想、新工具层出不穷，因此从某个角度来说，程序员总是“初学者”，需要不断的充电。在校学生没有在产业内的实践经

验,容易受各种风言风语(其中包括商家的炒作、周围的个别观点等等)的影响,面对这种情况,就容易变得无所适从。为什么呢?因为在校学生从滞后的专业课程中获取不到足够的知识和经验,本身就容易缺乏信心——昨天听说Delphi特别热门,今天又听说嵌入式开发有前途,明天又被告知C#"才是面向未来的",而专业前辈又老说其实C++才是最牛的。左顾右盼之际,自己不但没学到过硬的技术知识,又没跟上新技术的脚步,白白浪费时间在那里徘徊。

所谓触类旁通,只要先主抓一门技术学深学精,积累了经验,领悟了大局观,再跟进脚步学习任何新技术、拓展技术广度都不会很难。因此最后我就决定把话说绝一点,意思是:即便是去学Borland C++ 3.1,如果能借助BC3.1把面向对象、framework架构甚至C++对象模型这方面的知识吃透一点,比什么都不学要好得多。

其实大部分在校学生都有足够的学习条件和硬件,重要的是不要左顾右盼,要把眼前的事情做好学好。说到学习上的深度和广度,这里有一比:有广度没深度好像吹大的气球,针一扎就破;有深度没广度好像实心水晶球,一摔就碎;有广度又有深度是橡皮球,有韧性有弹性,针扎不破,摔下去弹起来。

Myan:对C++模板技术怎么看?你认为它代表C++未来的方向吗?

Kingofark:毫无疑问C++模板技术是这些年来C++领域里面最重要的关键技术之一,更是泛型技术发展的一个具体成果。我觉得泛型技术经过这么多年,应用程度还不算高。C++模板只能算个别几个引人注目的实现之一,比如Eiffel其实也有genericity,但是语言对它的支持非常有限,远不及C++。Eiffel的发明人Bertrand Meyer固然有他自己的看法,这一点可以参考OOSC2nd。所以泛型技术仍然还有蓬勃的发展前景和应用面——特别是这种理论不一定就只限于程序语言或者软件开发,作为一种高度抽象的理论,它甚至可以在其它相关领域体现价值;而C++的模板技术作为泛型的一种具体实现,借由STL经过这么多年的锤炼,我觉得已经从最初的迅猛发展转入到一个平稳而严峻的成熟发展期。

C++所提供的模板技术无法涵盖现代泛型技术的全部内容(甚至我们都不知道泛型技术的全部内容会是多少),而从《Modern C++ Design》看来,我觉得C++语言设施对其提供的支持已经在某些方面被挖到了极限。所以现在对C++模板技术应该不再是一个偏重探索的过程,而应该是一个偏重应用的过程,即如何有效的把模板技术应用到所有可应用的方面(这又会是一个很大的课题,即模板技术到底在哪些方面不适用,以及为什么不适用)。

如今C++在应用层面不再具有明显的优势,还在慢慢发展演化,谁也说不清它会以怎样的方式呈现在很多年之后的程序员面前。而C++模板技术也不应该完全代表C++未来的方向,但它是未来C++开发的一个重要组成部分,我认为会特别体现在"通过贯彻DbC(Design by Contract),产出可复用性极高的小型组件(比如系统组件、网络组件、功能单一的应用组件)"这方面。当然,C++的程序库也会扩充,其中某些程序库也应该会像STL那样,适合采用泛型技术来实现。

Myan:从条款的修订中,我发现你多次提到实践、编码,是不是两年的实践使你更务实了?

Kingofark:应该说经过两年的社会实践,我比过去任何时候都更感到实践的重要性。

程序开发是一个实践性极强的事情,不亲自动手编码,很难真正领悟技术的精义,把握复杂的实际情况。过去自认为看过不少书懂了不少技术,可到了工作岗位才发现,如果没有动手能力根本就无法把书上的知识反映到自己的设计和编码中去。要把书上学到的知识正确的反映到自己的设计和编码中去,需要通过实践积累经验——这也是为什么"工程经验类"书籍非常有价值的原因。就在前不久我还看到一段真实的"工业强度"代码:for(int i=0;i<100;i++) Sleep(300); //Wait 30Seconds 作者显然掌握了for循环语句的用法,你也不能说作者写错了,但是这段代码确实很蹊跷不是吗?把好书读少读精,用大量的时间动手实践,通过实践把理论知识反馈到大脑中去,才能很有效率的提高自身水平。

另一方面,做编码实践确实比较花时间,而在校学生可能面对课程压力,公司职员可能面对工作压力,时间很宝贵,所以我还是希望一次次的提醒他们,尽量挤出时间多做编码实践,挤一点是一点。这也是对我自己的一个提醒。

运用设计模式设计 MIME 编码类

——兼谈 Template Method 和 Strategy 模式的区别

▶撰文 / 温昱

本文讲述可重用、易扩充的MIME编码类的设计思路；并顺便对比了Template Method和Strategy模式的区别。

一、背景知识

MIME是一种Internet协议，全称为“Multipurpose Internet Mail Extensions”，中文名称为“多用途互联网邮件扩展”。其实，它的应用并不局限于收发Internet邮件——它已经成为Internet上传输多媒体信息的基本协议之一。本文仅关心MIME的编码算法。

MIME编码的原理就是把 8 bit 的内容转换成 7 bit 的形式以能正确传输，在接收方收到之后，再将其还原成 8 bit的内容。对邮件进行编码最初的原因是因为 Internet 上的很多网关不能正确传输8 bit 内码的字符，比如汉字等。MIME编码共有 Base64、Quoted-printable、7bit、8bit和 Binary等几种。

Base64算法将输入的字符串或一段数据编码成只含有{'A'-'Z', 'a'-'z', '0'-'9', '+', '/'}这64个字符的串，'='用于填充。其编码的方法是，将输入数据流每次取6 bit，用此6 bit的值(0-63)作为索引去查表，输出相应字符。这样，每3个字节将编码为4个字符($3\times8 \rightarrow 4\times6$)；不满4个字符的以'='填充。

Quoted-printable算法根据输入的字符串或字节范围进行编码，若是不需编码的字符，直接输出；若需要编码，则先输出'='，后面跟着以2个字符表示的十六进制字节值。

二、设计目标

我们计划开发一套MIME编码和解码的类，适用于可以想到的多种应用场合：

- Email客户端程序
- 乱码察看程序
- 图片等二进制对象存入XML文件

设计目标如下：

- 可重用
- 易使用
- 易扩充

三、设计过程

本部分分为下面3节，注意它们不是并列的3种设计方案，而是达到趋于合理的设计的思考过程：

- 设计成仅提供方法的Utility
- 设计成使用 Template Method 模式的 String Class
- 设计成使用 Strategy 模式的 String Class

1、设计成仅提供方法的 Utility

首先跳进我脑子的想法就是设计成Utility(仅仅提供方法的类)，我想可能是我受C影响太大的缘故吧。

它的接口会是什么样子呢？差不多象

```
bool UMime::Encode(unsigned char * outTargetBuf,
    int & outTargetBufLen,
    const unsigned char * const inSourceBuf,
    int inSourceBufLen);
bool UMime::Decode(unsigned char * outTargetBuf,
    int & outTargetBufLen,
    const unsigned char * const inSourceBuf,
    int inSourceBufLen);
```

吧。不行，为了满足易使用要求，应该支持CString类型的buffer吧，再增加2个接口函数

```
bool UMime::Encode(CString & outTargetStr,
    CString & inSourceStr);
bool UMime::Decode(CString & outTargetStr,
    CString & inSourceStr);
```

这样以来，UMime一共包括4个接口函数。

好像还不错？高兴得太早了。因为将来应用中很可能出现CString和unsigned char *协同工作的情形。比如应用从XML文件中读出一个字符串放到一个CString型变量中，而这个字符串是一个Bmp图片的MIME编码，它解码过后自然应放到unsigned char *的buffer中。所以我们还要增加下面4个接口函数：

```
bool UMime::Encode(CString & outTargetStr,
    const unsigned char * const inSourceBuf,
    int inSourceBufLen);
bool UMime::Decode(CString & outTargetStr,
    const unsigned char * const inSourceBuf,
    int inSourceBufLen);
bool UMime::Encode(unsigned char * outTargetBuf,
    int & outTargetBufLen,
    CString & inSourceStr);
bool UMime::Decode(unsigned char * outTargetBuf,
    int & outTargetBufLen,
    CString & inSourceStr);
```

以免用户类型转换之苦。

啊哈，这么8个极为相似的接口函数搅在一起，好像一团麻呀。可重用性似乎满足了，但易使用性和易扩展性完全谈不上。看来需要寻求更合理的设计。

2、设计成使用Template Method模式的String Class

第2种方案浮现在脑海中：

❑ 既然整个算法就是将一个Buffer转换成另一个Buffer，写成一个String Class是非常自然的设计

❑ 用Class的成员变量保存Target Buffer及其长度（因为Buffer中可能有'\0'），另外提供GetBuf()和GetBufLen()作为查询Target Buffer的接口

❑ 直接从构造函数传递Source Buffer的信息

该类大概象这样：

```
class CMimeString
{
public:
enum PROCESSTYPE
{
ENCODING = 0,
DECODING = 1
};
CMimeString(PROCESSTYPE inType, const unsigned char *
    const inBuf, int inBufLen);
CMimeString(PROCESSTYPE inType, CString & inStr);
    virtual ~CMimeString( );
    unsigned char * GetBuf( void );
    int GetBufLen( void );
    operator LPCTSTR( ) const;
};
```

哈，似乎很美妙。

❑ Source Buffer仍然支持unsigned char *和CString这2种类型，而Target Buffer由CMimeString本身来管理不必用户操心了。

❑ 但具体应用不是对二进制对象进行编码时，可以不用foo(s.GetBuf())而直接用foo(s)，因为operator LPCTSTR() const;自动负责类型转换。

❑ 直接从构造函数传递Source Buffer的信息，使得接口更为精简。

当具体使用CMimeString时，大概象这样：

```
CString  buf("sadfsdfsdf");
CMimeString mime(CMimeString::ENCODING, buf);
MessageBox( s );
```

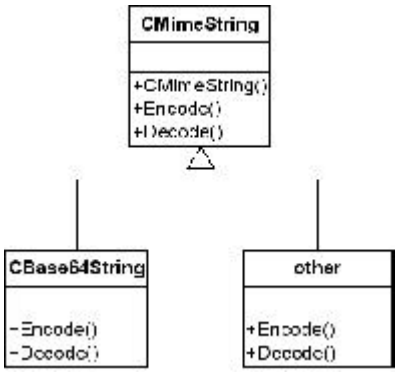
看来易使用性不错，下面要着重解决易扩展性了。CMimeString的实现部分会象这样：

```
class CMimeString
{
protected:
unsigned char *  mBuf;
int mBufLen;
virtual void Encode( unsigned char * inBuf, int inBufLen );
virtual void Decode( unsigned char * inBuf, int inBufLen );
};
```

其中的两个虚函数是专门为易扩展性准备的，要实现新的MIME编码算法，只需要从CMimeString继承一个子类：

```
class CBase64String : public CMimeString
{
protected:
virtual void Encode( unsigned char * inBuf, int inBufLen );
virtual void Decode( unsigned char * inBuf, int inBufLen );
};
```

类图如下：



这两个虚函数是在哪里被调用的呢？在基类的构造函数中。

```
CMimeString::CMimeString(WHICHTYPE inType,
                         CString & inStr)
{
   mBuf = 0;
   mBufLen = 0;

   if( inType == ENCODING )
   {
   Encode((unsigned char *)
      (inStr.operator LPCTSTR()), inStr.GetLength());
   }
   else if( inType == DECODING )
   {
   Decode((unsigned char *)(inStr.operator LPCTSTR()),
         inStr.GetLength());
   }
}
```

看上去是很不错的Template Method模式的运用，但是有问题——因为“在构造函数中调用虚函数”并无多态特性！

```
CBase64String::CBase64String(PROCESSTYPE inType,
    CString inStr)
{
   OnlyInitSelf();
}
```

之后

```
CString  buf("sadfsdfsdf");
CBase64String base64(CMimeString::ENCODING, buf);
MessageBox( base64 );
```

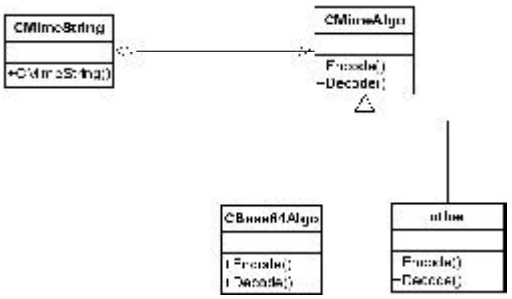
是不对的，仍然是基类的CMimeString::Encode()被调用了，而且OnlyInitSelf()在Encode()被调用之后才被调到。

是不是有些懊恼？别急。分析问题背后的问题：我们想用Template Method模式倒是没错，但是让构造函数扮演Template Method的角色是错误的，它先天(C++本身决定的)就不是这块料。

现在，摆在面前的至少有2条道路。第1种方法是，坚持使用Template Method模式，但要增加一个接口函数扮演Template Method角色。这样一来，我们使用CMimeString时就不如“直接从构造函数传递参数”方便。第2种方法是，坚持直接从构造函数传递参数，放弃Template Method模式，改用其它模式完成“改变算法”的职责。我决定采用第2种方法。

3、设计成使用Strategy模式的String Class

除了Template Method模式以为，Strategy模式也可以履行“改变算法”的职责，我们就用Strategy模式代替Template Method模式继续完成CMimeString的设计，类图如下：



新的CMimeString的类声明如下：

```
class CMimeString
{
public:
   enum PROCESSTYPE
   {
      ENCODING = 0,
      DECODING = 1
   };
   enum ENCODETYPE
   {
      WYMIME = 0,
      BASE64 = 1
   };
   CMimeString(PROCESSTYPE inType,
      ENCODETYPE inAlgoType, CString & inStr);
   CMimeString(PROCESSTYPE inType,
      ENCODETYPE inAlgoType,
   unsigned char * inBuf, int inBufLen);
   virtual ~CMimeString();
   int GetBufLen(void);
   unsigned char * GetBuf(void);
   operator LPCTSTR() const;
};
```

CMimeAlgo的类声明如下：

```
class CMimeAlgo
{
public:
   CMimeAlgo();
   ~CMimeAlgo();
   virtual void Encode( unsigned char ** outBuf,
      int & outBufLen,
   unsigned char * inSrcBuf, int inSrcLen );
   virtual void Decode( unsigned char ** outBuf,
      int & outBufLen,
      unsigned char * inSrcBuf, int inSrcLen );
};
```

子类CBase64Algo的类声明如下：

```
class CBase64Algo : public CMimeAlgo
{
public:
   CBase64Algo();
   ~CBase64Algo();
   virtual void Encode( unsigned char ** outBuf,
       int & outBufLen,
   unsigned char * inSrcBuf, int inSrcLen );
   virtual void Decode( unsigned char ** outBuf,
   int & outBufLen,
   unsigned char * inSrcBuf, int inSrcLen );
};
```

具体使用Base64算法时会象这样：

```
CString buf("sdfsdsdfsdfsdf");
CMimeString base64( CMimeString::ENCODING,
    CMimeString::BASE64, buf );
MessageBox(base64);
```

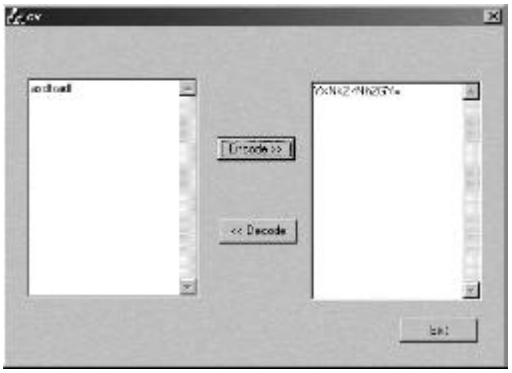
哈哈，基本满意。

四、使用举例

下面编一个小程序，重在演示CMimeString的用法。有2点需要说明：

❑ 程序比较简单，仅支持Base64编码和解码；

❑ 而且对一个串进行解码时并没有检查它是否是合法的Base64编码的结果串（有些字符串是不可能成为Base64编码的结果的），因此对串someString解码后再编码得到的串anotherString可能和someString并不相同。



五、Template Method和Strategy模式的区别

Template Method和Strategy模式对比如下：

❑ 相同点，都是行为型模式，目的都是方便地改变算法。

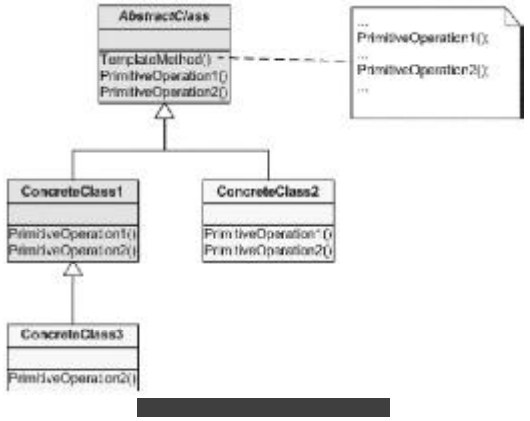
❑ 不同点，实现方式前者使用继承，称为类模式；后者使用委托，称为对象模式。

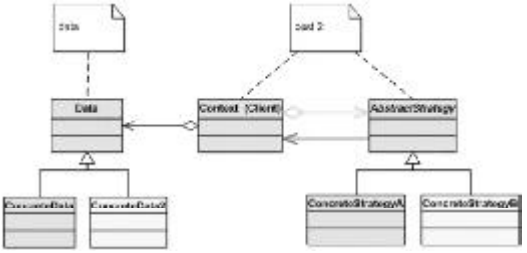
《设计模式》一书在讲到Template method模式和Strategy模式的关系时说：“模板方法使用继承来改变算法的一部分。Strategy使用委托来改变整个算法。”

“算法的一部分”和“整个算法”的区别，笔者认为“整个算法”是“算法的一部分”的特例（就象数学中全集是集合的特例），因此不是2个模式的根本区别。

“继承”和“委托”的区别，即“类模式”和“对象模式”的区别，笔者认为这是2个模式的根本区别。

顺便说明，《设计模式》一书中非常强调对象模式和类模式的区别，本文就提供了一个很极端的例子——用对象模式可行而用类模式不可行。





参考文献

❖《MIME邮件面面观》　作者：bhw98
出处：www.csdn.net

❖《设计模式》　作者：Gamma等
译者：李英军等

❖《Pattern Tips》　作者：温昱
出处：lcspace.nease.net

关键词:Delphi, .NET

Delphi.NET前瞻

▶撰文/韩磊

打开Delphi IDE的Delphi Direct（菜单：Help->Delphi Direct),是什么吸引了你的眼光？公元2003年的某个夜晚,Delphi Direct窗口中出现的内容吓了我一跳。在前十条内容中（分属于Delphi News和Delphi Community Articles两类），总共有7条跟.NET有关。尽管其中多数新闻早已变成旧闻,但对旧闻新闻的一体回味却颇让人顿感形势紧迫——再不跟上.NET的脚步,恐怕就会有丢饭碗之忧了。

Borland是第一个获Microsoft .NET Framework SDK授权的第三方开发工具厂商(参见http://www.internetnews.com/dev-news/article.php/1574721),这是今年一月底的事。但早在去年下半年发布的Delphi 7 Studio中，已经包括了一个预览版本的.NET编译器。作为对该预览版的补充，Borland随后推出了Update,其主要内容就是VCL for .NET。种种迹象表明，Delphi.NET已是呼之欲出。.NET是SUN One最可怕的竞争对手。更可怕的是McNealy的不可一世和Gates一贯的持久战策略——想想Sun Java Workshop推出之前的举世轰动和推出之后的黯然销魂,不禁暗中为Sun捏一把汗,同时也为Borland多方出击的战略击节赞叹。要知道,Jbuilder是Java开发工具市场的老大，而且Borland EJB也是数得上号的Java应用程序服务器产品,支持.NET就意味着自己对自己开战。不过,由于市场重合(同类型用户面临二中选一的产品抉择)的时间差,Borland能够在两个市场都讨得到巧。这已超出本文的讨论范围,但作为Win32平台的Delphi开发人员,Delphi对.NET的全面支持,却是不得不重视的。

本文简单介绍Delphi程序员比较应该关心的部分.NET架构,然后将就随D7发放的Delphi.NET编译器预览版进行一些有益的探讨。需要提请读者注意的是,文中涉及内容在正式发行的Delphi.NET中可能会有不同之处。

.NET Framework

.NET是一个庞大的系统,大体而言是通过.NET Enterprise Servers/.NET Framework/.NET WEB Services提供服务。未来的Delphi.NET,极有可能是通过与.NET Framework的互动向程序员提供开发运行在.NET平台软件的能力。那么,.NET Framework是怎样的呢？从下面的图中可以很容易地知道其结构和组成。

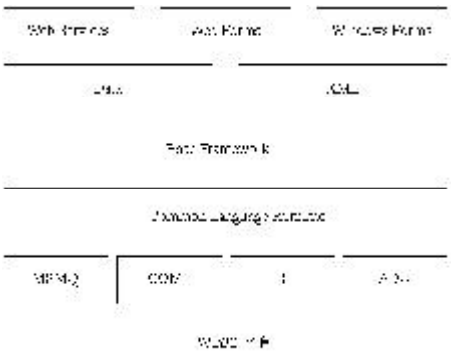


图1 .NET Framework

实际上,.NET Framework包括三个主要部分：Common Language Runtime（CLR)、Base Framework Class Library(基础框架类库，又称.NET class library）和ASP.NET。其中，CLR是.NET应用程序的执行引擎。开发工具将源代码编译为MSIL(Microsoft Intermediate Language，Microsoft中间语言)。顾名思义,IL是一种不依赖于具体操作系统的语言代码。执行时，CLR会利用JIT（Just-In-Time）编译器将其编译为本地操作平台相应代码。

.NET应用程序是自解释的。也就是说,它本身包括了能让CLR理解并照办的描述性元数据。这些元数据可能包括类、类型、方法定义等等内容。原数据和真正的IL代码都被包含在一个一个的Assembly中。而每个Assembly也都拥有一个manifest，其中描述了assembly

的名称、版本、文件列表和依赖性、类、方法列表等内容。通过assembly-manifest的组织方式，.NET应用程序能明确地向CLR解释自身，并帮助CLR将IL编译为最终的本地执行代码。

尽管Microsoft近水楼台先得月，快人一步推出VS.NET，但这并不意味着只有C#和VB.NET等微软开发工具才能用来开发.NET程序。相反，只要能编译出MSIL代码，任何语言都可以开发出执行于.NET平台的应用程序。从下图中可以清楚地看到，一个.NET应用程序是如何被创建及执行的。

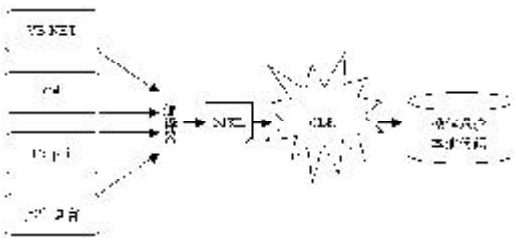


图2 .NET 应用程序的生成与执行

Base Class Library定义了一系列的类、方法等。所有的类和方法都通过namespace（命名空间）的形式来表述。例如Console这个命名空间就提供了ReadLine、WriteLine等方法。在稍后的例子中我们会使用这两个方法。当然，你也可以定义自己的类和方法，但它们最终会以namespace成员的形式呈现。这种方式对于Delphi程序员来讲并不难以理解。请看下面的例子：

```
type
  TDemoForm = class(System.Windows.Forms.Form);
```

这段代码定义了一个类TDemoForm，该类从System.Windows.Forms命名空间的Form类继承而来。如果我们把System.Windows.Forms名字空间放到uses子句中，则会变成下面的样子：

```
uses
  System.Windows.Forms;
type
  TDemo = class(Form)
```

是不是感觉很亲切呢？

事实上.NET是一种跨平台的解决方案。在整体架构上类似于Java，但在语言灵活性上却远胜于Java。程序员可以用任何熟悉的语言来开发（当然需要这种语言的开发工具厂商提供.NET编译器），然后通过CLR在支持.NET的任何平台上运行。但是，中间代码的运行方式会不会影响应用程序的执行效率呢？VB和Java（尤其是后者）程序的执行速度，实在是令人头疼。.NET有什么本事来解决这个问题？

首先，对于被多个应用程序调用的assembly（例如Base Class Library），.NET会在内存中始终保留一个备份。这样，调用这些assembly提供的方法将会十分快捷。再者，应用程序并非每次执行都需要进行编译。在版本未更新的情况之下，CLR不会再次进行编译，而是直接运行上次运行时编译好的本地代码。这些措施都保证了.NET应用程序的执行效率。

我们已经知道，.NET是一个开放式的平台，允许利用各种语言来开发运行于其上的应用程序。我们需要的，就是一个能将该种语言编译为MSIL的编译器。Borland尽管进入.NET世界的脚步稍显迟缓，但发挥了在编译器技术上的一贯优势，在2002年推出Delphi 7时，同时发放了Delphi for .NET compiler的Preview版本。跟下来我们就来看看这个Preview版的Delphi.NET编译器。

Delphi.NET 编译器

编译器技术一向是Borland引以为傲的强项。所以，并不令人意外地，Borland在支持.NET的过程中，第一步就是发布Delphi for .NET Compiler。如前所述，该编译器的作用就是将Object Pascal代码编译为.NET IL代码。通过后文的介绍，读者将会了解到，这个编译器已经接近成熟。再加上刚发布的VCL for .NET，Delphi两大杀手锏已然齐备。不过，在真正的Delphi .NET版本发布之前，程序员依旧无法凭借RAD工具快速地撰写.NET应用程序。

预览版本的Delphi for .NET Compiler编译速度不算很快。对于一个最简单的HelloWorld程序，在笔者的机器上大约需要2.9秒的时间。不过，Borland显然已经针对.NET平台特性作了许多的努力。在Delphi for .NET Preview安装目录的units目录下，我们可以看到后缀分别为dcuil和dcua的文件。前者是编译好的公用名字空间、类、类型、例程等，而后者则指导编译器从前者获取相关资料。这样，编译器就不必每次都从源代码级开始执行所有的编译工作。

李维先生的新著《Borland传奇》中也谈到Delphi .NET Complier Preview。但也许是作者的疏忽，有一个小小的错误。读者在看到该书相关章节时会注意到，

书中提及的在Delphi IDE中出现的.NET Compiler菜单，其实并不会在安装Delphi for .NET Compiler Preview后自行出现。当然那是一个非常有用的Open Tools工具（由Delphi QA Team开发），但读者需要在http://codecentral.borland.com/codecentral/ccWeb.exe/listing?id=18889下载安装后才能充分享受其方便。而相关文章可以在http://bdn.borland.com/article/0,1410,29159,00.html找到。我建议读者在继续阅读之前先安装这个非官方、但绝对好用的工具。我们稍后会详细地讨论它。

如果你没有在Delphi IDE中安装这个工具，那么就只能用命令行的方式来编译你的Delphi.NET应用程序。其实这也不是一件痛苦的事情，毕竟久违的黑底白字命令行方式也会给我们带来一丝对过往的回忆。在安装了Delphi for .NET Compiler Preview后，编译器路径已经被包括到系统搜索路径中，这意味着我们可以在任何地方运行该编译器。打开一个命令行窗口（最方便的做法：运行“cmd”），输入dccil后按回车。正常的话会看到命令行编译器的开关、选项的帮助。

图3 Delphi.NET Compiler命令行编译器开关和选项

使用命令行的好处是可以完全控制编译过程和编译结果。这一点读者可以从dccil帮助中很清楚地看出来。不过，Delphi QA Team提供的非官方Delphi for .NET Compiler IDE集成工具，却更为方便和快捷。后文如非特别说明，否则都将用该集成工具进行编译。好，让我们来熟悉一下这个好用的工具：

图4 Delphi for .NET IDE集成工具

最上面的六个菜单项不言自明。而跟下来的两项却又让人顿生不知所云之感。其实很简单，只要把Peverify的首三个字母全部大写就明白了——PEVerify。这是一个检查PE执行文件有效性的工具，在命令行模式下同样有用。但是用这个工具，验证结果即时在Debug窗显现出来，的确是相当方便。ILDasm是.NET提供的一个反编译工具，全称为IL Disassembler；在后文中将在使用中认识它。至于Refelector，是一个.NET Class浏览器，不过读者需要自己到http://www.aisto.com/roeder/dotnet/下载后才能使用。我个人觉得更有用的是.NET Framework SDK help一项，执行之后会通过Microsoft文档资源管理器调用.NET SDK帮助。建议读者好好阅读.NET SDK，因为其中的信息对于开发.NET程序是极重要、极具参考性的。当然，读者不可忽略的一点是，在初试Delphi for .NET Compiler Preview之前，需要先安装Microsoft .NET Framework。如果你已经安装VS.NET，则无须进行此步骤；否则请到http://www.microsoft.com/china/msdn/netframework/downloads/default.asp选择合适的Framework文件下载。建议同时安装Service Pack 2。

最简单的Delphi.NET程序

就像任何语言学习课程一样，让我们以“Hello World”作为我们的第一个Delphi.NET应用程序。这个程序的目的是在命令行窗口显示一行字：Hello World!虽然简单，但已透射出Borland在支持.NET平台上的巨大努力。好的，用记事本或是Delphi IDE创建一个.dpr文件，不妨叫做HelloWorld.dpr吧。代码如下：

```
ProgramHelloWorld;

{$APPTYPE CONSOLE}
begin
  WriteLn('Hello World!');
end.
```

请注意，代码和普通的Delphi Console应用程序没有任何不同之处。我们调用系统例程WriteLn，在命令行输出一个字符串Hello World。笔者去年曾在《程序员》杂志发表一系列关于Console Application的译文，有兴趣的读者可以参照。如果Delphi.NET Compiler能够将普通Delphi程序代码编译为能在.NET平台正常执行的程序，将会是多么激动人心的一刻！那将意味着Delphi程序员不需要撇弃喜爱的Delphi和Object Pascal，也能轻易地开发.NET应用程序。更让人振奋的是，旧的Windows原生程序，也能较容易地移植到.NET平台。

现在在Delphi IDE中调用Delphi for .NET Preview菜单的Build一项，或是在命令行中输入dccil -c HelloWorld.dpr。短暂的等待后我们得到HelloWorld.exe，一个.NET IL应用程序。然后把HelloWorld.dpr另存为HelloWorldWin.dpr，Build成为Windows本地代码的HelloWorld程序版本。一个是Windows原生应用程序，一个是.NET IL应用程序，执行起来会有什么不同呢？好，在命令行分别执行HelloWorld和HelloWorldWin。可以看到，字符串都被正常输出，完全看不出有任何的差别。

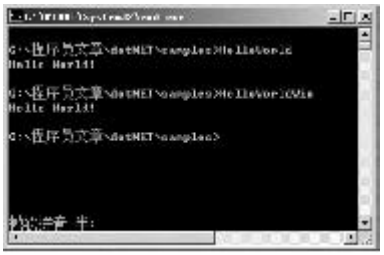


图5 .NET版本/Windows原生版本的HelloWorld程序

而且，以最终的.exe文件来比较，.NET版本要比Windows原生版本尺寸小。在上例中，分别是13Kb和15Kb。这是由于微软在操作系统级提供了支持。当然，对于多数程序，VB.NET编译出来的的程序尺寸还会更小。但以目前动辄数十G的硬盘容量来考较，实在不算得很值得重视的事。

除了原有的Pascal系统例程，Delphi for .NET Compiler还可以通过命名空间调用.NET提供的系统级类、类别、例程等。例如，System.Console命名空间同样提供了WriteLine、ReadLine等例程。将上例作少许改动，改用System.Console命名空间例程，并加上少少互动元素，变成下面的样子：

```
ProgramHelloSir;

{$APPTYPE CONSOLE}
var
  s: String;
begin
  Console.WriteLine('请输入名字:');
  s:=Console.ReadLine();
  Console.WriteLine('Hello, '+s+'!');
end.
```

将工程命名为HelloSir后保存并Build。运行一下：

图6 使用命名空间调用并具备交互性的HelloWorld程序

在这个例子中，至关重要的是我们可以选择使用旧的Object Pascal方式或是命名空间方式达到目的。在后一种情况中，应用程序的体积大幅减小了(HelloSir.exe只有区区5Kb)。使用ISDASM就可以清楚地看到，如果使用前一种方式，编译器需要在执行文件中放入更多的Borland名称空间信息供.NET平台调用。有兴趣的读者不妨作一番对比。

自定义类、方法和跨语言特性

.NET是跨语言的。这句话的意思不光是指任何语言理论上都可以开发.NET应用程序(理论上Java也可以，不过Sun可能要不高兴的)，还指在“中间语言”和CLR的机制底下，不同语言实现的类、方法等都可以通过名称空间互相调用甚至继承。平台的统一反而促成了语言的开放，这与Java刚好相反。而“未来的开发格局，语言不再重要，至关重要的是系统架构”这句话，多少也得到了验证。

为了展示跨语言特性，我们以一个简单的例子来说明。还是以刚才的HelloSir为蓝本，先用Delphi创建一个类TDelphiClass。整个dpr文件内容如下；

```
library SampleNameSpace.DelphiClassDemo;
uses
  System;
type
 TDelphiClass = class
   constructor Create;
   procedure SayHello(s: String);
end;
constructor TDelphiClass.Create;
begin
   Inherited Create;
end;
procedure TDelphiClass.SayHello(s: String);
begin
  Console.WriteLine('Hello, ' + s + '!');
end;

end.
```

对于任何一个有经验的Delphi程序员来说,这都是一个简单得不能再简单的库文件了。其中SampleNameSpace.DelphiClassDemo一句提供了重要的信息，将SampleNameSpace.DelphiClassDemo命名空间publish出来，供其它程序调用。而TDelphiClass则是该命名空间下的一个自定义类，其中定义了一个SayHello方法,接受传入的字符串,之后向命令行输出重新组合过的字符串。

在以往的经验中,VB等其它语言要实现TDelphi-Class类几乎是不可能的。而在.NET中一切都变得简单起来。用记事本编辑以下内容的vb源程序并将其保存为VBCallDelphi.vb：

```
Imports System,SampleNamespace.DelphiClassDemo

Module VBCallDelphi

Sub Main()

 Dim DemoClass As New TDelphiClass
 Dim InputName As String
 Console.WriteLine("请输入名字:")
 InputName = Console.ReadLine()
 DemoClass.SayHello(InputName)

End Sub

End Module
```

在.NET Framework中包括了一个命令行VB编译器。为了引入刚才的TDelphiClass类，需要用r参数来指明包括该类的Library文件。如此一来,在命令行输入：

```
vbc VBCallDelphiDemo.vb /r:DelphiClassDemo.dll
```

得到VBCallDelphi.exe可执行文件。让我们执行它，看看VB是否真的实现了Delphi自定义类TDelphiClass的一个实例，并调用其方法SayHello。

图7 在VB中实现Delphi自定义类

如果读者有志于第三方组件/函数库市场,那么,上面的执行结果肯定会令你兴奋。你用Delphi实现的组件/函数库将可不费周章地为其它语言服务。想想以前让人头疼的ActiveX……

.NET GUI应用程序

用惯Windows图形界面的读者，当然不会满意于Console Application的黑白世界。尽管我们还无法利用拖放等RAD方式快速搭建GUI用户界面,但仍然可以经由FCL（.NET Framework Class Library）构造标准窗口界面。

Delphi程序员习惯于使用VCL（Visual Component Library）构造用户界面，而FCL也提供了类似VCL的组件结构。System.Windows.Form命名空间包含了用于创建Windows图形程序的类。在Microsoft文档资源管理器中输入URL：ms-help://MS.NETFrameworkSDK.CHS/cpref/html/frlrfsystemwindowsforms.htm，读者可以自己查阅相关的.NET SDK资料。Delphi程序员对其中大部分内容不会感到陌生。

不幸的是我们目前只能手工编写所有的Form组件创建代码。不过由于FCL和VCL的相似性,这项工作并不具有很高的难度。习惯在运行时创建组件实例的程序员有福了,他们可以通过对System.Windows.Forms名称空间的引用,用自己熟悉的方式打造GUI应用程序。但即使是对于习惯RAD的用户,只要稍具OO概念,仍然不会被"自己生成Form"所吓倒。下面我们来生成一个最简单的.NET Windows GUI应用程序。

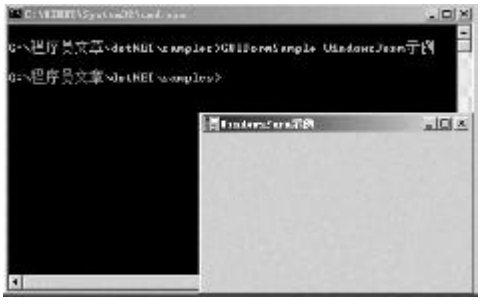
当然必须先引用System.Windows.Forms命名空

间,然后声明一个该命名空间的Form类型。不过,由于主窗口是手工生成的,我们必须在程序中显式地创建它。在Form的Create事件中,我们会对它的一些属性赋值。并且,我们还会使用一个技巧,使得程序可以从命令行接受参数作为Form的标题。完整代码如下;

```
program GUIFormSample;
uses System.Windows.Forms;
type
  TWinForm = class(Form)
  private
    procedure InitFormProperty;
  public
    constructor Create;
  end;
  { TWinForm }
var
  Arguments: array of String;
constructor TWinForm.Create;
begin
  inherited Create;
  InitFormProperty;
end;
procedure TWinForm.InitFormProperty;
begin
  Arguments:=System.Environment.GetCommandLineArgs;
  Text := Arguments[1];
  Size := Size.Create(300, 200);
end;
begin
  Application.Run(TWinForm.Create);
end.
```

将项目保存为GUIFormSample然后Build。成功后在命令行输入:GUIFormSample WindowsForm示例。GUIFormSample后面的字符串是我们想用来作为Form标题的参数。结果如何呢?果然不出所料,一个以传入字符串为标题的Form正确地创建了。但是请不要试图不带参数执行这个程序,否则会引起错误(如图8)。

在FCL内部,创建Form的请求最终被理解为对Windows API相关方法的调用。同样,在其它平台上构造GUI程序,也会有相应的平台自适应转换机制。



当然这里的讨论相当浅。真正的GUI界面要比这个复杂得多。例如,添加Form上的组件、对鼠标/键盘输入的响应事件等等,由于篇幅的关系,无法再做详述。不过要请读者相信的是,这些工作都可以经由FCL完成,而且,实际上并不十分费事。在Borland CodeCenter读者可以找到越来越多的.NET Windows GUI应用程序范例。在其中一些例子中,作者甚至使用VCL来构建.NET程序——当然这需要一定技巧。

ADO.NET 初探

Delphi在Windows平台上的数据应用开发能力,相信是任何开发者都会赞赏的。从Delphi3以来,对数据库应用的支持一直是Delphi的重要功能。而在Delphi.NET中,数据库应用同样是重中之重。

在以前的Delphi开发中,常用BDE、OLE DB、ODBC或是ADO作为数据库引擎。在.NET中,继Windows2000/XP力推ADO之后,ADO.NET成为主要的数据库引擎。ADO.NET正是ADO的换代产品,在特性上也有了许多的进步和改动。程序员也许需要一段时间的学习才能熟练掌握ADO.NET。

在.NET Framework中,ADO被映射到System.Data和System.XML两个命名空间。ADO.NET提供了两种数据库Provider,System.Data.SqlClient用来连接MS SQL Server数据库,而System.Data.Oledb则用来连接其它数据库。在微软网站可以下载针对Oracle和ODBC的数据库Provider,对于Interbase而言也可以使用Borland发布的Provider。根据不同的应用情景,可以采用不同的Provider或是不同的数据读取方式(dataset/datareader)。

开发基于ADO.NET数据库应用的基本流程,与开发基于ADO的数据库应用没有大的不同。简单说来就是连接数据库->执行查询或更新命令->获取数据->关闭连接。只不过我们得使用System.Data/System.XML命名空间中的类和方法而已。在下面的例子我们将连接MS SQL Server 2000的Northwind数据库并获取其中的一些数据。简单起见,将使用Console类型程序。

这个程序的流程如前所述,先打开一个数据库连接,然后执行一个SQL查询并得到返回数据集。鉴于我们只是读取数据,不需要对数据进行复杂处理,所以用DataReader的方式获取数据更为节省资源(dataset方式

将在内存中保留一个数据集备份)。在向用户界面输出查询结果之后,结束连接。代码如下:

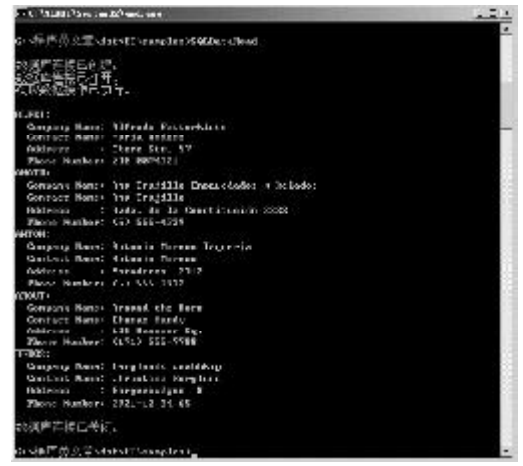
```
{$APPTYPE CONSOLE}

program SQLDataRead;

uses
  System.Data.SqlClient;

var
  Conn              : SqlConnection;
  Cmd               : SqlCommand;
  DR                : SqlDataReader;
begin
  Conn := SqlConnection.Create
    ('server=LEI\LEI;database=Northwind;uid=sa;pwd=');
  Console.WriteLine(#13#10+'数据库连接已创建。');
  Conn.Open;
  Console.WriteLine('数据库连接已打开。');
 Cmd := SqlCommand.Create('SELECT TOP 5 * FROM Customers
', Conn);
  DR := Cmd.ExecuteReader;
  Console.WriteLine('获取数据操作已执行。'+#13#10);
  while DR.Read() <> False do
  begin
    Console.WriteLine(DR['CustomerId'].ToString + ':
');
      Console.WriteLine('Company Name:'+DR
['CompanyName'].ToString);
      Console.WriteLine('Contact Name:'+DR
['ContactName'].ToString);
    Console.WriteLine('Address :'+DR['Address'].
ToString);
    Console.WriteLine('Phone Number: '+DR['Phone'].
ToString);
  end;
  Conn.Close;
  Console.WriteLine(#13#10+'数据库连接已关闭。');
end.
```

为了追踪数据库操作,代码中加入了一些"多余的"行。程序很简单,就不多做解释了。



小结

.NET Framework提供了强大的执行环境。通过CLR方式,任何具备IL编译器的语言工具都可以用来编写.NET应用程序。作为第一家与Microsoft缔结.NET合作协议的第三方厂商,Borland已推出基于.NET Framework的C# Builder,并将于今年内推出跨越Win32 Native platform和.NET platform的Delphi 8(据可靠消息,Delphi.NET将不作为单独产品推出,而是包括到Delphi 8中)。本文介绍了随Delphi 7发行的Borland .NET编译器预览版本,并与读者一起利用该编译器在.NET世界中小小游玩了一番。由于篇幅的关系,在ADO.NET方面深入探索原计划只好作罢。不过,我还想提醒读者注意,在李维和左轻侯介绍C# Builder的相关文章中,可以看到Borland是如何扩展.NET数据库访问功能的。

▶撰文 / 赵普舫

玩过游戏吗？

如果有兴趣可以玩一下，在你不忙的时候。

一个好的游戏可以拥有众多的玩家，这些人可以为了游戏放弃所有的兴趣，我也曾经痴迷过？呵呵。

反过来，我们这样一些程序员为什么从来没有设计出来一个应用程序和魔兽争霸，星际，反恐等等一样那样让他的使用者狂热呢？

玩过了游戏，回过头来发现大多数比较经典的游戏都有地图编辑器，角色编辑器，物品编辑器，逻辑插件和华丽的界面以及丰富的情节，例如，暴雪的星际，魔兽都有这些东西，反恐中的AMX插件扩展等等。通过这样一些做法游戏得以扩展、延续、加强、修正。

在这里，我给出了一个断言式的小结论“程序应该做的更灵活一些，留下一点可以挖掘的空间给你的用户，在你所能允许的范围之内。”然后我们开始面向应用的扩展之旅。我们会想尽一切办法来实现我们这一目标，让用户的“小需求”不再不断的困扰住我们，让我们有时间进行更多的项目开发。

一、 插件基本知识

插件(Plugins)是什么？

插件就是可以用来插入到主程序体中执行的组件或组件的逻辑体，主要是用来在不修改主程序体的条件下扩展主程序体的各项功能的实体。

也许我们都有过这样一个经历，当我们做完一个程序后，发现我们设计的软件缺少某种功能，或者用户用过一段时间后提出某种新的要求，然后打开我们的“项目包”开始简单而又繁琐的工作，一个单元或者一个窗体的修改、增加软件的功能。

插件有很多种种类划分办法，但是不管你如何去划分它，究其根本无外乎有两种区别，其一、是通过其他程序解释或响应其逻辑的脚本型插件(Script - Plugins)，例如反恐中大量使用的AMX；其二、是自己可以独立于主程序体独立使用或者以作为PE、LE等等已编译格式在内存中动态或静态中载入的插件执行体的编译型插件(Compiled - Plugins)，例如我们一些常用的COM组件等等。

其实，插件是一种思想，而这种思想突出表现的就是让我们设计的程序能够更灵活一些，这个想法有点效仿COM的本意，但又不同于COM，因为插件有的时候并不会如同COM那样单纯，插件将比COM复杂，灵活。

使用插件的原意只有一个，“让复杂的问题分解成多个简单集，在更短的时间内创建一个构架，让这个构架可以不断的插入更多的应用，以促使应用得到延续。”

记住我们最初的愿望，下面开始对插件的具体学习。

二、已编译型插件基础知识

在我们学习WINDOWS程序设计的时候，都会学习到动态连接库这样一个内容，但是一般来说，我们在学习这一章节的时候都会不太深入的学习一下。不过这并不会妨碍你阅读本文的内容。因为我们会从头开始学习。

关于已编译型插件的几个要素问题。

- ❒ 插件的基本编写步骤
- ❒ 插件的插入方式
- ❒ 内存的管理问题
- ❒ 同步插件会话。
- ❒ 插件安全。

首先,让我们来看一个简单的插件程序。

```
Library Example1
{库文件标识}
Uses
 Sysutils,classes;

Function GetMsg:string;
Begin
 Result:='这是我的消息插件';
End;

Export
 GetMsg;
End.
```

上面的代码是一个很简单的动态链接库的代码,GetMsg是我们设计的简单的插件方法,(注意:凡需要导出的方法,即需要插件寄主响应的方法必须在Export中声明),那么我们如何在程序主体中使用这样一个简单的插件呢?

