

Programmer 程序员

2004

适合开发者 项目经理 CTO&CIO 编程爱好者阅读

C++/CLI

凤凰的涅槃

“C++/CLI全景体验”开篇语

Stan Lippman谈C++/CLI

C++/CLI调查报告

C++/CLI会冲击C#吗?

C++/CLI: 鼎新革故

标准C++及C++/CLI发展综述

非典型C++/CLI教程

P26

迫在眉睫的
职业规划

P36 独行五年 开发交互图形控件

P42 迎接UML2.0

P92 了解Java规则引擎

P97 探讨与比较Java和.NET的事件处理框架

www.csdn.net

ISSN 1672-3252

Csdn!

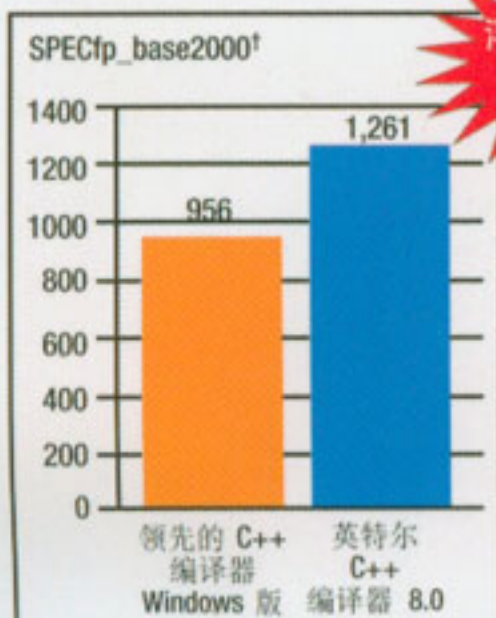


邮发代号: 2-665 定价: 10元

TLFeBOOK

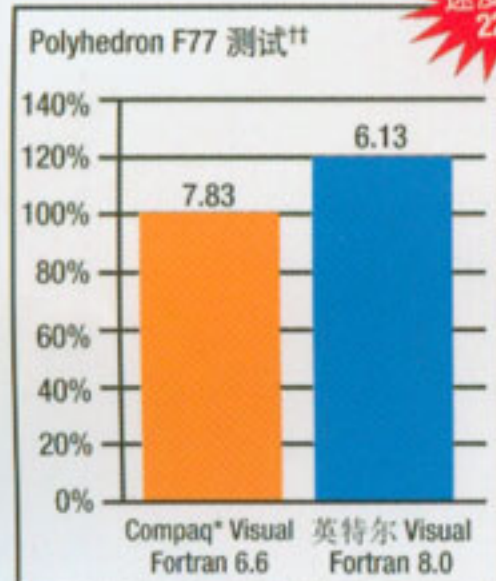
英特尔® C++ 与 Visual Fortran 8.1 编译器

最新版快速编译器 现已隆重推出!



浮点性能
提高达
24%!

† 英特尔® 奔腾® 4 处理器, 3.2 GHz, 512 KB 二级缓存, 256 MB 内存, Windows XP Professional



速度提升
22%!

†† 英特尔® 奔腾® 4, 1.8 GHz, 256 MB 内存, Microsoft Windows 2000
详见 www.polyhedron.com

性能

- 只需对源代码稍作修改甚至原封不动便能大幅提升应用程序性能
- 在英特尔体系结构上具有出众的性能。
(基于英特尔® 奔腾® 4、英特尔® 至强™ 及英特尔® 安腾® 2 处理器)

兼容性

- 两者均可与 Microsoft Visual Studio*.NET 集成
- C++ →**
- 与 Microsoft Visual C++*.NET 保持着源代码与二进制代码方面的兼容性
- Fortran →**
- 与 Compaq* Visual Fortran 保持着良好的兼容性

支持

- 附赠一年产品升级与“英特尔® 卓越支持”
- 可通过“英特尔® 软件学院”提供培训

“我们试用了‘英特尔® C++ 编译器 Windows 版’.....(该编译器)生成的代码令人印象深刻, 与我们‘集成开发环境’的兼容性配合之好令人惊讶, 同时还大幅提高了性能.....对于 C++ 开发人员, 如果要在基于英特尔处理器且运行 Windows 的系统上构建应用程序, 我们建议您使用这款产品。”