静态载入方式

```
unit LoadPlug

interface

function PlugGetMsg:stirng;stdcall;
//插件方法声明
implementation

function PlugGetMsg;external 'Example1.DLL' name
'GetMsg';
//插件载入

end.
```

上面这段代码是一个静态载入插件的主体程序,通过 function GetMsg;external 'Example1.DLL' name 'GetMsg';我们获得了一个内存转译的方式,也就是说原插件 Example1的 GetMsg方法在主体程序在 implementation静态载入后将系统描述附表中的GetMsg的地址“ 即内存地址方法指针 ”付给了主程序的静态常量PlugGetMsg(注意:这里的静态常量实际上是主程序的符号表,本地常量实际上是本地符号表,全局常量是全局符号表,实际上为了让不了解符号表的读者更好理解才这样来说, 而不是CONST中声明的常量), 在这里PlugGetMsg是一个本地的常量,也就是说大凡主程序单元中任何只要引用了LoadPlug单元的,都可以使用通过转译的GetMsg插件方法。

有的时候还可以通过序号的方式将其静态加入到静态程序体中。例如:

```
function PlugGetMsg;external 'Example1.dll' index 1;
```

也就是说静态载入具有两种方式,1、按名称方式引入,2、按序号方式引入。但是我们不推荐使用序号引入的方式,虽然序号引入的方式比名称引入的方式快,减少了查找名称对应关系的过程,但是一旦我们的插件体发生了异动,改变了export的顺序,那么整个调用体系将不再会正确的引入插件体,而名称引入相对安全,而且我们也可以更好的了解引入的具体方法的释义。

静态引入并不是我们最终在插件体中使用的型式,因为虽然静态方式解决了主程序体不能扩展的型式,但是它不够灵活,往往我们面向应用的扩展并不是可以预期的,当我们完成了程序体时,并不知道以后应用会是什么样,会不会需要主程序体与插件体的交互,而这正是本文急于要解决的问题,那就是在任何一种条件都能够使用的插件体,能够在我们想用的时候载入,不想使用的时候禁止。这样的东西有吗?

回顾一下上面的内容,理解了吗?好,我们继续。

在没有正式讲解问题之前,简要提及几个我们后面会用到的几个函数。

```
❖ DLLEntrypoint(
      hinstDLL: HINSTANCE;
      dwReason:DWORD;
      lpvReserved:LPVOID
  ):BOOL;
❖ DisableThreadLibraryCalls(
      hLibModule:HMODULE
  ):BOOL;
❖ FreeLibrary(
      hLibModule:HMODULE;
  );
❖ FreeLibraryAndExitThread(
      hLibModule:HMODULE;
      dwExitCode:DWORD;
  );
❖ GetModuleFileName(
      hMODULE:HINST;
      lpFileName:PChar;
      nSIZE:DWORD
  ):DWORD
❖ GetModuleHandle(
      lpModuleName:PChar
  ):HMODULE;
❖ GetProcAddress(
      hModule:HMODULE;
      ProcName:LPCSTR;
  ):FARPROC;
❖ LoadLibrary(
      lpLibFileName:Pchar;
  ):HMODULE;
❖ LoadLibraryEx(
      lpLibFileName:Pchar;
```

```
        hFile:Thandle;
        dwFlags:DWORD;
);
```

由于篇幅的限制就不在这里介绍这几个函数的具体作用了,你可以先查询WIN32API相关的文档,结合你查得的函数描述在我所提供的例程中具体去掌握它的应用;

让我们先来看一段动态载入并释放插件的例子:

```
0001  procedure Tform1.BtnLoadClick(Sender:TObject);
0002   var
0003     hMod:Thandle;
0004     ModuleFileName:array[0..255] of char;
0005     GetMsg:function:stirng;
0006     //一个插件体过程,回顾一下静态载入插件体;
0007   begin
0008     hMod:=LoadLibrary(Example1.dll');
0009     if (hMod=0) then Exit;
         //如果载入不成功则退出载入过程。
0010    @GetMsg=GetProcAddress(hMod,'GetMsg');
  //将Example1插件中的GetMsg方法地址付给本地方法地址。
0011     if (@GetMsg<>nil) then
0012     begin
0013       showMessage(GetMsg);//调用插件方法
0014       GetModuleFileName(GetModuleHandle('Example1.
            dll'),@ModuleFileName[0],
0015          Sizeof(ModuleFileName));
            //获得插件模组的名称
0016    Showmessage(ModuleFileName +' Plug is loading');
0017     End
0018     Else
0019       Showmessage('Plug Load is Failed');
0020       FreeLibrary(hMod);
0021   End;
```

在上面的程序体中0008行代表着将插件体装入到内存中,将装入的信息存放到hMod句柄变量中。LoadLibrary(或者LoadLibraryEx)是插件动态载入的必须方式也是唯一方式,请注意区别它与静态载入的方式,LoadLibrary中lpLibFileName参数是具体载入的插件库名称,由LoadLibrary载入后就会在内存创建lpLibFileName的内存映射。GetProcAddress将是我们获得插件方法的唯一手段。这是一个方法指针,该指针指向一个内存映射的方法地址,也就是要将载入的插件的方法地址付给本地方法;GetModuleFileName用来标示载入插件的包含路径在内的完整名称的,有的时候我们不使用它,因为用户并没有必要知道我们使用的文件位置;一定要记住在我们使用完毕后释放插件使用的内存地址空间或退出插件线程,这完全是出于对操作系统负责的考虑,以免造成内存异常的出现,FreeLibrary和FreeLibraryAndExitThread就是用来执行这一处理过程的手段。

好,让我们回顾一下载入插件的具体过程后继续。

LoadLibrary(插件)——GetProcAddress(获得方法)——FreeLibrary(释放插件)

按照上面的例子和说明自己尝试比较一下,看看是不是这样容易的就可以写一个插件。

为了更好的理解并掌握编译型插件在应用型软件中的编写技巧,让我们来看一个比较完整的可以动态管理插件的VCL组件。

```
{
插件组件1.0版
write by 3boy
eamil: 3boy@sohu.com
说明:
在本插件管理组件中使用了目录文件搜索方式和INI文件配置方式来确定所需载入的插件,可以通过选择不同的方式来载入插件。
基本方法有:
function InitPlugin(FilePath:string):TStrings; 初始化载入插件。
function LoadPlugins:Boolean; 载入插件列表
procedure IniPlugInsMenu(Plugins:TStrings); 初始化插件菜单项
procedure PlugInClike(sender:tobject); 定义插件响应处理过程
procedure Load(FActiveX:boolean); 激活插件管理
function ClosePlugins:boolean; 关闭插件列表
function Close:boolean; 关闭插件列表
}
unit Plugins;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs,inifiles,menus,DesignIntf,
  PropertyCategories,DesignEditors;
type
 TPluginInition=function(AHandel:THandle;
     ACaption:string):longint;stdcall;
 {1.0.1 增加了释放方法}
 TPluginDestion=procedure(Aref:longint);stdcall;
 {插件方法}
 IniError=class(Exception)
 public
  constructor Create(const Msg: string);overload;
//构造插件异常
 end;
 {属性页}
  TFreeLibraryEvent=procedure(sender:TObject;
      AHandel:Integer) of object;
  //释放过程指针
  TPlugFilePropertyEditor=class (TStringProperty)
//插件路径属性页
  public
  //创建属性页
 function GetAttributes: TPropertyAttributes; override;
  procedure GetValues(Proc: TGetStrProc); override;
```

```
    end;
  {异常方法}
  TLoadKind=(Ini,Auto);//建立插件打开方式集合
  TPlugIn=class
    public
      PlugInName: String;//插件输入名称
      PlugInTiTle:string;//插件在寄主程序中的显示名称
      PlugProcName:string;//插件的过程名称
      address: Integer;//插件的载入地址
      ProcShow:Pointer;//插件的方法指针
      ProcFree:Pointer; //插件的方法指针
   end;//创建一个用来保存插件的对象
   TPlugins = class(TComponent)
   private
     FActive:boolean;
     FPath:string;//文件路径
     FIniPath:string;//ini文件路径
     FPlugins:TStrings;//插件列表
     FLoadKind:TLoadKind;//打开方式
     FMenuItem:TMenuItem;//菜单单元
     FOnFreeLibrary:TFreeLibraryEvent;//释放过程指针
     { Private declarations }
   protected
  function InitPlugin(FilePath:string):TStrings;
  function LoadPlugins:Boolean;
     procedure IniPlugInsMenu(Plugins:TStrings);
     procedure PlugInClike(sender:tobject);
     procedure Load(FActiveX:boolean);
     function ClosePlugins:boolean;
     function Close:boolean;
   public
     constructor Create(AOwner: TComponent);overload;
     destructor Destroy;override;{ Public declarations }
   published
     property LoadKind:Tloadkind read FLoadKind write
FloadKind;
     property PlugMenuItem:TMenuItem read FMenuItem write
FMenuItem;
     property Plugpath:string read FPath write FPath ;
     property PlugInipath:string read FIniPath write
FIniPath;
  property OnFreeLibrary:TfreeLibraryEvent read
FOnFreeLibrary
  write FOnFreeLibrary;
     property Active:boolean read Close write Load;
      { Published declarations }
   end;
  procedure Register;
  implementation
  procedure Register;
  begin
   RegisterComponents('zVCL', [TPlugins]);
   RegisterPropertyEditor(Typeinfo(string),TPlugins,
'PlugInipath',TPlugFilePropertyEditor);
   RegisterPropertyEditor(Typeinfo(string),TPlugins,
'Plugpath',TPlugFilePropertyEditor);
    RegisterPropertiesInCategory('Path',TPlugins,
['Plugpath','PlugInipath']);
   [*注意:属性页的注册方式,RegisterPropertiesInCategory是用
来将某几个属性注册成一个Category,好处是在属性较多的时候便于
区别属性间的关系]
  end;
```

```
  { TPlugins }
  {SDK Ó¦ÓÃÀý³Ì
  这段代码主要描述编写的DLL文件中的内容,也就是相应载入
插件的插件模版,插件的方法,必须与插件管理其中的Loadlibrary方
法名称一致,也就是我们必须用愿名称取回内存映射的方法地址}
  { var
    TempHandel:THandle;
    function PluginInition(AHandle:Thandle;ACaption:
string):longint;stdcall;
    var
    Dllfrom:TFrmPlugManager;
    begin
    // 加入自己的处理代码
     Application.Handle:=AHandle;
     DllFrom:=TFrmPlugManager.Create(Application);
     Result:=longint(DllFrom);
     Dllfrom.Show;
    end;
    procedure PlugInDestion(Aref:Longint);stdcall
    begin
     if Aref>0 then
    TFrmPlugManager(Aref).Release;
    end;
    exports
    PluginInition,PlugInDestion;
    //dll 's form 一定要加入在关闭窗口的时候释放窗体
    procedure TFrmPlugManager.FormClose(Sender: TObject;
    var Action: TCloseAction);
    begin
      Action:=cafree;
     TFrmPlugManager(sender):=nil;//一定要将插件单元中
创建的无模式窗体释放掉,不然会引起内存异常的情况出现。
    end;
  }
  {1.0.1增加取文件非扩展名的名称}
  function ExtractUnExtFileName(filename:string):string;
  var
   i:integer;
  begin
   i:=Pos('.DLL',UpperCase(filename));
  //当然插件的后缀可以根据具体的要求改写成其他的文件后缀
名称。
   Result:=Copy(filename,0,i-1);
  end;
  {去掉末尾的扩展名}

  function TPlugins.Close : boolean;
  begin
   Result:=FActive;
  end;

  function TPlugins.ClosePlugins: boolean;
  var
   i:integer;
  begin
   //释放插件,在不再使用插件的时候将所有的插件所占用
的系统资源释放掉。
   if FPlugins<>nil then
   begin
   if FPlugins.Count>0 then
   begin
    for i:=0 to FPlugins.Count-1 do
```

```
      begin
       if FPlugins.Objects[i]<>nil then
        begin
        try
       FreeLibrary(TPlugIn(FPlugins.Objects[i]).address);
        except
          on E:Exception do
        IniError.Create(E.Message);
        end;
       FPlugins.Objects[i].Free;
       FPlugins.Objects[i]:=nil;
        end;
      end;
      // Result:=false;
       FPlugins.Free;
       FPlugins:=nil;
      end;
      end;
    end;
    constructor TPlugins.Create(AOwner: TComponent);
    begin
      //
      inherited ;
    end;
    destructor TPlugins.Destroy;
    begin
      inherited ;
    end;
    procedure TPlugins.IniPlugInsMenu(Plugins: TStrings);
    var
     i:integer;
     PluginsMenu:TMenuItem;
    begin
      with Plugins do
      begin
       for i:=0 to count-1 do
       begin
         PluginsMenu:=TMenuItem.Create(self);
           PluginsMenu.Caption:=TPlugIn(Objects[i]).
PlugInTiTle;
         PluginsMenu.OnClick:=PlugInClike;
         //定义执行插件方法响应的过程。
         PluginsMenu.Tag:=i;
         FMenuItem.Add(PluginsMenu);
       end;
      end;
    end;

    function TPlugins.InitPlugin(FilePath: string):
TStrings;
    var
        IniFile:TIniFile;
        i:integer;
        PlugIn:TPlugIn;
        FileName:string;
        Founds: TSearchRec;
        Sub:string;
        founded:integer;
        Stop:boolean;
        Dirs:TStrings;
       begin
        case FLoadKind of
          Ini:
          Begin
           //ini初始化插件方式,可以更好的,更安全的确定所
要载入的插件。
           FileName:=FilePath+'Plugins.ini';
           if FileExists(FileName) then
           begin
            FPlugins.Clear;
           IniFile:=TIniFile.Create(FileName);
           IniFile.ReadSections(FPlugins);
           for i:=0 to FPlugins .Count-1 do
           begin
             Plugin:=TPlugIn.Create;
               Plugin.PlugInName:=IniFile.ReadString
(FPlugins.Strings[i],'FileName','');
              Plugin.PlugInTiTle:=IniFile.ReadString
(FPlugins.Strings[i],'FileTiTle','');
      //不用在外部引出的函数中
    Plugin.PlugProcName:=IniFile.ReadString(FPlugins.
Strings[i],'ProcName','');
             FPlugins.Objects[i]:=PlugIn;
            end;
          end;
           //载入Ini文件结束
          Result:=FPlugins;
          end;
          Auto:begin
           //自动加载,进行指定目录搜索,找出所有以dll为后
缀得插件文件。
             stop:=false;
             FPlugins.Clear;
              founded:=FindFirst(FilePath+'*.dll',63,
Founds);
             while (founded=0) and not(stop) do
             begin
              if (Founds.Name[1]<>'.') then
                 begin
                   PlugIn:=TPlugIn.Create;
                   PlugIn.PlugInName:=Founds.Name;
                            PlugIn.PlugInTiTle:
=ExtractUnExtFileName(Founds.Name);
                       FPlugins.AddObject(PlugIn.
PlugInName,PlugIn);
                 end;
                founded:=FindNext(founds);
             end;
             FindClose(Founds);
             //搜索目录
              Result:=FPlugins;
          end;
          end;
    end;
    procedure TPlugins.Load(FActiveX: boolean);
    begin
      //
      case FActiveX of
      True:begin
        if FPath<>'' then
        begin
         FPlugins:=TStringList.Create;
         if LoadPlugins then
            begin
```

```
              IniPlugInsMenu(FPlugins);
              FActive:=true;
            end;
        {载入插件和插件菜单}
        end;
      end;
      false:begin
          FActive:=ClosePlugins;
      end;

      end;
    end;

    function TPlugins.LoadPlugins: Boolean;
     var
          i:integer;
          LibHandle:Integer;
          FilePath,FileName:String;
          Flfile:TextFile;
        begin
          FilePath:=FPath;
          Result:=False;
         with InitPlugin(FIniPath) do
         begin
          if count<0 then
    raise IniError.Create('PlugIni文件没有被打开或则该文件并
不存在');
          for i:=0 to count-1 do
           begin
              FileName:=FilePath+TPlugIn(Objects[i]).
PluginName;
            if FileExists(FileName) then begin
            //检查插件是否存在。
             TPlugIn(Objects[i]).Address:=LoadLibrary
(PChar(FileName));
            if TPlugIn(Objects[i]).Address=0 then raise
IniError.Create('PlugIn无法打开')
    else
             begin
               TPlugIn(Objects[i]).ProcShow:=
    GetProcAddress(TPlugIn(Objects[i]).address,
'PluginInition');
    //打开插件方法,装入插件列表中的对象过程;
    TPlugIn(Objects[i]).ProcFree:=
    GetProcAddress(TPlugIn(Objects[i]).address,
'PlugInDestion');       //1.0.1用于在主调程序中释放的过程
                IniError.Create('$'+inttostr(TPlugIn
(Objects[i]).address));
             end;
            end else begin
              IniError.Create('PlugIn文件并不存在');
            end;
           end;
          end;
          Result:=True;
        end;

    procedure TPlugins.PlugInClike(sender: TObject);
    var
     PluginInition:TPluginInition;
    begin
      @PluginInition:=TPlugIn(FPlugins.Objects[TMenuitem
(sender).Tag]).ProcShow;
       [*注意@xxxx表示的是某某过程的地址指针]
       PluginInition(Application.Handle,
    TPlugIn(FPlugins.Objects[TMenuitem(sender).Tag]).
PlugInTiTle);
       //将过程地址付给相应的对象;
    end;

    { TPlugFilePropertyEditor }
    //属性页获得属性地址,付给Fpath;
    function TPlugFilePropertyEditor.GetAttributes:
TPropertyAttributes;
    begin
      Result:=inherited GetAttributes+ [paDialog];
    end;

    procedure TPlugFilePropertyEditor.GetValues(Proc:
TGetStrProc);
    var
     FileDilag:TOpenDialog;
    begin
      FileDilag:=TOpenDialog.Create(nil);
       try
      if FileDilag.Execute then
      begin
        Proc(ExtractFilePath(FileDilag.FileName));
      end;
      finally
       FileDilag.Free;
      end;
    end;
    { IniError }
    constructor IniError.Create(const Msg: string);
    var
     ErrorFile:TextFile;
     FHandel:Integer;
    begin
      if not FileExists('C:\Plug.log') then begin
      //创建日志,检查文件是否存在
      FHandel:=FileCreate('C:\Plug.log');
      FileClose(FHandel);
      end;
      AssignFile(ErrorFile,'C:\plug.log');
      Append(ErrorFile);
      Writeln(ErrorFile,FormatDateTime('[yyyy-mm-dd hh:nn:
ss]',now)+msg);
      CloseFile(ErrorFile);
       inherited;
    end;
    //创建异常日志
    end.
```

为了让组件的功能更加强大,我们可以不断的加入一些属性和方法,在稍后我们将加入线程控制,内存块管理,文件加密/解密压缩,等等更加复杂的方式来使我们的插件管理器组件更加强大。

今天就到这里了,好戏还在后头。

Java Learning Path
——带你走上Java开发之路

▶撰文/范凯

Java Learning Path（一）、工具篇

一、JDK(Java Development Kit)

JDK是整个Java的核心，包括了Java运行环境(Java Runtime Envirnment)，一堆Java工具和Java基础的类库(rt.jar)。不论什么Java应用服务器实质都是内置了某个版本的JDK。因此掌握JDK是学好Java的第一步。最主流的JDK是Sun公司发布的JDK，除了Sun之外，还有很多公司和组织都开发了自己的JDK，例如IBM公司开发的JDK，BEA公司的Jrocket，还有GNU组织开发的JDK等等。其中IBM的JDK包含的JVM（Java Virtual Machine）运行效率要比Sun JDK包含的JVM高出许多。而专门运行在x86平台的Jrocket在服务端运行效率也要比Sun JDK好很多。但不管怎么说，我们还是需要先把Sun JDK掌握好。

1、JDK的下载和安装

JDK的一个常用版本J2SE（Java2 SDK Standard Edition），可以从Sun的Java网站上下载到，http://java.sun.com/j2se/downloads.html，JDK当前最新的版本是J2SDK1.4.2，建议下载该版本的JDK，下载页面在这里：http://java.sun.com/j2se/1.4.2/download.html。

下载好的JDK是一个可执行安装程序，默认安装完毕后会在C:\Program Files\Java\目录下安装一套JRE（供浏览器来使用），在C:\j2sdk1.4.2下安装一套JDK（也包括一套JRE）。然后我们需要在环境变量PATH的最前面增加java的路径C:\j2sdk1.4.2\bin。这样JDK就安装好了。

2、JDK的命令工具

JDK最重要的命令行工具：

java：启动JVM执行class

javac：Java编译器

jar：Java打包工具

javadoc：Java文档生成器

这些命令行必须要非常非常熟悉，对于每个参数都要很精通才行。对于这些命令的学习，JDK Documentation上有详细的文档。

二、JDK Documentation

Documentation在JDK的下载页面也有下载连接，建议同时下载Documentation。Documentation是最最重要的编程手册，涵盖了整个Java所有方面的内容的描述。可以这样说，学习Java编程，大部分时间都是花在看这个Documentation上面的。总之我是随身携带它的，写Java代码的时候，随时查看，须臾不离手。

三、应用服务器(App Server)

App Server是运行Java企业组件的平台，构成了应用软件的主要运行环境。当前主流的App Server是BEA公司的Weblogic Server和IBM公司的Websphere以及免费的Jboss，选择其中一个进行学习就可以了，个人推荐Weblogic，因为它的体系结构更加干净，开发和部署更加方便，是Java企业软件开发人员首选的开发平台。下面简要介绍几种常用的App Server：

1、Tomcat

Tomcat严格意义上并不是一个真正的App Server，它只是一个可以支持运行Serlvet/JSP的Web容器，不过Tomcat也扩展了一些App Server的功能，如JNDI，数据库连接池，用户事务处理等等。Tomcat被非常广泛的应用在中小规模的Java Web应用中，因此本文做一点下载、安装和配置Tomcat的介绍：

Tomcat是Apache组织Jakarta项目下的一个子项目，它的主网站是：http://jakarta.apache.org/tomcat/，Tomcat最新版本是Tomcat4.1.27，软件下载的连接是：http://www.apache.org/dist/jakarta/tomcat-4/binaries/。

下载Tomcat既可以直接下载zip包，也可以下载exe安装包（个人建议zip更干净些），不管哪种情况，下载完毕安装好以后（zip直接解压缩就可以了）。需要设置两个环境变量：

```
JAVA_HOME=C:\j2sdk1.4.2
CATALINA_HOME=D:\tomcat4（你的Tomcat安装目录）
```

这样就安装好了，启动Tomcat运行CATALINA_HOME\bin\startup.bat，关闭Tomcat运行shutdown.bat脚本。Tomcat启动以后，默认使用8080端口，因此可以用浏览器访问http://localhost:8080来测试Tomcat是否正常启动。

Tomcat提供了两个Web界面的管理工具，URL分别是：

http://fankai:8080/admin/index.jsp

http://fankai:8080/manager/html

在启用这两个管理工具之前，先需要手工配置一下管理员用户和口令。用一个文本工具打开CATALINA_HOME\conf\tomcat-users.xml这个文件，加入如下几行：

```
<role rolename="manager"/>
<role rolename="admin"/>
<user username="robbin" password="12345678"
      roles="admin,manager,tomcat"/>
```

这样用户“robbin”就具备了超级管理员权限。重新启动Tomcat以后，你就可以使用该用户来登陆如上的两个管理工具，通过Web方式进行Tomcat的配置和管理了。

2、BEA Weblogic

Weblogic可以到BEA的网站上免费注册之后下载到最新的Weblogic8.1企业版，License可以免费使用1年时间，其实这已经完全足够了。Weblogic的下载连接：http://commerce.bea.com/index.jsp，Weblogic的在线文档：http://edocs.bea.com/。

3、IBM Webshpere

Websphere同样可以下载到免费的试用版本，到IBM的developerWorks网站可以看到Websphere试用产品的下载和相关的Websphere的资料，developerWorks中文网站的连接是：http://www-900.ibm.com/developerWorks/cn/wsdd/，Websphere的下载连接：http://www7b.software.ibm.com/wsdd/downloads/WASsupport.html 。

4、Jboss

Jboss是免费开源的App Server，可以免费的从Jboss网站下载：http://www.jboss.org/index.html，然而Jboss的文档是不免费，需要花钱购买，所以为我们学习Jboss设置了一定的障碍。在Jdon上有几篇不错的Jboss配置文档，可以用来参考：http://www.jdon.com/idea.html

四、Java应用的分类及其运行环境

Java的应用可以简单分为以下几个方面：

1、Java的桌面应用

桌面应用一般仅仅需要JRE的支持就足够了。

2、Java Web应用

Java的Web应用至少需要安装JDK和一个web容器（例如Tomcat），以及一个多用户数据库，Web应用至少分为三层：

Browser层：浏览器显示用户页面

Web层：运行Servlet/JSP

DB层：后端数据库，向Java程序提供数据访问服务

3、Java企业级应用

企业级应用比较复杂，可以扩展到n层，最简单情况会分为4层：

Browser层：浏览器显示用户页面

Client层：Java客户端图形程序（或者嵌入式设备的程序）直接和Web层或者EJB层交互

Web层：运行Servlet/JSP

EJB层：运行EJB，完成业务逻辑运算

DB层：后端数据库，向Java程序提供数据访问服务

4、Java嵌入式应用

Java嵌入式应用是一个方兴未艾的领域，从事嵌入式开发，需要从Sun下载J2ME开发包，J2ME包含了嵌入式设备专用虚拟机KVM，和普通JDK中包含的JVM有所不同。另外还需要到特定的嵌入式厂商那里下载模拟器。

Java Learning Path（二）、书籍篇

学习一门新的知识，不可能指望只看一本，或者两本书就能够完全掌握。需要有一个循序渐进的阅读过程。我推荐Oreilly出版的Java系列书籍。

在这里我只想补充一点看法，很多人学习Java是从《Thinking in Java》这本书入手的，但是我认为这本书是不适合初学者的。我认为正确的使用这本书的方法应该是作为辅助的读物。《Thinking in Java》并不是在完整的介绍Java的整个体系，而是一种跳跃式的写作方法，是一种类似tips的方法来对Java很多知识点进行了深入的分析和解释。

对于初学者来说，最好是找一本Java入门的书籍，可以比较完整的循序的介绍Java的语法、面向对象的特性、核心类库等等，在阅读此书的同时，可以同步来看

《Thinking in Java》，以加深对Java的理解和原理的运用，同时又可以完整的了解Java的整个体系。

对于Java的入门书籍，蔡学镛推荐的是Oreilly的《Exploring Java，2nd Edition》或者《Java in a Nutshell，2nd Edition（针对C++背景）》，我并没有看过这两本书。其实我觉得电子工业出版社的《Java 2编程详解》或者《Java 2从入门到精通》就很不错。

在所有的Java书籍当中，其实最最有用的，并不是O'reilly的 Java Serials，真正最最有用处是JDK的Documentation！几乎你想获得的所有的知识在Documentation里面全部都有，其中最主要的部分当然是Java基础类库的API文档，是按照package来组织的，对于每一个class都有详细的解释，它的继承关系，是否实现了某个接口，通常用在哪些场合，还可以查到它所有的public的属性和方法，每个属性的解释，意义，每个方法的用途，调用的参数，参数的意义，返回值的类型，以及方法可能抛出的异常等等。可以这样来说，所有关于Java编程方面的书籍其实都不过是在用比较通俗易懂的语言，和良好的组织方式来介绍Documentation里面的某个package里面包含的一些类的用法而已。所以万变不离其宗，如果你有足够的能力来直接通过Documentation来学习Java的类库，那么基本上就不需要看其他的书籍了。除此之外，Documentation也是编程必备的手册，我的桌面上有三个Documentation的快捷方式，分别是J2SDK1.4.1的Documentation，Servlet2.3的Documentation和J2SDKEE1.3.1的Documentation。有了这个三个Documentation，什么其他的书籍都不需要了。

对于Java Web编程来说，最核心的是要熟悉和掌握HTTP协议，这个就和Java无关了，在熟悉HTTP协议之后，就需要熟悉Java实现HTTP协议的类库，也就是Servlet API，所以最重要的东西就是Servlet API。当然对于初学者而言，直接通过Servlet API来学习Web编程有很大的难度，我推荐O'reilly的《Java Server Pages》这本书来学习Web编程。

EJB的书籍当中，《Enterprise JavaBeans，2nd Edition》是一本很不错的书，EJB的学习门槛是比较高，入门很难，但是这本书完全降低了学习的难度，特别重要的一点是，EJB的学习需要结合一种App Server的具体实现，所以在学习EJB的同时，必须同步的学习某种App Server。与此相关的图书有三本，它们分别是在Weblogic6.1、Websphere4.0和JBoss3.0三种Server上部署EJB的实践。真是既有理论，又有实施。在学习EJB的同时，可以边看边做，EJB的学习会变得很轻松。

但是这本书也有一个问题，就是版本比较旧，主要讲EJB1.1规范和部分EJB2.0的规范。而Ed Roman写的《Mastering EJB 2.0》这本书完全是根据EJB2.0规范写的，深入浅出，覆盖了EJB编程的各个方面，并且还有很多编程经验tips，也是学习EJB强烈推荐的书籍之一。

如果是结合Weblogic来学习J2EE，《J2EE应用与BEA Weblogic Server》绝对是首选读物，虽然讲述的是Weblogic6.0，仍然值得购买，这本书是BEA官方推荐的教材，作者也是BEA公司的工程师。现在中文版已经随处可见了。这本书结合Weblogic介绍了J2EE各个方面的技术在Weblogic平台上的开发和部署，实践指导意义非常强。

在掌握了Java平台基础知识和J2EE方面的知识以后，更进一步的是学习如何运用OO的方法进行软件的设计，那么就一定要学习“设计模式”。Sun公司出版了一本《J2EE核心模式》，是每个开发Java企业平台软件的架构师必备的书籍。这本书全面的介绍了J2EE体系架构的各种设计模式，是设计师的必读书籍。

Java Learning Path（三）、过程篇

每个人的学习方法是不同的，一个人的方法不见得适合另一个人，我只能是谈自己的学习方法。因为我学习Java是完全自学的，从来没有问过别人，所以学习的过程基本上完全是自己摸索出来的。我也不知道这种方法是否是比较好的方法，只能给大家提供一点参考了。

学习Java的第一步是安装好JDK，写一个Hello World，:）其实JDK的学习没有那么简单，关于JDK有两个问题是一直困扰Java程序员的地方：一个是CLASSPATH的问题，其实从原理上来说，是要搞清楚JRE的ClassLoader是如何加载Class的；另一个问题是package和import问题，如何来寻找类的路径问题。把这两个问题摸索清楚了，就扫除了学习Java和使用JDK的最大障碍。推荐看一下王森的《Java深度历险》，对这两个问题进行了深入的探讨。

第二步是学习Java的语法。Java的语法是类C++的，基本上主流的编程语言不是类C，就是类C++的，没有什么新东西，所以语法的学习，大概就是半天的时

间足够了。唯一需要注意的是有几个不容易搞清楚的关键字的用法，public, protected, private, static, 什么时候用，为什么要用，怎么用，这可能需要有人来指点一下，我当初是完全自己琢磨出来的，花了很久的时间。不过后来我看到《Thinking in Java》这本书上面是讲了这些概念的。

第三步是学习Java的面向对象的编程语言的特性的地方。比如继承，构造器，抽象类，接口，方法的多态，重载，覆盖，Java的异常处理机制。对于一个没有面向对象语言背景的人来说，我觉得这个过程需要花很长很长时间，因为学习Java之前没有C++的经验，只有C的经验，我是大概花了一个月左右吧，才彻底把这些概念都搞清楚，把书上面的例子反复的揣摩，修改，尝试，把那几章内容反复的看过来，看过去，看了不下5遍，才彻底领悟了。不过我想如果有C++经验的话，应该一两天时间足够了。那么在这个过程中，可以多看看《Thinking in Java》这本书，对面向对象的讲解非常透彻。可惜的是我学习的时候，并没有看到这本书，所以自己花了大量的时间，通过自己的尝试和揣摩来学会的。

第四步就是开始熟悉Java的类库。Java的基础类库其实就是JDK安装目录下面jre\lib\rt.jar这个包。学习基础类库就是学习rt.jar。基础类库里面的类非常非常多。据说有3000多个，我没有统计过。但是真正对于我们来说最核心的只有4个，分别是

java.lang.*;
java.io.*;
java.util.*;
java.sql.*;

对这四个包的学习，每个都可以写成一本厚厚的教材，而O'reilly也确实是这样做的。我觉得如果时间比较紧，是不可能通过读四本书来学习。我觉得比较好的学习方法是这样的：

首先要通读整个package的框架，了解整个package的class, interface, exception的构成，最好是能够找到介绍整个包框架的文章。这些专门介绍包的书籍的前几章应该就是这些总体的框架内容介绍。

对包整体框架的把握并不是要熟悉每个类的用法，记住它有哪些属性，方法。想记也记不住的。而是要知道包有哪些方面的类构成的，这些类的用途是什么，最核心的几个类分别是完成什么功能的。我在给人培训的时候一般是一次课讲一个包，所以不可能详细的介绍每个类的用法，但是我反复强调，我给你们讲这些包不是要告诉你们类的方法是怎么调用的，也不要求你们记住类的方法调用，而是要你们了解，Java给我们提供了哪些类，每个类是用在什么场合，当我遇到问题的时候，我知道哪个类，或者哪几个类的组合可以解决我的问题，That'all！，当我们具体写程序的时候，只要你知道该用哪个类来完成你的工作就足够了。编码的时候，具体的方法调用，是边写代码，边查Documentation，所有的东西都在Documentation里面，不要求你一定记住，实际你也记不住3000多个类的总共将近10万个方法调用。所以对每个包的总体框架的把握就变得极为重要。

第五步，通过上面的学习，如果学的比较扎实的话，就打好了Java的基础了，剩下要做的工作是扫清Documentation里面除了上面4个包之外的其他一些比较有用处的类。相信进展到这一步，Java的自学能力已经被培养出来了，可以到了直接学习Documentation的水平了。除了要做GUI编程之外，JDK里面其他会有用处的包是这些：

java.text.*;
java.net.*;
javax.naming.*;

这些包里面真正用的比较多的类其实很少，只有几个，所以不需要花很多时间。

第六步，Java Web 编程

Web编程的核心是HTTP协议，HTTP协议和Java无关，如果不熟悉HTTP协议的话，虽然也可以学好Servlet/JSP编程，但是达不到举一反三，一通百通的境界。所以HTTP协议的学习是必备的。如果熟悉了HTTP协议的话，又有了Java编程的良好的基础，学习Servlet/JSP简直易如反掌，我学习Servlet/JSP就用了不到一周的时间，然后就开始用JSP来做项目了。

在Servlet/JSP的学习中，重头仍然是Servlet Documentation。Servlet API最常用的类很少，花比较少的时间就可以掌握了。把这些类都看一遍，多写几个例子试试。Servlet/JSP编程本质就是在反复调用这些类来通过HTTP协议在Web Server和Brower之间交谈。另外对JSP，还需要熟悉几个常用JSP的标记，具体的写

法记不住的话,临时查就是了。

此外Java Web编程学习的重点要放在Web Application的设计模式上,如何进行业务逻辑的分析,并且进行合理的设计,按照MVC设计模式的要求,运用Servlet和JSP分别完成不同的逻辑层,掌握如何在Servlet和JSP之间进行流程的控制和数据的共享,以及Web Application应该如何配置和部署。

第七步,J2EE编程

以上的学习过程如果是比较顺利的话,进行到这一步,难度又陡然提高。因为上面的知识内容都是只涉及一个方面,而像EJB, JMS, JTA等核心的J2EE规范往往是几种Java技术综合运用的结晶,所以掌握起来难度比较大。

首先一定要学习好JNDI, JNDI是App Server定位服务器资源(EJB组件, Datasouce, JMS)查找方法,如果对JNDI不熟悉的话, EJB, JMS这些东西几乎学不下去。JNDI其实就是javax.naming.*这个包,运用起来很简单。难点在于服务器资源文件的配置。对于服务器资源文件的配置,就需要看看专门的文档规范了,比如web.xml的写法, ejb-jar.xml的写法等等。针对每种不同的App Server,还有自己的服务资源配置文件,也是需要熟悉的。

然后可以学习JTA,主要是要理解JTA对于事务控制的方法,以及该在什么场合使用JTA。这里可以简单的举个例子,我们知道一般情况可以对于一个数据库连接进行事务控制(conn.setAutoCommit(false),....,conn.commit()),做为一个原子操作,但是假设我的业务需求是要把对两个不同数据库的操作做为一个原子操作,你能做的到吗?这时候只能用JTA了。假设操作过程是先往A数据库插一条记录,然后删除B数据库另一个记录,我们自己写代码是控制不了把整个操作做为一个原子操作的。用JTA的话,由App Server来完成控制。

在学习EJB之前要学习对象序列化和RMI, RMI是EJB的基础。接着学习JMS和EJB,对于EJB来说,最关键是要理解EJB是如何通过RMI来实现对远端对象的调用,以及在什么情况下要用到EJB。

在学习完EJB、JMS这些东西之后,你可能会意识到要急不可待学习两个领域的知识,一个是UML,另一个是Design Pattern。Java企业软件的设计非常重视框架(Framework)的设计,一个好的软件框架是软件开发成功的必要条件。在这个时候,应该开始把学习的重点放在设计模式和框架的学习上,通过学习和实际的编程经验来掌握EJB的设计模式和J2EE的核心模式。

J2EE规范里面,除了EJB、JMS、JTA、Servlet/JSP、JDBC之外还有很多很多的企业技术,这里不一一进行介绍了。

另外还有一个最新领域Web Services。Web Services也完全没有任何新东西,它像是一种黏合剂,可以把不同的服务统一起来提供一个统一的调用接口,作为使用者来说,我只要获得服务提供者给我的WSDL(对服务的描述),就够了,我完全不知道服务器提供者提供的服务究竟是EJB组件,还是.Net组件,还是什么CORBA组件,还是其他的什么实现,我也不需要知道。Web Services最伟大的地方就在于通过统一的服务提供方式和调用方式,实现了整个Internet服务的共享,是一个非常令人激动的技术领域。Web Services好像目前还没有什么很好的书籍,但是可以通过在网络上面查资料的方式来学习。

Java Learning Path(四)、方法篇

Java作为一门编程语言,最好的学习方法就是写代码。当你学习一个类以后,你就可以自己写个简单的例子程序来运行一下,看看有什么结果,然后再多调用几个类的方法,看看运行结果,这样非常直观的把类给学会了,而且记忆非常深刻。然后不应该满足把代码调通,你应该想想看如果我不这样写,换个方式,再试试行不行。记得哪个高人说过学习编程就是个破坏的过程,把书上的例子,自己学习Documentation编写的例子在运行通过以后,不断的尝试着用不同的方法实现,不断的尝试破坏代码的结构,看看它会有什么结果。通过这样的方式,你会很彻底的很精通的掌握Java。

举个例子,我们都编过Hello World:

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

很多初学者不是很理解为什么main方法一定要这样来定义public static void main(String[] args),能不能不这样写?包括我刚学习Java的时候也有这样的疑问。

想知道答案吗？很简单,你把main改个名字运行一下,看看报什么错误,然后根据出错信息进行分析;把main的public取掉,再试试看,报什么错误;static去掉还能不能运行;不知道main方法是否一定要传一个String[]数组的,把String[]改掉,改成int[],或者String试试看;不知道是否必须写args参数名称的,也可以把args改成别的名字,看看运行结果如何。

我当初学习Java的时候就是这样做的,把Hello World程序反复改了七八次,不断运行,分析运行结果,最后就彻底明白为什么main方法是这样定义的了。

此外,我对于staic、public、private、Exception、try{ }、catch {}、finally{}等等一开始都不是很懂,都是把参考书上面的例子运行成功,然后就开始破坏它,不断的根据自己心里面的疑问来重新改写程序,看看能不能运行,运行出来是个什么样子,是否可以得到预期的结果。这样虽然比较费时间,不过一个例子程序这样反复破坏几次之后。我就对这个相关的知识彻底学通了。有时候甚至故意写一些错误的代码来运行,看看能否得到预期的运行错误。这样对于编程的掌握是及其深刻的。

其中特别值得一提的是JDK有一个非常棒的调试功能:

```
java -verbose
```

其它很多JDK工具都有这个选项,-verbose可以显示在命令执行的过程中,JVM都依次加载哪些Class,通过这些宝贵的调试信息,可以帮助我们分析出JVM在执行的过程中都干了些什么。

另外,自己在学习过程中,写的很多的这种破坏例程,应该有意识的分门别类的保存下来,在工作中积累的典型例程也应该定期整理,日积月累,自己就有了一个代码库了。遇到类似的问题,到代码库里面Copy & Paste, Search & Replace,就好了,极大提高了开发速度。最理想的情况是把一些通用的例程自己再抽象一层,形成一个通用的类库,封装好。那么可复用性就更强了。

所以我觉得其实不是特别需要例程的,自己写的破坏例程就是最好的例子,如果你实在对自己写的代码不放心的话,我强烈推荐你看看JDK基础类库的Java源代码。在JDK安装目录下面会有一个src.zip,解开来就可以完整的看到整个JDK基础类库,也就是rt.jar的Java源代码,你可以参考一下Sun是怎么写Java程序的,规范是什么样子的。我自己在学习Java的类库的时候,当有些地方理解的不是很清楚的时候,或者想更加清晰的理解运作的细节的时候,往往会打开相应的类的源代码,通过看源代码,所有的问题都会一扫而空。

Java Learning Path（五）、资源篇

1、http://java.sun.com/（英文）

Sun的Java网站,是一个应该经常去看的地方。不用多说。

2、http://www-900.ibm.com/developerWorks/cn/

IBM的developerWorks网站,英语好的直接去英文主站点看。这里不但是一个极好的面向对象的分析设计网站,也是Web Services, Java, Linux极好的网站。强烈推荐！！！

3、http://www.javaworld.com/（英文）

关于Java很多新技术的讨论和新闻。想多了解Java的方方面面的应用,这里比较好。

4、http://dev2dev.bea.com.cn/index.jsp

BEA的开发者园地,BEA作为最重要的App Server厂商,有很多独到的技术,在Weblogic上做开发的朋友不容错过。

5、http://www.huihoo.com/

灰狐动力网站,一个专业的中间件网站,虽然不是专业的Java网站,但是在J2EE企业应用技术方面有深厚的造诣。

6、http://www.theserverside.com/home/（英文）

TheServerSide是一个著名的专门面向Java Server端应用的网站。

7、http://www.javaresearch.org/

Java研究组织,有很多优秀的Java方面的文章和教程,特别是在JDO方面的文章比较丰富。

8、http://www.cnjsp.org/

JSP技术网站,有相当多的Java方面的文章和资源。

9、http://www.jdon.com/

Jdon论坛,是一个个人性质的中文J2EE专业技术论坛,在众多的Java中文论坛中,Jdon一个是技术含量非常高,帖子质量非常好的论坛。

10、http://sourceforge.net/

SourgeForge是一个开放源代码软件的大本营,其中也有非常非常丰富的Java开放源代码的著名软件。

采用扩展机制增强Java平台

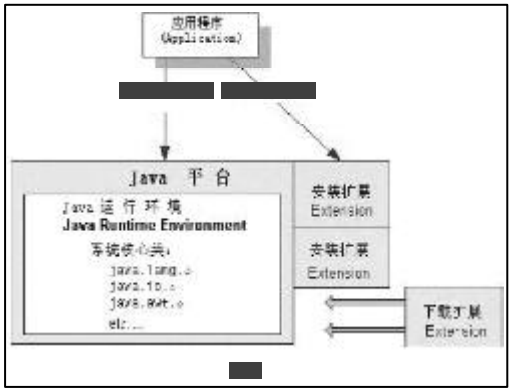
▶撰文/李利　王秀峰

一、引言

从1.2版开始,Java平台引入了一种机制——扩展(Extension),这种机制提供了一种标准的、方便的方式,以使在同一Java平台上运行的应用程序都能使用客户提供的(非平台本身提供的)API,这一机制更加强了"Write Once, Run Anywhere"的理念。对我们自己所写的类施以扩展之后,不用在CLASSPATH环境变量上添加这些类的路径,运行环境也能找到并加载这些类。从这一点看,通过扩展,我们自己所写的类就像Java平台提供的核心类一样,在此平台上部署的所有应用程序都可以象使用Java核心类那样直接使用它们,这些类犹如对Java平台核心API进行了扩展,增强了Java平台的功能。

二、何谓扩展

如图1所示,"扩展"就是将类打包成"JAR"文件,成为Java平台的可添加的模块,其中的类和公开的API对于运行在Java平台上的任何应用程序都是可以直接使用的,也就是不用特别指明这些类的路径。相当于使我们自己写的API和java运行环境自带的API处于同等的地位。尽管在一般项目中,我们极少通过扩展机制来部署java应用,但是通过这一点,我们至少可以认清JDK的确是一个开放的,不断发展的应用开发包。



三、扩展机制的背后

为什么使用"扩展"就不用指明类路径了呢?这与Java平台(1.2版之后)的新类加载机制有关(参见http://java.sun.com/docs/books/tutorial/ext/basics/load.html)。当需要为应用程序加载一个新的类时,运行环境从以下位置并且按以下顺序搜索这个新类:

1. 引导(Bootstrap)类:rt.jar文件中的运行时类以及i18n.jar文件中的国际化类。

2. JRE中lib/ext目录下JAR文件中的类(即下面文中所指的"安装扩展")。

3. 指明的类路径:系统属性java.class.path所指明路径下的类以及这些路径下JAR文件中的类。如果类路径中JAR文件的manifest(清单)文件带有Class-Path属性,那么Class-Path所指定的类(即下面文中所指的"下载扩展")也会被搜索。默认情况下,java.class.path属性的值是".",即当前路径,我们可以通过设置环境变量"CLASSPATH"或者在命令行中使用"-classpath"或"-cp"选项改变这个属性值。

按照上面的顺序,我们可知只有在rt.jar、i18n.jar以及"安装扩展"和"下载扩展"中没有找到所需的类时,才会在CLASSPATH所提供的类路径中进行搜索。我们采用扩展机制时,由于运行环境自动从"安装扩展"或"下载扩展"中加载所需的类,因此,就不用再特别指明类路径了。

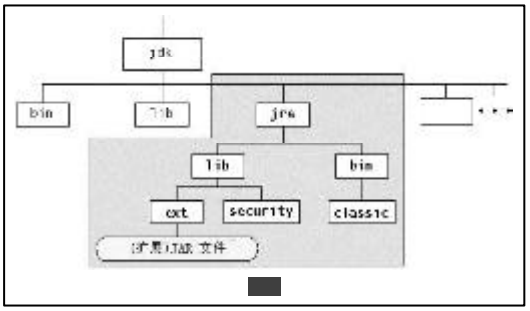
四、如何进行扩展

有两种扩展方式——安装方式扩展和下载方式扩展,分别适用于不同的环境。为便于表述,我们称用于安装方式扩展的类为"安装扩展",用于下载方式扩展的类为"下载扩展"。

方式一:安装方式扩展

一般情况,Java开发工具包(Java Development Kit, JDK,可以到http://java.sun.com/j2se/1.4.2/download.html下载,目前版本是J2SDK1.4.2)软件安

装后的目录树结构类似图2:



图中灰色部分是Java运行环境(Java Runtime Environment，JRE)，它是JDK的真子集，它为Application或Applet提供运行环境，是与Java程序运行密切相关的部分。JRE既可以单独使用，也可以作为JDK的一部分而使用(如果只是提供运行环境，而不是用以开发，仅仅安装JRE就可以了)。不论JRE是单独使用，还是作为JDK的一部分，在JRE中的 /lib/ext目录下的任何JAR文件都自动作为运行环境的"扩展"，在此平台上部署的所有应用程序都可以象使用Java核心类那样直接使用它们。

安装方式扩展就是指将"安装扩展"打包成JAR文件放在JRE软件安装目录中的/lib/ext目录下(注意：是JRE软件安装目录下的/lib/ext/目录)，也就是用作扩展的类是直接在Java运行平台上安装的。这种方式一般适用于直接在本地运行的应用程序或Applet。

例如，我们有一个类Square，用于计算一个整数的平方。代码如下：

```
public final class Square{
   public static int getSquare(int a){
          return a*a;
   }
}
```

Square 类含有一个方法——getSquare(int a)，它以一个整数为参数，返回这个整数的平方。

假使有一个应用程序（Application）——ComputeSquareApp，要使用 Square类，代码如下：

```
public class ComputeSquareApp{
   public static void main(String[] args){
          int s=10;
          System.out.println("整数 "+s+" 的平方是
"+Square.getSquare(s));
   }
}
```

假如我们已将Square类的.class文件打包成square.jar 文件，那么在不使用扩展机制的情况下怎么运行ComputeSquareApp这个应用程序呢？因为Square类不是Java平台的一部分(是我们自己定义的类)，所以，如果 square.jar 是在目录 c:\myjar\ 下，为了正常运行 ComputeSquareApp，则应当先在命令行中转到 ComputeSquareApp.java 文件所在的目录（c:\myapp\compute\）下，使用如下命令进行编译：

```
java -classpath c:\myjar\square.jar ComputeSquareApp.java
```

然后，再使用如下命令以使程序运行：

```
java -classpath .; c:\myjar\square.jar ComputeSquareApp
```

也就是在编译和运行时都要显式地指明square.jar的路径，是不是很麻烦？

下面我们看一下在采用扩展机制时如何运行ComputeSquareApp 程序：

1. 用命令行工具 javac 编译 Square 类的源文件 Square.java，生成Square.class文件。将命令的当前目录转到Square.java所在目录，执行如下命令即可：

```
javac Square.java
```

2. 将类文件打包，生成JAR文件。命令如下：

```
jar -cvf square.jar Square.class
```

3. 安装"扩展"，即将 square.jar拷贝到你的JRE的 lib/ext目录下。(如果你除了JDK下的JRE之外，另装有JRE，在这两个JRE下都需拷贝；拷贝到JDK中的JRE下，目的是编译时使用，而在单独的JRE下目的是运行时使用)。

经过上面的三个步骤，安装方式扩展已经完成，我们可以通过如下命令直接进行编译和运行程序，无需指明 square.jar的路径：

```
javac ComputeSquareApp.java
java ComputeSquareApp
```

同样，在这个做了扩展的系统下运行的任何Applet或 Application，如果使用 Square类，也无需指明类路径，直接使用即可，如同使用Java核心类一样，这样做是很方便吧！

方式二：下载方式扩展

下载方式扩展是指在其它JAR文件清单(manifest)的Class-Path头中列出"下载扩展"JAR文件中的类，"下载扩展"不像"安装扩展"那样处于JRE中，它仅是在需要时从所指定的URL加载。例如，a.jar和b.jar是两个在同一目录下的JAR文件，a.jar的manifest文件包含下面的头：

```
Class-Path: b.jar
```

那么b.jar中的类就可用作a.jar中类的"下载扩展"，这样b.jar中的类不用写在类路径上，a.jar中的类就可以调用b.jar中的类(a.jar自身可以是也可以不是"扩展")。

为了更好的理解“下载扩展”,让我们看一个例子。

假如我们创建了一个要使用前面Square类的Applet——ComputeSquareApplet:

```
import java.applet.Applet;
import java.awt.*;

public class ComputeSquareApplet extends Applet {
    int s=10;

    public void paint(Graphics g) {
        g.drawString("整数"+s+"的平方是"
                    + Square.getSquare(s), 10, 10);
    }
}
```

这个Applet通过调用Square类的方法getSquare计算一个整数的平方。假如Applet是嵌入到网页中在调用方的浏览器中运行的,并用调用方的系统中并没有Square类,所以若不采取措施,ComputeSquareApplet是不能正常运行的。解决这个问题的方式之一就是将Square类做成“下载扩展”,在ComputeSquareApplet运行时加载“下载扩展”。

如何做呢?

1. 按照上面代码书写Square类和ComputeSquareApplet类的源文件,分别存为Square.java和ComputeSquareApplet.java,存到一个目录下,如:c:\myapplet\下。

2. 编译Square.java和ComputeSquareApplet.java,生成Square.class和ComputeSquareApplet.class文件,命令如下:

```
javac Square.java
javac ComputeSquareApplet.java
```

3. 将Square类的class文件打包成square.jar文件。命令如下:

```
jar -cvf square.jar Square.class
```

4. 写一个文本文件,存为mymani.txt,其内容如下:

```
Class-Path: square.jar
```

注意:此文本文件的末尾至少要有一个空行。

5. 将ComputeSquareApplet类的class文件打包成comsqapplet.jar文件。命令如下:

```
jar -cvmf mymani.txt comsqapplet.jar ComputeSquareApplet.class
```

注意:在上面的命令中使用的参数多加了一个“m”,并提供了“mymani.txt”,这表示要在生成的comsqapplet.jar文件的清单文件(manifest)的头部增添“mymani.txt”文件中的内容“Class-Path: square.jar”。上面comsqapplet.jar的manifest文件的Class-Path没有特别指明路径,表示square.jar和comsqapplet.jar处于同一个目录中。这样“square.jar”文中的类就会成为ComputeSquareApplet类的“下载扩展”。

6. 在网页文件中加入使用ComputeSquareApplet的标签。假如我们要在一个网页文件“test.htm”中使用ComputeSquareApplet,可以用类似如下的代码:

```
<HTML>
<HEAD>
<TITLE> 使用下载扩展的Applet </TITLE>
</HEAD>
<BODY>
本网页中用到下载扩展:<br>
<APPLET CODE="ComputeSquareApplet.class" WIDTH=150
HEIGHT=25 ARCHIVE="comsqapplet.jar">
</APPLET>
</BODY>
</HTML>
```

7. 将test.htm,comsqapplet.jar,square.jar三个文件发布到网站的同一目录下。我们通过tomcat(Web服务器的一种,支持JSP)进行测试,下载(http://jakarta.apache.org/tomcat/)一个tomcat的windows平台的安装版,安装好以后,将应用在“webapp”目录下建一个文件夹“mytest”,然后将网页文件test.html和其它另外两个文件拷贝到此目录下,并在“mytest”下再建一个文件夹“WEB-INF”,重新启动tomcat;然后在另一台计算机上的浏览器的地址栏中输入http://192.168.3.78:8080/mytest/test.htm即可。

另外,如果Applet或Application使用了不止一个扩展,我们可以在manifest文件中加入多个URL,例如,下面就是一个有效的Class-Path头:

```
Class-Path: area.jar servlet.jar images/
```

在Class-Path头中列出的URL如果不是以“/”结尾,就表示是JAR文件,而不论是否带有扩展文件名“.jar”;如果以“/”结尾则表示是目录,在上面的例子中,images/就是目录,其中含有Applet或Application所需的其它资源。

我们还可以使用多个Class-Path头指定多个扩展的URL。例如:

```
Class-Path: area.jar
Class-Path: servlet.jar
```

“下载扩展”还可以是“扩展链”,也就是一个“下载扩展”还可以有一个“Class-Path”头指向另一个“扩展”,第二个“扩展”还可以指向第三个“扩展”……

通过标签库

实现 Java Web 编程中的 Session 管理

▶撰文 / 杨鹏

Web应用,作为新一代的应用方式,已经开始取代传统应用程序的地位,成为当今流行的事务处理方式,而Java作为新一代面向对象的语言,不仅提供传统的界面编程,而且作为一种结构体系,深入到企业服务内部,对于Web应用方面的编程,更是提供了强大的支持,通过Java编写Web应用,无疑是一种非常高效、经济的选择,我们今天就来讨论Java Web编程中的一些问题和技巧。首先来看一些相关的知识。

预备知识

我们知道,所谓Web应用,是指基于HTTP超文本传输协议的应用。它的实现,必须完全符合HTTP协议的标准,但HTTP协议一开始只是简单的处理资源标记,并未设计为处理复杂的事务,因此是无连接(connectionless)和无状态(stateless)的。也就是说,当用户向Web服务器发送一个请求,如果请求的资源可用,则Web服务器返回该资源,然后马上关闭连接。Web服务器无法识别每次请求是否同一用户,但如今越来越复杂的Web应用,恰恰要求保留用户的信息和状态,所以,Web编程中,最困难的是状态的保持。好在人们想到了利用Cookie来保存信息。所谓Cookie,就是保存在客户端上的小文本,它保存了标识该客户的唯一ID号,及其他一些信息。这样,当该客户再次访问某Web服务器时,服务器就知道,这个用户在什么时候访问过自己,并保留了哪些信息。而利用Cookie,保留用户在整个访问过程中的状态和信息,这种方法,就叫做Session(会话)。

接下来,我们再看看Java。众所周知,Java包括三个版本,面向普通应用的标准版J2SE、面向移动及微型应用的J2ME和面向企业级应用的J2EE。而对于Web编程的支持,就包括在J2EE中。最初,J2EE通过Servlet提供对Web编程的支持,Servlet是一种服务器端的Java程序,和普通的服务器端脚本不同的是,它是预先编译的,而非执行时解释,所以效率较高;但它不能和HTML语句混合使用,因此编写强度较大,在Servlet源程序中,常常是连篇累牍的out.println("HTML文本");而后,针对Servlet的缺陷,Sun推出了JSP、Java Server Pages。它基本和其他服务器端脚本如ASP、PHP类似,都是在Web页中混合HTML代码和特定的语言代码,这样,既能够迅速实现动态Web页,又能极大减少工作强度。

但是,在服务器脚本如此众多的今天,单纯的JSP并不能得到Web程序员的青睐,之所以能够被大量应用,很大程度上是由其中的一门技术所影响——TagLib(通常翻译为自定义标签库、标签集、行为库等,这里,为避免混乱,我们使用TagLib这个词)。可能很多人都有这样的体会,在学习JSP的时候。总是弄不明白,很多教材为什么一上来就是介绍如何使用Java Beans,如何制作及利用TagLib,怎么不能像ASP、PHP一样先把JSP基本内容搞清楚再说呢?后来才慢慢开窍,不用Java Beans、TagLib,JSP根本毫无用处,甚至可以说,TagLib才是JSP的精华所在。

通过TagLib,JSP可以编写自己完整的类、能够调用大部分类库,例如JDBC、JavaBeans等等,这样,借助Java的庞大架构,使JSP拥有其他服务器脚本难以超越的强大功能,再加上预编译技术,JSP还可以同时具备足够快的运行速度。

下面,就让我们一步一步的来学习,如何在Java Web编程中,使用TagLib对Session进行管理,编写出强大的Web应用来。

简单入门

首先,我们来看一个简单的Session应用,在很多网站上,都有这样的显示:“您是第xxx位客人”,你会说,这是个计数器,和Session有什么关系呢?当然有关系了。相信很多人都知道,计数器通常只是简单的记录了当前页面的点击率,也就是说,如果访问者在这个页面上刷新十次,计数器就会增加10次,这对于想要分析真

实的网站访问量来说,是完全没有价值的。我们所需要的,是每天有多少用户访问该网站,而不是每个用户在上面刷新了多少次。这时,Session就派上用场了,当用户的浏览器开始访问Web服务器的时候,Session启动,当用户离开网站或达到一定时间没有动作,Session停止,所以,我们只在Session开始时记录一次访问就可以了,在Session有效期间,无论用户怎么刷新,计数器都不会再增加了。那么,这个简单的效果如何实现呢?

在JSP页面中,默认提供了很多隐含对象,如全局对象application、对话对象session和当前请求对象request等,其中session就是专门处理用户会话的。我们可以在JSP页面中这样处理计数器:

```
    /* 首先判断session中是否存在我们指定的属性counted,该属性标记此次会话已被计数器记录,如果不存在,说明这次会话需要执行计数器操作:*/
    if(session.getAttribute("counted") == null)
    {
    /* 在假设的计数器Bean中执行+1操作。*/
       yourCount.addCount();
    /* 计数器+1后,我们就必须在session中加上有关属性,当用户刷新此页面或在Session有效期内再次访问此页面时,不执行这段操作。someObject可以是任意的对象,因为这里我们并没有用到它。*/
       session.setAttribute("counted", someObject);
    }
```

三行代码就增强了计数器的可信度,这就是最简单的Session应用。

与此类似,还有很多地方我们可以用到session。例如,如果某个网站通过数据库为用户提供信息,如果获取每条信息时都创建一次数据库连接,并取得相应的数据,这些开销是相当庞大而且毫无必要的。我们可以在Session中设置一个数据库连接,当Session建立时,连接到数据库,以后所有该用户的数据库请求都通过该连接进行,无疑将大大提高Web应用的性能。

进阶应用

上面所说的,只是最初级的Session应用,还有很多网站会提供当前访问人数,如"现在有xxxx人访问本站,其中访客xx人,其余为...",这个技巧其实用Java实现也很简单。我们暂不讨论注册用户的问题,看看如何实现在线人数。

在J2EE中,有一个专门的接口(interface)可以很方便的实现此功能,它就是javax.servlet.http中的HttpSessionBindingListener,它可以监视用户Session的建立和注销。其中有两个方法,valueBound(HttpSessionBindingEvent event)会在用户Session建立的时候被触发,另一个valueUnbound(...)则相反,在用户Session注销时触发。有了它们,实现在线人数,真是易如反掌。

由于在JSP页面中自定义方法不太实用,很多权威资料都建议不使用这种方式,这里,我们就要用到TagLib了。

首先,我们来简单的看看JSP标签库的实现过程。所谓标签库,其实还是一些Java程序,唯一的区别是它可以获得当前Web应用的所有环境及数据。标签程序必须由TagSupport扩展,在调用标记开始时执行doStartTag()方法,在调用结束时调用doEndTag()方法,所以我们只要覆盖并实现自己的doEndTag()就可以了。以下是一个简单的例子:

```
import java.io.*;
import java.util.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class myTag extends TagSupport
{
   public int doEndTag()
   {
          JspWriter out = pageContext.getOut();
          try
          {
             out.println("Hello, Tag!");
             return EVAL_PAGE;
          }
          catch(IOException e)
          {}
          return EVAL_PAGE;
   }
}
```

然后,我们需要在Web应用的目录下(通常是WEB-INF目录)创建tlds目录,并在其中创建一个tld文件如my.tld,其内容为:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
    PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
       "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<taglib>
   <tlibversion>1.0</tlibversion>
   <jspversion>1.1</jspversion>
   <shortname>ns</shortname>

   <tag>
```

```
            <name>myTag</name>
            <tagclass>mypackage.myTag</tagclass>
            <bodycontent>empty</bodycontent>
    </tag>
</taglib>
```

其关键部分是<tag>标签，它指定了在JSP页面中要应用的标签名(<name>),以及该标签使用的类。

最后，在JSP页面中，我们就可以通过taglib来调用该标签库了：

```
<%@ taglib uri="/WEB-INF/tlds/my.tld" prefix="mytag" %>
```

并调用该标签库的相应标记：

```
<myTag:myTag />
```

注意：上面的语句都没有包含在JSP语句中，而是普通的HTML语句。

当JSP解释器执行到这里时，就会根据tld文件中的指示，找到相应的类，并执行其中的doEndTag()方法。

接下来的重点，就是在标签库中实现我们的HttpSessionBingingListener了。

前面我们谈到了JSP的隐含对象，以及标签库也可以调用JSP环境，那么，究竟如何调用呢？所有的调用都是通过pageContext实现的。凡是扩展自TagSupport的标签库，都有默认的setPageContext()方法，可以得到当前JSP页面的所有环境，其中pageContext.getServletContext()对应于JSP中的application全局对象，pageContext.getSession()对应于JSP中的Session会话对象，pageContext.getRequst()对应预JSP中的request请求对象。这样，我们就可以很方便地将TagLib与JSP结合起来。

要显示在线用户，仅使用Session对象是不行的，因为Session只针对当前用户，无法和当前所有用户联系起来，所以我们需要注册一个全局属性，供所有当前用户使用。和前面用过的方法类似，还是判断是否有一个属性，没有就创建，有了就使用，不同的是，对象是application而非session了：

```
package com.yourname;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

/* TagLib中的实现，fileName:userMonitor.java. */
/* 1.首先实现一个HttpSessionBindingListener类myListener */
class myListener implements HttpSessionBindingListener
{
    private int count = 0;
    // 这个Vector用来保存用户的IP地址
    private Vector users = new Vector();
    // 标准的Bean存取方法
    public int getCount()
    {
       return this.count;
    }
    public Vector getUsers()
    {
       return this.users;
    }
    //两个必须实现的方法
    public void valueBound(HttpSessionBindingEvent e)
    {
        this.count++;
       /* 我们在下面的代码中，会将用户IP地址作为属性加入session中，这样，通过获取session属性，就可以得到用户IP地址，但是为了防止出现意外，仍然要对代码进行保护。*/
        if(e.getSession().getAttribute("ip") != null)
        {
            this.users.add(e.getSession().getAttribute("ip"));
        }
       else
        {
            this.users.add("noip");
        }
     }
     public void valueUnbound(HttpSessionBindingEvent e)
     {
        this.count--;
        if(this.users.size() != 0)
        {
            this.users.remove(this.users.lastElement());
        }
     }
};
public class userMonitor extends TagSupport
{
   /* 2.创建一个该类的变量*/
   private myListener ml = null;
   public void doListen()
   {
      //全局对象
       ServletContext app = pageContext.getServletContext();
       //当前Session对象
       HttpSession session = pageContext.getSession();
       // 得到请求，以获取IP地址
       ServletRequest req = pageContext.getRequest();
      // 得到输出
       JspWriter out = pageContext.getOut();
      /* setMax...方法用于设置用户最长不响应时间，单位是秒，即如果用户20秒没有动作，则认为该用户已离开。*/
       session.setMaxInactiveInterval(20);
       if(app.getAttribute("lisenter") == null)
      {
        /* 3.把这个监听器作为属性值设置到全局对象中。要注意，这个"listener"虽然可以任意取，但由于是全局属性，不要取容易和其他属性混淆的名字 */
         this.ml = new myListener();
```

```
            app.setAttribute("lisenter", this.ml);
        }
    /* 4.然后在当前session中调用该全局变量,这样,当前session
    实际上启用了该监听器,session的启动和注销会影响到该监听
    器的动作。*/
    session.setAttribute("listener", (myListener)app.
            getAttribute("listener"));
    session.setAttribute("ip",req.getRemoteAddr());
    /* 5.调用myListener的getCount()方法就得到了在线人数,当
    然也可以通过获取下面的Vector元素个数得到这个值*/
    int count =0;
    count = ((myListener)app.getAttribute("listener")).
            getCount();
    /* 调用getUsers()方法就得到当前所有在线用户的IP地址*/
    Vector users = ((myListener)app.getAttribute
            ("listener")).getUsers();
    try
    {
       out.println("当前共有" + count + "人在线！他们来
            自:");
       for(int c2 = 0; c2 < users.size(); c2++)
       {
            out.println("[" + users.get(c2) + "] 
                ");
       }
    }
    catch(IOException e)
    {}
  }
  /* 6.最后,在doEndTag()中调用doListen(),就全部OK了 */
  public int doEndTag() throws JspException
  {
     doListen();
     return EVAL_PAGE;
  }
}
```

具体实现

由于TagLib涉及到具体实现的多个方面,我们在这里给出一个具体的例子:在UNIX平台上,Apache1.3.27整合Tomcat 4.1.18,JDK版本是1.3.1。

假设我们将Tomcat的webapp默认应用目录设置为/var/webapp,在其下建一个WEB-INF目录,WEB-INF中通常包含这样一些子目录:

classes(保存编译过的Java类,包括Beans、Tag);

src(Java源程序);

tlds(TagLib的定义文件);

lib(打包的Java类);

当然还有整个web应用的配置文件web.xml。

在src中编写完Java程序后,先用javac进行编译,如果把所有自己的Java类放到一个特定的包中,那么既易于管理,又易于使用,假设在所有的java源程序中,第一行都是:

```
package com.yourname;
```

那么执行

```
javac -d ../classes myTag.java
```

如果编译正确的话,将创建/var/webapp/WEB-INF/classes/com/yourname目录树,并把myTag.class保存到这里。

注意:通常JSP/Servlet容器可以自动将当前web应用目录下的/WEB-INF/classes目录作为类的搜索目录,因此,其下的class文件可以自动被找到,但在编译java源程序时,由于系统CLASSPATH变量中并没有该路径,因此如果我们在源程序中引入自己的类,如import com.yourname.*;那么将该路径加入CLASSPATH是有必要的。

然后,我们应当编写合适的tld文件,前面我们已经通过my.tld了解了tld文件的编写,这里只是介绍一些tld文件中的一些注意事项。

最简单的tag标签如下,

```
<tag>
   <name>myTag</name>
   <tagclass>com.yourname.myTag </tagclass>
   <bodycontent>empty</bodycontent>
</tag>
```

但是,我们可以在使用标签的时候,同时为它指定一些属性,例如在数据库应用中,很可能需要传递一个SQL查询,那么,我们可以在<tag>标记中增加一项<attribute>子项:

```
...
<tag>
   ...
<bodycontent>empty</bodycontent>
   <attribute>
          <name>queryString</name>
          <required>true</required>
          <rtexprvalue>true</rtexprvalue>
   </attribute>
</tag>
...
```

其中,<name>项标记了该属性的名字,和标准的JavaBean一样,在指定的Java程序中,必须有一个同名的变量,和一个设置方法,如queryString变量和setQueryString()方法。而<required>标识该变量是否是必须的,如果为真,那么在JSP页面中调用该标签库时,就必须提供该变量,否则JSP解释器会出错。<rtextprvalue>标识该变量是否可以通过另外的JSP变量获得,举个例子,假设在某个数据库查询中,queryString并不是由手工写入,而是根据当天时间动态产生,那么必须生成一个字符串保存这个SQL语句,如:

```
<%
/* JSP代码 */
String sqlString = "select * from myData where myDate
      = '" + new java.util.Date() + "'";
%>
```

然后调用该Tag:

```
<%@ taglib uri="/WEB-INF/tlds/my.tld" prefix="my" %>
<my:myTag queryString="<%=sqlString%>" />
```

如果未设置<rtextprvalue>值,由于它默认为false,那么,在myTag中我们实际获得的queryString将会是"<%=sqlString%>"这样一个字符串,如果设置<rtextprvalue>为真,则获得的queryString才可能是"select * from myData where myDate='2003-8-12'"。这就是<rtextprvalue>的作用。

最后,我们可以在Tomcat默认的首页/var/webapp/index.jsp中,检查TagLib是否制作正确:

```
<%@ page language="java" contentType="text/html;
charset=gb2312" %>
...
HTML code
...
<%@ taglib uri="/WEB-INF/tlds/my.tld" prefix="my" %>
<my:myTag />
...
HTML code
...
```

通过访问默认的Web服务器地址,我们就可以得到这样的显示:

...

当前共有 x 人在线!

...

总结

看了这么多,相信大家对TagLib和Session一定有了深入的了解,上述的例子虽然简单,却包含大多数Web应用的原理。例如,目前流行的网上购物,就是Session的大量使用:在Session中保存用户名和密码,这样无论在哪个页面,都可以保障用户操作的简便和安全;在Session中保存用户定购的物品和数量,在用户逛了一圈回来后,仍可以记住他所有购买的项目;在Session中记录下用户的定购品种,可以在最后动态为用户提供相关的商品信息,等等。

而且,我们不难发现,熟练应用TagLib,可以实现很多JSP脚本难以实现的功能,还可以轻松的实现Web应用的MVC模式,甚至,可以做到jsp页面中完全不包含Java代码,大大增强应用的性能和可维护性。

只要了解了Session的基本模式,和Java对Session编程的有力支持,我们就可以编写出功能强大、易于管理的Web应用来。

知识点

JSP标签库(也被称作定制标签)是一种通过JavaBean生成基于XML的脚本的方法。从概念上讲,标签就是很简单而且可重用的代码结构。定制标签使得JSP项目中很容易创建重用的开放源代码模块。而你的全部需要不过就是标签库及其文档。定制标签库有那些好处呢?

1. 易于安装在多个项目上　标签很容易从一个JSP项目迁移到其他项目。一旦建立了一个标签库,你只需要把这个标签库包装成一个JAR文件就可以在其他JSP项目中重新使用了。不能重用的是你作为程序员在建立标签时所加进标签的内容。因为标签可以重新使用,所以标签库可以轻松地用于你自己的项目。目前,最好的标签资源可以在JSPTags.com这个站点找到。.

2. 扩展JSP　标签库可以具备JSP规范(JSP 1.2)中的任何特性和功能。这也意味着你拥有了无限的能力可以扩展和增加JSP的强大功能却无需等待新版本JSP的发布。所以说,你完全可以取消页面上的JSP include调用了——只需用include标签建立自己的规范就可以了。

3. 易于维护　标签库使得JSP Web应用程序变得很容易维护。主要有以下几个原因:

◆ 标签对任何人而言都很容易使用、易于理解。

◆ 你的所有逻辑都驻留在处于中心的标签处理器和JavaBean内。这样一来,如果你不得不更新你的代码,你只需要处理这些中心文件而无需修正使用这些代码的其他页面。

◆ 如果你需要增加新的功能,你不必改变任何已经存在的页面。你可以把额外的属性包含到你的标签内,从而在引进新的行为同时保留以前的属性,实现旧页面的正常运行。

◆ 标签提升了代码的重用性。那些经过多次测试和使用的代码肯定具有更少的bug。所以,使用定制标签的JSP页面也同样具有更少的缺陷,维护起来自然方便多了。

4. 更快的开发速度　标签库是一种重用代码的好办法。我们知道,服务器端语言标准的重用代码方式是使用模版。标签库和模版库这种方式相比则好得多。采用模版库,你就需要针对每个项目修改模版或者建立生硬的接口。标签库则没有这些限制,而其所具有的面向对象特性则让标签库不仅用法灵活而且扩展能力极为强大。并且,因为你重用代码,结果在项目开发上花费的时间就大大降低了,而更多的时间则可以用来设计自己的Web应用程序。标签库的简单接口使得这些代码用法简单、易于调试。

需要说明的是,虽然标签库用起来特别简单,但是,建立其内部支持层次比建立简单的JavaBean复杂得多!

从VB到VB.NET
——VB6程序员如何转向.NET

▶编译/韩磊

Visual Basic .NET是Microsoft Visual Studio .NET套件的主要组成部分之一。.NET版本的Visual Basic增加了更多特性,而且演化为完全面向对象(就像C++)的编程语言。本文将介绍VB.NET的新特性,并比较VB 6.0/VB.NET之间的区别,阐述如何利用VB .NET编写简单的应用程序。

1.1 什么是 VB.NET?

VB.NET是VB 6.0的后续版本。Microsoft推出全新的编程和操作系统Framework——.NET,支持多种语言利用公共.NET库开发应用程序,这些应用程序在.NET Framework上运行。使用Visual Basic在.NET Framework上编程,这就是VB.NET。

首先,让我演示在VB.NET中写最简单的控制台程序:Hello World。

1.2 Hello, World!

"Hello World!"是初学者学习Windows编程的代表性程序。我们的第一个程序就叫做'Hello VB.NET World!"。该程序在控制台输出一句话:"Hello VB .NET World!",代码如下所示:

代码1.1:Hello VB.NET World例子

```
Imports System

Module Module1
    Sub Main()
        System.Console.WriteLine("Hello VB.NET
World!")
    End Sub
End Module
```

1.3 VB.NET 编辑器和编译器

你可以在记事本或VS.NET IDE等任意文本编辑器中撰写上述代码,然后保存为HelloWorld.vb。

代码编写完成之后,要么在命令行、要么在VS.NET IDE中编译它。在Microsoft .NET Framework SDK中已经包括VB.NET编译器vbc.exe①,从IDE或是命令行都可以调用。要从命令行编译HelloWorld.vb,请在命令行窗口输入

```
vbc HelloWorld.vb /out:HelloWorld.exe /t:exe
```

编译结束后,HelloWorld.exe被创建到当前目录下。在资源管理器其中双击图标或在命令行执行,程序正确地运行了。祝贺你进入VB.NET开发者的行列。

Imports 语句

如你所知,大部分的.NET类型都在名字空间(namespace)中定义。Namespace是定义和管理类别的范畴。察看.NET Framework Class Library,可以看到数以百计的namespace。例如,System namespace就包括了Console、Object等类型定义。如果想使用Console类,需要用Imports指令导入System namespace。如下所示:

```
Imports System
```

甚至可以明确地调用namespace而无需用Import导入。下面的例子展示了不用Import的'Hello World!"程序:

代码1.2:Hello VB.NET World例子

```
Module Module1
    Sub Main()
        System.Console.WriteLine("Hello VB.NET
World!")
    End Sub
End Module
```

1.4 解析 "Hello VB.NET World!"

程序第一行是:

```
Imports System;
```

System namespace定义了Console类,该类用于读写控制台(命令行窗口)。

然后你定义了一个module:

```
Module Module1
...
End Module
```

所有的VB程序都包括一个Main()方法，即应用程序入口点。在例子程序中，我们调用Console.WriteLine()向控制台写入“Hello VB.NET World!”:

```
Sub Main()
    Console.WriteLine("Hello VB.NET World!")
End Sub
```

WriteLine()方法归属于Console类，它负责向控制台写一个带有行结束符的字符串。如前所述，Console类定义于System namespace，你通过直接引用来控制类成员。

Console类负责读写系统控制台。读控制台输入用Read和ReadLine方法，向控制台输出用WriteLine方法。

方法	用途	例子
Read	读入单个字符	int i = Console.Read();
ReadLine	读入一行	string str = Console.ReadLine();
Write	写一行	Console.Write("Write: 1");
WriteLine	写一行，并带上行结束符	Console.WriteLine("Test Output Data with Line");

表1.1 Console类定义的方法

1.5 VB.NET有什么新特点？

作为VB 6.0的后续版本，VB.NET更加稳定，而且完全面向对象。也许你还记得，VB 6.0不支持继承、重载和接口，所以不是真正面向对象的。而VB.NET则支持这些面向对象特性。VB6.0有两个薄弱环节——多线程和异常处理。在VB.NET中，开发多线程应用和使用C++/C#别无二致，结构化异常处理也得到支持。稍后我们会详细解释这些特性。

下面是VB.NET的特性列表——

- 面向对象的编程语言。支持继承、重载、接口、共享成员和构造器。
- 支持所有的CLS特性，如存取控制.NET类、与其它.NET语言交互、元数据、公共数据类型、委托等等。
- 多线程支持。
- 结构化异常处理。

1.6 VB.NET：完全面向对象的编程语言

抽象、封装、多态、继承是面向对象语言的四个基本属性。VB 6.0不支持继承，而VB.NET则不然。所以，和C++一样，VB.NET也是完全面向对象的编程语言。

Class 和 Module

VB.NET用Class...End Class语句创建class。每个VB.NET至少包括一个Module（模块）。Module在Module...End Module语句中实现。应用程序的主要模块是Main方法，亦即应用程序入口点。

和VB 6.0一样，使用Function/Sub关键字可以定义方法。下面的例子显示了如何创建class、添加方法，并从主程序调用方法:

```
Imports System

Module Module1

    Sub Main()
        Dim cls As TestClass = New TestClass
        Console.WriteLine(cls.MyMethod)
    End Sub

    Class TestClass
        Function MyMethod() As String
            Return "Test Method"
        End Function
    End Class
End Module
```

Property

属性（Property）是类变量的公共描述。用Property...End Property语句创建property。属性的Get/Set方法分别用于取得和设置属性值。下面的例子中，Data是TestClass的属性。

```
Imports System
Imports System.Console

Module Module1
    Sub Main()
        Dim cls As TestClass = New TestClass
        WriteLine(cls.MyMethod)
        WriteLine(cls.Data)
        cls.Data = "New Data"
        WriteLine(cls.Data)
    End Sub
End Module

Class TestClass
    Private strData As String = "Some Data"
```

```
    Function MyMethod() As String
        Return "Test Method!"
    End Function

    ' Adding Data property to the class
    Public Property Data() As String
        Get
            Return strData
        End Get
        Set(ByVal Value As String)
            strData = Value
        End Set
    End Property
```

重载

VB.NET通过overload关键字支持方法重载。使用这个关键字,你可以定义同名但不同参数的方法。

类成员访问域

除了原有的Private和Public, VB.NET引入了几个新关键字。全部访问域关键字列表如下:

关键字	作用域
Private	限于class内部
Public	可以从class外访问
Friend	限于class所属的应用程序内
Protected	只能被class和其派生类访问
Protected Friend	能被class、应用程序和派生类访问

继承

继承是面向对象编程语言中最常用的技术。继承让你能够重用类代码和函数。

VB.NET支持继承, 而VB 6.0则不支持。继承的好处在于你能使用任何人编写的类,从这些类派生自己的类,然后在自己的类中调用父类功能。在下面的例子中, Class B派生自 Class A, 我们将从 Class B中调用 Class A的方法MethodA。

```
Imports System
Imports System.Console

Module Module1
    Sub Main()
        Dim bObj As B = New B
        WriteLine(bObj.MethodA())
    End Sub
End Module

' Class A defined
Public Class A
    Function MethodA() As String
        Return "Method A is called."
    End Function
End Class

'Class B, inherited from Class A. All members
(Public and Protected)
' can be access via B now.

Public Class B
    Inherits A
    Function MethodB() As String
        Return "Method B is called."
    End Function
End Class
```

可以从一个class中派生多个自定义class,也可以从多个 class派生一个自定义class。

共享的成员

类的共享成员被类的所有实体共享。共享成员可能是属性、方法或字段/值域。在你不想让用户全面控制自己的类时,共享成员相当有用。例如,你可以开发一个类库,让用户通过共享成员使用其中的部分功能。

可以通过class本身引用共享成员,而无需通过类的实体。例如:

```
Module Module1
    Sub Main()
        WriteLine(A.MethodA())
    End Sub
End Module

' Class A defined
Public Class A
    Shared Function MethodA() As String
        Return "Method A is called."
    End Function
End Class
```

多线程

VB语言的一大弱点就是缺乏编写自由线程(free-threaded)程序的能力。在.NET Framework中, 所有语言共享CRL (Common Runtime Library, 公共运行库),也就是说,你可以用VB.NET、C#或其它.NET语言编写同样的程序。

System.Threading namespace定义了线程类。我们只需要引入System.Threading namespace,即可使用线程类。

System.Threading.Thread类提供线程对象,可以使用Thread类创建或破坏线程。

创建线程

使用 Thread 类的实体创建一个新线程, 然后用Thread.Start方法开始执行线程。线程建构函数接受一

个参数,该参数指明你要在线程中执行的procedure。在下例中,我想在oThread1(Thread类的一个实体)的第二线程中执行SecondThread过程:

```
oThread1 = New Thread(AddressOf SecondThread)
SecondThread procedure looks like below:
Public Sub SecondThread()
    Dim i As Integer
    For i = 1 To 10
        Console.WriteLine(i.ToString())
    Next
End Sub
```

然后,调用Thread.Start()开始线程:

```
oThread1.Start()
```

下面的代码创建两个第二线程:

```
Imports System

Module Module1
    Public oThread1 As Thread
    Public oThread2 As Thread

    Sub Main()
        oThread1 = New Thread(AddressOf
SecondThread)
        oThread2 = New Thread(AddressOf ThirdThread)
        oThread1.Start()
        oThread2.Start()
    End Sub

    Public Sub SecondThread()
        Dim i As Integer
        For i = 1 To 10
            Console.WriteLine(i.ToString())
        Next
    End Sub

    Public Sub ThirdThread()
        Dim i As Integer
        For i = 1 To 10
            Console.WriteLine("A" + i.ToString())
        Next
    End Sub
End Module
```

破坏线程

调用Abort方法来破坏(中止)一个线程。在调用Abort之前,确保用IsAlive判断线程处于活动状态。

```
If oThread1.IsAlive Then
  oThread1.Abort()
End If
```

暂停线程

可以使用Sleep方法来暂停线程执行。Sleep方法接受一个以毫秒为单位的参数,指明线程应当暂停多长时间。

下面的例子让线程暂停1秒钟:

```
oThread2.Sleep(1000)
```

你也可以使用Suspend和Resume方法来挂起和继续线程执行。

设定线程优先级

Thread类的Priority属性用于设定线程优先级。该属性可以设置为Normal, AboveNormal, BelowNormal, Highest和Lowest。如:

```
oThread2.Priority = ThreadPriority.Highest
```

线程与Apartment

使用ApartmentState属性设置线程的apartment类型,该属性值可以为STA, MTA或是Unknown[②]:

```
oThread.ApartmentState = ApartmentState.MTA
```

MTS意味着可以使用多线程模式,而STA则只能是单线程执行。

```
Public Enum ApartmentState
{
STA = 0,
MTA = 1,
Unknown = 2,
}
```

1.7 结构化异常处理

异常处理也被称之为错误处理。作为VB程序员,你一定对On Error Goto和On Error Resume Next这些VB6.0错误处理语句耳熟能详。这种类型的错误处理被称为非结构化异常处理(Unstructured Exception Handling)。而在VB.NET中,Microsoft推出了结构化异常处理机制。VB.NET支持类似C++的Try..Catch..Finally控制。Try..Catch..Finally结构如下:

```
Try
'可能导致异常的代码
Catch
'当异常发生时处理异常的代码
Finally
'清理现场
End Try
```

Try语句块用来抛出异常。如果异常发生,在Catch语句块中处理。Finally语句块是可选的,在需要释放资源时特别有用。

1.8 VB 6.0与VB.NET的不同之处

除了上面谈到的语言进化,还有一些语法上的变化。所有这些语言和语法的变化在MSDN中均可查到,本文

只作简单介绍。

数据类型(Data Type)的改变

VB.NET中有些数据类型得到改进。下面是变化对照表。

数据类型	VB6.0	VB.NET
Integer	16 bit size	32 bit size
Long	32 bit size	64 bit size
Currency	用于存储大浮点数	被decimal替代,支持更高精度
Variant	可以存储任意类型数据	被Object类型替代,也可以存储任意类型数据,但结果更好
Date	Date类型被存储为double	引入DateTime类型,用于存储不同格式的日期

在VB.NET中,Short数据类型是16 bit的。Short,Integer和Long等同于CLR的System.Int16、System.Int32和System.Int64类型。

变量声明的变化

在VB 6.0中,变量声明有许多限制。其中之一就是不能同行声明多个变量。如果一定要在一行中声明多个变量,就得指明每个变量的类型,否则将被默认为Variant类型。

```
Dim a1, a2 As Integer
Dim a3 As Integer, a4 As Integer
```

第一行中的a1是Variant类型,a2是Integer类型。第二行中两个变量都是Integer类型。

VB.NET支持同行声明多个变量,举例如下:

```
Dim a1, a2, a3 As Integer
```

变量初始化是另一个问题。在VB 6.0中不能同时声明和初始化变量,而VB.NET则支持这个特性。

```
Dim name As String = "Mahesh"
System.Console.Write(name)
```

声明常量也可以照此办理:

```
Const DT_COUNT As Integer = 23
```

New关键字。在VB.NET中,New关键字用于创建对象。由于数据类型是对象,所以New关键字用以创建一个数据类型对象。

```
Dim i As Integer = New Integer()
i = 10
System.Console.WriteLine(i.ToString())
```

代码块级别支持。像C++一样,VB.NET支持代码块级别的作用域检查。在语句块中声明的变量只在块内有效。

```
For i = 1 To 10
Dim p As Long
System.Console.WriteLine(i.ToString())
Next
System.Console.WriteLine(p.ToString())
```

这段代码在VB.NET中会得到一个编译错误,因为p在For..Next语句块之外不可访问。在VB 6.0中这段代码可以通过。

改进了的类型安全

在VB 6.0中,当你声明一个对外部过程的引用时,可以指定任意类型的参数为As Any。Any关键字禁止了类型检查,允许任意数据类型传入和返回。

VB.NET不支持Any关键字。你必须指定每个参数和返回值的数据类型。

数组

VB.NET对数组作了明显的改动。

数组范围。在VB.NET中,你需要格外留意数组范围问题。VB 6.0默认数组下界为0,故数组中的元素数量等于数组上界值加一。下面的数组界限从A(0)到A(10),共有11个元素:

```
Dim A(10) As Single
```

可以使用Option Base改变下界值为1。在VB.NET中,数组和C++一样,下界值为0,不支持Option Base。

> 注意:MSDN文档指出数组只能包括与其尺寸相等的元素数量,例如:
>
> ```
> Dim A(10) As Integer
> ```

只能包括10个元素(从A(0)到A(9)),但在编译下面这段代码时我发现它运行良好,看起来数组中容纳了11个元素。

```
Dim A(10) As Integer
A(0) = 12
A(2) = 24
A(10) = 23

System.Console.WriteLine(A(0).ToString())
System.Console.WriteLine(A(10).ToString())
System.Console.WriteLine(UBound(A).ToString())
System.Console.WriteLine(LBound(A).ToString())
```

Lbound和Ubound分别返回 0与10。

ReDim和Fixed Array。你可以在VB 6.0中指定固定长度的数组。

```
Dim ArrWeekDays(0 To 6) As Integer
```

这里的ArrWeekDays数组是固定长度的,不能用

ReDim语句改变长度。VB.NET不支持固定长度数组，所以ReDim总是有效。

可以用下面两种方式声明数组：

```
Dim ArrWeekDays(6) As Integer
Dim ArrWeekDays() As Integer = {1, 2, 3, 4, 5, 6}
```

ReDim语句。在VB 6.0中，ReDim用于初始化动态数组。在VB.NET中你不能把它当作声明用。ReDim只能用于改变数组长度，不过不能改变数组维度。

Variant对阵Object

VB 6.0中的Variant数据类型能存储任意类型变量，VB.NET中Object具备相同能力。

算术操作符

VB.NET支持类似C++的快捷方式。下面的表格显示了常规操作与快捷操作的不同之处。快捷方式也可用于*、/、\、&等操作符。

操作符	常规语法	快捷方式
加法	A = A+5	A +=5
减法	A = A -5	A -=5

固定长度字符串

在VB 6.0中，可以在声明字符串时指定其长度。VB.NET不支持固定长度字符串。

布尔操作符

VB 6.0中的And、Or或是Xor语句是按位操作符。而在VB.NET中，它们是布尔操作符。执行这些操作将返回true或false。VB.NET引入新操作符来完成按位操作。

操作符	描述
BitAnd	按位And
BitOr	按位Or
BitXor	按位Xor
BitNot	按位Not

结构与自定义类型

在VB 6.0中，你使用Type...End Type语句块创建结构或自定义类型。例如：

```
Type StdRec
  StdId As Integer
  StdName As String
End Type
```

VB.NET引入新的语法：Structure。Type...End Type不再被支持。Structure...End Structure与C++用法相同。可以指定结构中每个元素的可访问域，如Public、Protected、Friend、Protected Friend、Private等。例如：

```
Structure StdRec
  Public StdId As Integer
  Public StdName As String
  Private StdInternal As String
End Structure
```

VB.NET中的Structures就像类一样，也可以拥有方法和属性。

New和Nothing关键字

VB 6.0中，AS New和Nothing关键字用于声明一个对象并初始化它。

VB.NET不支持隐式创建对象。如前所言，甚至连数据类型都是对象。你可以采用以下两种方法创建数据类型或对象：

```
Dim i As Integer
Dim i As Integer = New Integer()
// Do something
if i = Nothing Then
End If
```

不支持Set语句。VB 6.0使用Set语句指派对象。例如：

```
Set myObj = new MyObject
Set a = b
```

在VB.NET中，不需要使用Set指派对象。例如：

```
myObj = new MyObj()
a = b
```

过程（procedure）语法的变化

在VB.NET中过程语法有了很多变化。例如类似C++的过程调用方式、ByVal（传值）为默认类型、Optional关键字、return语句等等。

类似C++的过程调用方式

VB 6.0允许不用加圆括号调用过程（sub）。不过，用Call语句调用函数或sub时，一定要使用圆括号。例如：

```
Dim I as Integer
Call EvaluateData(2, i)
EvaluateData 2, i
```

在VB.NET中，所有的方法调用都需要圆括号，而Call语句则是可选的。

ByVal是默认参数类型

在VB 6.0中，在调用函数或sub时ByRef（传址）是默认类型。那意味着所有改变将反映到传入的变量。VB.NET改变了这种方式。现在，默认的参数类型是ByVal（传值）。

Optional关键字

VB 6.0使用Optional关键字可用来让用户决定传入一个默认值，之后在调用IsMissing函数判断参数是否有效。

而在VB.NET中，每个可选参数必须声明其默认值，无需调用IsMissing函数。例如：

```
Sub MyMethod(Optional ByVal i As Integer = 3)
```

Return语句

VB.NET的Return语句与C++相似。使用Return语句把控制权从过程返还给调用者。在VB 6.0中，Return语句与GoSub语句一起使用。VB.NET不再支持GoSub语句。

流程控制的改变

下面是VB.NET对流程控制语句的修改：

1. GoSub不再受到支持。

2. Call、Function和Sub语句均可用于调用过程。

3. On ... GoSub和On ... GoTo语句不再受支持。可以使用Select Case语句来替代。

4. While ... Wend语句现在改为While...End While语句。不再支持Wend关键字。

小结

Visual Basic .NET是.NET版本的Visual Basic。通过本文你了解到VB.NET的基本概念，而且也从VB 6.0开发者的角度对VB.NET进行了考察。

① 译注：这意味着你无需购买VS.NET即可进行简单的VB.NET开发。

② 译注：STA - Single Threaded Apartment；MTA - Multi Threaded Apartment

---

（上接80页）

```
            foreach (Attribute attr in type.
GetCustomAttributes(true))
             {
                HelpAttr = attr as HelpAttribute;
                if (null != HelpAttr)
                 {
                   Console.WriteLine("Description of
AnyClass:\n{0}",
                            HelpAttr.Description);
                 }
             }
             //查询Class中的Method属性
            foreach(MethodInfo method in type.GetMethods())
             {
                foreach (Attribute attr in method.
GetCustomAttributes(true))
                 {
                   HelpAttr = attr as HelpAttribute;
                   if (null != HelpAttr)
                    {
                      Console.WriteLine("Description
of {0}:\n{1}",
                                        method.Name,
                                HelpAttr.Description);
                    }
                 }
             }
             //查询Class中的Field（公有的）属性
            foreach(FieldInfo field in type.GetFields())
             {
                foreach (Attribute attr in field.
GetCustomAttributes(true))
                 {
                   HelpAttr= attr as HelpAttribute;
                   if (null != HelpAttr)
                    {
                      Console.WriteLine("Description
of {0}:\n{1}",
                                        field.Name,
HelpAttr.Description);
                    }
                 }
             }
         }
    }
```

程序的输出为：

```
Description of AnyClass:
This is a do-nothing Class.
Description of AnyMethod:
This is a do-nothing Method.
Description of AnyInt:
This is any Integer.
Press any key to continue
```

在C# Builder中使用ADO.NET

▶撰文/Bob Swart

提要

在本文中，Bob Swart向你展示了如何使用C#Builder个人版结合“plain”ADO.NET来连接MSDE数据库，创建表，插入记录，再从表中选择记录并将记录显示在DataGrid组件中这一实践过程。

C# Builder的一个重要新特性就是提供了Borland Data Providers for ADO.NET，它具有为连接和使用数据库而提供的强大的设计时(design-time)支持，拥有专门用于DB2、InterBase、Oracle、SQL Server/MSDE的数据库驱动程序，还具备添加第三方驱动程序(例如dbExpress)的潜力。然而，Borland Data Providers并不包括在C# Builder个人版中，这意味着你不得不借助于“plain(纯代码的)”ADO.NET；这是与数据库交互的标准.NET方式。

在本文中，我将向你展示如何使用C# Builder个人版结合ADO.NET来连接一个MSDE数据库，创建表，插入一些记录，从表中选择这些记录，最后将它们显示在DataGrid中。对于最容易的“play-along”的方式，我假设你已经安装了MSDE(MSDE是与C# Builder捆绑在一起的——C# Builder个人版也是如此)。

SqlConnection

启动C# Builder，选择File|New-C# Application以开始一个新的项目。拖放一个SqlConnection组件，并将ConnectionString属性设为：

```
Data Source=.; Initial Catalog=master; Integrated security=SSPI
```

注意：master是与MSDE一起发布的数据库，你也可以使用任何其它数据库或者.NET DBMS(只不过需要修改ConnectionString)。

将第一个按钮拖到WinForm上，将其标题设为“Connect”，然后在button_Click事件处理程序中写入以下代码，以便打开sqlConnection(注意，这里我再次设置了ConnectionString属性，这样做只是为了向你展示“纯粹”通过代码也能设置它的属性)：

```
    private void button1_Click(object sender, System.
EventArgs e)
      {
        try
        {
          sqlConnection1.ConnectionString = "Data
Source=.; Initial Catalog=master; Integrated
security=SSPI";
          sqlConnection1.Open();
        }
       catch (Exception ex)
        {
         MessageBox.Show(ex.Message);
        }
      }
```

这是首次尝试，也是最容易失败的尝试(其它步骤都比较简单)。如果你遇到一个异常，很可能需要修改ConnectionString，确保它包含了正确的连接字符串(注意：还应确保SQL Server/MSDE正在运行，这是理所当然的事情)。

SqlCommand

一旦你连接成功，你就可以拖放一个SqlCommand组件。这个组件可用于执行SQL语句。使用Object Inspector将其Connection属性指向sqlConnection组件，并将要执行的命令写入CommandText属性中。注意，Connection属性值框的下拉列表可能不会列出sqlConnection组件(即使该组件位于同一窗体)，但是

有一个“诀窍”，就是在Connection属性值框上双击鼠标，这样就可以循环选择可能的值。另一种方法是直接输入你要使用的组件的名称，假设输入“s”，所有“s”开头的连接都会列出，选中sqlConnection1即可。如果要清除Connection 属性，只需要输入"("，这样就会得到“(none)”。

用“纯”代码来完成工作会比较有教育意义，我将用源代码来设置所有属性（包括要执行的SQL命令）。

拖放另一个按钮，将其标题设置为“SQL”，并在button_Click事件处理程序中写入以下代码，创建一个表（如果该表已经存在，则首先将其删除）并插入三条记录：

```
      private void button2_Click(object sender, System.
EventArgs e)
      {
        try
        {
         sqlCommand1.Connection = sqlConnection1;
         sqlCommand1.CommandText = "drop table test42";
          try
          {
          sqlCommand1.ExecuteNonQuery(); // create table
          }
         catch {}; // ignore
          sqlCommand1.CommandText = "create table test42
(id int NOT NULL, name nvarchar(42))";
         sqlCommand1.ExecuteNonQuery(); // create table

          sqlCommand1.CommandText = "insert into test42
values(1, 'Bob Swart')";
         sqlCommand1.ExecuteNonQuery(); // insert table
          sqlCommand1.CommandText = "insert into test42
values(2, 'Erik Mark Pascal Swart')"; // my son
         sqlCommand1.ExecuteNonQuery(); // insert table
          sqlCommand1.CommandText = "insert into test42
values(3, 'Natasha Louise Delphine Swart')"; // my daughter
         sqlCommand1.ExecuteNonQuery(); // insert table
        }
       catch (Exception ex)
        {
         MessageBox.Show(ex.Message);
        }
      }
```

注意：第一个try-catch 程序块只用于当表不存在时（在这种情况下，不能删除表）忽略掉异常。

我使用SqlCommand的ExecuteNonQuery 方法，因为这条SQL语句不需要返回数据，而只是删除表、创建新表或者插入新记录（显然，当删除记录时，也必须调用ExecuteNonQuery）。

SqlDataAdapter、DataSet 和 DataGrid

现在我们已经有了一个对 MSDE master 数据库的连接，并创建了一个名为test42 的表，这个表有3条记录。接下来该选择这些记录，并将其与一个Grid组件关联起来。为了实现这一目标，我想使用3个组件：一个SqlDataAdapter、一个DataSet 和一个用于显示数据的DataGrid。

首先拖放一个SqlDataAdapter 组件。该组件有一些为SelectCommand、InsertCommand、UpdateCom-mand 和 DeleteCommand而设的子属性。这些命令中的每一个都与一个SqlCommand组件相对应，我们在前一节中曾用过这种组件。不过现在我们只需要SelectCommand。要使用SelectCommand，你必须首先为Connection属性赋值，然后将一条SELECT 语句赋给CommandText 属性（注意，我将在代码中再次做这件事情）。

现在，拖放一个DataSet 组件。这个DataSet 将被SelectCommand执行后返回的结果集填充。顺便提一下，.NET DataSet 可以包含多个表，这与Delphi的TdataSet有所不同。

最后，拖放一个DataGrid 组件。为该DataGrid的DataSource 和 DataMember属性赋值，使其与DataSet和Table相关联，以便显示数据。随便配置一下DataGrid，让它显得好看一些（现在我将略过这一步，而把注意力放到Data 组件上）。

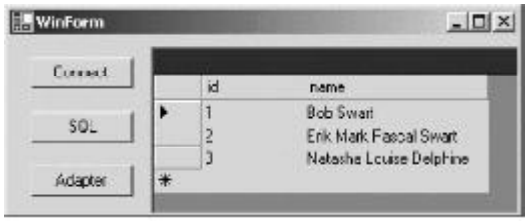
以下代码用于初始化SqlDataAdapter组件的SelectCommand属性，并使用SelectCommand 填充DataSet，最后再将其绑定到DataGrid 组件：

```
    private void button3_Click(object sender, System.
EventArgs e)
    {
      try
      {
       sqlDataAdapter1.SelectCommand.Connection =
sqlConnection1;
       sqlDataAdapter1.SelectCommand.CommandText =
"select * from test42";
       sqlDataAdapter1.Fill(dataSet1, "MyTable42");

       dataGrid1.DataSource = dataSet1;
       dataGrid1.DataMember = "MyTable42"; // same
name as above
      }
      catch (Exception ex)
      {
       MessageBox.Show(ex.Message);
      }
    }
```

注意：在（通过SqlDataAdapter）填充DataSet时，指定的名称与我在将DataSet绑定到DataGrid时作为DataMember属性值来指定的那个名称是一致的。

上面所示的代码将把SQL SELECT语句的输出显示在DataGrid中，如下图所示：



你可以看到，与数据库、SqlCommand以及SqlDataAdapter进行交互的是SqlConnection，SqlCommand 和SqlDataAdapter。可以执行一条纯操作（如DELETE、UPDATE）的命令，也可以利用SELECT语句返回一个结果集。

小结

如果你想快速地重复上述操作（RAD方式），那么下面的一些步骤将很有帮助（假设主数据库以及带有记录的test42表已经存在）。

1．开始一个新的C# Builder 项目。

2．拖放一个sqlConnection 组件。如下所示设置其ConnectionString 属性：

```
    Data Source=.; Initial Catalog=master; Integrated
security=SSPI
```

3．拖放一个SqlDataAdapter组件。将SelectCommand.Connection属性指向SqlConnection组件。在CommandText属性中输入一条SQL SELECT语句，例如：

```
    select * from test42
```

4．拖放一个DataGrid 组件。使用Auto Format功能随意设置DataGrid 的格式。

5．在Form上双击鼠标，这将把你带到WinForm_Load 事件处理程序。在那里，我们将编写代码来使用SqlDataAdapter.SelectCommand.CommandText的结果填充一个新的DataSet，并将其绑定到DataGrid，如下所示：

```
    private void WinForm_Load(object sender, System.
EventArgs e)
    {
      try
      {
       DataSet ds = new DataSet();
      //   sqlConnection1.Open(); // implicit
       sqlDataAdapter1.Fill(ds, "MyTable42");
       dataGrid1.DataSource = ds;
       dataGrid1.DataMember = "MyTable42"; // same
name as above
       sqlConnection1.Close(); // explicit
      }
      catch (Exception ex)
      {
       MessageBox.Show(ex.Message);
      }
    }
```

6．保存、编译并运行你的应用程序（确保SQL Server / MSDE正在运行）。

DataGrid 将显示SQL SELECT语句执行得到的结果集。

注意：在最后的代码片段中，我还使用了SqlConnection.Open() 和Close() 语句来演示连接可以自动地打开，但是必须显式地关闭（如果你不想让连接在完全不需要的情况下还继续开放下去的话，你就得显式地关闭它）。

总结

在本文中，我展示了如何使用C# Builder 个人版结合ADO.NET 来连接MSDE 数据库，创建表，插入几条记录，从表中选择这些记录并将它们显示在DataGrid组件中。

详解 Attributes

▶撰文 / Sadaf alvi

简介

Attribute（属性）是一种新的说明性信息。使用属性既可以定义设计时信息(例如帮助文件、文档的URL),又可以定义运行时信息(例如与类字段相关联的XML字段),还可以使用属性生成“自描述”组件。在本指南中将讲述如何生成属性并将它附加到各种应用程序项中,以及如何在运行时环境中检索属性信息。

使用预定义属性

C#中有一组预定义属性。在学习生成自定义属性前,首先来看如何在代码中使用这些预定义属性。

```
using System;
public class AnyClass
{
   [Obsolete("Don't use Old method, use New
method", true)]
    static void Old( ) { }

    static void New( ) { }

    public static void Main( )
    {
       Old( );
    }
}
```

看看这一例子。在这个例子中使用了Obsolete属性,它用于标识不能使用的程序项。第一个参数是字符串,用于解释为什么该项被废弃,应该使用什么来替代它。实际上这儿可以使用任意的文本。第二个参数告诉编译器是否将该项的使用看作错误。默认值为false,也就是说编译器为该项的使用生成警告信息。

编译上面的应用程序将得到一个错误:

```
AnyClass.Old()' is obsolete: 'Don't use Old method,
use New method'
```

开发自定义属性

现在来看如何开发自己的属性。这里有个生成自己属性的诀窍。

如C#语言规范中所述,从System.Attribute类派生出自己的属性类(由抽象类System.Attribute直接或间接派生的类是一个Attribute类。Attribute类的声明定义了一个新类型的属性,它可以放置在声明中),形式如下所示。

```
using System;
public class HelpAttribute : Attribute
{
}
```

信不信由你,这里已经生成了一个自定义属性。与Obsolete属性一样,可以使用它来修饰自己的类。

```
[Help()]
public class AnyClass
{
}
```

注意:对于属性类的名字使用单词Attribute作为后缀,这是约定。但是附加属性到程序项时,可以自由地不包含Attribute后缀。编译器首先在System.Attribute导出类中查找属性。如果没有找到类,编译器将单词Attribute添加到指定的属性名后,再次查找。

但是这一属性到目前为止没有什么实际作用。现在为它添加其它内容使其更有用。

```
using System;
public class HelpAttribute : Attribute
{
   public HelpAttribute(String Descrition_in)
    {
       this.description = Description_in;
    }
   protected String description;
   public String Description
    {
```

```
        get
        {
            return this.description;

        }
    }
}
[Help("this is a do-nothing class")]
public class AnyClass
{
}
```

上面的例子中,为属性类添加了一个属性,最后一节中,将在运行时查询它。

定义或控制属性的使用

AttributeUsage类是另一个预定义类,用于帮助我们控制自定义属性的使用。也就是说,可以定义自己的Attribute类的属性。描述了如何使用自定义Attribute类。AttributeUsage有三个属性,在将它放入自定义Attribute类时可以设置这三个属性。第一个属性是:

ValidOn

通过这个属性,可以定义可以放置自定义属性的程序项。在AttributeTargets枚举器中列出了可以放置属性的所有程序项集。可以使用按位或运算符连接几个AttributeTargets值。

AllowMultiple

该属性表明自定义属性是否可以在同一个程序项中出现多次。

Inherited

使用这个属性可以控制自己属性的继承规则。该属性表明自己的属性是否可以被由包含了该属性的类的导出类继承。

来做一些练习。将AttributeUsage属性放入自己的Help属性中,并控制带有这一帮助属性的使用。

```
using System;
[AttributeUsage(AttributeTargets.Class),
AllowMultiple = false,
 Inherited = false ]
public class HelpAttribute : Attribute
{
    public HelpAttribute(String Description_in)
    {
        this.description = Description_in;
    }
    protected String description;
    public String Description
    {
        get
        {
            return this.description;
        }
    }
}
```

首先来看AttributeTargets.Class。它声明了Help属性只能在一个类中出现一次。这意味着下面的代码将会产生错误:

```
AnyClass.cs: Attribute 'Help' is not valid on this
declaration type.
It is valid on 'class' declarations only.
```

现在试一试在方法中加入Help属性:

```
[Help("this is a do-nothing class")]
public class AnyClass
{
    [Help("this is a do-nothing method")]    //错误
    public void AnyMethod()
    {
    }
}
```

可以使用AttributeTargets.All来允许Help属性出现在任意程序项中。可能的取值有:

- Assembly,
- Module,
- Class,
- Struct,
- Enum,
- Constructor,
- Method,
- Property,
- Field,
- Event,
- Interface,
- Parameter,
- Delegate,
- All = Assembly | Module | Class | Struct | Enum | Constructor | Method | Property | Field | Event | Interface | Parameter | Delegate,
- ClassMembers = Class | Struct | Enum | Constructor | Method | Property | Field | Event | Delegate | Interface)

现在考虑AllowMultiple = false。这表明属性不能出现多次。

```
[Help("this is a do-nothing class")]
[Help("it contains a do-nothing method")]
public class AnyClass
{
    [Help("this is a do-nothing method")]    //错误
    public void AnyMethod()
    {
    }
}
```

这将产生编译错误。

```
AnyClass.cs: Duplicate 'Help' attribute
```

好,现在来讨论最后一个属性。当属性放置在基类中, Inherited表明该属性是否也被基类的导出类继承。如果属性类的Inherited为true, 则该属性被继承。但是如果属性类的Interited为false或者未指定,则该属性不能被继承。

假设有下列类之间的关联。

```
[Help("BaseClass")]
public class Base
{
}

public class Derive : Base
{
}
```

有四种可能的组合方法:

❒ [AttributeUsage(AttributeTargets.Class, AllowMultiple = false, Inherited = false]

❒ [AttributeUsage(AttributeTargets.Class, AllowMultiple = true, Inherited = false]

❒ [AttributeUsage(AttributeTargets.Class, AllowMultiple = false, Inherited = true]

❒ [AttributeUsage(AttributeTargets.Class, AllowMultiple = true, Inherited = true]

第一种情况

如果查找(我们将在后面部分介绍如何在运行时查找一个类的属性)派生类中的Help属性,由于继承属性的设置为 false, 所以找不到该属性。

第二种情况

在这种情况中继承属性的设置也为false,第二种情况与第一种情况类似。

第三种情况

要解释第三种和第四种情况,先在派生类中添加相同的属性。

```
[Help("BaseClass")]
public class Base
{
}
[Help("DeriveClass")]
public class Derive : Base
{
}
```

现在如果我们查找Help属性,由于虽然继承属性设置成了true但不允许出现多个,基类中的Help被导出类的Help属性覆盖,因此只能得到导出类中的属性。

第四种情况

第四种情况中,查询派生类的Help属性时将得到两个属性,因为这种情况下继承和多次出现都是允许的。

> 注意:AttributeUsage属性只对由System.Attribute导出的类并且该属性的AllowMultiple和Inherited都为false时才有效。

Positional 与 Named 参数的比较

Positional参数是Attribute的构造函数的参数。它是强制的,每次将Attribute放入一个程序项必须传递一个值。另一方面Named参数实际上是可选的,不是Attribute构造函数的参数。

在Help类中添加另一个属性来更详细地解释这一点。

```
[AttributeUsage(AttributeTargets.Class,
AllowMultiple = false,
 Inherited = false)]
public class HelpAttribute : Attribute
{
    public HelpAttribute(String Description_in)
    {
        this.description = Description_in;
        this.verion = "No Version is defined for
this class";
    }
    protected String description;
    public String Description
    {
        get
        {
            return this.description;
        }
    }
    protected String version;
    public String Version
    {
        get
        {
            return this.version;
        }
        //如果希望用户可以设置该属性,
```

```
        //必须为它指定set方法
        set
        {
            this.verion = value;
        }
    }
}
[Help("This is Class1")]
public class Class1
{
}
[Help("This is Class2", Version = "1.0")]
public class Class2
{
}
[Help("This is Class3", Version = "2.0",
 Description = "This is do-nothing class")]
public class Class3
{
}
```

当查询Class1的Help属性和它的属性时，将得到：

```
Help.Description : This is Class1
Help.Version :No Version is defined for this class
```

由于没有定义Version属性的值，将使用构造函数中设置的值。如果没有定义值，将使用这种类型的默认值（例如：这种情况下int的默认值为零）。

现在，查询Class2将得到：

```
Help.Description : This is Class2
Help.Version :  1.0
```

对可选参数不要使用多个构造函数。相应的，可以将它们标识为命名参数。之所以称它们为命名参数，因为在构造函数中应用它们的值时，必须为它们命名。例如，在第二个类中我们定义Help。

```
[Help("This is Class2", Version = "1.0")]
```

在AttrubuteUsage例子中，参数ValidOn是一个Positional参数，而Inherited和AllowMultiple是命名参数。

注意：要在属性的构造函数中设置命名参数的值，必须为该属性应用设置方法，否则将会产生编译错误：

```
'Version' : Named attribute argument can't be a
read only property
```

现在查询Class3中的Help属性和它的属性将会怎样呢？结果同样是编译错误：

```
'Desciption' : Named attribute argument can't be a
read only property
```

修改Help类，添加Description的设置方法。这样输出为：

```
Help.Description : This is do-nothing class
Help.Version : 2.0
```

发生的是：首先调用带有Positional参数的构造函数，然后为每个命名参数调用设置方法。在构造函数中设置的值将被set方法覆盖。

属性标识符

假设希望Help属性在整个程序集中都有效。出现的第一个问题是要将Help属性放置在什么位置，这样编译器可以确定该属性在整个程序集中都有效？考虑另一种情况：希望某个方法返回类型是属性。编译器如何确定属性放置在方法返回类型中，而不是整个方法中？

可以使用属性标识符来解决这样的二义性问题。借助于属性标识符可以清楚地确定属性被放置的数据项。

例如：

```
[assembly: Help("this a do-nothing assembly")]
```

Help属性前面的assembly标识符清楚地告诉编译器该属性附属于整个程序集。可用的标识符有：

- assembly
- module
- type
- method
- property
- event
- field
- param
- return

运行时查询属性

已经介绍了如何生成属性以及如何将它们附加到程序元素上。现在来学习类的用户如何在运行时查询信息。

要查询程序项的附加属性必须使用Reflection（反射）。反射能够在运行时找到类型信息。

可以使用.NET Framework Reflection API通过整个程序集的元数据进行迭代，生成所有类、类型以及为该程序集定义的方法的列表。

```
using System;
using System.Reflection;
using System.Diagnostics;

//将Help属性附加到整个程序集
```

```
[assembly : Help("This Assembly demonstrates custom
attributes
 creation and their run-time query.")]
//自定义属性类
public class HelpAttribute : Attribute
{
   public HelpAttribute(String Description_in)
    {
        //
       //完成的工作:在这儿添加构造函数逻辑
       this.description = Description_in;
        //
    }
   protected String description;
   public String Description
    {
       get
        {
          return this.deescription;
        }
    }
}
//将Help属性附加到AnyClass
[HelpString("This is a do-nothing Class.")]
public class AnyClass
{
//将Help属性附加到AnyMethod
   [Help("This is a do-nothing Method.")]
   public void AnyMethod()
    {
    }
//将Help属性附加到AnyInt字段
   [Help("This is any Integer.")]
   public int AnyInt;
}
class QueryApp
{
   public static void Main()
    {
    }
}
```

在接下来的两节中将在Main方法中添加属性查找代码。

查询程序集属性

在下面的代码中,将获取当前进程的名字并使用Assembly类的LoadForm方法加载程序集。然后使用GetCustomAttributes方法得到附属于当前程序集的所有自定义属性。接着的foreach语句循环扫描所有的属性,尝试将每个属性强制转换成Help属性(使用as关键字进行对象强制转换有一个优点:如果转换无效,不必担心会抛出一个异常。相对应的,结果为null)。下一行检查转换是否有效,如果不等于null则显示该属性的Help性质。

```
class QueryApp
{
   public static void Main()
    {
       HelpAttribute HelpAttr;

        //查询Assembly属性
       String assemblyName;
       Process p = Process.GetCurrentProcess();
       assemblyName = p.ProcessName + ".exe";
       Assembly a = Assembly.LoadFrom
(assemblyName);
       foreach (Attribute attr in a.
GetCustomAttributes(true))
        {
          HelpAttr = attr as HelpAttribute;
          if (null != HelpAttr)
           {
             Console.WriteLine("Description of
{0}:\n{1}",
                                 assemblyName,
HelpAttr.Description);
           }
        }
}
}
```

程序的输出为:

```
Description of QueryAttribute.exe:
This Assembly demonstrates custom attributes
creation and
their run-time query.
Press any key to continue
```

查询类、方法和字段属性

下面的代码中,在Main方法中我们唯一不熟悉的是第一行代码。

```
Type type = typeof(AnyClass);
```

它使用typeof运算符得到与AnyClass类相关联的Type对象。查询类属性的剩余部分的代码类似于上面的例子,这里不再解释了。

要查询方法和字段属性,首先得到类中出现的所有方法和字段,然后以与查询类属性相同的方式来查询相关联的属性。

```
class QueryApp
{
   public static void Main()
    {

       Type type = typeof(AnyClass);
       HelpAttribute HelpAttr;


        //查询Class属性
```

(下转72页)

数据库设计经验谈系列之：

数据对象的命名

▶撰文 / 熊建国

编者按

注意到一个很实际的现象:经过大学四年的学习,经过“数据库系统原理”或“数据库系统概论”等课程学习后的本科生毕业后,虽然他们能够对第一到第五范式的定义对答如流,甚至还能进行一些复杂的关系运算,但在数据建模和数据库设计方面依然是一筹莫展。这里,我并不是说理论不重要,而是说这些试图把学生都变成计算机科学家的理论,的确不应该在大学本科阶段出现。经验的出现对于学院派的理论将是一个有力的补充和印证。为此,本刊特约从事多年数据库开发和设计的软件工程师撰稿,从本期开始将推出一系列数据库设计的经验性文章,使读者在数据库设计方面不再茫然。

我们知道,造成这个世界混乱而复杂的重要原因在于名称或称谓的混乱！同一对象的1000种“说法”只会产生1000个诸如“这是什么东东啊？”之类的问题。

经验是最好的老师。在一个学院派的老师和经历过多次数据库项目的开发人员之间,我宁愿选择后者做我的数据库设计课的老师。

好了,言归正传,希望在我的开发故事中,你能够了解那些设计数据库的宝贵的经验。

在分任务分模块设计的今天,程序员除了负责自己模块的编码以外,还要负责自己模块的数据库设计,于是,每个人都有了命名的权利！在崇尚按流程开发的今天,依然有软件公司固执的这么干。遗憾的是,我们不可能指望每个程序员都是数据库设计方面的高手,他的任务应该只是“code”。然而目前许多软件公司的现实情况是,程序员依然干着他们并不十分在行的数据库设计的工作,依然要从后台做到前台,从系统需求分析到编码实现。没有办法,唉！谁要你是中国的程序员呢？你不把自己训练成一个十项全能冠军,软件公司怎么会看得上你呢？

这样的情形会产生什么样的后果呢？我只说说我看到的两种极端的情况。一种是有人将DBMS仅仅当成一个放数据表的地方,DBMS的许多强大功能,完全没有使用,甚至连视图也不建(这种情况一般是出现于那些对该DBMS不熟悉或对数据库设计毫无经验的程序员身上)。另外一种情况是在后台数据库上面过多的开发,加入了太多的商业逻辑处理(这种情况肯定会出现在那些对该DBMS非常熟悉的程序员身上)。我们当然知道,这两种方式绝不是理想的数据库设计方案。

怎样才能产生一个最佳的设计呢？我的建议是由专门的系统分析人员(数据库专家)设计数据库。一般的软件公司即便做不到这一点,也应该在数据库设计之前,大家商定一些基本原则和规范,然后由程序员们在需求分析过程中一起交流设计,切切不能各自为政,互不沟通。

下面我要说的是数据库设计过程中数据对象命名的一些规范。

有一个总的原则是,所有的数据对象千万别用汉语拼音或其缩写来命名。早年,我曾经做过一个项目,属于系统改造。一个并不十分复杂的桌面应用,数据库是access, 要将其功能扩充, 变成一个后台数据库为sql server的网络应用程序, 原先的数据、功能保留。我们打开数据库一看,当时就懵了,因为所有的数据对象都是拼音缩写,并且经过多年运行,系统文档也丢了。我们就那样一个一个对象的猜测,一行行代码的分析,最后才算完成了任务。能够在那个项目中幸存下来,我只想说感谢上帝！所以,我想说的是,数据对象的命名最好用英文,或用英文缩写。数据表的命名对于数据表,如果能用一个英文单词表示,就不要缩写(我不明白有些家伙为什么生怕多打了一个字母！)比如员工表,就用Employees(单复数无所谓,但是一定要统一。如有人可能写成Employee,那么所有的表名就请都用单数。),干嘛非要写成“Empl”让人家猜呢？

对数据表不是很多的小型数据库,可以这样做。但是对于数据对象多达上百,上千的大型数据库,必须加上前缀, 用以标明对象的属性。如“T_Employees”,这

里的“T_”表示数据对象是“表(table)”,如果是试图,则以“V_”(view)开头,索引可以用“I_”(index)开头,存储过程用“SP_”(store process)开头,触发器用“TR_(triggers)”。

对oracle里面许多数据对象,如包、序列、同义词等都可以依次地来命名。如果表名是两个单词表示的怎么办?如有很多考核表,项目考核表,安全考核表等,我们可以这样命名,“ProjectAssess”“SecurityAssess”。但是,表名长度不要超过30个字符,如果发现字符很长,我建议可以换一个同义的字符数较少的单词来表达。当然,对表名超过两个单词的情况也应该适当进行转换,在不影响理解的前提下。我还从未遇到过非要三个或5个英文单词才能表达清楚意思的情况。

如在微软的范例数据库“北风”(northwind)中“顾客统计表”就被命名为“CustomerDemographics”。每个单词首字母大写,这一点很多人不习惯,大多是些面向对象编程经验不足的家伙。如果做过多年面向对象编程的人就知道,一般是一表对应一类,和我们类名的命名规范一样。但是,“北风”中有个很不好的命名就是“定购主细表”的定购细目表的命名是“Order Details”,在两个单词之间留有空格,这一点在数据库的设计中我认为实在是大忌!千万不要加什么空格,大家都应该知道在程序调试时空格的危害和可怕之处。

如果你觉得这样定义的表名还是太长,那么你可以通过定义别名来缩短表名称的字符数。不过,值得注意的是,别名的出现,将会在代码理解和实际编程中产生新的混乱。

关于表的别名命名规则:一般做法是,如果表名是一个单词,别名就取单词的前4 个字母;如果表名是两个单词,就各取两个单词的前两个字母组成4 个字母长的别名;如果表的名字由3 个单词组成,可以从头两个单词中各取一个然后从最后一个单词中再取出两个字母,结果还是可以组成4 字母长的别名。

视图的命名规则就可以建立在别名的基础上,如我们建立一个来自于两个表的视图,可以命名为V_表A(别名)_表B(别名)。如果连接的表过多可以将表别名适当简化,但一定要列出所有表名。这样可能写起来麻烦,但是看起来清楚,方便日后的维护。(另:在编程过程中书写sql语句之前,我经常将数据对象全部导出,包括字段名和类型,放在写字板里面,按照字母排序放好,需要哪个对象名,直接过来copy就行了,我才不会每次都敲键盘呢!)

数据表字段命名原则,推荐两种常见方式:其一:数据库字段名全部采用小写英文单词,单词之间用“_”隔开,命名规则是表别名+单词,如:user_id,user_name等;其二:不用下划线分隔符号,如“employeeId”,“firstName”其中包含的所有单词都应紧靠在一起,而且大写中间单词的首字母。这一点符合面向对象的编程规则,字段首字母应该小写,当我们在定义类的字段时,直接就可以使用数据表的字段名。需要说明的是,如果表有一个自动ID,是数据库自动生成的编码,则应该统一命名为“ID”;对自定义的逻辑上的编码则用表名或其缩写加“ID”的方法命名。

如果系统比较复杂、庞大,也可以用系统模块名称将数据对象区别开。比如说,两个不同的模块中都有用户表的情况,怎么办?那么,我们可以这样:属于论坛模块的表,表示为“forum_T_Uers”,即在前面加上模块名称。当然,这样命名要结合系统的特点和复杂度。我曾经就遇到过系统模块比较多,程序员在后台“到处寻找”自己的表的情形。

除了对数据对象进行按统一的规则命名以外,还必须确定“数据对象”的定义和注释规范。每个数据对象包括表、存储过程、函数和字段都必须作相关的注释说明(说明用途,修改信息等)。如果数据对象是函数或存储过程,还必须说明作者,功能和创立时间(修改时间),输入输出参数等信息,这些信息都应该用列表详细说明。如:下面是对表的注释规范

表名	T_USERS
功能	记录系统用户信息
创建时间	2003-8-5
修改记录	
字段	字段类型
userId	int
userName	Varchar(20)
……	……

关于数据库设计中数据对象的命名原则(或规则),每个人都可以来说上几句,就象我在这里喋喋不休一样。但是,大部分人说的无非都是自己的习惯而已。因为,没有那一本教科书告诉我们一定得这样命名,而那样就是错的。这个世界就是如此,有时我们渴望异彩纷呈,有时,我们又希望天下大同,如,要是只有一种编程语言就好!要是只有一种命名规则就好……,当然,这一天是永远也不可能到来了!

Oracle8i与MS SQL SERVER之比较

▶撰文 / xxj

编者按

很多学习数据库的人，从SQL server过渡到oracle时，都感觉非常痛苦，主要是一些概念理解起来发生了变化，很多人因此而放弃了继续学习的打算。这篇文章，也许能为您扫清学习道路上的障碍。

下面是我个人的一点体会，欢迎指正。对于Oracle 8i初学者，很有可能会把MS SQL Server中的概念拿来与Oracle对照，我个人认为不需要做这种比较，其实一个没有学过其它数据库管理的人，可能更容易学习Oracle。现在我来说一下两者的区别和联系。

1、 关于数据库的概念

怎么找到或者创建自己的数据库，Oracle的数据库的概念已经完全不同于MS SQL。Oracle的服务和数据库，相当于MS SQL的数据库服务，Oracle的服务=后台进程+相关内存，数据库=数据文件的集合，而且Oracle的服务是完全可以与数据文件脱离开来的。那么MS SQL的数据库，在Oracle中哪能找到呢？答案是Oracle的Schema。Schema的中文意思是：方案，指一个用户所拥有所有对象的集合。这里的对象包括表、视图、实例化视图、序列、过程、函数、程序包、同义词（下面我会详细解释其中的一些陌生概念）。

下面我们要建立一个在Oracle中的类似于MS SQL的数据库，大致过程如下：

❒ 建立数据表空间（相当于MS SQL的数据库的数据文件）

❒ 建立临时表空间（相当于MS SQL的数据库的事务日志文件）

❒ 建立用户，它的缺省表空间和临时表空间是刚建立的两个表空间

❒ 给用户授权，最起码的角色权限是Connect 的角色

❒ 以该用户登录，创建自己的表、视图、实例化视图、序列、过程、函数、程序包、同义词等

2、 关于Oracle Manager Server与MS SQL的Enetrprise manager

应该说这两者有很大的差别，DBA Studio工具提供了与MS SQL的Enetrprise manager同样的功用，DBA Studio工具对一般使用来说已经足够了。但这里有必要对Oracle Manager Server解释一下：

❒ 它是Oracle管理分布式数据库的服务，注意它是服务

❒ 它缺省并不安装

❒ 它需要有自己管理所需要的资料库，在某个数据库中需要建立相应的用户

❒ 它的登录需要身份验证，注意这里的身份验证，与管理所需要的资料库的用户不是一个概念，也不是数据库中的Sys和system用户，它的缺省用户是oemtemp

❒ 使用它，在管理端要启动Manager Server服务和智能代理服务（OracleAgent），被管理的数据库服务器要启动智能代理服务（OracleAgent）

3、关于Oracle 的Sys、System的用户与MS SQL的master的比较

MS SQL的master数据库存储了当前数据库服务的一些配置信息，如数据库设备（在MS SQL已经弱化）、字符集、数据文件、登录帐号、拥有的数据库、整个服务参数配置等。

Oracle的Sys存储了Oracle服务或者实例的信息及所有用户的数据字典信息，这一点不同于MS SQL。MS SQL中每一个数据库拥有自己的对象的数据字典信息。System用户拥有的数据字典是视图信息，有了这些视图，我们查询数据库的信息时就特别方便。缺省情况下，system用户拥有DBA系统角色权限，而sys不仅拥有DBA 的权限还拥有SysDBA的权限。

那DBA、SysDBA这两个系统角色有什么区别呢？

在说明这一点之前,我需要说一下Oracle服务的创建过程:

❒ 创建实例

❒ 启动实例

❒ 创建数据库(system表空间是必须的)

启动过程:

❒ 实例启动

❒ 装载数据库

❒ 打开数据库

SysDBA, 是管理Oracle实例的。它的存在不依赖于整个数据库完全启动,只要实例启动了,它就已经存在。以SysDBA身份登录,装载数据库、打开数据库,只有数据库打开了,或者说整个数据库完全启动后,DBA角色才有了存在的基础!

4、Oracle中新的数据库对象:实例化视图、快照、序列、程序包、同义词、抽象的数据类型

❒ 实例化视图又称显形图:实例化说明它有自己的存储空间,视图说明它的数据来源于其它表数据。

实例化视图中的数据,设置为隔一段时间更新数据,更新的模式可以定义为完全更新和增量更新。

❒ 快照基本上相同于实例化视图,只不过数据来源不同,快照数据来源于远程数据库,而实例化视图则来源于本地数据表。

❒ 序列, 相当于MS SQL中的identity列, 它是一个数字顺序列表。

❒ 程序包:它是过程、函数、全局变量的集合,它封装了私有变量、私有过程和私有函数。

如:dbms_out包

❒ 同义词:是数据库中的对象的别名,同义词可以是全局的也可以是私有的(属于某个用户的)。

如:Tab,col等

❒ 抽象的数据类型,类似于C中的结构体或Pascal记录类型。

关于类型,这里还有一个题外话:

A Tab%RowType,这是一个特别的抽象的数据类型,该类型的分量就是TAB的字段。

B Tab.TName%Type,这定义了一个和tab的字段TNAME相同的数据类型的变量,想想它有什么好处。

5、Oracle数据库连接和MS SQL远程连接

两者都是为了实现分布式数据库的操作。

两者都能实现分布式事务,具体请参见MS SQL的联机帮助。

6、Oracle回滚段和MS SQL的数据库事务日志文件

回滚段提供了事物回滚需要使用的数据变化以前的映象。这些映象是按条目存储的,如果这些条目过少,一个事务等待另一个事务的几率增大,就会影响数据库的性能。缺省安装时,提供一个系统回滚段,它在System表空间。为了提高性能,system表空间不应存储有任何数据字典信息以外的信息。

MSSQL数据库事务日志文件功能雷同于回滚段,只不过它是同特定的数据库密切相关的。

7、关于数据表的管理

❒ 超大型数据表的管理

一个数据表,尤其是那种流水帐表,长年累月后急剧膨胀,最后影响查询性能怎么办?Oracle和MS SQL都提供了一种方法:把数据文件及其索引存放在一个特定的数据文件或表空间里,但这个还是不能解决根本问题。怎么办?开发人员只能给表添加时间的标志:如CWSJ2000(财务数据2000)、CWSJ2001、CWSJ2002。这种命名方式,确实能提高查询性能,但是给开发带来了不少的麻烦(浪潮财务就是这样做的),而且当你不知道数据在哪个表中的时候,你要联合这些表进行查询,岂不长哉!Oracle提供了解决这个问题的很好手段:表及索引分区存储。按字段值的范围进行分区存储.具体做法不再说明。

❒ Oracle索引组织表和MS SQL簇索引表

两者的数据存放顺序都是按照索引值的顺序存放的。

❒ 无事务回滚概念的表,对这种表的操作不存在事务的概念

记得以前MySQL是不提供事务的回滚(不知现在有否变化),而Oracle提供了建表参数nologging,使对该表的操作不参与事物的回滚。

❒ 索引

Oracle提供了多种MS SQL没有的索引类型

①位图索引

比方说性别:仅有男女

第1条记录:男

第2条记录:男

第3条记录:女

第4条记录:男

第5条记录:女

第6条记录:男

.....

那它的索引:110101.....

看这种索引,多节省空间,它适用于字段值是已知几个中的一个。

②基于函数或者说表达式的索引,这个功能可谓强大。

8、外连接

MS SQL SERVER 支持两种形式表间连接:

①从Sybase继承来的形式:

字段1 *= 字段2(左连接)

字段1 =* 字段2(右连接)

没有这种形式的全外连接语法

②标准的外连接语法

left [outer] join on 逻辑表达式

right [outer] join on 逻辑表达式

full [outer] join (全外连接) on 逻辑表达式

这里的逻辑表达式,可以是很复杂的表达式,例如:

A.ID=B.ID AND (A.Parebt_ID=1 OR A.Parent_ID=2)

需要提醒是:你写的查询语句报告过这样的错误

```
Joined tables cannot be specified in a query containing outer join operators.Joined tables cannot be specified in a query containing outer join operators. View or function 'dbo.VU_CAF_BILLS' contains joined tables
```

这句话告诉你,查询语句引用的视图或者子查询也用到了外连接,但是引用视图或者子查询外连接语法与你的外连接语法不一致导致出错。

例如:select A.[ZONE],A.FLAG,A.FlagDesc,A.CAF_NO

```
from dbo.VU_CAF_BILLS A,TU_Flag
where A.CAF_NO*=TU_Flag.ObjNo
```

视图dbo.VU_CAF_BILLS的外连接语法是标准的SQL语法,而本语句中的外连接语法却是Sybase式的外连接语法。

❒ Oracle不支持标准的外连接语法,也没有全外连接,这是它的缺陷

字段1 = 字段2(+)(左连接)

字段1(+) = 字段2(右连接)

❒ 使用外连接语句的用处

①不想因为表连接而使主表数据行丢失,这一点毋庸多说。

②找某条记录在A表存在,而在B表不存在,按常规做法使用not in(select 查询子句)语法,使用not in最大的缺点速度慢,原因是每个数据行都去做:select查询子句,而使用下面的语法:

```
select TU_COMPANY.* from TU_COMPANY left join TU_Comp_Agent on TU_COMPANY.ID=TU_Comp_Agent.CompCode
where TU_Comp_Agent.Id is null
```

9、触发器

❒ 我的了解到, MS SQL SERVER。仅有表的触发器,而且触发时机不够丰富。

如插入触发机制,不区分单条插入还是多条插入,也不区分插入前触发还是插入后触发;碰到多条数据的插入,需要使用游标处理每条插入的数据。

❒ Oracle提供的触发器不仅有基于表的触发器,而且有其它类型的,例如数据库级的触发器:数据库启动、数据库关闭。对于表级的触发器,区分单条插入还是多条插入,也区分插入前触发还是插入后触发。

10、表数据复制

库内数据复制

❒ MS SQL Server

Insert into 复制表名称select语句(复制表已经存在)

select 字段列表into复制表名称from表(复制表不存在)

❒ Oracle

Insert into 复制表名称 select 语句(复制表已经存在)

create table 复制表名称 as select 语句(复制表不存在);文本文件转入、转出的批量处理

❒ MS SQL Server

BCP命令行程序

❒ Oracle

SQLLDR命令行程序

11、关于存储过程或函数中使用的临时表，两者都提供了这个功能

临时表最主要的好处是,操作不留任何痕迹、不产生日志,所以速度快。

❒ MS SQL SERVER

CREATE TABLE #表名称.....或者 SELECT 字段表达式列表INTO#表名称FROM。

表名称前加#即可,这些临时表都是只在一个数据库连接会话期间有效。

❒ Oracle

create [Global] Temporary Table ,加上[Global]就是全局的临时表(所有数据库连接会话都是可见的),不加则为私有的(在一个数据库连接会话期间有效)。

12、两者都提供了自己的桌面版的数据库

❒ MS SQL SERVER(毋庸我多说了)

基本上桌面版的数据库提供的功能跟服务器版的功能没多大区别;缺少的只是全文搜索服务功能。

❒ Oracle

Oracle Lite 8,我就看过别人用Oracle lite 8做过海关报关系统。

补充：Sybase提供了桌面版的 Sybase SQL any-where

13、动态执行SQL语句

❒ MS SQL SERVER 7.0好象没有这个功能,MS SQL SERVER 2000已经具备这个功能。

你是不是想在存储过程的参数中传递一个表名或者在过程体里动态生成一个SQL语句,你会发现很难办到。看了下面的例子:你以前的问题全解决了。

```
declare @count int
declare @SQL nvarchar(200)
set @SQL = N'select count(*) from sysobjects'
exec sp_executesql @SQL,N'@i int output',@count
output
```

❒ Oracle提供了两种方法实现这个功能

①程序包DBMS_SQL,执行一个语句的过程:

打开游标(open_cursor,对于非查询语句,无此过程)

分析语句(Parse)

绑定变量(bind_variable)

执行语句(execute)

关闭游标(close_cursor,对于非查询语句,无此过程)

② execute immediate ls_SQL

14、数据库备份和恢复

两者都提供了很强的数据库的备份和恢复能力,Oracle提供了更多的手段,Oracle宣称它的数据库是不可摧毁的也不是瞎吹牛。

❒ MS SQL SERVER

①数据库的导入导出DTS工具,如果数据源和目的都是MS SQL SERVER,则可以完全复制数据库的结构(包括表、视图、索引、触发子、规则、默认、存储过程、用户定义函数、表数据)。

②数据库备份和恢复命令

backup database

restore database

导出的数据文件还可以压缩,这一点不同于Sybase,而且这种数据库备份和恢复的方式可以是增量的或完全的。

③数据库的附加Attach

只要数据库的原始数据和日志文件没有损坏,就可以用命令:SP_ATTACH_DB

这种情况,有时候会出问题:比方说你键了登录帐号,不是使用缺省的登录帐号sa,由于登录帐号的信息存放在master数据库中,所以你使用原来建立的帐号就登录不上,所以要做一些侯补工作。

❒ Oracle

①导出导入工具exp和imp工具

导入导出的参数何其多。

②冷备份或者称脱机备份,备份的时候数据库是停止的

备份所有的数据文件、日志文件、控制文件。

③热备份或者联机备份

数据库处在ARCHIVELOG模式,注意缺省情况下都是 ARCHIVELOG。

恢复时可以选择完全恢复、时间点恢复、SCN点恢复、用户自由干预的CANCEL恢复。

可以说它的热备份的恢复功能是非常之强大的。

④ RMAN恢复管理器,本人正在研究中,请待后续。

高手的风范——淡妆浓抹总相宜

▶撰文 / 行舟

剑客高手与软件大师

现在的年轻一族,不知道古龙的人并不多;而不了解古龙的程序员,恐怕更是少之又少。当然,逝去的古大侠并不是程序员,把他作为这篇文章的主角自然也并不合适;而我要回想的,也只不过是他留给我们的一个经典的故事。

话说有一个来自东瀛的白袍剑客,专向中原的武林高手挑战,而他的战术也非常特别,即他自己用剑削下的一根枯枝。这个枯枝在一般人的眼里,可以说是毫无价值,但它却足够吓破许多威震江湖的武林高手的胆,也足够打动三十余年来不屑与人交手的紫衣侯的心,因为这根看似随手削下的枯枝,已经显露了白袍剑客的绝世武功。

行家一伸手,便知有没有,这句广为人知的谚语里蕴含了深刻的道理。对于我们程序员来说,这句话也丝毫未见例外。武功高超的剑客,除非不出剑,一出剑便要夺人心魄;而功力精湛的程序员,除非不出手,一出手也将尽昭才华。这既是因为他们多年经验所养成的良好习惯,也因为他们在不自觉地维护自己所应该具有的风范。

示例代码与书的质量

众所周知,挑选好的电脑书籍是异常困难的事情,介绍选书的方法和怎样判断书籍是否优秀的文章也时常得见。虽然公说公有理,婆说婆有理,但根据书中的示例代码来判断全书的含金量,窃以为这是一个放之四海而皆准的原则。如果用这个原则来比较中外书籍,我们可以看出一个明显的对比,或者说是差距——让人悲哀的差距:外版的经典书籍中的示例都是作者精选题材、细雕代码而成,每个示例既精彩,又实用,而且可用,印刷版式也是编排适当、格式井然;而当我们对比国内的书籍时,愤慨的情绪便几乎是忍无可忍了,这些书中的代码往往是拷贝的相关语言的帮助文档,或是从网上搜集到的、当然是随处可见的源码,排版也就更不必说了,往往格式杂乱,谬误纷呈。国人作者能够自己真正有实力编写示例代码的寥寥无几啊,文抄公耶,文造公也!

这里不妨以谭浩强编写的系列教材为例,其优劣我不加妄论,我要说的是看看其书中的示例代码(如《C语言程序设计》),再比较国外的同类书籍,结果如何,我想各位心中都有了谱吧。

Jeffrey Richter和《Windows核心编程》

如果你是一名Windows程序员,如果你不知道Jeffrey Richter是干什么的,那未免太煞风景了。简而言之,这个带有传奇色彩的人对于Windows的了解,也许比微软那些开发Windows的工程师还要多那么一点点,而他的经典著作《Windows核心编程》(英文原名:Programming Applications for Microsoft Windows, Fourth Edition),可谓在Windows下进行开发的圣经,譬如我曾经苦苦寻觅才得到的许多技巧,都在这本书中找到了源头。

当然,这本书也完全满足上面提及的评书原则,书中的每一个示例都是Jeffrey精制而成。每一个示例都有用意、每一段代码都体现功底;每一个示例无论长度,都是完整的,每一段代码也都是冗繁削尽,没有刻意的花招,也鲜有无用的废码,不经意之间,显得恰到好处。

Hello, error

“Hello, world!”,这是我们最为亲切和熟悉的东西了,自从Turbo C以来,许多电脑书籍都把显示这行文字作为第一个例子。但在Jeffrey看来,这个例子显然还不够好,因为他觉得这个例子并没有多少实用性,他追求的是每一个例程都可以给读者带来真正的价值。

编写Windows程序,不可避免地要调用大量的Windows函数。有经验的程序员都知道,由于Windows系统的复杂性,在调用Windows函数时发生错误是不可避免的现象。Windows的函数在操作失败后,将设置一个与当前线程相关的错误代码,以标记出错的原因。我们可以用GetLastError函数取得这个代码,然后根据它

进行相应的处理。如果要成为优秀的Windows程序员，在程序中进行这些善后的处理是必要的，但是如果没有足够多的经验，单凭这个错误代码，我们还是难以了解到底发生了什么意外，我们需要获得更多的有关信息。

所以，虽然用“Hello, world！”来欢迎刚开始学习编程的人很合适，但是对于Windows程序员来说，我们更需要的是说一声，“Hello, error！”。于是，Jeffrey的第一个例子便是设计一个工具，根据错误代码来查阅详细的错误信息。

必须说明，这是一个很有用的工具，甚至于还有人设计了这样的共享软件，我就下载了一个。其实，Visual Studio中也附带了这样的工具——“Error Lookup”，在Visual Studio.NET 中，这个工具继续得到了保留。试想，当我们完成了这个例子后，就得到了一个“Visual Studio.NET等级”的工具，甚至于有可能把它“Share”出去，是不是特别有成就感呢？

浓妆淡抹总相宜

如果看了这个示例，我们会觉得，作为全书的第一个示例，的确较为简单。但是，Jeffrey并没有因为其简单，就忽略了程序设计的某个方面，Jeffrey展示的是一个完完整整的软件源码。

清单1：ErrorShow.cpp

```
//关于整个模块文件的说明,标出文件名称、最后改写时间、作者
/******************************************************
Module: ErrorShow.cpp
Notices: Copyright (c) 2000 Jeffrey Richter
******************************************************/
//统一的全程头文件
#include "..\CmnHdr.h"          /* See Appendix A. */
#include <Windowsx.h>   // 使用Windows提供的宏,简化代码
#include <tchar.h>              // 支持Unicode编码
#include "Resource.h"
//////////////////////////////////////////////////////
//通过自己消息在进程之间通信
#define ESM_POKECODEANDLOOKUP    (WM_USER + 100)
//自己的窗口名称
const TCHAR g_szAppName[] = TEXT("Error Show");
//////////////////////////////////////////////////////
//初使化对话框
bool Dlg_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM
 lParam) {
  // 设置自己的图标
  chSETDLGICONS(hwnd, IDI_ERRORSHOW);
  // 目前,最大的错误代码不超过5位,这里限制了允许输入
     的字符个数
  // Don't accept error codes more than 5 digits long
  Edit_LimitText(GetDlgItem(hwnd, IDC_ERRORCODE), 5);
  // 可以处理命令行的参数
  // Look up the command-line passed error number
  SendMessage(hwnd, ESM_POKECODEANDLOOKUP, lParam, 0); return(TRUE);
}
//////////////////////////////////////////////////////
//处理命令消息
void Dlg_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
   // 采用switch语句,最经典的做法
   switch (id) {
   // 没有必要对消息的处理都设计为一个函数,所以优先处理简单的消息
   case IDCANCEL:
      EndDialog(hwnd, id);
      break;
   // 设置主窗口总在最前面
   case IDC_ALWAYSONTOP:
      SetWindowPos(hwnd, IsDlgButtonChecked(hwnd,
                  IDC_ALWAYSONTOP)
         ? HWND_TOPMOST : HWND_NOTOPMOST, 0, 0, 0, 0,
            SWP_NOMOVE | SWP_NOSIZE);
      break;
   // 无效的数据时,禁止查找按钮
   case IDC_ERRORCODE:
      EnableWindow(GetDlgItem(hwnd, IDOK),
            Edit_GetTextLength(hwndCtl) > 0);
      break;
   case IDOK:
      // Get the error code
      DWORD dwError = GetDlgItemInt(hwnd, IDC_ERRORCODE,NULL, FALSE);
      // 定义指针时赋以NULL初值
      HLOCAL hlocal = NULL;   // Buffer that gets the
                                 error message string
      // Get the error code's textual description
      BOOL fOk = FormatMessage(
       FORMAT_MESSAGE_FROM_SYSTEM |
       FORMAT_MESSAGE_ALLOCATE_BUFFER,
         NULL, dwError,
         MAKELANGID(LANG_ENGLISH, SUBLANG_ENGLISH_US),
         (PTSTR) &hlocal, 0, NULL);
      // 直接调用FormatMessage失败后,再试试
      if (!fOk) {
         // Is it a network-related error?
         HMODULE hDll = LoadLibraryEx(TEXT("netmsg.dll"), NULL,
            DONT_RESOLVE_DLL_REFERENCES);
         if (hDll != NULL) {
            FormatMessage(
               FORMAT_MESSAGE_FROM_HMODULE |
                  FORMAT_MESSAGE_FROM_SYSTEM,
               hDll, dwError, MAKELANGID(LANG_ENGLISH,
                  SUBLANG_ENGLISH_US),
                (PTSTR) &hlocal, 0, NULL);
            FreeLibrary(hDll);
         }
      }
      if (hlocal != NULL) {
         SetDlgItemText(hwnd, IDC_ERRORTEXT, (LPCTSTR) LocalLock(hlocal));
         LocalFree(hlocal);
      } else {
         SetDlgItemText(hwnd, IDC_ERRORTEXT, TEXT("Error number not found."));
}
// 虽然是最后的选择了,但还是加上了break
      break;
   }
}
```

```
//////////////////////////////////////////////////////
// 对话框消息循环函数
INT_PTR WINAPI Dlg_Proc(HWND hwnd, UINT uMsg, WPARAM
 wParam, LPARAM lParam) {
  // 采用预定义的宏,使得代码大为简化了
  switch (uMsg) {
    chHANDLE_DLGMSG(hwnd, WM_INITDIALOG, Dlg_OnInitDialog);
    chHANDLE_DLGMSG(hwnd, WM_COMMAND,  Dlg_OnCommand);
  // 处理重复启动的实例传来的消息
  case ESM_POKECODEANDLOOKUP:
     SetDlgItemInt(hwnd, IDC_ERRORCODE, (UINT) wParam, FALSE);
    FORWARD_WM_COMMAND(hwnd, IDOK, GetDlgItem(hwnd,
      IDOK), BN_CLICKED,
        PostMessage);
    SetForegroundWindow(hwnd);
    break;
  }
  return(FALSE);
}
//////////////////////////////////////////////////////
// 入口函数,注意,不是我们熟悉的WinMain
int WINAPI _tWinMain(HINSTANCE hinstExe, HINSTANCE,
 PTSTR pszCmdLine, int) {
  // 先坚持是否已经启动了实例
  HWND hwnd = FindWindow(TEXT("#32770"), TEXT("Error Show"));
  if (IsWindow(hwnd)) {
    // An instance is already running, activate it and
     send it the new #
    SendMessage(hwnd, ESM_POKECODEANDLOOKUP, _ttoi
     (pszCmdLine), 0);
  } else {
    DialogBoxParam(hinstExe, MAKEINTRESOURCE(IDD_ERRORSHOW),
      NULL, (DLGPROC)Dlg_Proc, _ttoi(pszCmdLine));
  }
  return(0);
}
//////////////////////// End of File ////////////////////
```

清单1[2]是其除开头文件之外唯一的源文件,Jeffrey用尽量简单而清晰的代码,完成了看似简单而并非无用的功能。虽然这段代码只有寥寥数十行,但仍然显露了Jeffrey深厚功力的冰山一角,同时也表现了Jeffrey良好的编程风格。虽然这些代码需要占用一些篇幅,但是我想,把它包含在这篇文章中,让我们细细品味一番,是不无收获的。

1、模块说明:在源文件的最前面,有关于模块的说明。对于简单的工程,这样做也许还体现不了多大的作用,但是如果工程中包含了大量的模块文件,或者在他人阅读你的源文件时,对模块进行说明的作用就会充分体现了。虽然这只是一个简单的示例,但Jeffrey并不以事小而不为,对模块进行了说明。

2、支持Unicode:如果我们要让自己的软件能够走出国门,就必须处理使用该软件的外国人所使用的文字,这实际上是软件国际化的问题。程序员使用最多的是ANSI编码,但是如果我们在程序中使用ANSI编码,则显然难以满足软件国际化的要求。解决这个问题的方法便是使用Unicode编码,也即如果我们在开发软件时,眼睛看得远一点,那么使用Unicode编码将是十分明智的选择。我们看到,尽管ErrorShow只是一个简单的示例,Jeffrey仍然提供了对于Unicode编码的支持。我们也可以从这个了解到,其实在程序中使用Unicode编码并没有想象的难,我们需要做到的无非是,在源文件中包含tchar.h头文件,这个文件中定义了一系列的宏,如果我们在程序中使用这些宏来代替具体的API名称和变量类型,那么根据是否定义了_UNICODE标志,编译器将为你自动选择相应的API函数或变量类型。

3、容错处理:由于误操作或参数输入错误等情况,将影响程序的运行结果,而好的程序也有责任对于一些意外的情况进行适当的防范。对于本示例来说,似乎没有进行容错处理的需要,但Jeffrey竟然在四处进行了容错处理,可谓事无巨细,明察秋毫。首先,他在对话框的资源中,把输入错误编号的Edit控件设置了“Number”属性,因为错误代码是一个整数。另外,在初使化对话框时,他限制这个Edit控件最多只能输入5位,因为目前Windows的错误代码都不超过5位数。再者,如果用户没有输入任何数据,他就禁止了查找按钮,这也是很显然的容错代码。顺便指出,Visual Studio提供的Error Lookup一直没有进行这一处理。最后,根据错误代码取得错误字符串,一般调用FormatMessage函数都会成功,但是也有意外,当然,Jeffrey照样不会留下这样的漏洞,如果没有取得错误信息,他就再次尝试,在与网络相关的错误信息资源的netmsg.dll文件中查找错误信息,以期获得每一个错误代码的含义。

4、支持命令行参数:我觉得这个程序完全是没有必要支持命令行参数的,但是Jeffrey追求的是一种完美,他的程序支持传入一个错误代码的参数。当然,我们也看到,藉着他高超的编程技巧,这一切的实现似乎并没有增加任何的代码。

5、防止多次启动:有时我们不希望程序启动多个实例,在这个例子中,启动多个实例显然没有什么用处,所以Jeffrey在启动实例时加了个判断,在创建主窗口之前,先查找是否已经创建了指定属性的窗口,如果发现了,就激活这个窗口,并且把命令行参数传递过去。在这里,Jeffrey还顺便示例了通过消息传递在进程间通讯的方法,可谓处处留心。当然,进行这些处理的代码,也已经被精简到最少了。

6、自己的图标:Windows窗口程序都可以拥有自己独特的图标,但是如果我们稍微留心一下,就会发现许多Windows例子都没有自己的图标,而是采用了开发工具提供的缺省图标,甚至有不少的软件都是这样。我们看到Jeffrey特意为程序准备了一个图标。虽然Jeffrey准备的图标很简单,但是他毕竟没有忽视这点,而且准备的图标还算恰当。如果说图标是窗口的形象代表,那么连自己图标都没有的软件,都将是不完整的东西。(下转122页)

保龄球计分程序设计实践

——《敏捷软件开发》精彩节选

【编者按:相对于编程技术而言,设计是一种艺术。想要成为这门艺术的高手,除了多思多想多实践多交流之外,恐怕没有什么更好的办法。可是以往我们的书籍和文章是怎么做的呢,干巴巴的理论,刻板的面孔,形式化的叙述方式,令人避之唯恐不及。《CSDN开发高手》认为,实践,只有实践,才是提高设计能力的不二法门。为此,我们推出了"大话Design"这个栏目,通过对话记录、讲故事这样的方式,把软件设计者们的实践生动地展示在读者的面前。作为开篇,我们特别经过了清华大学出版社的许可,从即将由清华大学出版社出版的2003年计算机图书Jolt震撼大奖作品《敏捷软件开发》中截取了一段精彩片断,既是借此机会向大家推荐这本出色的技术书籍,同时也是为我们的栏目制定基调。欢迎大家对这个栏目发表自己的批评和见解,更欢迎您的建议和参与。】

为了演示一下XP的编程实践, Bob Koss (RSK) 和Bob Martin (RCM) 要在一个小型的应用程序中使用结对编程 (pair programming) 的方法, 你可以在一边进行观看。在创建该应用程序的过程中,会使用测试驱动的开发方法以及大量的重构。接下来的一幕是这两个Bob于2000年末在一家旅馆中实际编程情景的真实再现。

在创建这个程序的过程中,我们犯了很多的错误。这些错误包括代码方面的、逻辑方面的、设计方面的以及需求方面的。在学习本章时,会看到我们围绕这些方面所进行的活动:识别出错误和误解,然后处理它们。过程是混乱的——过程中只要有人参与都是这样。结果……唔,令人吃惊,竟然能够从这样一个混乱的过程中出现秩序。

这个程序是计算保龄球比赛的得分的,所以如果知道保龄球比赛的规则,会有助于理解本章内容。如果对保龄球比赛的规则不了解的话,可以察看章末的补充内容。

保龄球比赛

RCM:可以帮忙编写一个保龄球记分小程序吗?

RSK:(自言自语,"XP中结对编程的实践规定当有人请求帮助时,不能够说'不'。若请求的人是你的老板,就更不能拒绝了。")当然可以, Bob, 非常高兴帮助你。

RCM:太好了,我想编写一个应用程序来记录一届保龄球联赛。需要记录下所有的比赛、确定团队的等级、确定每次周赛的优胜者和失败者,并且准确地记录每场比赛的成绩。

RSK:棒极了。我曾经是个很好的保龄球选手。这件事情很有趣。你已经列出了一些用户素材,想先做哪一个呢?

RCM:先来实现记录一场比赛成绩的功能吧

RSK:好。它指的是什么呢?该素材的输入和输出是什么呢?

RCM:在我看来, 输入只是一个投掷 (throw) 的序列。一次投掷仅仅是一个整数,表明了此次投球所击倒的木瓶数目。输出就是每一轮 (frame) 的得分。

RSK:如果你在这个练习中担任客户的角色,会希望什么形式的输入和输出呢?

RCM:好, 我担任客户。我们需要一个函数, 调用它可以添加投掷,还需要另外的函数用来获取得分。有几分像下面的样子

```
ThrowBall(6);
ThrowBall(3);
assertEquals(9, getScore( ) );
```

RSK：好，我们需要一些测试数据。我来画一张记

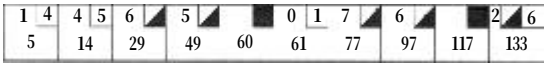
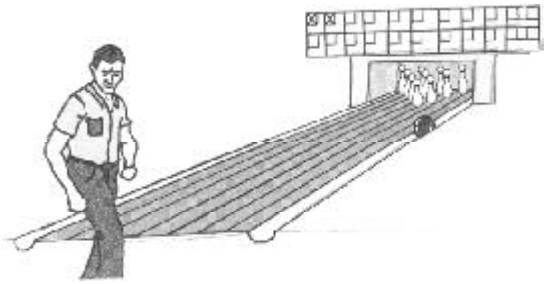


图1　典型的保龄球比赛记分卡

分卡的小草图。(参见图1)

RCM:这名选手发挥的很不稳定。

RSK:或许喝醉了,但是可以作为一个相当好的验收测试用例。



RCM:我们还需要其他的验收测试用例,稍后再考虑吧。该如何开始呢？要做一个系统设计吗？

RSK：我不介意用UML图来说明从记分卡中得到的一些问题领域概念。从中会发现一些候选对象,可以在随后的编码时使用。

RCM:(戴上他那顶强大的对象设计者的帽子)好,显然，Game对象由一系列共10个Frame对象组成，每个Frame对象可以包含1个、2个或者3个Throw对象。

RSK: 好主意。这也正是我所想的。我立刻把它画出来（参见图2)。

图2　保龄球记分卡的UML 图

RSK：好，来选取一个要测试的类。从依赖关系链的尾部开始,依次往后如何？这样会容易测试一些。

RCM:当然可以。我们来创建Throw类的测试用例。

RSK：(开始键入代码)

```
//TestThrow.java--------------------------------
import junit.framework.*;
public class TestThrow extends TestCase
{
   public TestThrow(String name)
   {
      super(name);
   }
   //public void test????
}
```

RSK：Throw对象应该具有什么行为呢？

RCM:它保存着比赛者所击倒的木瓶数。

RSK：好，你只用了寥寥数语，可见它确实没做什么事情。也许我们应该重新审视一下,来关注具有实际行为的对象,而不是仅仅存储数据的对象。

RCM：嗯，你的意思是说实际上Throw这个类也许不必存在？

RSK:是的,如果不具有任何行为,能有多重要呢？我还不知道它是否应该存在。我只是觉得如果我们去关注那些不仅仅只有setter和getter方法的对象的话,会更有效率。但是如果你想控制的话……(将键盘推到RCM面前。)

RCM：好吧，我们上移至依赖链上的Frame类，看看是否能在编写该类的测试用例时,完成Throw类。(把键盘推回给RSK。)

RSK:(想知道RCM是在通过这种方式将我引入一个死胡同来教育我呢,还是他确实同意我的观点)好,新的文件，新的测试用例。

```
//TestFrame.java--------------------------------
import junit.framework.*;
public class TestFrame extends TestCase
{
   public TestFrame(String name)
   {
      super(name);
   }
   //public void test????
}
```

RCM：嗯，这是第二次键入这样的代码了。现在，你想到一些针对Frame类的有趣的测试用例吗？

RSK：Frame类可以给出它的分数，每次投掷击倒的木瓶数,以及是否为全中或者补中……

RCM：好，用代码来说明问题。

RSK：(键入)

```
//TestFrame.java--------------------------------
import junit.framework.*;
public class TestFrame extends TestCase
{
   public TestFrame(String name)
   {
      super(name);
   }
   public void testScoreNoThrows()
   {
```

```
    Frame f = new Frame();
    assertEquals(0, f.getScore() );
  }
}
//Frame.java---------------------------------
public class Frame
{
  public int getScore()
  {
    return 0;
  }
}
```

RCM：好，测试用例通过了，但是getScore实际上是一个愚蠢的方法。如果向Frame中加入一次投掷的话，它就会失败。所以我们来编写这样的测试用例，它会加入一些投掷，然后检查得分。

```
//TestFrame.java
public void testAddOneThrow()
{
  Frame f = new Frame();
  f.add(5);
  assertEquals(5, f.getScore() );
}
```

RCM：这不能编译通过。Frame类中没有add方法。

RSK：我打赌如果你定义这个方法，就会编译通过；-）

RCM：

```
//Frame.java---------------------------------
public class Frame
{
  public int getScore()
  {
    return 0;
  }
  public void add(Throw t)
  {
  }
}
```

RCM：（自言自语）这不可能编译通过的，因为还没有编写Throw类。

RSK：和我说说，Bob。在测试中传给add方法的是一个整数，而该方法期望一个Throw对象。add方法不能具有两种形式。在我们再次关注Throw类前，你能描会一下Throw类的行为吗？

RCM：哦，我甚至都没有注意到我写的是f.add(5)。我应该写f.add(new Throw(5))，但那太不优雅了。我真正想写的就是f.add(5)。

RSK：先不管是否优雅，我们暂时把美学的考虑放到一边。你能描绘一下Throw对象的行为吗？是二元表示吗，Bob？

RCM：10110101101010010l。我不知道Throw是否具有一些行为。我现在觉得Throw就是int。不过，我们不必再考虑它了，因为我们可以让Frame.add接受一个int。

RSK：我觉得这样做的根本原因就是简单。当出现问题时，可以再使用一些复杂的方法。

RCM：同意。

```
//Frame.java---------------------------------
public class Frame
{
  public int getScore()
  {
    return 0;
  }
  public void add(int pins)
  {
  }
}
```

RCM：好，编译通过而测试失败了。现在，我们来通过测试。

```
//Frame.java---------------------------------
public class Frame
{
  public int getScore()
  {
    return itsScore;
  }
  public void add(int pins)
  {
    itsScore += pins;
  }
  private int itsScore = 0;
}
```

RCM：编译和测试都通过了，这明显太简单了。下一个测试用例是什么？

RSK：先休息一会儿好吗？

---------- Break ----------

RCM：不错。但Frame.add是一个脆弱的方法。如果用11作为参数去调用它会怎样呢？

RSK：如果发生这种情况，可以抛出异常。但是谁会去调用它呢？这个程序会成为被数千人使用的应用框架以至于我们必须要对这种情况进行防护吗？还是仅仅被你一人使用呢？如果是后者，只要调用它时不传入11就没问题了。（暗笑。）

RCM：好主意，系统的其他测试会捕获无效的参数。如果我们遇到麻烦，再把这个检查加进来也不迟。目前，add函数还不能处理全中和补中的情况。我们编写一个测试用例来表现这种情况。

RSK:嗯……如果用调用add(10)来表示一个全中,那么getScore应该返回什么值呢?我不知道该如何写这个断言,也许我们提出的问题是错误的,或者我们选择提问的对象是错误的。

RCM:如果调用了add(10),或者调用了add(3)后又调用了add(7),那么随后调用Frame的getScore方法是没有意义的。此时当前Frame对象必须要根据随后几个Frame实例的得分才能计算自己的得分。如果后面的Frame实例还不存在,那么它会返回一些令人讨厌的值,像-1。我不希望返回-1。

RSK:是的,我也不喜欢返回-1这个想法。你刚刚引入了一个概念,就是Frame之间要互相知晓。谁会持有这些不同的Frame对象呢?

RCM:Game对象。

RSK:那么Game依赖于Frame,而Frame反过来又依赖于Game。我不喜欢这样。

RCM:Frame不必依赖于Game,可以把它们放置在一个链表中。每个Frame持有指向它前面以及后面Frame的指针。要获取一个Frame的得分,该Frame会获取前一个Frame的得分;如果该Frame中有补中或者全中的情况,它会从后面Frame中获取所需的得分。

RSK:好的,不过不太形象,我感觉有些不清楚。写一些代码看看吧。

RCM:好。我们首先要编写一个测试用例。

RSK:是针对Game呢,还是另一个针对Frame的呢?

RCM:我认为应该针对Game,因为是Game构建了Frame并把它们互相连接起来。

RSK:你是想停下我们正在做的有关Frame的工作,而跳转到Game上去呢?还是只想要一个MockGame对象来完成Frame正常运转所需要的工作呢?

RCM:我们停止在Frame上的工作,转到Game上来吧。Game的测试用例应当可以证明我们需要Frame链表。

RSK:我不知道它们是怎样证明的。我需要代码。

RCM:(键入代码)

```
//TestFrame.java-------------------------------
import junit.framework.*;
public class TestFrame extends TestCase
{
   public TestFrame(String name)
   {
      super(name);
   }
   public void testoneThrows()
   {
     Game g = new Game();
     g.add(5);
       assertEquals(5, g.score() );
   }
}
```

RCM:看上去合理吗?

RSK:当然合理,但是我仍然在寻找需要Frame链表的证据?

RCM:我也是,我们继续这些测试用例,看看会有什么结果。

```
//Game.java-----------------------------------------
public class Game
{
   public int score()
   {
     return 0;
   }
   public void add(int pins)
   {
   }
}
```

RCM:好,编译通过,而测试失败了。现在我们来让测试通过。

```
//Game.java-----------------------------------------
public class Game
{
   public int score()
   {
      return itsScore;
   }
   public void add(int pins)
   {
     itsScore += pins;
   }
   private int itsScore = 0;
}
```

RCM:测试通过了,很好。

RSK:是不错,但是我仍在寻找需要Frame对象链表的重要证据。最初就是它致使我们认为需要Game。

RCM:是的,这也是我正在寻找的。我肯定一旦加进了有关补中和全中的测试用例,就必须得构建Frame,并把它们用链表链接在一起。但是在代码迫使这样做前,我不想这样做。

RSK:好主意。我们来继续逐步完成Game。编写另外一个关于有两次投掷但没有补中的情况的测试怎么样?

RCM:好,这应该会立刻通过测试。我们来试试。

```
//TestGame.java-------------------------------------
public void testTwoThrowsNoMark()
```

```
{
   Game g = new Game();
   g.add(5);
   g.add(4);
   assertEquals(9, g.score());
}
```

RCM:是的,这个测试通过了。现在我们来试一下有4次投掷但没有补中和全中的情况。

RSK:嗯,这个测试也能通过。但这不是我所期望的,我们可以一直增加投掷数,甚至根本不需要一个Frame。但是我们还不曾考虑补中或者全中的情况。也许到那时我们就会需要一个Frame。

RCM:这也是我正在考虑的。不管怎样,考虑一下这个测试用例:

```
//TestGame.java-----------------------------------------
public void testFourThrowsNoMark()
{
   Game g = new Game();
   g.add(5);
   g.add(4);
   g.add(7);
   g.add(2);
   assertEquals(18, g.score());
   assertEquals(9, g.scoreForFrame(1));
   assertEquals(18, g.scoreForFrame(2));
}
```

RCM:这看上去合理吗?

RSK:当然合理。我忘了必须要能显示每轮的得分。啊,我把咱们画的记分卡草图当可乐杯垫来用了。这是我忘记的原因。

RCM:(叹气)好,首先我们给Game加入scoreForFrame方法使测试失败。

```
//Game.java---------------------------------------------
   public int scoreForFrame(int frame)
   {
       return 0;
   }
```

RCM:好极了,编译通过,测试失败了,现在,怎样通过测试呢?

RSK:我们可以定义Frame对象了,但这是通过测试的最简单方法吗?

RCM:不是,事实上,我们只需要在Game中创建一个整数数组。每次对add的调用都会在这个数组里添加一个新的整数。每次对scoreForFrame的调用只需要前向遍历这个数组并计算出得分。

```
//Game.java-----------------------------------------
public class Game
{
    public int score()
    {
       return itsScore;
    }
    public void add(int pins)
    {
        itsThrows[itsCurrentThrow++] = pins;
        itsScore += pins;
    }
    public int scoreForFrame(int frame)
    {
        int score = 0;
        for (int ball = 0;
          frame > 0 && (ball < itsCurrentThrow);
          ball += 2, frame--)
        {
          score += itsThrows[ball] + itsThrows[ball+1];
        }
        return score;
    }
    private int itsScore = 0;
    private int[] itsThrows = new int[21];
    private int itsTurrentThrow = 0;
}
```

RCM:(对自己很满意)看,可以工作了。

RSK:为什么要用21这个魔数(magic number)呢?

RCM:它表示一场比赛中最大可能的投掷数。

RSK:讨厌。让我猜猜,你年轻时,是一个Unix hacker,并自豪于把整个应用程序编写在没人理解的一条语句中。需要重构scoreForFrame,这样可以更好的理解它。但是在考虑重构前,我来问另外一个问题。Game是放置这个方法的最好地方吗?我认为Game违反了单一职责原则(Single Responsibility Principle,简称SRP)。它接收投掷并且知道如何计算每轮的得分。你觉得增加一个Scorer对象如何?

RCM:(粗鲁地摆了一下手)目前我还不知道这个函数该放在哪里,现在我感兴趣的只是让记分程序工作起来。完成所需的功能后,我们再来讨论SRP的价值。不过,我明白你所说的Unix hacker指的是什么。我们来简化这个循环。

```
public int scoreForFrame(int theFrame)
{
   int ball = 0;
   int score = 0;
   for (int currentFrame = 0;
         currentFrame < theFrame;
         currentFrame++)
       {
         score += itsThrows[ball++] + itsThrows[ball++];
```

```
        }
        return score;
}
```

RCM:好了一点,但是score+= 这个表达式具有副作用。不过,这个表达式中两个加数表达式的求值顺序无关紧要,所以这里不会造成副作用。(是这样吗?两个增量操作会不会在任意一个数组运算前完成呢?)

RSK:我认为可以做个实验来证明这里不会有任何副作用,但是这个函数还不能处理补中和全中的情况。我们是应该继续使它更易读些呢?还是应该给它添加更多的功能呢?

RCM:实验只对特定的编译器有意义。其他的编译器可能采用不同的求值顺序。我不知道这是不是一个问题,但是我们还是先来去除这种可能的顺序依赖,然后再编写更多的测试用例来添加功能。

```
public int scoreForFrame(int theFrame)
{
   int ball = 0;
   int score = 0;
   for (int currentFrame = 0;
         currentFrame < theFrame;
         currentFrame++)
   {
      int firstThrow = itsThrows[ball++];
      int secondThrow = itsThrows[ball++];
      score = firstThrow + secondThrow;
   }
      return score;
}
```

RSK:好,下个测试用例。我们来试试补中的情况。

```
public void testSimpleSpare()
{
   Game g = new Game();
}
```

RCM:我已经厌倦总是写这个了,我们来重构一下测试,把Game对象的创建放到setUp函数中吧。

```
//TestGame.java--------------------------------
import junit.framework.*;
public class TestGame extends TestCase
{
    public TestGame(String name)
   {
      super(name);
   }
   private Game g;
   public void setUp()
  {
     g = new Game();
  }
   public void testOneThrows()
   {
     g.add(5);
   assertEquals(5, g.score() );
  }
   public void testTwoThrowsNoMark()
   {
     g.add(5);
     g.add(4);
     assertEquals(9, g.score());
  }
   public void testFourThrowsNoMark()
   {
     g.add(5);
     g.add(4);
     g.add(7);
     g.add(2);
     assertEquals(18, g.score());
     assertEquals(9, g.scoreForFrame(1));
     assertEquals(18, g.scoreForFrame(2));
  }
   public void testSimpleSpare()
   {
   }
}
```

RCM:好多了,现在来编写关于补中的测试用例。

```
public void testSimpleSpare()
  {
       g.add(3);
       g.add(7);
       g.add(3);
       assertEquals(13, g.scoreForFrame(1));
 }
```

RCM:好,测试失败了,现在我们要让它通过。

RSK:我来写吧。

```
public int scoreForFrame(int theFrame)
{
   int ball = 0;
   int score = 0;
   for (int currentFrame = 0;
         currentFrame < theFrame;
         currentFrame++)
  {
   int firstThrow = itsThrows[ball++];
   int secondThrow = itsThrows[ball++];
    int frameScore = firstThrow + secondThrow;
    // spare needs next frames first throw
    if (frameScore ==10 )
        score += frameScore + itsThrows[ball++];
    else
        score += frameScore;
    }
    return score;
}
```

编者后记:

由于篇幅所限,本刊无法把这篇精彩的设计对话完整刊载。意犹未尽的读者不妨去领略一下原书的风采。

用C# 实现弹出广告谋杀器
—— Popup Killer

▶撰文 / Stephane Rodriguez.

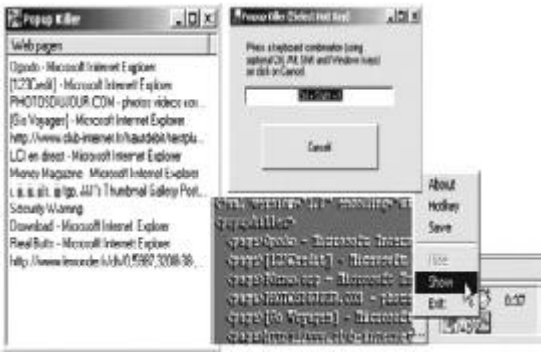
编者：

您在上网的时候，一定被无穷无尽的弹出广告所困扰过，它们就像一只苍蝇，不对，是一千只苍蝇在你眼前嗡嗡.....。好了，打开你的开发工具，手起刀落，从此整个世界清静了！

确切地说，这不是一个新主题，因为在Internet上已有许多这种程序。但是我还是想就这个机会，向大家展示用C#很容易地完成这种事情。

Popup Killer是自动关闭禁止窗口的程序。Popup Killer工作在系统托盘（system tray）上，基于用户设定的规则检查浏览器窗口，更新XML文件中描述的禁止窗口，并能够借助快捷键提供了易于使用性。

已经有一个用C++开发的类似程序，该程序随一篇文章（参见网址：http://www.codeproject.com/dialog/killpopups.asp）发表在 Code Project 上。但是本文中展示的程序将用C#完成，并且能够更快更好地实现禁止窗口查找。



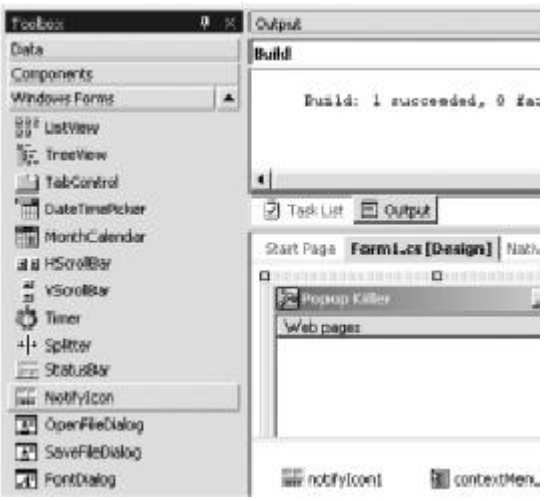
自动关闭禁止窗口

开发步骤：

- 建立系统托盘应用程序
- 应用程序开关
- 添加上下文菜单
- 实现计时器
- 本地Win32窗口查找
- 注册 Windows热键

建立系统托盘应用程序

开始新的Windows C# Application，并从Toolbox窗口拖放NotifyIcon组件，如下图所示：



向Windows C# 应用程序加入一个系统托盘

要确保系统托盘图标和Application.exe图标相匹配，需要选择一个.ico文件，并设置NotifyIcon的Icon属性。然后，编辑项目中的general属性，然后选择该图标作为 Application Icon的值。

如果在任务栏中还能看见这个，系统托盘应用程序就有一点怪。要避免这一点，只需设置ShowInTaskbar窗体属性为false。这可以通过直接在Form Properties窗口中更改来完成。也可以通过如下程序来完成：

```
this.ShowInTaskbar = false;
```

系统托盘显示基本完成了,C#太容易了!当然,还没有上下文菜单,并且还不能使应用程序可见和隐藏。

应用程序开关

首先,主Form必须根据其状态要么显示,要么隐藏。除此之外,通过调整Form的WindowState属性,我们可以最小化Form,或者让它返回到正常状态:

```
public void HideApp()
{
    this.WindowState = FormWindowState.Minimized;
    Hide();
}
public void ShowApp()
{
    Show();
    this.WindowState = FormWindowState.Normal;
}
```

一个需要完成的有趣特性是,当窗体本身不存在时,可以让用户关闭Form。为了完成这一特性,必须重载Form类实现的OnClosing事件:

```
protected override void OnClosing(CancelEventArgs e)
{
    //方法重载,以使窗体能够最小化,而不是关闭
    e.Cancel = true;
    //最小化窗体,并隐藏它
    this.WindowState = FormWindowState.Minimized;
    Hide();
}
```

当然,必须提供显式的可选项来退出应用程序。这可以通过系统托盘上下文菜单的Exit选项来完成,其单击事件如下:

```
private void menu_App_Exit(object sender, System.EventArgs e)
{
   NativeWIN32.UnregisterHotKey(Handle, 100);
    //从系统托盘隐藏图标
    notifyIcon1.Visible = false;
    Application.Exit();
}
```

添加上下文菜单

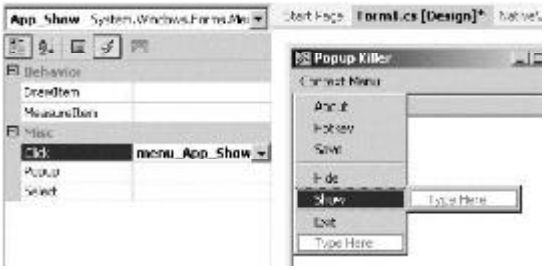
加入上下文菜单与加入系统托盘同样简单,在Toolbox窗口中有一个ContextMenu组件。通过将其与系统托盘的ContextMenu属性相关联,上下文菜单在单击右键时会自动弹出。

这里有一件有趣的事件需要提一下,除了采用这种关联外,没有其他方法可以用来弹出上下文菜单,虽然System.Windows.Forms.ContextMenu类提供了与此类似的一种方法:

```
[redacted]
```

但这种方法不能使用,因为在特定的实例中,唯一可用的控件对象是应用程序Form,在Form隐藏时ContextMenu的Show()方法失败。

将菜单项加入到菜单中很直观。单击上下文菜单,就会显示实际的上下文菜单实例,就可以加入菜单项或者是分隔符了。双击每一菜单项,可以添加单击事件处理程序。



在.Net环境中建立上下文菜单

一旦上下文菜单设置完毕,选项菜单必须根据应用程序状态来判断是启用或者是禁用的。出于此目的,上下文菜单总是触发与其相关联的BeforePopup事件,从而允许我们设置每一菜单项的Enabled属性,或者即使是从头开始重建上下文菜单(通过代码):

```
private void menu_App_BeforePopup(object sender,
     System.EventArgs e)
{
  if ( this.WindowState == FormWindowState.Minimized )
  {
    //App_Show是一个System.Windows.Forms.MenuItem对象
     App_Show.Enabled = true;
    //App_Hide是一个System.Windows.Forms.MenuItem对象
       App_Hide.Enabled = false;
  }
   else
  {
       App_Show.Enabled = false;
       App_Hide.Enabled = true;
  }
}
```

实现计时器

.Net框架Timer与标准的Win32计时器非常雷同(它甚至不需要id)。实现时也不需要线程,这确实是非常好的。需要我们做的所有事情就是声明一个计时器,

赋予正确的设置,并为其关联一个回调程序。

```
//显式命名空间 (Timer也在System.Threading中)
m_Timer = new System.Timers.Timer();
m_Timer.Elapsed += new ElapsedEventHandler
      (OnTimerKillPopup);
//比如3000毫秒
m_Timer.Interval = m_nInterval;
m_Timer.Enabled = true; //启动计时器
protected void OnTimerKillPopup(Object source,
          ElapsedEventArgs e)
{
   m_Timer.Enabled = false; //暂停计时器
   FindPopupToKill();
   m_Timer.Enabled = true;
}
```

本地 Win32 窗口查找

应用程序完成的是检查所有打开的Internet Explorer窗口,获取其标题,并将标题与禁止名称的已知列表进行比较。如果匹配成功,就自动地关闭Internet Explorer窗口,这就好象我们使用手工右键单击窗口的边框,并从系统菜单中选择“Close”选项。

比较本文开始提及的C++实现方法,不得不说我们更喜欢这种智能程序,并避免每3秒枚举所有打开窗口的开销。为什么会这样呢?我已经进行了性能测试,当时系统只有8个应用程序正在运行并显示在任务栏,在多于180个窗口时本程序确实是可用的。在这180个窗口中,只有2个是顶层的Internet Explorer窗口,而这两个窗口中只有一个是被禁止的。

此时,我们在KillPopup()实现中,叫做每n秒一次。我们需要检索所有Internet Explorer窗口标题。困扰我们的是不能使用单一的.NET框架函数来很容易地完成这项功能。实际中我们可以使用System.Diagnostics.Process方法搜索出所有的iexplore.exe进程,然后从中获得主窗口句柄,但这还不能解决问题。每个Internet Explorer进程可能打开了许多窗口,实际上是每个线程打开一个窗口。但没有办法获得附加到每个正在运行线程的窗口。

第一种有效的实现方式是,我们使用System.Diagnostics.Process来列出正在运行的进程,然后从其Threads属性中获得System.Diagnostics.ProcessThreadCollection,从而获得线程id。接着使用一个本地WIN32 API调用,即EnumThreadWindows(DWORD threadId, WNDENUMPROC lpfn, LPARAM lParam),并进行一次回调,以枚举出所有来自该特定线程的窗口。一旦获得窗口句柄,注意这里使用了IntPtr,因为在C#中没有HWND对象,再次调用一个本地Win32 API方法,GetWindowText(HWND hwnd,/*out*/LPTSTR lpString, int nMaxCount),从而获得窗口标题。对已知的禁止窗口,再次使用一个本地Win32 API方法调用,选择发送关闭方法,SendMessage(HWND hWnd, int msg, int wParam, int lParam)。代码类似如下(这里只是示例,实际中可以灵活实现):

```
Process[] myProcesses = Process.GetProcessesByName("IEXPLORE");
  foreach(Process myProcess in myProcesses)
{
   FindPopupToKill(myProcess);
}
protected void FindPopupToKill(Process p)
{
    //遍历所有线程,并枚举所有与该线程相关的窗口
    foreach (ProcessThread t in p.Threads)
    {
       int threadId = t.Id;
       NativeWIN32.EnumThreadProc callbackProc =
       new NativeWIN32.EnumThreadProc
         (MyEnumThreadWindowsProc);
       NativeWIN32.EnumThreadWindows(threadId,
         callbackProc, IntPtr.Zero /*lParam*/);
    }
}
//回调,用于枚举与其中一个线程相关的窗口
bool MyEnumThreadWindowsProc(IntPtr hwnd, IntPtr lParam)
{
    public const int WM_SYSCOMMAND = 0x0112;
    public const int SC_CLOSE = 0xF060;

    //获取窗口标题
    NativeWIN32.STRINGBUFFER sLimitedLengthWindowTitle;
    NativeWIN32.GetWindowText(hwnd, out
     sLimitedLengthWindowTitle, 256);
    String sWindowTitle = sLimitedLengthWindowTitle.szText;
    if (sWindowTitle.Length==0) return true;
    //查找禁止标题列表
    foreach (ListViewItem item in listView1.Items)
    {
      if ( sWindowTitle.StartsWith(item.Text) )
       NativeWIN32.SendMessage(hwnd,
                               NativeWIN32.WM_SYSCOMMAND,
                               NativeWIN32.SC_CLOSE,
                               IntPtr.Zero); // try soft kill
    }
    return true;
}
public class NativeWIN32
{
 public delegate bool EnumThreadProc(IntPtr hwnd, IntPtr lParam);
   [DllImport("user32.dll", CharSet=CharSet.Auto)]
 public static extern bool EnumThreadWindows(int threadId,
       EnumThreadProc pfnEnum, IntPtr lParam);
//用于方法调用的输出LPCTSTR参数
```

```
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Auto)]
   public struct STRINGBUFFER
   {
     [MarshalAs(UnmanagedType.ByValTStr, SizeConst=256)]
       public string szText;
    }
   [DllImport("user32.dll", CharSet=CharSet.Auto)]
   public static extern int GetWindowText(IntPtr hWnd,
       out STRINGBUFFER ClassName, int nMaxCount);
   [DllImport("user32.dll", CharSet=CharSet.Auto)]
   public static extern int SendMessage(IntPtr hWnd,
       int msg, int wParam, int lParam);
}
```

就性能而言,本代码漂亮地完成了工作,因为它自动从所有窗口中过滤掉那些来自Internet Explorer进程的禁止窗口,这非常类似于过滤掉显示在任务栏中的正在运行的应用程序。

但是有一个甚至更简单的实现方法。我们使用本地WIN32 FindWindowEx(HWND hWndParent, HWND hWndNext, /*in*/LPCTSTR szClassName, /*in*/LPCTSTR szWindowTitle)方法,为什么要使用它呢?因为我们可以多次调用它来获得匹配registered window class name标准(所有由Internet Explorer打开窗口的IEFrame)的所有窗口:

```
protected void FindPopupToKill()
{
   IntPtr hParent = IntPtr.Zero;
   IntPtr hNext = IntPtr.Zero;
   String sClassNameFilter = "IEFrame";
    //所有IE窗口的CLASSNAME
   do
    {
      hNext = NativeWIN32.FindWindowEx(hParent,hNext,
       sClassNameFilter,IntPtr.Zero);
       //获得hwnd
      if ( !hNext.Equals(IntPtr.Zero) )
      {
       //获得窗口标题
       NativeWIN32.STRINGBUFFER sLimitedLengthWindowTitle;
       NativeWIN32.GetWindowText(hNext,
                  outsLimitedLengthWindowTitle, 256);
       String sWindowTitle = sLimitedLengthWindowTitle.szText;
        if (sWindowTitle.Length>0)
        {
         //找到禁止标题列表中的这个标题
          foreach (ListViewItem item in listView1.Items)
          {
            if ( sWindowTitle.StartsWith(item.Text) )
                 NativeWIN32.SendMessage(hNext,
                 NativeWIN32.WM_SYSCOMMAND,
                 NativeWIN32.SC_CLOSE,
                 IntPtr.Zero); //尝试软件关闭窗口
          }
        }
      }
    }
   while (!hNext.Equals(IntPtr.Zero));
}
public class NativeWIN32
{
  [DllImport("user32.dll", CharSet=CharSet.Auto)]
  public static extern IntPtr FindWindowEx
   (IntPtr parent /*HWND*/,IntPtr next /*HWND*/,
   string sClassName,IntPtr sWindowTitle);
}
```

注册Windows快捷键

Windows快捷键简化了应用程序(比如Popup Killer)的操作。简单的注册热键组合(默认情况下是Ctrl+Shift+J)允许Popup Killer来指出激活的窗口,并将其加入到禁止窗口列表。这避免了手工添加和编辑窗口名称(也可以通过这种方式在Form中实现,比如ListView上下文菜单)。

注册热键可以随着系统托盘上下文菜单动态改变。注册热键还可以与禁止窗口名列表一起保存在XML文件中。

现在来关注实现,我们需要再次使用本地Win32 API调用,即RegisterHotkey(HWND hWnd, int id, UINT fsModifiers, UINT vkey)。代码类似如下:

```
public void SetHotKey(Keys c, bool bCtrl, bool bShift,
  bool bAlt, bool bWindows)
{
    m_hotkey = c;
    m_ctrlhotkey = bCtrl;
    m_shifthotkey = bShift;
    m_althotkey = bAlt;
    m_winhotkey = bWindows;
     //更新热键
    NativeWIN32.KeyModifiers modifiers =
     NativeWIN32.KeyModifiers.None;
 if (m_ctrlhotkey)
        modifiers |= NativeWIN32.KeyModifiers.Control;
    if (m_shifthotkey)
        modifiers |= NativeWIN32.KeyModifiers.Shift;
    if (m_althotkey)
        modifiers |= NativeWIN32.KeyModifiers.Alt;
    if (m_winhotkey)
        modifiers |= NativeWIN32.KeyModifiers.Windows;
//Keys.J);
    NativeWIN32.RegisterHotKey(Handle,100,
                                modifiers, m_hotkey);
}
```

在应用程序中使用热键需要几个步骤,列表如下:

```
/* -------在C#应用程序中使用HOTKEY-------
  --James J Thompson编写的代码片段--
窗体装载: Ctrl+Shift+J
   bool success = RegisterHotKey(Handle,100,
```

```
                                KeyModifiers.Control |
                                KeyModifiers.Shift,Keys.J);
窗体关闭：
    UnregisterHotKey(Handle, 100);
只需按键即可处理一个热健:
protected override void WndProc( ref Message m )
{
    const int WM_HOTKEY = 0x0312;
        switch(m.Msg)
    {
         case WM_HOTKEY:

            MessageBox.Show("Hotkey pressed");
            ProcessHotkey();
             break;
    }
    base.WndProc(ref m );
}
public class NativeWIN32
{
    [DllImport("user32.dll", SetLastError=true)]
    public static extern bool RegisterHotKey(
          IntPtr hWnd, //窗口句柄
          int id, //热键标识符
          KeyModifiers fsModifiers,// key-modifier选项
          Keys vk     //virtual-key代码
    );
    [DllImport("user32.dll", SetLastError=true)]
    public static extern bool UnregisterHotKey(
                    IntPtr hWnd, //窗口句柄
                    int id       //热键标识符
    );
    [Flags()]
    public enum KeyModifiers
    {
        None = 0,
        Alt = 1,
        Control = 2,
        Shift = 4,
        Windows = 8
    }
}
-------在C#应用程序中使用HOTKEY-------*/
```

按下热键时，工作流程如下：获得激活的窗口，这由本地 Win32 API HWND GetForegroundWindow() 方法调用完成。接下来唯一需要做的事情就是检索其标题，这由对本地方法 Win32 API GetWindowText (HWND hwnd,/*out*/LPTSTR lpString,int nMaxCount)调用来完成。

```
protected void ProcessHotkey()
{
    IntPtr hwnd = NativeWIN32.GetForegroundWindow();
    if (!hwnd.Equals(IntPtr.Zero))
    {
      NativeWIN32.STRINGBUFFER sWindowTitle;
      NativeWIN32.GetWindowText(hwnd, out sWindowTitle, 256);
       if (sWindowTitle.szText.Length>0)
            //加入到ListView(Form)
          AddWindowTitle( sWindowTitle.szText );
    }
}
```

注：

本文的英文原文地址，即源码下载目录为：http://www.codeproject.com/csharp/popupkiller.asp?print=true,翻译整理时有所调整。读者只需简单注册一个帐号便可下载。

about author:

Addicted to reverse engineering. At work, I am developing business intelligence software in a team of smart people (independant software vendor). I am currently working on a personal project code-named "LongSleeves". Drop in an email if you think you know what's all the fuss about.

小知识

认识一下托盘程序

Windows状态栏也称系统托盘，位于任务栏的右侧。托盘程序是指这样一类程序：当程序运行后，会在系统的托盘区（也有说是状态区域）创建此程序的图标，使用者可以通过点击图标出现的菜单来控制程序的运行状态。

在Windows 9x中已有系统时钟、音量控制、计划任务、输入法的图标；一些应用程序在安装完后也将它们的图标放入了状态栏中，如超级解霸、RealPlayer、金山毒霸实时监控程序等；此外运行一些程序后也会在此出现它们的图标，如拨号连接、网络蚂蚁、QQ、Foxmail……

托盘程序有很多优点，如不占屏幕，后台运行，便于控制等。所以现在越来越多的程序都做成了托盘程序。

.Net框架中为所有.Net平台开发语言提供了公用类库——.Net FrameWork SDK。在这个类库中，为编写托盘程序提供了具体的类，调用这些类就可以实现程序的托盘效果了。

一次惊心动魄的调试
——都是“单线程”惹的祸

▶撰文 / 周松奕

编者按：

马上就要“交货”，而程序联合测试突然出现意想不到的重大问题，相信很多程序员们对这种情形并不陌生。难得作者写得如此紧张生动，让读者的心也一起和他提到了嗓子眼。不仅如此，还让我们明白了多线程技术使用的一个重要环境，而这一点才是最关键的。

引子

记得那是一个阳光明媚的早晨，我的心情就像天气一样晴空万里，这是因为我第一次负责开发的“即时通信系统”服务端系统已经完工，紧张的项目工作终于接近尾声，今天就是内部系统测试的日子，如果顺利通过，就可以交付客户验收了。

风平浪静

8点一过，我就已经到达公司，然后用最短的时间再次review了一下所有程序，确认没有任何语法和逻辑上的错误。我把它Checkin进公司的版本服务器，等候调试人员调试。提交了我的处女作，一种成就感油然而生，充满了信心。

10点，联调开始。这是个让人紧张的时候。首先是系统联编，没有出现任何问题，顺利通过了。接下来测试人员开始按照测试计划一步一步地进行测试。一开始，我在边上看得胆颤心惊，生怕出现什么问题。不过随着每个功能项的顺利测试通过，服务端也依然很健壮地跑着，项目经理的脸上露出了难得的微笑，我心情慢慢放松了，对自己的程序充满着前所未有的自信。

10点30分，开始进行异常处理调试。我也带着轻松的心情，溜到一旁休息室喝咖啡去了。因为，调试进入这个阶段，就是基本功能已经都实现了，我觉得没有什么必要在旁边看着了。我在咖啡室中很悠闲的边喝着咖啡，边听着音乐，回忆着项目开发中的日日夜夜，重新拾掇着自己的喜怒哀乐。

石破天惊

就在我沉浸在回忆中时，一个声音打断了我的思绪，调试人员小李跑进来了，神色很严重地说：“快来，你的服务端有问题了！”

我抬头看了看他，说：“不可能，我绝对相信我的程序！你别开玩笑了！可能是你们出了什么问题吧？”

小李气急败坏地说：“谁和你开玩笑啊！真的有问题了！我们是屡试屡败，次次出现相同的问题，快去看看！”

我这才意识到事情的严重性，跳了起来，赶快跑到调试现场。

“怎么回事？什么问题？”我大声地问道。

调试组长小陈用他那很有特点的沙哑的声音说：“我们测试到异常处理第三条，也就是：客户端在通信中突然中断，非正常退出，而后重连。当我们让客户端在通信中非正常退出，然后重新连入服务端，服务端没有响应，客户端连不上。可是过了一段时间，大概一分钟左右，重新连接，又可以了。”

“那么在这一分钟内，没有任何客户端能够连得上？用别的用户ID也连不上？”我问。

“是的，都连不上。”小陈说。

“这是个很大的问题，因为实际使用过程中，用户的通信条件有可能很不好，那么会出现一个用户出现非正常退出，就让服务端失去响应别的用户的能力。这个必须改！”项目经理发话了，“我建议调试中断，程序发回重新修改，修改后重新联调，今天务必改完调完，保证明天提交给客户验收。”

完了，完了，这个项目不会因为我而推迟交付吧，我心里开始打鼓了。

重返威武山

我从版本服务器再次checkout出我的程序,重新审查我的代码,连续过了两遍,没有发现任何流程上的问题啊,可是为什么会出现异常呢?相关模块的程序代码如下:(经过部分的删节)

```
DWORD WINAPI HandleConnect(LPVOID lpParam)
{
    SOCKET            sListen, sClient;
    struct sockaddr_in local, client;
    CWinThread*       hThread;
    char              szmsg[200];
    char              szBuff[2048];
    sListen = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
    if (sListen == SOCKET_ERROR)
    {
       sprintf(szmsg,"socket() failed! Error:%i",
                WSAGetLastError());
       cout << szmsg << endl;
       return 1;
    }
    local.sin_addr.s_addr = htonl(INADDR_ANY);
    local.sin_family = AF_INET;
    local.sin_port = htons(8888);

   if (bind(sListen, (struct sockaddr *)&local,
     sizeof(local)) == SOCKET_ERROR)
    {
       sprintf(szmsg,"bind() failed! Error:%i",
                WSAGetLastError());
       cout << szmsg << endl;
       return 1;
    }
   listen(sListen, SOMAXCONN);
   while (true)
    {
     int iAddrSize = sizeof(client);
     sClient = accept(sListen,(struct sockaddr *)
                      &client, &iAddrSize);

     if (sClient == INVALID_SOCKET)
      {
       Sleep(10);
       continue;
      }
      sprintf(szmsg,"Accepted client: %s:%d\n",
        inet_ntoa(client.sin_addr), ntohs(client.sin_port));
      cout << szmsg << endl;
      int ims=60000;
      setsockopt(sClient, SOL_SOCKET, SO_RCVTIMEO,
       (char*)& ims, sizeof(ims) );//接收客户端数据
      ret = recv(sClient, szBuff, 2048, 0);
      if (ret == 0)        // Graceful close
      {
        cout << "client close!" << endl;
        return -1;
      }
      else if (ret == SOCKET_ERROR)
      {
        sprintf(szmsg,"recv() failed: %d\n",
                WSAGetLastError());
        cout << szmsg << endl;
        return -1;
       }
       szBuff[ret] = '\0';
       sprintf(szmsg,"Receive from Client: %i bytes",ret);

       cout << szmsg << endl;
       DealWithMsg(szBuff);
       closesocket(sClient);
     }
    closesocket(sListen);
    WSACleanup();
    cout << "Exit Handle Connect!"<< endl ;
}
```

我仔细地回想着项目经理和我描述的出错现象,突然一个词语跳入了我的脑海中:一分钟!为什么是一分钟?而且为什么每次都是一分钟?在我的程序中涉及到一分钟的只有设置接收超时的地方。难道接收数据过程中客户端非正常中断数据发送,而服务端还在那里等候,导致无法再次接受别的客户端的接入?为了证明我的这个想法的正确,我另外写了个测试服务端和客户端。

11点,测试服务端和客户端完成,我开始测试。当在通信过程中,我强行使客户端非正常退出,然后马上重新连接,连不上了!过了一分钟,我再次连接,成功!然后我又将超时时间设置为2分钟,就发现异常断开后,需要过2分钟才能够再次连接成功,看来这就是问题的症结所在。

既然已经发现了问题,那么我如何解决它呢?不可能把服务端的连接接收超时调整到0,那会引起在网络速度较慢的情况下,服务端来不及接收客户端的任何数据,就把socket给关掉了。可是如果把连接接收超时调到任何数值都会引起有那么一段时间服务端无法接受任何连接。问题依然存在,怎么办??!!

云里雾里

12点,时间在一秒一秒的流逝,中午饭时间到了,可是我还一筹莫展。我开始着急。难道就因为我的原因,明天项目不能按时提交给客户?我可不希望第一次参与的项目就成为拖延进度的家伙。

“周,吃饭了!”旁边的同事叫我。

“噢”,我漫不经心地回应,可是这时的我哪有心情

吃饭啊！

12点10分，正当我对着电脑屏幕上的程序冥思苦想、发愣的时候、一个MSN的消息窗口跳了出来，“还没有去吃饭啊？”我的网友在问我。

“别理我，烦着呢！”我随手敲了一句话回了过去。

“怎么了，遇到什么问题了？”网友很关心地问我。

“没什么，我在思考问题，你别理我！”

“不会吧，那么酷，那你边思考问题，边和我聊天吧！”

“我思考问题啊，怎么和你聊啊，别理我了！！”

“你不是经常说你的脑袋好用吗？可以并行处理，哈哈，怎么现在短路了？只能单项处理了？看来你的CPU也不过尔尔哟”

“你！！！！！！！别理我！！！！”，我恼怒地回了一句很不客气的话，就隐身了。

一语惊醒梦中人

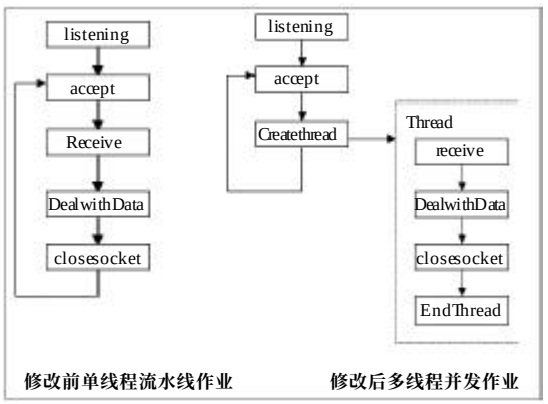
他无意间的话，却使我突然灵光一闪，并行处理！！！！对啊，为什么不用多线程呢？我的程序是单线程流水线式的处理。完全可以采用多线程啊！多线程处理可以在处理一个连接的同时，等待另外一个连接的接入。对，就是它了，多线程！我怎么就没有想到呢？我很懊恼地拍了一下自己的脑袋。

“太好了，我知道了，你帮了我大忙了！！！！”我很兴奋地给我的网友发了一条消息。

“噢？怎么了？我帮了你什么忙？？”网友奇怪的问。

“回头说！”

我顾不上和他说那么多，我已经兴奋得手都有点抖了。我重新看了一遍我的通信模块，没错，就是单线程流水线处理，完全可以改成多线程并发处理。多线程并发处理即每接入一个客户端，就建立一个线程来处理和这个客户端的通信。这样做可以保证同时接受多个客户端的接入，并发处理多个客户端的数据。如果一个客户端出现通信错误，不会影响到系统和别的客户端通信和数据处理。我在纸上大致画了一下修改前和修改后的原理如上图所示。



我迫不及待的开始重构测试服务端和客户端代码。

12点27分，测试服务端和客户端代码重构完成。开始测试，当我在客户端连上服务端进行通信过程中，使客户端非正常退出，然后重连，连接成功。连续作了几次，都是成功的，太棒了，解决方案可行。

12点35分，我开始重构即时通信系统的服务端代码。重构后的代码如下：

```
DWORD WINAPI HandleConnect(LPVOID lpParam)
{
  SOCKET              sListen, sClient;
  struct sockaddr_in  local, client;
  CWinThread*         hThread;
  char                szmsg[200];
  char                szBuff[2048];
  sListen = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
  if (sListen == SOCKET_ERROR)
    {
      sprintf(szmsg,"socket() failed! Error:%i",
              WSAGetLastError());
             cout << szmsg << endl;
      return 1;
    }
  local.sin_addr.s_addr = htonl(INADDR_ANY);
  local.sin_family = AF_INET;
  local.sin_port = htons(8888);
  if (bind(sListen, (struct sockaddr *)&local,
         sizeof(local)) == SOCKET_ERROR)
   {
      sprintf(szmsg,"bind() failed! Error:%i",
              WSAGetLastError());
             cout << szmsg << endl;
      return 1;
   }

  listen(sListen, SOMAXCONN);
     while (true)
     {
       int iAddrSize = sizeof(client);
       sClient = accept(sListen, (struct sockaddr *)
        &client,&iAddrSize);
       if (sClient == INVALID_SOCKET)
        {
         Sleep(10);
         continue;
        }
       sprintf(szmsg,"Accepted client: %s:%d\n",
         inet_ntoa(client.sin_addr),
         ntohs(client.sin_port));
       cout << szmsg << endl;
       hThread = AfxBeginThread(HandleLink, sClient,
```

```
            THREAD_PRIORITY_NORMAL, 0, 0, NULL );
        if (hThread == NULL)
        {
          cout << "Error Create HandleConnect thread!" << endl;
          Sleep(10);
          continue;
        }
         CloseHandle(hThread);
     }
    closesocket(sListen);
    WSACleanup();
     cout << "Exit Handle Connect!"<< endl ;
}

UINT HandleLink(LPVOID lpParam)
{
   Socket sClient=(Socket)lpParam;
    CWinThread * hThread;
    char szBuff[2048];
    int iret;
    char szmsg[200];
    struRcvMsg * srmsg=(struRcvMsg*)lpParam;
    int ims=60000;
    setsockopt(sClient, SOL_SOCKET, SO_RCVTIMEO,
               (char*)& ims, sizeof(ims) );
              // Perform a blocking recv() call
            //接收客户端数据
    iret = recv(sClient, szBuff, 2048, 0);
    if (iret == 0)        // Graceful close
    {
      cout << "client close!" << endl;
      return -1;
    }
    else if (iret == SOCKET_ERROR)
    {
      sprintf(szmsg,"recv() failed: %d\n",
              WSAGetLastError());
              cout << szmsg << endl;
      return -1;
    }
    szBuff[iret] = '\0';
    sprintf(szmsg,"Receive from Client: %i bytes",ret);
    cout << szmsg << endl;
   DealWithMsg(szBuff);
   closesocket(sClient);
}
```

12点45分,代码重构完成。编译通过,我重新review了一下,没有任何问题,Checkin到版本控制服务器上。等候调试小组的联调。自己也彻底地松了一口气,这时才感觉到自己的胃早已在闹革命了。

13点,测试小组开始工作。这次我没敢再溜走,心情紧张的在旁边看着他们的每一项联调工作。

15点30分,当最后一项调试报告通过的时候,我的心终于放下了,我的第一个即时通信服务端通过了所有联调。再没有比这种成就感更让人兴奋的了,我都忘记了没有吃午饭带来的饥饿感,完全沉浸在极度的兴奋中。

小知识:认识线程

要理解线程,先看一下进程,进程是应用程序的执行实例,每个进程是由私有的虚拟地址空间、代码、数据和其它系统资源组成。进程在运行时创建的资源随着进程的终止而死亡。

线程的基本思想很简单,它是一个独立的执行流,是进程内部的一个独立的执行单元,相当于一个子程序,它对应Visual C++中的CwinThread类的对象。单独一个执行程序运行时,缺省的运行包含的一个主线程,主线程以函数地址的形式,如main或WinMain函数,提供程序的启动点,当主线程终止时,进程也随之终止,但根据需要,应用程序又可以分解成许多独立执行的线程,每个线程并行的运行在同一进程中。

一个进程中的所有线程都在该进程的虚拟地址空间中,使用该进程的全局变量和系统资源。操作系统给每个线程分配不同的CPU时间片,在某一个时刻,CPU只执行一个时间片内的线程,多个时间片中的相应线程在CPU内轮流执行,由于每个时间片时间很短,所以对用户来说,仿佛各个线程在计算机中是并行处理的。操作系统是根据线程的优先级来安排CPU的时间,优先级高的线程优先运行,优先级低的线程则继续等待。

线程被分为两种:用户界面线程和工作线程(又称为后台线程)。用户界面线程通常用来处理用户的输入并响应各种事件和消息,其实,应用程序的主执行线程CWinAPP对象就是一个用户界面线程,当应用程序启动时自动创建和启动,同样它的终止也意味着该程序的结束,进程终止。

工作者线程用来执行程序的后台处理任务,比如计算、调度、对串口的读写操作等,它和用户界面线程的区别是它不用从CwinThread类派生来创建,对它来说最重要的是如何实现工作线程任务的运行控制函数。工作线程和用户界面线程启动时要调用同一个函数的不同版本;最后需要读者明白的是,一个进程中的所有线程共享它们父进程的变量,但同时每个线程可以拥有自己的变量。

多线程的好处在于可以提高CPU的利用率——任何一个程序员都不希望自己的程序很多时候没事可干,在多线程程序中,一个线程必须等待的时候,CPU可以运行其它的线程而不是等待,这样就大大提高了程序的效率。

然而我们也必须认识到线程本身可能影响系统性能的不利方面,以正确使用线程:线程也是程序,所以线程需要占用内存,线程越多占用内存也越多;多线程需要协调和管理,所以需要CPU时间跟踪线程;线程之间对共享资源的访问会相互影响,必须解决竞用共享资源的问题;线程太多会导致控制太复杂,最终可能造成很多Bug。

举重若轻的 ASP.NET 开发工具
—— Web Matrix

▶撰文 / 韩磊

Matrix让你想起了什么？

——对！黑客帝国。就在该片第二集火爆上映之时，一个以“ Matrix ”命名的ASP.NET开发工具也Reloaded（重装上阵）了。

对使用过正版 Visual Studio.Net的人，一定知道 Visual Studio.Net占用的内存、硬盘和它惊人的价格。举例来说，Visual Studio.Net的企业版价格是几千美金，一套软件的光盘就有好几张，如果真正进行软件开发的话，即使你的计算机有128M内存，在开发的时候你也要有很好的耐心来等待计算机的反应。另外，依照微软的霸气，Visual Studio.Net这套软件设计的目的就是满足微软开发者的所有需求，这个开发工具除了可以开发.NET应用以外，还可以设计WEB Service，设计网页，甚至设计程序图标、位图等。我们可以想像，如果不是专业的软件公司，一般开发者甚至一些小一点的软件公司都不会喜欢这样庞大而高价的开发工具。如果仅仅为了开发ASP.NET应用，一般公司都不会希望有这么巨大的投资。

如果你既买不起正版又不齿于盗版，那么你一定要试试Web Matrix。它体积细小，功能强大，系出名门，完全免费……实在有太多的理由让我们接受它。

这是一款完全用C＃和.NET FRAMEWORK开发出来的工具。它的原型最初被命名为Web Studio，其间经历了 Mongoose、Project Saturn、Tahiti Project，最后才定名为Web Matrix。

那么，Web Matrix到底有何高招？笔者最初也是抱着这个疑惑下载并安装了Web Matrix，一用之下不忍释手，再用之下竟为之叹服。且让我把它的“官方(其实Web Matrix何来官方可言）”特性介绍一番，请看：

（我想告诉读者的是，下列的特性我一一试过，绝非广告。）

● 以所见即所得方式设计 ASP.NET和HTML页面。直接从工具盒拖放控件到页面，双击控件自动生成事件代码。更妙的是，当你放置一个自定义控件(.ascx)到页面，Web Matrix会正确地绘制它，这一点连VS.NET都没有做到。当然你也可以修改控件属性，使之符合要求。

● 支持 MS SQL Server/MSDE和 Access数据库。直接拖放数据库对象到页面，通过向导生成ADO.NET访问代码。几乎不需要手工编写代码，就可以通过Web Matrix完成捆绑数据的页面。拖拉数据库的表到页面，可以创建捆绑数据的网格，使用数据模板创建报表或者Master/Detail页面。而且你还可以通过内建工具管理数据库（包括数据库、数据表、关系、存储过程、记录的创建、修改、删除等）。

● 支持C＃、VB.NET、J＃ 编码。通过.NET Framework支持其它任何语言编写的ascx和aspx文件。

不过“代码完成（code complete）”的功能欠佳，不能不说是极大的遗憾。

● 自带Class Browser工具，.NET Framework对象体系一目了然。我还把它当作简明手册来用，非常方便。

● 支持XML Web Service的创建和修改。开发人员可以非常方便的创建并列出基于SOAP的XML Web服务，当然也可以让开发人员调用其它服务器上的Web服务。

● 支持移动应用（PDA、智能电话、呼机）编写。

● 基于FTP或本地文件系统的项目管理。也就是说，你可以在远程FTP站点上“直接”编写ASP.NET应用页面。上传、下载等细节由Web Matrix内置的FTP功能执行，开发者无须进行繁琐的文件管理操作。想想Macromedia DreamWeaver的FTP站点管理功能，你不用关心站点更新问题，只要集中精力于开发本身。不过我建议读者在测试站点端口使用这个功能，因为你根本保证不了每次编译结果都是完美的。我一般用81或82之类的端口和独立的目录做测试版本，在整个开发过程结束之后，再把站点移到正式目录。

● 内建的Web Server。这款WEB服务器的目的是调试程序，因此，它和ASP.NET Web Matrix有比较完善的结合，可以直接在ASP.NET Web Matrix打开WEB服务器进行调试。按下F5键，可以看到页面在本机8080端口运行了。这个特性的优点不言而喻——你甚至不需要安装IIS，就可以享受本机调试ASP.NET应用的方便。该内建WEB服务器还支持Web Service。8080端口的设置，是希望不要和你的IIS的端口相冲突。

● 我想特别提及的是Web Matrix的社区集成。在右下角toolbox的Community Tab中，有ASP.NET、Web Matrix相关站点、论坛、新闻组等社区资源列表。你可以在集成环境中立刻得到此中高手的技术支持。

实际上，Web Matrix并不是由微软公司内部正式的开发小组开发，而是由ASP.NET团队中的一群人利用它们的业余时间（主要是晚上和周末）完成的。在某些方面，Web Matrix甚至是ASP.NET新功能的试验场。因此，你不但可以感受到源自微软的技术实力，更可抢先尝试未来版本ASP.NET的某些特性。

那么Web Matrix与Visual Studio .NET究竟有何不同呢？为什么微软有了Visual Studio .Net，还要再推出Web Matrix这个Asp.Net工具呢？是因为这两个工具面向的是不同类型的开发，Web Matrix有点像Macromedia公司的Dreamweaver UltraDev，而Visual Studio是一组企业开发工具，它使用的是SourceSafe，它的理念是工程。ASP.NET Web Matrix是一个文件编辑器，它没有工程的概念。

Visual Studio.Net使用Code-Behind技术来创建ASP.NET项目，而Web Matrix使用的是内联代码（inline code）技术。在Visual Studio中，服务器端脚本代码的标准设置是在一个叫做filename.aspx.cs或filename.aspx.vb的单个文件中。这个文件被编译成工程级别DLL，不可视地被Web服务器所引用。Web Matrix之中，像上述所讲，服务器端代码在所使用的ASPX文件内部被引用，这在改善了简易性的同时降低了可升级性。由于Web Matrix的目标就是为那些在ASP.NET架构中需要更多的基础性服务的人提供更高的简易性，因此这样做是很好的。当然，这种方法的好坏最终取决于你或者是开发小组实际怎么开发应用程序。如果你们雇了一个图形设计师来设计页面的可视部分，雇了其它人，一般来说是程序员来完成页面的代码部分，你也许更愿意将它们分开，不同的部分使用不同的文件。

最后，Visual Studio.Net是一个全面性的开发工具，而ASP.NET Web Matrix只针对Web。

好了，下面是有关Web Matrix的一些资源地址，供读者参考。

Web Matrix“官方”站点：http://www.asp.net/webmatrix/default.aspx

Web Matrix下载：http://www.asp.net/webmatrix/download.aspx?tabindex=4

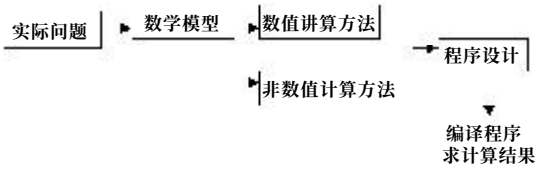
Web Matrix在线论坛：http://www.asp.net/Forums/ShowForum.aspx?tabindex=1&ForumID=30

另外，微软提供了新的ASP.NET Starter Kits（初级开发者工具包）。ASP.NET Starter Kits中含有五个ASP.NET应用的样例。编程人员通过参考这五个样例可以迅速地完成ASP.NET应用的开发。这些样例从电子商务店面应用到社区入口数据报表等，编写十分详细。微软还为这五个样例提供了源代码。这些工具包目前可以从www.asp.net/starterkits上面下载。

计算机科学数学理论浅谈

▶撰文/曾毅

计算机自从其诞生之日起,它的主要任务就是进行各种各样的科学计算。文档处理,数据处理,图像处理,硬件设计,软件设计等等,都可以抽象为两大类:数值计算与非数值计算。作为研究计算机科学技术的人员,我们大都对计算数学在整个计算机科学中的重要性有一些了解。但是数学对我们这些专业的研究和应用人员究竟有多大的用处呢?我们先来看一下下面的一个流程图:



上图揭示了利用计算机解决科学计算的步骤,实际问题转换为程序,要经过一个对问题抽象的过程,建立起完善的数学模型,只有这样,我们才能建立一个设计良好的程序。从中我们不难看出计算数学理论对用计算机解决问题的重要性。下面我们将逐步展开对这个问题的讨论。

计算机科学的数学理论体系是相当庞杂的,笔者不敢随意划分,参考计算机科学理论的学科体系,我们谈及的问题主要涉及:数值计算,离散数学,数论,计算理论四大方向。

一 数值计算(Numerical Computation)

主要包括数值分析学、数学分析学、线性代数、计算几何学、概率论与数理统计学。

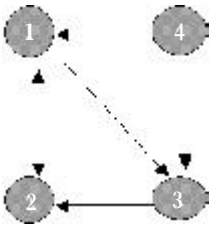
数值分析学又常被称为计算方法学,是计算理论数学非常重要的一个分支,主要研究数值型计算。研究的内容中首先要谈谈数值计算的误差分析,误差是衡量我们的计算有效与否的标准,我们的算法解决问题如果在误差允许的范围内,则算法是有效的,否则就是一个无效的问题求解。另外就是数值逼近,它研究关于如何使用容易数值计算的函数来近似地代替任意函数的方法与过程,感觉应用比较广的不得不提切雪比夫逼近和平方逼近了。笔者曾经尝试过的就是通过最佳平方逼近进行曲线的拟合,开发工具可以选择VC++或者Matlab。插值函数是另外一个非常重要的方面,现代的计算机程序控制加工机械零件,根据设计可给出零件外形曲线的某些型值点,加工时走刀方向及步数,就要通过插值函数计算零件外形曲线及其他点函数值。至于方程求根、线性方程组求解,一般的计算性程序设计问题都会多多少少的涉及一些,我们这里就不赘述了。关于数值分析学的一个学习误区就是仅仅学习理论知识,而很难和程序设计结合起来,实际上通过上面的论述,大家已经能够初步地认识到这个学科是应当与程序设计紧密联系才能够体现它的重要性的。关于理论的学习,推荐华中科技大学李庆扬老师的《数学分析》。然而理论学习毕竟是个过程,最终的目标还是要用于程序设计解决实际的计算问题,向这个方向努力的书籍还是挺多的,这里推荐高等教育出版社(CHEP)和施普林格出版社(Springer)联合出版的《计算方法(Computational Methods)》,由华中理工大学数学系编写(现华中科技大学),这方面华科大做的工作在国内应算是比较多的,而个人认为以这本最好,至少程序设计方面涉及了:任意数学函数的求值、方程求根、线性方程组求解、插值方法、数值积分、场微分方程数值求解。

数学分析学很多学校在近些年已经替代高等数学被安排到了本科教学当中。原因是很简单的,高等数学虽然也是非常有用的工程数学,介绍的问题方法也被广泛的应用,但是正如大家所知道的,高等数学不太严格的说,基本上就是偏向于计算的数学分析,当然省去了数学分析非常看重的推理证明,然而我们认为这一部分正是我们最需要的。这对我们培养良好的分析能力和推理能力极有帮助。我的软件工程学导师北工大数理学院的王仪华先生就曾经教导过我们,数学系的学生到软件企业中大多作软件设计与分析工作,而计算机系的学生做初级程序员的居多,原因就在于数学系的学生分析推理能力,从所受训练的角度上要远远在我们平均水平之上。谈到这方面的书籍,公认北京大学张筑生老师的《数学分析新讲》为最好。张筑生教授一生写的书并不太多,但是只要是写出来的每一本都是本领域内的杰作,这本当然更显突出。这种老书看起来不仅是在传授你知识,而

是在让你体会科学的方法与对事物的认识方法。现在多用的多似乎是复旦大学的《数学分析》,高等教育出版社的,也是很好的教材。但关于如何去利用从中获得的推理证明能力,我们在遇到具体问题的时候,可以在今后的文章详细讨论。

线性代数是我们在工科本科学习的必修课程,似乎大家找不到到底这个有什么用,其实很明显,线性代数作为工程数学的重要分支,在计算机领域的研究有很广泛的应用。最为突出的可以谈谈数组和矩阵的相关知识:

下面谈一个我经常作为例子和同学讨论的问题:四个城市之间的航线如图所示:



令aij=1,表示从i市到j市有1条航线

令aij=0, 表示从i市到j市没有单项航线

则图可用矩阵表示:

$$A=(aij)=\begin{pmatrix}0110\\1000\\0100\\1010\end{pmatrix}$$

我们可以采用程序设计实现这个问题,如果辅以权值,可以转化为最短路径的问题,再复杂化一点还可以转化为具有障碍物的最短路径问题,这就会涉及一些如Dijkstra算法等高级程序设计算法话题。这些都依靠着数组、矩阵的基本知识。数组的应用主要在图像处理以及一些程序设计理论。矩阵的运算领域极为广泛,比如在计算机图形学当中曲线曲面的构造,图像的几何变换,包括平移、镜像、转置、缩放。在高级图像问题更有广泛应用,例如在图像增强技术,投影技术中的应用。

计算几何学研究的是几何外形信息的计算机表示。包括几何查找、多边形、凸包问题、交与并、几何体的排列、几何拓扑网络设计、随机几何算法与并行几何算法。它构成了计算机图形学中的基本算法,是动画设计,制造业计算机辅助设计的基础。如果从事这方面的深入研究,可以参考中国计算机学会周培德先生的《计算几何--算法分析与设计》。

概率论与数理统计学是这个领域最后一门关键的课程。概率论部分提供了很多问题的基本知识描述,比如模式识别当中的概率计算,参数估计等等。数理统计部分有很多非常经典的内容,比如伪随机数、蒙特卡罗法、回归分析、排队论、假设检验、以及经典的马科夫过程。尤其是随机过程部分,是分析网络和分布式系统,设计随机化算法和协议非常重要的基础。

二 离散数学(Discrete Mathematics)

随着计算机科学的出现与广泛应用,人们发现利用计算机处理的数学对象与传统的分析有明显的区别:分析研究的问题解决方案是连续的,因而微分,积分成为基本的运算;而这些分支研究的对象是离散的,因而很少有机会进行此类的计算。人们从而称这些分支为"离散数学"。离散数学经过几十年发展,方向上基本上稳定下来。当然不同时期还有很多新内容补充进来。就学科方向而言,一般认为,离散数学包含:集合论、逻辑学、代数学、图论、组合学。

逻辑学(Logics)我们主要指数理逻辑,形式逻辑在推理问题中也有比较广泛的应用。(比如我们学校还为此专门开设了选修课程)这方面的参考推荐中科院软件所陆钟万教授的《面向计算机科学的数理逻辑》。现在可以找到陆钟万教授的讲课录像, http://www.cas.ac.cn/html/Dir/2001/11/06/3391.htm。总的来说,学集合/逻辑一定要站在理解的高度上去思考相关的问题。集合论(Set Theory)和逻辑学构成了计算机科学最重要的数学问题描述方式。

代数学(Algebra)包括:抽象代数、布尔代数、关系代数、计算机代数

(1)抽象代数(Abstract Algebra)研究的主要内容涵盖群、环、域。抽象代表的是将研究对象的本质提炼出来,加以高度概括,来描述其形象。"欧式环"就是在将整数和多项式的一些相同的特点加以综合提炼引入的。抽象代数提供的一些结论为我们研究一些具体问题时所需使用的一些性质提供了依据。推荐一个最简单的,最容易学的材料:http://www.math.miami.edu/~ec/book/ 这本《Introduction to Linear and Abstract Algebra》非常通俗易懂,而且把抽象代数和线性代数结合起来,对初学者来说非常理想。

(2)布尔代数(Boolean Algebra)是代数系统中最为基础的部分,也是最核心的基本理论。主要包括了集合的基本概念与运算,自对偶的公理系统。是数据表示的重要基础。相信大家都很清楚它的重要性。

(3)关系代数(Relational Algebra)应用也是极为广泛,比如数据库技术中的关系数据库的构建就要用到关系代数的相关理论。

(4)计算机代数(Computer Algebra)大家可能比

较生疏,其实它研究的主要内容即是围绕符号计算与公式演算展开的,是研究代数算法的设计、分析、实现及其应用的学科。主要求解非数值计算,输入输出用代数符号表示。计算机代数的开发语言主要有:ALTRAN,CAMAL,FORMAL。主要应用于:射影几何,工业设计,机器人手臂运动设计。

图论(Graph Theory)主要研究的内容包括:图的基本概念、基本运算、矩阵表示,路径、回路和连通性,二部图、平面图,树,以及网络流。图论的应用领域太过广泛,仅举两个例子:比如在计算机网络拓扑图的设计与结构描述中,就必须用到相当多的图的结构和基本概念。关于网络流更是在电流网络与信息网络的流量计算当中广泛应用。树的相关应用则无须多言了。

组合学(Combinatorics)有两部分单独的研究领域:组合数学与组合算法。组合学问题的算法,计算对象是离散的、有限的数学结构。从方法学的角度,组合算法包括算法设计和算法分析两个方面。关于算法设计,历史上已经总结出了若干带有普遍意义的方法和技术,包括动态规划、回溯法、分支限界法、分治法、贪心法等。应用是相当广泛的,比如旅行商问题、图着色问题、整数规划问题。关于组合数学,主要研究的内容有:鸽巢原理、排列与组合、二项式系数容斥原理及应用,递推关系和生成函数、特殊计数序列、二分图中的匹配、组合设计。推荐Richard A.Brualdi的《Introductory Combinatorics》作为参考。

三 数论(Number Theory)

数论这门学科最初是从研究整数开始的,所以叫做整数论。后来更名为数论。它包括以下几个分支:

初等数论是不求助于其他数学学科的帮助,只依靠初等方法来研究整数性质的数论分支。比如在数论界非常著名的"中国剩余定理",就是初等数论中很重要的内容。对于程序设计来说这部分也是相当有价值的,如果你对中国剩余定理比较清楚,利用它,你可以将一种表达式经过简单的转换后得出另一种表达式,从而完成对问题分析视角的转换。

解析数论是使用数学分析作为工具来解决数论问题的分支。是解决数论中比较深刻问题的强有力工具。我国数学家陈景润在尝试解决"哥德巴赫猜想"问题中使用的就是解析数论的方法。以素数定理为基础解决计算素数的问题及其算法实现应是我们多多关注的。

代数数论是把整数的概念推广到一般代数数域上去,建立了素整数、可除性等概念。程序设计方面涉及比较多的是代数曲线的研究,比如说椭圆曲线理论的实现。

几何数论研究的基本对象是"空间格网"。空间格网就是指在给定的直角坐标系上,坐标全是整数的点,叫做整点;全部整点构成的组就叫做空间格网。空间格网对计算几何学的研究有着重大的意义。几何数论涉及的问题比较复杂,必须具有相当的数学基础才能深入研究。

总的说来,由于近代计算机科学的发展,数论得到了广泛的应用。比如在计算方法、代数编码、组合学理论等方面都广泛使用了初等数论范围内的许多研究成果;现在有些国家应用"孙子定理"来进行测距,用原根和指数来计算离散傅里叶变换等。如果你曾经系统的学习过数论算法,你会发现这个分支学科研究的一些基本问题对程序设计是相当有用的,比如说素数问题、素性测试、因子分解、最大公约数、模取幂运算、求解同余线性方程。其中的很多问题都是程序设计的基本问题。但这些问题都不能小视,举个例子来说吧,关于求最大公约数的程序,笔者曾经尝试的就可以采用循环语句结构和递归结构。另外,以大素数为基础的密码体系的建立是近些年数论算法广泛应用的一个重要的原因。原理是大素数的乘积重新分解因数十分困难。RSA公钥加密系统的构建就是基于这个原理的(三位发明人因此也获得了2002年美国计算机协会颁发的图灵奖)。

四 计算理论(Theory of Computation)

这一部分涉及的内容是科学计算非常重要的一个分支,也是大家研究相当多的一部分。主要包括:算法学、计算复杂性、程序理论。

算法学(Algorithms)在计算机科学理论中有着举足轻重的地位。是解决很多数值型,非数值型问题的基础。记得一次学校接收招标项目,很多中小型软件厂商都无法完成一个软件的功能模块,就是因为当时他们对一个具体问题的算法不能做出正确的抽象,最后由我们学校数理学院的一支软件团队承担了这项任务,他们的最终报告体现出来,问题的解决策略只有通过人工神经元网络的反向传播算法。可见在比较有深度的程序设计中,算法的重要性更为突出。学习算法学要有一个长期的理论和实践的过程。遇到一个具体算法问题时,首先要通过自己描述的数学抽象步骤,看看自己以前有没有处理过这种问题。如果没有,很可能这个问题是多个算法的综

合,或者是需要我们自己去构造算法。这就需要我们有扎实的算法功底,为了打好这个功底,推荐两套圣经级的书籍首先是Thomas H.Cormen等著的《Introduction to Algorithms》。对算法学习而言,这一本内容相当的全面。再深一点的就是大家作为常识都知道的《The Art of Computer Programming》,目前已经出版3册。两本书的价值大家应当都是清楚的。

计算复杂性研究的内容很广,其中包括NP完全性理论、可计算性理论、自动机理论、形式语言理论(包括广泛应用于编译原理领域的文法,还包括Petri网论的相关内容)以及大家熟知的复杂性度量。时间复杂度、空间复杂度的计算是度量算法非常重要的参数,也是我们衡量程序优劣程度的重要依据。

程序理论(Theory of programs)包含了形式语义学,程序验证和并发模型的研究。关于程序验证学习的重要性大家都很清楚,学习的方法自然也是多多结合具体的问题去分析。关于并发模型,主要研究的就是进程代数,通信系统演算,通信顺序进程。这部分是研究操作系统理论与实现的重要基础。

上面我们按照计算机科学数学理论的架构来谈了各方面的内容和一些应用,下面我们再单独来看一些上面没有涉及到的学科与这些理论的具体结合情况:

软件程序设计方面的应用刚才谈的很多,我再说说数据库原理与技术,这方面用到的重要数学基础主要包括:集合论、二元关系及其推理(尤其是研究关系数据库)、研究数据分布与数据库结构又涉及很多图论知识。

计算机科学的发展有赖于硬件技术和软件技术的综合。在设计硬件的时候应当充分融入软件的设计思想,才能使硬件在程序的指挥下发挥极致的性能。在软件设计的时候也要充分考虑硬件的特点,才能冲破软件效率的瓶颈。达到硬件和软件设计的统一,严格的说这并不轻松,一般的程序设计者很难将这样的思想贯穿在其程序设计当中。仅举个简单的例子:我们在写一些C语言的程序,必要的时候都会采取内嵌一段汇编指令,这就是比较充分地考虑了硬件的工作情况,从而能够提高程序运行的效率。所以我们也有必要了解一些硬件的基础知识。关于学习硬件的时候常会用到的基本数学思想也是相当多的,拿电路基础与模拟电路来说,我们就经常要利用多元函数,不等式计算进行电流电压的计算。能量的计算还常常涉及微积分学的很多计算。在数字电子技术当中(有时也称数字逻辑学)数理逻辑,尤其是逻辑演算部分运用相当广泛,数制转换更是非常重要的基础,各种数字电路参数的计算则是多元函数,不等式计算解决的问题。

如果你是从事计算机硬件程序设计的程序员,则不可回避的就是数字信号处理。这门科学所用到的数学基础主要有:三角函数、微积分、高次方程求解、数值逼近、傅里叶变换。在滤波器的设计当中还会用到矩阵运算。笔者曾经研究过一个VC++环境下开发的滤波器的模拟软件,就是利用莱文逊-杜宾递推算法,在较大规模的矩阵运算基础上进行的。当然,开发的环境不一定是这个,你也可以选择MATLAB或者纯C语言编译器。如果我们不了解相关的数学基础,不要说程序设计,就算是建立运算模型都是相当困难的。

平时接触一些周围的同学和一些在职的程序员,大家经过一段时间的学习,普遍都觉得数学对学习计算机和研究计算机程序设计等问题来说非常重要,但是又苦于无从下手。上面比较全面地谈及了计算机科学数学理论的相关内容。需要特别指明的是,我们研究问题的精力是有限的,如果您是在校的计算机系学生,则可以对上面的方方面面都有所涉及,以尝试计算数学这个强大的理论工具,为今后的工作奠定一个坚实的基础。但是如果您研究的是比较具体的工作,我们并不推荐您研究所有的内容,最好的方法就是对上面的数学基础都有些了解,然后遇到具体工作,需要哪部分内容,再进行深入的学习与研究。这样针对性比较强的学习效果是会比较显著的。对于上面推荐的一些参考材料,除非你要花相当长的一段时间来提高你的计算机数学理论。否则也没必要每一页,每一本都字字精读,还是那个原则,按需索取其中的内容。学习的方法描述起来就一句话:结合具体的问题,深入的理解数学理论知识,将理论程序化,尝试用程序设计实现理论原理。达到这样的程度,问题基本上都可以解决的。(限于篇幅,很多问题不能展开,您可以通过zengyi820@hotmail.com与我联系或者把问题直接反馈给杂志社)

参考文献:

《计算机科学技术百科全书》中国计算机学会 清华大学出版社

《工程数学-线性代数》同济大学数学教研室 同济大学出版社

《数值分析》李庆扬 华中科技大学出版社

面向对象设计思辩一则

▶撰文/区思

C++ Gotchas(中译名《C++程序设计陷阱》)的出版,正赶上好时候。不过在我看来,这本书值得一读,却难称经典。为什么呢?盖因天时已变,此书如果是十年前问世,成就当可比肩Effective C++,十年之后,C++应用高峰已过,渐趋退守之势,即便是坚持不懈之人,要么已经习惯成形,积重难返,要么早已升堂入室,无暇后顾。再加上Scott Meyers在此领域中已经是体察甚微,Effective C++和More Effective C++在前,此书之意义,早已被时光消磨大半,怎能不令人扼腕长叹!

不过在全书最后两章中,作者确实显示出异乎寻常的勇气,针对多个长期存在争议的设计思想进行了思辩。虽然作者的那些观点在C++阵营内早已算不上新鲜,但是把这些观点一条条写下来,放在书里面大声宣扬,这可是需要相当胆量的。比如Gotchas 93抨击了单根继承树结构。书里是这么说的:

“从设计角度来看,单根层次结构经常会导致所谓的‘通用对象容器’。这些容器中包含的元素是无可预料的,因此可能导致意外的运行时行为。在Bjarne Stroustrup的经典反例中,他考察了把战列舰放在笔筒中的可能性。在单根层次结构中,这绝对可能,而笔筒的主人定然会对这种可能性感到莫名其妙。

缺乏经验的程序员往往有一个危险的假设,即体系结构应该尽可能灵活。这种假设是错误的。相反,体系结构应该尽可能与问题域接近,而同时保持足够的灵活性,以允许将来进行扩展。”

好大胆的抨击!

自从70年代Smalltalk出现以来,从MFC到VCL,从Java到.NET BCL,所有成功的面向对象框架,几乎无一例外全部采用了单根继承树结构。“实践是检验真理的唯一标准”,这些框架都历经实践,难道说它们都错了?或者Dewhurst是在信口开合?

笔者认为,单根不单根,这事无关大局,原因如下:

对于C++、Java和C#等语言来说,class基本上只是一个静态观念。程序一旦运行,则在概念上只有一个个的对象在协同运作。既然如此,我们应该牢牢记住,作为程序员,我们是要设计对象,而不是类。之所以我们把大部分时间花在类的设计之上,实在是迫不得已之举:一来类是产生对象的唯一途径,二来通过类来生成大量的同质对象也会比较便捷。

澄清这个观念十分重要,因为对象与对象之间的关系比类与类之间的关系简单。对象之间只有两大关系——包容和使用,而类之间的关系要复杂得多,使用、组合、接口继承、实现继承、参数类型化等等。因此,如果我们忘记了类设计的目的是对象设计,就很有可能一开始就陷入到极其复杂的类关系中,不能自拔。因此,我强烈建议,在设计系统时,先进行动态设计,之后进行静态设计,而不是相反。先进行动态设计,得出来的系统即使不是最优的,至少是可用的。而先进行静态设计,也许可以得到最优设计,但也很可能得到一个非常糟糕的设计。

如果我们先设计对象系统,再设计类型系统,那我们会发现,继承也好,模板也好,其实只不过是实现层面的事情,其唯一目的就是实现合适的对象。反过来说,如果能够实现合适的对象,则具体采用什么机制,无关乎系统大局。

有了这个认识,我们回到单根层次结构的讨论。一个framework,采用单根结构也好,不采用单根结构也好,对于运行着的系统本身并没有意义。因为在运行系统中,只有对象,而对象之间,不存在继承关系。连继承关系都不存在,单根不单根谁在乎?

因此,无论是单根还非单根,都可以设计出优秀的系统,也都可以设计出垃圾。Bjarne Stroustrup批评单根结构可能会“把战列舰装到笔筒里”,批评的虽然有理,不过从MFC到.NET,实践证明单根系统还是不错的。反过来说,即使没有单根系统,糟糕的设计者也能够做出比“把战列舰装到笔筒里”更荒唐的事情。我们探讨单根的优劣,只能从这样的问题出发,即:单根是否有助于程序员更方便地设计出合适的对象?

对这个问题的回答,其实就比较简明了:对于拥有强大泛型机制的C++来说,单根可有可无,而对于Java和.NET来说,单根是决不可少。说来说去,大家都有道理,这不是无关大局吗?大家可以去看Eiffel的设计,既有泛型又有单根,不也不亦乐乎吗?只不过在容器的实现上,用单根来弥补缺乏泛型机制的尴尬,实在显得有些吃力。这也就是为什么Java和.NET都将加入泛型能力的原因。

异议大师,罪过罪过。

数据压缩技术简史

▶撰文/王咏刚

电脑里的数据压缩其实类似于美眉们的瘦身运动，不外有两大功用。第一，可以节省空间。拿瘦身美眉来说，要是八个美眉可以挤进一辆出租车里，那该有多省钱啊！第二，可以减少对带宽的占用。例如，我们都想在不到100Kbps的GPRS网上观看DVD大片，这就好比瘦身美眉们总希望用一尺布裁出七件吊带衫，前者有待于数据压缩技术的突破性进展，后者则取决于美眉们的恒心和毅力。

简单地说，如果没有数据压缩技术，我们就没法用WinRAR为Email中的附件瘦身；如果没有数据压缩技术，市场上的数码录音笔就只能记录不到20分钟的语音；如果没有数据压缩技术，从Internet上下载一部电影也许要花半年的时间……可是这一切究竟是如何实现的呢？数据压缩技术又是怎样从无到有发展起来的呢？

1．概率奇缘

一千多年前的中国学者就知道用“班马”这样的缩略语来指代班固和司马迁，这种崇尚简约的风俗一直延续到了今天的Internet时代：当我们在BBS上用“7456”代表“气死我了”，或是用“B4”代表“Before”的时候，我们至少应该知道，这其实就是一种最简单的数据压缩呀。

严格意义上的数据压缩起源于人们对概率的认识。当我们对文字信息进行编码时，如果为出现概率较高的字母赋予较短的编码，为出现概率较低的字母赋予较长的编码，总的编码长度就能缩短不少。远在计算机出现之前，著名的Morse电码就已经成功地实践了这一准则。在Morse码表中，每个字母都对应于一个唯一的点划组合，出现概率最高的字母e被编码为一个点“.”，而出现概率较低的字母z则被编码为“--..”。显然，这可以有效缩短最终的电码长度。

信息论之父C．E．Shannon第一次用数学语言阐明了概率与信息冗余度的关系。在1948年发表的论文“通信的数学理论（A Mathematical Theory of Communication）”中，Shannon指出，任何信息都存在冗余，冗余大小与信息中每个符号（数字、字母或单词）的出现概率或者说不确定性有关。Shannon借鉴了热力学的概念，把信息中排除了冗余后的平均信息量称为“信息熵”，并给出了计算信息熵的数学表达式。这篇伟大的论文后来被誉为信息论的开山之作，信息熵也奠定了所有数据压缩算法的理论基础。从本质上讲，数据压缩的目的就是要消除信息中的冗余，而信息熵及相关的定理恰恰用数学手段精确地描述了信息冗余的程度。利用信息熵公式，人们可以计算出信息编码的极限，即在一定的概率模型下，无损压缩的编码长度不可能小于信息熵公式给出的结果。

有了完备的理论，接下来的事就是要想办法实现具体的算法，并尽量使算法的输出接近信息熵的极限了。当然，大多数工程技术人员都知道，要将一种理论从数学公式发展成实用技术，就像仅凭一个$E=mc^2$的公式就要去制造核武器一样，并不是一件很容易的事。

2．数学游戏

设计具体的压缩算法的过程通常更像是一场数学游戏。开发者首先要寻找一种能尽量精确地统计或估计信息中符号出现概率的方法，然后还要设计一套用最短的代码描述每个符号的编码规则。统计学知识对于前一项工作相当有效，迄今为止，人们已经陆续实现了静态模型、半静态模型、自适应模型、Markov模型、部分匹配预测模型等概率统计模型。相对而言，编码方法的发展历程更为曲折一些。

1948年，Shannon在提出信息熵理论的同时，也给出了一种简单的编码方法——Shannon编码。1952年，R．M．Fano又进一步提出了Fano编码。这些早期的编码方法揭示了变长编码的基本规律，也确实可以取得一定的压缩效果，但离真正实用的压缩算法还相去甚远。

第一个实用的编码方法是由D．A．Huffman在1952年的论文“最小冗余度代码的构造方法（A Method for the Construction of Minimum Redundancy Codes）”中提出的。直到今天，许多《数据结构》教材

在讨论二叉树时仍要提及这种被后人称为Huffman编码的方法。Huffman编码在计算机界是如此著名，以至于连编码的发明过程本身也成了人们津津乐道的话题。据说，1952年时，年轻的Huffman还是麻省理工学院的一名学生，他为了向老师证明自己可以不参加某门功课的期末考试，才设计了这个看似简单，但却影响深远的编码方法。

Huffman编码效率高，运算速度快，实现方式灵活，从20世纪60年代至今，在数据压缩领域得到了广泛的应用。例如，早期UNIX系统上一个不太为现代人熟知的压缩程序COMPACT实际就是Huffman 0阶自适应编码的具体实现。20世纪80年代初，Huffman编码又出现在CP/M和DOS系统中，其代表程序叫SQ。今天，在许多知名的压缩工具和压缩算法（如WinRAR、gzip和JPEG）里，都有Huffman编码的身影。不过，Huffman编码所得的编码长度只是对信息熵计算结果的一种近似，还无法真正逼近信息熵的极限。正因为如此，现代压缩技术通常只将Huffman视作最终的编码手段，而非数据压缩算法的全部。

科学家们一直没有放弃向信息熵极限挑战的理想。1968年前后，P.Elias发展了Shannon和Fano的编码方法，构造出从数学角度看来更为完美的Shannon-Fano-Elias编码。沿着这一编码方法的思路，1976年，J.Rissanen提出了一种可以成功地逼近信息熵极限的编码方法——算术编码。1982年，Rissanen和G.G. Langdon一起改进了算术编码。之后，人们又将算术编码与J.G.Cleary和I.H.Witten在1984年提出的部分匹配预测模型（PPM）相结合，开发出了压缩效果近乎完美的算法。今天，那些名为PPMC、PPMD或PPMZ并号称压缩效果天下第一的通用压缩算法，实际上全都是这一思路的具体实现。

对于无损压缩而言，PPM模型与算术编码相结合，已经可以最大程度地逼近信息熵的极限。看起来，压缩技术的发展可以到此为止了。不幸的是，事情往往不像想象中的那样简单：算术编码虽然可以获得最短的编码长度，但其本身的复杂性也使得算术编码的任何具体实现在运行时都慢如蜗牛。即使在摩尔定律大行其道，CPU速度日新月异的今天，算术编码程序的运行速度也很难满足日常应用的需求。没办法，如果不是后文将要提到的那两个犹太人，我们还不知要到什么时候才能用上WinZIP这样方便实用的压缩工具呢。

3．异族传说

逆向思维永远是科学和技术领域里出奇制胜的法宝。就在大多数人绞尽脑汁想改进Huffman或算术编码，以获得一种兼顾了运行速度和压缩效果的"完美"编码的时候，两个聪明的犹太人J.Ziv和A.Lempel独辟蹊径，完全脱离Huffman及算术编码的设计思路，创造出了一系列比Huffman编码更有效，比算术编码更快捷的压缩算法。我们通常用这两个犹太人姓氏的缩写，将这些算法统称为LZ系列算法。

按照时间顺序，LZ系列算法的发展历程大致是：Ziv和Lempel于1977年发表题为"顺序数据压缩的一个通用算法（A Universal Algorithm for Sequential Data Compression）"的论文，论文中描述的算法被后人称为LZ77算法。1978年，二人又发表了该论文的续篇"通过可变比率编码的独立序列的压缩（Compression of Individual Sequences via Variable Rate Coding）"，描述了后来被命名为LZ78的压缩算法。1984年，T.A.Welch发表了名为"高性能数据压缩技术（A Technique for High Performance Data Compression）"的论文，描述了他在Sperry研究中心（该研究中心后来并入了Unisys公司）的研究成果，这是LZ78算法的一个变种，也就是后来非常有名的LZW算法。1990年后，T.C.Bell等人又陆续提出了许多LZ系列算法的变体或改进的版本。

说实话，LZ系列算法的思路并不新鲜，其中既没有高深的理论背景，也没有复杂的数学公式，它们只是简单地延续了千百年来人们对字典的追崇和喜好，并用一种极为巧妙的方式将字典技术应用于通用数据压缩领域。通俗地说，当你用字典中的页码和行号代替文章中每个单词的时候，你实际上已经掌握了LZ系列算法的真谛。这种基于字典模型的思路在表面上虽然和Shannon、Huffman等人开创的统计学方法大相径庭，但在效果上一样可以逼近信息熵的极限。而且，可以从理论上证明，LZ系列算法在本质上仍然符合信息熵的基本规律。

LZ系列算法的优越性很快就在数据压缩领域里体现了出来，使用LZ系列算法的工具软件数量呈爆炸式增长。UNIX系统上最先出现了使用LZW算法的compress程序，该程序很快成为了UNIX世界的压缩标准。紧随其后的是MS-DOS环境下的ARC程序，以及PKWare、

PKARC等仿制品。20世纪80年代，著名的压缩工具LHarc和ARJ则是LZ77算法的杰出代表。

今天，LZ77、LZ78、LZW算法以及它们的各种变体几乎垄断了整个通用数据压缩领域，我们熟悉的PKZIP、WinZIP、WinRAR、gzip等压缩工具以及ZIP、GIF、PNG等文件格式都是LZ系列算法的受益者，甚至连PGP这样的加密文件格式也选择了LZ系列算法作为其数据压缩的标准。

没有谁能否认两位犹太人对数据压缩技术的贡献。我想强调的只是，在工程技术领域，片面追求理论上的完美往往只会事倍功半，如果大家能像Ziv和Lempel那样，经常换个角度来思考问题，没准儿你我就能发明一种新的算法，就能在技术方展史上扬名立万呢。

4．音画时尚

LZ系列算法基本解决了通用数据压缩中兼顾速度与压缩效果的难题。但是，数据压缩领域里还有另一片更为广阔的天地等待着我们去探索。Shannon的信息论告诉我们，对信息的先验知识越多，我们就可以把信息压缩得越小。换句话说，如果压缩算法的设计目标不是任意的数据源，而是基本属性已知的特种数据，压缩的效果就会进一步提高。这提醒我们，在发展通用压缩算法之余，还必须认真研究针对各种特殊数据的专用压缩算法。比方说，在今天的数码生活中，遍布于数码相机、数码录音笔、数码随身听、数码摄像机等各种数字设备中的图像、音频、视频信息，就必须经过有效的压缩才能在硬盘上存储或是通过USB电缆传输。实际上，多媒体信息的压缩一直是数据压缩领域里的重要课题，其中的每一个分支都有可能主导未来的某个技术潮流，并为数码产品、通信设备和应用软件开发商带来无限的商机。

让我们先从图像数据的压缩讲起。通常所说的图像可以被分为二值图像、灰度图像、彩色图像等不同的类型。每一类图像的压缩方法也不尽相同。

传真技术的发明和广泛使用促进了二值图像压缩算法的飞速发展。CCITT（国际电报电话咨询委员会，是国际电信联盟ITU下属的一个机构）针对传真类应用建立了一系列图像压缩标准，专用于压缩和传递二值图像。这些标准大致包括20世纪70年代后期的CCITT Group 1和Group 2，1980年的CCITT Group 3，以及1984年的CCITT Group 4。为了适应不同类型的传真图像，这些标准所用的编码方法包括了一维的MH编码和二维的MR编码，其中使用了行程编码（RLE）和Huffman编码等技术。今天，我们在办公室或家里收发传真时，使用的大多是CCITT Group 3压缩标准，一些基于数字网络的传真设备和存放二值图像的TIFF文件则使用了CCITT Group 4压缩标准。1993年，CCITT和ISO（国际标准化组织）共同成立的二值图像联合专家组（Joint Bi-level Image Experts Group，JBIG）又将二值图像的压缩进一步发展为更加通用的JBIG标准。

实际上，对于二值图像和非连续的灰度、彩色图像而言，包括LZ系列算法在内的许多通用压缩算法都能获得很好的压缩效果。例如，诞生于1987年的GIF图像文件格式使用的是LZW压缩算法，1995年出现的PNG格式比GIF格式更加完善，它选择了LZ77算法的变体zlib来压缩图像数据。此外，利用前面提到过的Huffman编码、算术编码以及PPM模型，人们事实上已经构造出了许多行之有效的图像压缩算法。

但是，对于生活中更加常见的，像素值在空间上连续变化的灰度或彩色图像（比如数码照片），通用压缩算法的优势就不那么明显了。幸运的是，科学家们发现，如果在压缩这一类图像数据时允许改变一些不太重要的像素值，或者说允许损失一些精度（在压缩通用数据时，我们绝不会容忍任何精度上的损失，但在压缩和显示一幅数码照片时，如果一片树林里某些树叶的颜色稍微变深了一些，看照片的人通常是察觉不到的），我们就有可能在压缩效果上获得突破性的进展。这一思想在数据压缩领域具有革命性的地位：通过在用户的忍耐范围内损失一些精度，我们可以把图像（也包括音频和视频）压缩到原大小的十分之一、百分之一甚至千分之一，这远远超出了通用压缩算法的能力极限。也许，这和生活中常说的“退一步海阔天空”的道理有异曲同工之妙吧。

这种允许精度损失的压缩也被称为有损压缩。在图像压缩领域，著名的JPEG标准是有损压缩算法中的经典。JPEG标准由静态图像联合专家组（Joint Photographic Experts Group，JPEG）于1986年开始制定，1994年后成为国际标准。JPEG以离散余弦变换（DCT）为核心算法，通过调整质量系数控制图像的精度和大小。对于照片等连续变化的灰度或彩色图像，JPEG在保证图像质量的前提下，一般可以将图像压缩到原大小的十分之一到二十分之一。如果不考虑图像质量，JPEG甚至可以将图像压缩到“无限小”。

JPEG标准的最新进展是1996年开始制定,2001年正式成为国际标准的JPEG 2000。与JPEG相比,JPEG 2000作了大幅改进,其中最重要的是用离散小波变换(DWT)替代了JPEG标准中的离散余弦变换。在文件大小相同的情况下,JPEG 2000压缩的图像比JPEG质量更高,精度损失更小。作为一个新标准,JPEG 2000暂时还没有得到广泛的应用,不过包括数码相机制造商在内的许多企业都对其应用前景表示乐观,JPEG 2000在图像压缩领域里大显身手的那一天应该不会特别遥远。

JPEG标准中通过损失精度来换取压缩效果的设计思想直接影响了视频数据的压缩技术。CCITT于1988年制定了电视电话和会议电视的H.261建议草案。H.261的基本思路是使用类似JPEG标准的算法压缩视频流中的每一帧图像,同时采用运动补偿的帧间预测来消除视频流在时间维度上的冗余信息。在此基础上,1993年,ISO通过了动态图像专家组(Moving Picture Experts Group, MPEG)提出的MPEG-1标准。MPEG-1可以对普通质量的视频数据进行有效编码。我们现在看到的大多数VCD影碟,就是使用MPEG-1标准来压缩视频数据的。

为了支持更清晰的视频图像,特别是支持数字电视等高端应用,ISO于1994年提出了新的MPEG-2标准(相当于CCITT的H.262标准)。MPEG-2对图像质量作了分级处理,可以适应普通电视节目、会议电视、高清晰数字电视等不同质量的视频应用。在我们的生活中,可以提供高清晰画面的DVD影碟所采用的正是MPEG-2标准。

Internet的发展对视频压缩提出了更高的要求。在内容交互、对象编辑、随机存取等新需求的刺激下,ISO于1999年通过了MPEG-4标准(相当于CCITT的H.263和H.263+标准)。MPEG-4标准拥有更高的压缩比率,支持并发数据流的编码、基于内容的交互操作、增强的时间域随机存取、容错、基于内容的尺度可变性等先进特性。Internet上新兴的DivX和XviD文件格式就是采用MPEG-4标准来压缩视频数据的,它们可以用更小的存储空间或通信带宽提供与DVD不相上下的高清晰视频,这使我们在Internet上发布或下载数字电影的梦想成为了现实。

就像视频压缩和电视产业的发展密不可分一样,音频数据的压缩技术最早也是由无线电广播、语音通信等领域里的技术人员发展起来的。这其中又以语音编码和压缩技术的研究最为活跃。自从1939年H. Dudley发明声码器以来,人们陆续发明了脉冲编码调制(PCM)、线性预测(LPC)、矢量量化(VQ)、自适应变换编码(ATC)、子带编码(SBC)等语音分析与处理技术。这些语音技术在采集语音特征,获取数字信号的同时,通常也可以起到降低信息冗余度的作用。像图像压缩领域里的JPEG一样,为获得更高的编码效率,大多数语音编码技术都允许一定程度的精度损失。而且,为了更好地用二进制数据存储或传送语音信号,这些语音编码技术在将语音信号转换为数字信息之后又总会用Huffman编码、算术编码等通用压缩算法进一步减少数据流中的冗余信息。

对于电脑和数字电器(如数码录音笔、数码随身听)中存储的普通音频信息,我们最常使用的压缩方法主要是MPEG系列中的音频压缩标准。例如,MPEG-1标准提供了Layer I、Layer II和Layer III共三种可选的音频压缩标准,MPEG-2又进一步引入了AAC(Advanced Audio Coding)音频压缩标准,MPEG-4标准中的音频部分则同时支持合成声音编码和自然声音编码等不同类型的应用。在这许多音频压缩标准中,声名最为显赫的恐怕要数MPEG-1 Layer III,也就是我们常说的MP3音频压缩标准了。从MP3播放器到MP3手机,从硬盘上堆积如山的MP3文件到Internet上版权纠纷不断的MP3下载,MP3早已超出了数据压缩技术的范畴,而成了一种时尚文化的象征了。

很显然,在多媒体信息日益成为主流信息形态的数字化时代里,数据压缩技术特别是专用于图像、音频、视频的数据压缩技术还有相当大的发展空间——毕竟,人们对信息数量和信息质量的追求是永无止境的。

5. 回到未来

从信息熵到算术编码,从犹太人到WinRAR,从JPEG到MP3,数据压缩技术的发展史就像是一个写满了“创新”、“挑战”、“突破”和“变革”的羊皮卷轴。也许,我们在这里不厌其烦地罗列年代、人物、标准和文献,其目的只是要告诉大家,前人的成果只不过是后人有望超越的目标而已,谁知道在未来的几年里,还会出现几个Shannon,几个Huffman呢?

谈到未来,我们还可以补充一些与数据压缩技术的发展趋势有关的话题。

1994,M.Burrows和D.J.Wheeler共同提出了一种全新的通用数据压缩算法。这种算法的核心思想是对

字符串轮转后得到的字符矩阵进行排序和变换,类似的变换算法被称为Burrows-Wheeler变换,简称BWT。与Ziv和Lempel另辟蹊径的做法如出一辙,Burrows和Wheeler设计的BWT算法与以往所有通用压缩算法的设计思路都迥然不同。如今,BWT算法在开放源码的压缩工具bzip中获得了巨大的成功,bzip对于文本文件的压缩效果要远好于使用IZ系列算法的工具软件。这至少可以表明,即便在日趋成熟的通用数据压缩领域,只要能在思路和技术上不断创新,我们仍然可以找到新的突破口。

分形压缩技术是图像压缩领域近几年来的一个热点。这一技术起源于B.Mandelbrot于1977年创建的分形几何学。M.Barnsley在20世纪80年代后期为分形压缩奠定了理论基础。从20世纪90年代开始,A.Jacquin等人陆续提出了许多实验性的分形压缩算法。今天,很多人相信,分形压缩是图像压缩领域里最有潜力的一种技术体系,但也有很多人对此不屑一顾。无论其前景如何,分形压缩技术的研究与发展都提示我们,在经过了几十年的高速发展之后,也许,我们需要一种新的理论,或是几种更有效的数学模型,以支撑和推动数据压缩技术继续向前跃进。

人工智能是另一个可能对数据压缩的未来产生重大影响的关键词。既然Shannon认为,信息能否被压缩以及能在多大程度上被压缩与信息的不确定性有直接关系,假设人工智能技术在某一天成熟起来,假设计算机可以像人一样根据已知的少量上下文猜测后续的信息,那么,将信息压缩到原大小的万分之一乃至十万分之一,恐怕就不再是天方夜谭了。

回顾历史之后,人们总喜欢畅想一下未来。但未来终究是未来,如果仅凭你我几句话就可以理清未来的技术发展趋势,那技术创新的工作岂不就索然无味了吗?依我说,未来并不重要,重要的是,赶快到Internet上下载几部大片,然后躺在沙发里,好好享受一下数据压缩为我们带来的无限快乐吧。

为什么是1、2、4、7、21?——浅谈MPEG标准的编号方式

▶编译/王咏刚

MPEG-1、MPEG-2、MPEG-4、MPEG-7还有MPEG-21,MPEG标准中的数字编号有规律吗?究竟有没有MPEG-3、MPEG-5、MPEG-6或类似的标准呢?有人说,未曾用过的编号是动态图像专家组(MPEG)专门预留的,今后可以更方便地补充的新的标准;也有人说3、5、6、8这些编号早已被用在了那些经讨论没有被通过的实验性标准上,这和美国空军的战机编号里有F-16、F-22却没有F-17和F-19的道理一模一样。

看上去,这两种说法都言之凿凿。但在事实面前,它们就不能自圆其说了。其实,MPEG标准的编号规则里并没有太多的玄机:MPEG-3早年夭折,MPEG-5和MPEG-6则根本不存在;历史上的偶然外加动态图像专家组里大多数成员的偏好造就出了1、2、4、7、21这样奇怪的编号序列。

起初,动态图像专家组把他们的第一个视频编码标准称为MPEG-1,然后又在其基础上开发出了面向机顶盒与DVD应用的MPEG-2标准。与此同时,专家们着手开发一种适用于数字电视(HDTV)的编码标准MPEG-3,但没过多久他们就发现,MPEG-2标准可以覆盖数字电视领域的应用。于是,MPEG-3被抛在一边,数字电视的编码标准被纳入了MPEG-2(顺便说一句,半途而废的MPEG-3和我们今天所说的MP3绝不是一回事,后者的真正含义是MPEG-1 Audio Layer 3)。

接下来,动态图像专家组开发出了新一代的视频压缩标准MPEG-4。到此为止,动态图像专家组内部的标准编号是顺序递增的。但在外人开来,因为MPEG-3没有发布,MPEG标准的编号成了2的指数序列1、2和4。那么,下一个标准,究竟该按顺序递增的编号方式,叫MPEG-5呢,还是依2的指数序列增长,叫MPEG-8呢?动态图像专家组开发多媒体元数据标准时,也为此大伤脑筋。最终,经过讨论,组员们决定放弃明显有序的编号方式:他们既不用5,也不用8,偏偏选择不成序列的MPEG-7作为新标准的名字。——有趣的是,尽管动态图像专家组试图避免人们从编号中看出某种规律,喜欢索隐附会的人还是忙不迭地声称:MPEG-7之所以用7来编号,是因为MPEG-7标准集前3个标准之大成,而1加2再加4不正好等于7吗?

MPEG-21是动态图像专家组在编号上标新立异的又一次尝试。据知情人士透露,之所以选用21这个与众不同的数字,主要是因为MPEG-21是一个异常新颖和独特的多媒体标准,它配得上这样"另类"的名字。

今后,MPEG的家族中还会出现什么样的编号呢?是35还是81?抑或是108?我们拭目以待。

声音与评论
(Voice and Comment)

▶评论人/一刀

任何一个笨蛋,只要有几个脑细胞,能够读懂手册,就可以拼凑出一个病毒。这又不是航天科技,我实在不知道做病毒到底什么地方需要“智商”?真正需要智商的地方是开发一个软件,一个别人愿意花钱来买的软件。

(人们往往认为开发电脑病毒的人都是高智商程序员。Robert C. Martin不知道是被什么人刺激了,在网上大发雷霆,痛斥病毒制造者。)

以前我不懂Web Service是什么,到处向人打听,结果没人能给我讲清楚;现在我算是基本明白了,可是却不知道怎么给人讲清楚。

(互联网时代的计算机技术名词,似乎越来越难懂,也越来越难表述了。)

我宁可使用Socket。

(还是鲍勃大叔Robert C. Martin,最近参加了一个小型讨论会,会上一个听众问,SOAP、RMI、CORBA、RPC及其它类似的消息中间件各自前途如何。与会者各抒己见,轮到Robert发言时,他说宁可用最基本的套接字,引起听众的热烈掌声。)

1964年,IBM的George Radin开发了PL/1语言,号称是Fortran,COBOL和ALGOL“最佳特性”的组合;1995年,SUN的James Gosling开发了Java语言,号称是C++和Smalltalk“最佳特性”的组合;2000年,Microsoft的Anders Hejlsberg开发了C#,号称是C++,Visual BASIC和Java“最佳特性”的组合。下一个语言又将组合哪些语言的“最佳特性”呢?我们什么时候才能看到一个新语言,象当年的C那样,由程序员而不是某个公司开发,用起来既有趣又令人兴奋,没有累赘,简洁易用,完整一致,又容易理解,什么时候才能看到这样一场变革呢?我们真的需要这样的变革吗?或者,我们真的就在Java和C#中乐不思蜀,甚至再也想象不出更好的语言了吗?

(德国著名的C++专家伴侣Klaus Kreft和Angelika Langer说出了很多计算机科学家的心声。今天的软件世界似乎越来越变成了大公司政治角逐和商业斗争的竞技场,难道曾经充满传奇的领域,再也没有英雄的长啸了吗?)

要防止孩子被电风扇的叶片打伤,你可以做两件事情:要么反复告诫他千万不要把手指伸到旋转的叶片里,要么把前面的风扇门关上。

(Ken Arnold,一位Java技术专家,当他提起C++和Java对待内存管理问题时举了这个有趣的例子。)

根据我的经验,一个根本性的观念,是导致一次重大进步的触发因素。我最近发现的一个具有根本性的观念就是:程序员也属于人类。有人可能对此表示怀疑,但我有明显的证据来证明这一点。虽然我也注意到程序员很少睡觉,而且因此而散发出恶臭气味,但是除了这些明显的动物特征之外,程序员有一个显著的人类特征——他们能够使用语言,而这被广泛认为是人类独有的特征之一。

(仍然来自Ken Arnold,他最近发现了“程序员也属于人类”这一事实,并因此意识到由于长期忽略这一事实,我们的编程语言和工具已经满是非人类的气味。)

那些只是原则,原则就是等着被破坏的。

(本刊编辑部记者阎辉,在某天中午大放厥词。虽然这句话并非他首创,但是暂且算在他的头上吧。)

然而终究还是有一些原则是与众不同的,一些在你心中响起的微弱声音,通过天使的喇叭传来的内心的裁决:“I WILL NOT SHIP SHIT”。

(再引一段鲍勃大叔的话,算是一位有良心的软件开发者对阎辉同志的响亮回答。)

继承层次在很多设计范例中已经成为一个重要的抽象工具。用好了,它确实可以体现出组织的力量。可是如果企图借助它来摆脱基本复杂性,则它就会带来不幸。遗憾的是,在大多数面向对象设计中,大部分的抽象都是糟糕的。它成了一个借口,我们用它来逃避那些我们本可以解决却不愿意去面对的问题。

(面向对象和设计模式先驱任务James Coplien似乎也对今天的面向对象应用深感不满。我们不禁要问,是这些前辈思想过时了,还是我们陷入了庸俗?)

程序人生

▶编译/韩磊

中学 - - - - - - - - - - - - - (Basic)

```
10 PRINT "HELLO WORLD"
20 END
```

大学一年级 - - - - - - - - - (Pascal)

```
program Hello(input, output)
  begin
    writeln('Hello World')
  end.
```

大学高年级 - - - - - - - - (Lisp)

```
(defun hello
  (print
    (cons 'Hello (list 'World))))
```

进入职场 - - - - - - - - - - (C)

```
#include <stdio.h>
void main(void)
{
  char *message[] = {"Hello ", "World"};
  int i;

  for(i = 0; i < 2; ++i)
    printf("%s", message[i]);
  printf("\n");
}
```

专业开发者 - - - - - - - - (C++)

```
#include <iostream.h>
#include <string.h>

class string
{
private:
  int size;
  char *ptr;

public:
  string() : size(0), ptr(new char('\0')) {}
  string(const string &s) : size(s.size)
  {
    ptr = new char[size + 1];
    strcpy(ptr, s.ptr);
  }

  ~string()
  {
    delete [] ptr;
  }

  friend ostream &operator <<(ostream &, const string &);
  string &operator=(const char *);
};

ostream &operator<<(ostream &stream, const string &s)
{
  return(stream << s.ptr);
}
string &string::operator=(const char *chrs)
{
  if (this != &chrs)
  {
    delete [] ptr;
    size = strlen(chrs);
    ptr = new char[size + 1];
    strcpy(ptr, chrs);
  }
  return(*this);
}
int main()
{
  string str;

  str = "Hello World";
  cout << str << endl;

  return(0);
}
```

编程大师 - - - - - - - - - - (COM)

```
[
uuid(2573F8F4-CFEE-101A-9A9F-00AA00342820)
]
library LHello
{
    //引入主库文件
    importlib("actimp.tlb");
    importlib("actexp.tlb");

    //引入接口
    #include "pshlo.idl"

    [
    uuid(2573F8F5-CFEE-101A-9A9F-00AA00342820)
    ]
    cotype THello
  {
  interface IHello;
  interface IPersistFile;
  };
```

```
};
[
exe,
uuid(2573F890-CFEE-101A-9A9F-00AA00342820)
]
module CHelloLib
{
    //引入各种头文件
    importheader(<windows.h>);
    importheader(<ole2.h>);
    importheader(<except.hxx>);
    importheader("pshlo.h");
    importheader("shlo.hxx");
    importheader("mycls.hxx");

    //引入类库
    importlib("actimp.tlb");
    importlib("actexp.tlb");
    importlib("thlo.tlb");

     [
    uuid(2573F891-CFEE-101A-9A9F-00AA00342820),
    aggregatable
     ]
    coclass CHello
 {
 cotype THello;
 };
};

#include "ipfix.hxx"
extern HANDLE hEvent;
class CHello : public CHelloBase
{
public:
    IPFIX(CLSID_CHello);

    CHello(IUnknown *pUnk);
    ~CHello();

    HRESULT  __stdcall PrintSz(LPWSTR pwszString);

private:
    static int cObjRef;
};

#include <windows.h>
#include <ole2.h>
#include <stdio.h>
#include <stdlib.h>
#include "thlo.h"
#include "pshlo.h"
#include "shlo.hxx"
#include "mycls.hxx"

int CHello::cObjRef = 0;

CHello::CHello(IUnknown *pUnk) : CHelloBase(pUnk)
{
    cObjRef++;
    return;
}

HRESULT  __stdcall  CHello::PrintSz(LPWSTR pwszString)
{
    printf("%ws\n", pwszString);
    return(ResultFromScode(S_OK));
}
 CHello::~CHello(void)
 {

 //对象计数归零,停止服务
 cObjRef--;
 if( cObjRef == 0 )
    PulseEvent(hEvent);
 return;
 }

 #include <windows.h>
 #include <ole2.h>
 #include "pshlo.h"
 #include "shlo.hxx"
 #include "mycls.hxx"

 HANDLE hEvent;

  int _cdecl main(
 int argc,
 char * argv[]
 ) {
 ULONG ulRef;
 DWORD dwRegistration;
 CHelloCF *pCF = new CHelloCF();

 hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

  //初始化OLE库
 CoInitializeEx(NULL, COINIT_MULTITHREADED);

 CoRegisterClassObject(CLSID_CHello, pCF, CLSCTX_LOCAL_SERVER,
     REGCLS_MULTIPLEUSE, &dwRegistration);

 //等待事件停止
 WaitForSingleObject(hEvent, INFINITE);

 //清除类对象
 CoRevokeClassObject(dwRegistration);
 ulRef = pCF->Release();

  //通知OLE
 CoUninitialize();
  return(0); }
  extern CLSID CLSID_CHello;
  extern UUID LIBID_CHelloLib;

  CLSID CLSID_CHello = { /* 2573F891-CFEE-101A-9A9F-
00AA00342820 */
     0x2573F891,
     0xCFEE,
     0x101A,
     { 0x9A, 0x9F, 0x00, 0xAA, 0x00, 0x34, 0x28, 0x20 }
  };
  UUID LIBID_CHelloLib = { /* 2573F890-CFEE-101A-9A9F-
00AA00342820 */
```

```
        0x2573F890,
        0xCFEE,
        0x101A,
        { 0x9A, 0x9F, 0x00, 0xAA, 0x00, 0x34, 0x28, 0x20 }
    };
    #include <windows.h>
    #include <ole2.h>
    #include <stdlib.h>
    #include <string.h>
    #include <stdio.h>
    #include "pshlo.h"
    #include "shlo.hxx"
    #include "clsid.h"
    int _cdecl main(
    int argc,
    char * argv[]
    ) {
    HRESULT  hRslt;
    IHello        *pHello;
    ULONG  ulCnt;
    IMoniker * pmk;
    WCHAR  wcsT[_MAX_PATH];
    WCHAR  wcsPath[2 * _MAX_PATH];
    //取得对象路径
    wcsPath[0] = '\0';
    wcsT[0] = '\0';
    if( argc > 1) {
        mbstowcs(wcsPath, argv[1], strlen(argv[1]) + 1);
        wcsupr(wcsPath);
         }
   else
         {
        fprintf(stderr, "Object path must be specified\n");
        return(1);
         }
    //取得打印串
    if(argc > 2)
        mbstowcs(wcsT, argv[2], strlen(argv[2]) + 1);
    else
        wcscpy(wcsT, L"Hello World");
    printf("Linking to object %ws\n", wcsPath);
    printf("Text String %ws\n", wcsT);
     //初始化OLE库
    hRslt = CoInitializeEx(NULL, COINIT_MULTITHREADED);
    if(SUCCEEDED(hRslt)) {
        hRslt = CreateFileMoniker(wcsPath, &pmk);
    if(SUCCEEDED(hRslt))
     hRslt = BindMoniker(pmk, 0, IID_IHello, (void **)
&pHello);
         if(SUCCEEDED(hRslt)) {
      //打印字符串
     pHello->PrintSz(wcsT);
     Sleep(2000);
     ulCnt = pHello->Release();
      }
         else
     printf("Failure to connect, status: %lx", hRslt);
         //通知OLE
        CoUninitialize();
         }
    return(0);
    }
```

初级黑客 - - - - - - - - - - - - (Perl)

```
#!/usr/local/bin/perl
[redacted]
1;
```

中级黑客 - - - - - - - - - - - - (C)

```
#include <stdio.h>
[redacted]
main(){exit(printf(S) == strlen(S) ? 0 : 1);}
```

高级黑客 - - - - - - - - - - - - (gcc)

```
% cc -o a.out ~/src/misc/hw/hw.c
 % a.out
```

大师级黑客 - - - - - - - - - - - - (bash)

```
% echo "Hello, world."
```

新经理 - - - - - - - - - - - - - - (Basic)

```
10 PRINT "HELLO WORLD"
20 END
```

中级经理

```
mail -s "Hello, world." bob@b12
[redacted]
^D
```

高级经理

```
[redacted]
I need a "Hello, world." program by this afternoon.
```

首席执行官

```
% letter
[redacted]
% logout
```

Typedef一次搞掂

▶撰文/Dr. CSDN

Dr. CSDN,就是本人,从克莱登大学获得博士,算不上资深程序员,入职也就是两三年的光景,才学虽不敏,倒也有点子小聪明。诸位有什么编程方面的问题,不才愿意动动脑筋,替各位分忧。初次跟大家见面,先送上一份见面礼,请大家笑纳。

一个工作了七八年的老大哥给我来了一封邮件:

“嗨,我知道你小子基础还不错,你给我看看,这行代码究竟是什么意思:

```
typedef char* (*PT)(char*, size_t);
```

郁闷中...这typedef是怎么用的?”

我给他回信:

“这行代码声明了一个类型PT,这个类型是一个函数指针,指向一个函数,该函数接收两个参数,第一个是char*,第二个是size_t,函数返回一个char*指针——够累的,怎么样,老大明白了吧。”

老大满意而去,我却陷入沉思。

照说这个问题在C语言中算是基础问题,可是老大工作了七八年,还是没搞明白,可见C语言的声明语法确实是一大难点。还记得以前读过C++之父Bjarne Stroustrup的文章,他就曾经说,为了与C兼容,C++不得不接受了C语言那丑陋的声明语法,那语气好像是被迫咽下了一口变质牛奶。在很多平台下有一个工具叫cdecl,可以把C语言的声明语句翻译成英语。初学C的人用用它,倒无伤大雅。若是搞了好几年还是离不开cdecl,那可就贻笑大方了。C的声明语法自然是有章可循的,其实经过一些训练,对于一般的声明语句还是不难理解的。可是很多人一看到typedef就茫然了。比如:

```
void (*pf)(void);
```

很多人都知道,这行代码是说:pf是一个指针,指向一个函数,该函数没有参数,没有返回值。可是:

```
typedef void (*pf)(void);
```

是什么?

```
typedef int* (*pt)(int* (*pn)(int* p1, int* p2), int * p3);
```

又是什么呢?

其实typedef的用法十分简单,只不过通常的C语言教材里对这个机制的介绍,实在是太幼稚,太简单了。我们一般是从这样的例子里学习typedef基本用法的:

```
typedef unsigned int UINT;
```

老师会告诉我们,此声明之后,UINT就与unsigned int同义。

通常我们在学习typedef之前,都会先接触到#define,所以我对typedef的第一个印象是:它是#define的反格式。因为

```
#define UINT unsigned int
```

和

```
typedef unsigned int UINT;
```

等价(这话要打大大的疑问,不过大家就稍微马虎马虎好了)。所以以前每次我想用typedef的时候,就会先在心里把#define语句写出来,然后翻转一下次序就可以了。因此,在我看来,typedef不过是#define的一件漂亮的花外衣

于是我抱着这种理解快乐地生活了很久,直到我第一次遇到下面的代码:

```
typedef void (*pf)(void);
```

我“#define外衣论”在一瞬间崩溃。无论我怎么用#define去拼装组合,都无法解释这行代码究竟说的是什么。

搞明白这个问题花了我很长时间,于是我希望自己能够真正理解typedef,下次遇到类似问题的时候不至于再花时间。钻研了一些文档之后,我找到了typedef的理解方法,此后,对于typedef我可以说是逢山开路,遇水架桥,再也没有出过差错。

理解typedef的方法是:

1. 确定被声明的类型:遇到typedef声明语句时,

从左至右进行扫描,找到第一个生词,这个生词就是该语句所声明的类型名称。例如:

[redacted]

中,如果pt是个生词(既不是保留字,也不是曾经被声明过的符号),则真正被声明的类型是pt,而不是其它。其它的名字全部是为了阅读方便的占位符,可有可无。也就是说,上面的声明式与:

[redacted]

完全等价。

2. 之后一旦遇到用该类型声明的变量,则:在该类型的typedef式中用变量代替类型,去掉typedef关键字,所得到的声明式等价与原来的声明。例如:

[redacted]

这个声明式,可以经过两步变换为等价的声明式。首先,回到pt的typedef式:

[redacted]

用p替代pt:

[redacted]

然后把typedef去掉,得到:

[redacted]

这个语句与

[redacted]

意义相同。

如果你的基础扎实,就应该知道,这个声明的意思是:

p是一个函数指针,指向一个函数,该函数有两个参数,第一个参数也是一个函数指针,所指向的那个函数有两个int*参数,返回一个int*值,第二个参数是int*,整个函数返回一个int*。

这样以来,我们就知道,原来那个typedef声明了这样一个指针的类型。由于typedef的运用,不仅使这类指针的声明变得非常简洁,而且有些地方如果不用typedef,写声明式的过程会成为一场恶梦。比如:

```
typedef std::basic_string<char> String;
class A;
....
typedef int (A::*PAMF)(int, String);
typedef std::map<String, PAMF> CallingTable_T;
CallingTable_T table;
```

如果直接声明table,会是这样的(请深吸一口气):

```
std::map<std::basic_string<char>, int (A::*)(int,
basic_string<char>)> table;
```

看得懂的请举手,嘿嘿。

(上接89页)

7、适当的注释:对于代码进行注释也是非常必要的,什么地方加上注释,什么地方不必进行注释,每个程序员都有自己的习惯和原则。我们可以看到,Jeffrey的注释还是十分恰当的,他只注释了几个关键的地方,对于一目了然、可以望文生义的语句,则一律没有注释。

8、一致的代码风格,如变量命名规则、代码缩进方式、{使用的位置,空格、空行的使用,这就不必多说了吧。

9、灵活使用宏简化代码:Windows是消息驱动机制,而一个消息只有两个参数:LPARAM和WPARAM。这两个参数都是无符号的整型变量,显然在相当多的情况下他们都不能满足要求,因为许多消息需要传递更多的参数信息,所以不得不用这两个参数传递指向参数结构的地址。于是,我们不得不采用变通的方法,即把传送参数时,把原本的数据类型都强制转换到整形,而处理消息的函数中,再又强制把参数转换回来。所以,编写Windows程序时不得不进行大量的数据类型的转换操作。为了方便程序员处理这些操作,Windows提供了一个专门处理这一类转换操作的头文件,即WindowsX.h。WindowsX.h的宏分为三组:消息分流器、子控件宏和API宏,灵活使用这些宏,可以大大地简化代码,而且使得代码的可读性更好、往往易于理解、也更易用。但是必须承认,我个人很少使用这个头文件,甚至在大多数情况下忽略这个文件的存在。我也发现,其实还有许多人像我一样忽略了它。但是从上面的代码中可以看出,Jeffrey无疑非常了解这些宏,也非常善于运用这些宏。我们不妨试试,如果不使用chHANDLE_DLGMSG这些宏,代码的复杂性将会增加多少!

1 见古龙的《浣花洗剑录》。

2 为了加以说明,我在其中添加了一些中文注释。

.NET网络资源库

我们从ccboy的站点(http://www.dotnettools.org/dotnettools.htm)中精选了一些.NET网上资源介绍给大家。编者一一访问过下面的站点,并确认值得推荐。如果你的收藏夹中有其它好的站点,不妨告诉我们,大家一起来分享。

Framework类

.NET Framework SDK

http://www.microsoft.com/downloads/details.aspx?displaylang=zh-cn&FamilyID=9B3A2CA6-3647-4070-9F41-A333C6B9181D

这个不用解释了。我刚检查过地址,但并不保证未来继续有效。

ASP.NET Start Kits

http://www.asp.net/Default.aspx?tabindex=9&tabid=47

五个简单的ASP.NET应用程序,帮你入门。

MONO

http://www.go-mono.com/

在Linux上的开源.NET开发执行环境。

CsLex

http://www.cybercom.net/~zbrad/DotNet/Lex/Lex.htm

C#语义分析生成器。

XGen

http://www.codeworks.nl/Xml/XGen/

用C#实现的开源项目,用于从各种不同数据源生成统一的XML文档。

Remoting类

RemotingFAQ

http://www.ingorammer.com/RemotingFAQ/

相当完整的Remoting常见问题集。

Remoting.Corba

http://remoting-corba.sourceforge.net/

旨在实现集成Corba和Remoting的开源项目,刚开始进行,值得关注。

图形图像/游戏类

ScPI

http://www.netcontrols.org/scpl/

免费的.NET图表、图形绘制库。

OpenGL for .NET

http://www.randyridge.com/Releases/Tao.OpenGl.0.1.Windows.zip

作者最近正在找工作,或许读者能帮帮他?

DbDiagrams

http://www.carlosag.net/Tools/DbDiagrams/Default.aspx

WebMatrix的插件,用来绘制数据库关系图、工作流程图等很不错。

GapiDraw.NET

http://www.intuitex.com/gapidraw.html

免费的Pocket PC图形库,封装了GapiDraw。

GeoTools.NET

http://geotoolsnet.sourceforge.net/Default.html

实现了OpenGIS规范的.NET类。安装时要小心,因为它还需要一些其它库的支持。

设计模式类

C#设计模式

http://www.dofactory.com/Patterns/Patterns.aspx

每个设计模式都提供了C#的范例代码。

工具/杂锦类

FormBuilder.NET

http://www.xmlforasp.net/Formbuilder.net.aspx

在IE中创建ASP.NET窗体的应用。

RegExDesigner

http://www.sellsbrothers.com/tools/#regexd

用可试化方式设计和测试正则表达式。

ASP.NET Source Projects

http://www.asp.net/Default.aspx?tabindex=7&tabid=41

微软ASP.NET开发组提供的范例程序代码。

投稿信箱:tougao@csdn.net

应试绝招，帮你轻松过软考

▶撰文/徐锋

谈到软件资格和水平考试(以下简软考),总能引起我很多思考与回忆,自从1992年接触电脑以来,软考就成了我不断挑战自己、检验与完善自己的一个试金石。从1993年通过初级程序员到2002年通过系统分析员,真可谓"十年磨一剑",软考在我的发展历程中扮演了十分重要的角色,它为我理清了学习的思路、找到了发展的方向、督促着自己每一天向前进。

作为一场以考代评的职称考试,又得到了国际上的广泛认可,这都使软考参加者越来越多,用人单位越来越重视。在这一年一度的软考即将来临的时候,笔者结合自己的复习与参考的经验,和大家交流一下如何更充分地准备高级程序员(系统设计师)和系统分析员这两个级别的考试,希望对大家通过这场考试有所帮助。

上午考试——基础知识考察

高级程序员和系统分析员两个级别的上午考试的范围、形式基本一致。都是75个选择题(通常会将几个选择题组合成为一题)。考试时间都相当充足,均提供了150分钟。考试的内容主要包括计算机软件知识、硬件知识,数学以及计算机英语。

专业知识

不过这两级考试中,对专业知识的考察原则还是有很大不同的:

高级程序员上午考试的原则是"广度优先",主要要求应试者能够对各种不同的知识都有一些了解,出题的方向通常是涉及面很广,但每个点的深度不会太大。

#ziplib
http://www.icsharpcode.net/OpenSource/SharpZipLib/
用C#实现Zip、GZip、BZip2和Tar的.NET库。

资源/资料类

Microsoft Research
http://www.research.microsoft.com/research/downloads/
微软尖端技术站点。
MSDN Visual Studio.NET
http://msdn.microsoft.com/vstudio/
MSDN中关于VSStudio的部分。
CodeProject
http://www.codeproject.com/
有大量高质量的文章和代码。
ASP.NET
http://www.asp.net
ASP.NET大本营,做ASP.NET WEB应用的必去之地。
OnDotNet
http://www.ondotnet.com/
O'Reilly公司的.NET资源站点(你不会不知道该公司著名的坚果系列图书吧?)。
C# Journal
http://www.csharpjournal.net/
关于C#、.NET的新闻、技术文章。
.NET Wire
http://www.dotnetwire.com/
设计简洁明了的技术文章站,正在庆祝三周岁生日!
Weblogs at ASP.NET
http://weblogs.asp.net/
不仔细找还真不知道ASP.NET有自己的Blog,不过我没找到RSS源。
GotDotNet
http://www.gotdotnet.com/
.NET Framework社区,热闹得很。
DotNetJunkies
http://www.dotnetjunkies.com/
许多见解独到的文章。

系统分析员上午考试的原则是“深度优先”,要求应该者不仅拥有广博的领域知识,还需要能够深入地掌握这些技术。因此,应试者在复习时,需要“大字小字都要读”,需要建立扎实的基本功才能够应对。

英语和数学

在75道选择题中,数学与英语的考题就占了25分,鉴于各年度通过的分数线都是45-50分左右,而且数学与英语的考题通常难度也并不是太大,因此对于想通过这场考试的朋友,一定要重视数学与英语的复习。

在这两个级别的考试中,英语试题通常都是计算机时文,考察点通常在于专业名词和时态。数学则主要包括离散数学、高等数学、线性代数三个方面。

应试指南

对于专业知识方面,在考前应加强计算机报刊、杂志的阅读,笔者推荐大家着重于《计算机世界》、《中国计算机报》、《微电脑世界》、《程序员》4本杂志,重点在于了解新技术、行业新动向。对于报考高级程序员考试的,应该着重梳理,建立良好的知识框架结构;而对于系统分析员的应试者,则应该力求深入,一方面梳理自己的强项,另一方面针对热点知识与自己薄弱方面的交集下苦工。

对于英语方面,建议应试者在考前一个月,坚持每天阅读一篇计算机时文,语法知识薄弱者应对时态进行一个总结性的复习。因为,对于已经通过英语4级以上考试的应试者而言,丢一分都可惜,因为这部分实在不能够算难。

而对于数学方面,建议应试者在考前半个月前,进行粗略性复习:

1) 离散数学:通过做习题的方式,对概率论、集合论、代数系统、数理逻辑、自动机、图论建立初步的印象;

2) 高等数学:将知识点扫描一遍,将公式整理在一张纸上,针对性记忆一下,重点在于导数、微分;

3) 线性代数:重点是公式整理,针对性记忆。

通过以上的努力,15天的复习与练习相信一定能够让你在数学题上拿到至少10分。

高程下午考试——程序设计

高级程序员下午考试的考察重点在于数据结构与算法!而不在于语言本身。

不得不说

在可视化编程的今天,VC、VB、PB都被误认为“语言”,因此许多“编程高手”无一例外地栽倒在下午考试中,然后也引发了“软考无用,过时”的言论。其实不然,一个伟大的程序员应该能够精通数据结构、算法,比尔·盖次曾经说过“如果你读懂了《计算机程序设计艺术》,就到微软来报到吧!”,可见其对算法与数据结构之重视。

考试形式

高级程序员下午试题主要是程序填空题,也就是指定程序描述,给出了大部分程序,但挖去部分语句,甚至是一个语句中的关键部分,让你补充完整。其中包括一道必须的汇编题和3道高级语言(通常是C++,还有一种可选语言)。

应试技巧

考试之前,应该重点放在数据结构与算法的复习上。最好的复习方法就是找一本你最喜欢的“数据结构与算法”的书籍,但一定要有习题哦!先认真地阅读例子,然后上机编程,完成习题,实践才是学习编程的最好方法。复习的重点应该放在:链表、树和排序算法方面,这些都是高级程序员考试要考察的重中之重。

另外,由于现在许多程序员都几乎没有接触过汇编语言,所以总是感到十分恐惧,其实高级程序员考试中使用的是一种虚拟的汇编语言——CASL,其语法十分简单,如果你能够花一个月的时间来复习,答对这道题是十分简单的。软考指定教材中有一本专门讲述CASL的,对该语言的描述十分到位,也很简单,认真地阅读教材,然后下载一个CASL的模拟器,实际做几道习题,相信胜利一定属于你。

因为下午的题量明显大于上午,在考试的过程中,一定要合理地分配时间。在答题之前,应认真的审题,理解程序描述,然后认真地阅读程序,找到程序所采用的数据结构、算法、编程思路,这样找到答案就不是什么难事了。

在这里,很容易出现的一个误区就是,阅读完程序描述后,不自觉地浮现出了一个解决思路,这一思路却往往与题目中给定的程序所采用的思路完全不同,这样

在解题的时候就陷入了自己设下的条框,无法自拔,从而“费九牛二虎之力,却始终无门而入”。

系分下午考试——炼狱

系统分析员下午的考试安排,真可谓是“残酷”:1点半就开考,90分钟要完成3大题系统分析题,每题都包含3个小题,中间只休息20分钟,就要在150分钟之内完成3000字以上的论文。因此,我经常看到有人“临阵脱逃”。

考试时间紧凑、内容又很新、强调实践,这么残酷的炼狱,没有真才实学想蒙混过关着实不易,这也是本场考试开考10余年,全国通过者为数不多的原因。因此在这部分内容中别期望我能够救人于苦海,我最多只能够帮助你“锦上添花”,增加一些成功的机会罢了。

下午第一场

下午一是系统分析题,一道必答题,再从4道可选题中选答2道,共计3道。每个问题中包括2-3个子题。题目主要在开发技术分析、软件工程管理、成本/效益分析等方面,综合性强、难度大、时间短,经常会让你感觉到好像也能答上一些,但总答不好的感觉。根据我的经验,要想在这场考试中取得好成绩,需要过好以下几关:

1) 选题关

下午一的试题,题面都很长,因此很多应试者在选择题目时经常犹豫不决,这样会浪费大量时间。根据我的经验,由于总的考试时间不长,应该不要在这个阶段浪费太多的时间。

我的建议是,在阅卷时间里,就应该全面地扫视一下后面4题,由于每个题目都会涉及到不同的领域,结合你熟悉的领域、问题的难度,快速地做出选择。

2) 审题关

正式开始答卷时,应该认真地审题,在阅读题目时,可以圈出重点的词、句,认真地分析题目描述的内容,找出与问题相关联的部分,不要过早地回答问题。很多应试者容易在时间的压力之下,草率地答题,以致对题目理解不够、遗漏要点。

3) 答题关

在下午一考试中,对答题的字数有限制,因此需要你对回答的内容进行高度的概括与浓缩,抓住要点。因此,建议大家在审题时,将过渡性的答案写在草纸上,圈出回答的要点,然后认真地组织文字。不要忘了,在批改时,其评分原则是根据你点出的要点数做出的。

下午第二场

系统分析员考试的最后一关,就是论文了。论文的形式是实践手记,考卷将会给出4个主题,由你任选一个主题撰写。根据我的经验,在论文的考试中,应该注意:

1) 选题要慎重

时间很紧,因此一定要选择自己最熟悉的主题,建议在阅卷时完成选择。

2) 形式很关键

我第一次冲击系统分析员时,就在这里落马,系统分析员考试十分注重对应试者实践经验的考察。体现在论文这场考试中,就落实在文体上,千万不要写学术论文,一定要根据其给出的写作格式:你承担了什么项目、担任什么角色?在这个过程中你采用了什么措施、收到了什么效果?在这个过程中有什么不足、下次想如何改进它?

这接近八股文的形式,就是告诉你一定摆事实、讲道理,这不是议论文,而应该是记叙文。其实如果大家在平时养成了写“项目总结”的习惯,这会对你应对这场考试有很大的帮助。在文章的分配上,建议第一部分600-800字,第二部分1500-1800字,第三部分600-900字。

一定要让人感觉到你的内容是真实的,而项目的选择方面也不要过于忌讳其规模的大小,而应该重视其内容真实、实在。

3) 不要打草稿

时间不允许你这样做,应该先拟提纲,整理写作思绪,在这个方面分配30分钟左右是值得的。

4) 写好摘要

摘要是很重要的内容,我建议大家最后写。因为是一笔成文,在写作的过程中,总会有一些临时的变化,因此,待到全文完成后,总结出摘要是更理性的,一般只需留下10-20分就可以了。

小结

高级程序员、系统分析员考试的要求比较严格,对应试者的总体要求较高,应试者蒙混过关的几率很小,因此大家应该本着检验自己的态度备考,毕竟在应试中提高才是最重要的。“充分准备、沉着应试、重在参与”永远是最好的应试态度,预祝各位在2003年软考中取得满意的好成绩。

meng@csdn.net

Amone信箱

Amone 絮语

《CSDN 开发高手》的新鲜出炉凝聚了《程序员》编辑们大半年的心血和汗水，从年初的架构、3 月中的全面征稿到今天与读者见面，我们的心一直忐忑不安——都是为了你们心里的两个字“还行”。我们深知读者是检验《CSDN 开发高手》的唯一标准，她的定位、她的内容、她的发行都需要你们的真知灼见。同时，读者也是我们办好《CSDN 开发高手》的财富……。本期是amone 自己唱独角戏，就杂志的定位、内容、发行作一些描述，希望能伴你读完《CSDN 开发高手》的每一页。

编读互动

- 为什么要办一本纯技术的刊物？

很多读者在看完每期《程序员》后，都觉得技术部分不“过瘾”(实质上《程序员》每期都有50 页以上的精彩技术文章，非常厚实了)。因此，在改半月刊时，杂志社就顺应了读者呼声，将下半月定为纯技术刊物。

- 《CSDN 开发高手》和《程序员》的关系？

《CSDN 开发高手》和《程序员》的关系非常简单——就是《程序员》的半月刊。定位方面，在“致读者的一封公开信”中已阐述清楚了。为了方便读者的选择（当然更希望你们两本都喜欢），杂志社专门从邮局申请了2 个独立的邮发代号。其中，《程序员》为上半月发行，《CSDN 开发高手》是下半月出刊，我们的口号“上半月读《程序员》，下半月用《CSDN 开发高手》”。

meng@csdn.net

Amone 信箱

编读互动

- 为什么叫《CSDN 开发高手》？

编辑们决不敢以高手自居，编辑的初衷是帮助读者学好、用好技术，成为为中国软件崛起而奋斗的开发高手。如果读者对此刊名不满意，我们会虚心接受，不过改名需要一定时间……

- 怎样购买《CSDN 开发高手》？

读者服务

我们在全国大中城市都有经销商，一般在每月的15 号左右，读者会在当地的书报摊见到《CSDN 开发高手》。如有购买不到的地方，请及时与我们联系。

联系电话：010-51661202 转 209，电子邮箱：caili@csdn.net

- 怎样邮购《CSDN 开发高手》？

为方便购买，读者也可以直接汇款到杂志社邮购《CSDN 开发高手》，每期8.8 元（含邮资）、一年12 期共计105 元（含邮资）。更多优惠请关注CSDN 网站《程序员》、《CSDN 开发高手》的征订活动。

咨询电话：010-51661202 转 279　　电子邮箱：bcm@csdn.net

邮购：北京市朝阳区北三环东路8 号静安中心 26 层　100028

收款人：程序员读者服务部

银行转账：　　户名：北京百联美达美数码科技有限公司

账号：042101040004505

开户行：农行北京朝阳支行展览中心分理处　　网上购买：CSDN 网站商城

- 怎样订阅《CSDN 开发高手》？

《CSDN 开发高手》可直接到当地邮局订阅，破季也可。

订阅代号：2-423

单价10 元 / 季价30 元 / 半年价60 元 / 全年价120 元