—Mike Marchywka
EyeWonder, Inc. 公司首席科学家



英特尔® VTune™ 可视化性能分析器

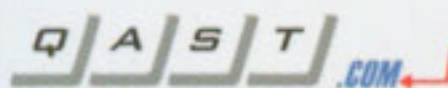
通过收集、分析, 显示各种性能数据, 迅速发现软件性能瓶颈。



英特尔® 集群数学内核库 7.0

为工程、科研计算提供了高度优化数学函数库, 集群数学函数库还为 Linux 集群新增了 ScaLAPACK

缩短开发周期 提高应用程序性能



比特瑞网集团软件事业部
010-6257-9990 (北京)
021-6211-3955 (上海)



上海亿盟信息技术有限公司
021-5352-2355 (上海)
021-5396-4504 (上海)

英特尔软件开发工具产品咨询热线: 021-52574545-3798

提供免费试用版: xlsoft.com/cn/intel

经销商: **XLISOFT**

© 2004 英特尔公司版权所有。英特尔、英特尔徽标、奔腾、安腾、英特尔至强及 VTune 均为英特尔公司或其子公司在美国以及其它国家或地区的商标或注册商标。
*其它品名与品牌可能是属于其它公司的资产。



选硬件，不用左右为难。

无论使用何种硬件，VERITAS都能帮助您降低存储成本。您不必再为硬件选择而左右为难。专为效用计算而设计。想了解更多，请访问www.veritas.com/cn


VERITAS™

©2004 VERITAS Software Corporation版权所有。VERITAS与VERITAS标识以及NetBackup是VERITAS Software Corporation或其在
在美国及其它国家合作伙伴的商标或注册商标。其它名称可能是各自所有者的商标。



China SoftCon 04 中国软件技术大会

参与，就有收获

咨询热线: 010-62610684 / 82623146 / 62539143 / 62538720

Http://www.softcon.cn Email: service@softcon.cn

- 在这里，不会有顾此失彼的担心，更不会有偏听偏信的忧虑；
- 在这里，不会有望梅止渴的虚望，而是会有感同身受的平实和共鸣；
- 在这里，不会有隔靴搔痒的无奈，而是会有酣畅淋漓、茅塞顿开的快意。
- 这里不会全是赞歌，还会有很多争鸣；
- 这里不会全是成功的经验，还会有很多失败的教训；
- 这里不会只描绘美好的前景，还会揭示可能的陷阱。

这里就是中国软件业的年度压轴盛会——中国软件技术大会！

本届大会的看点：

- 60人、60场高水准技术讲座、研讨：德高望重的专家指点江山，精于实践的中青年技术骨干传道解惑，内容涵盖应用实践技术、通用平台及组件技术、系统产品技术、管理及方法学等；
- 最新技术成果的全面汇聚：熟知的国内外知名软件厂商、企业悉数到场，全面展示各自的最新技术成果；
- 丰富的行业应用实践：既有电信业、银行业、保险业、烟草业等完整的垂直应用解决方案，又有能自如搭建各类应用的成熟的水平应用平台；
- 纷呈的热点技术和应用：深入解剖那个让人捉摸不定的开源，全景展示新兴的数码娱乐技术。

北京国际会议中心

2004年12月4日—6日

2004年11月1日 第11期 总第98期

主管：中国社会科学院
主办：中国社会科学院文献信息中心
协办：北京百联美达美数码科技有限公司
网址：http://www.csdn.net
国际刊号：ISSN 1672-3252
国内刊号：CN11-5038/G2
邮发代号：2-665
广告经营许可证号：京工商广字0188号
出版：《程序员》杂志社

总编：黄长著
社长\常务副总编：张悦校
副社长：蒋清
编委会：黄长著 张悦校 陈洋彬
蒋清 唐琦 曾登高

编辑部主任：孟迎霞
采访主任：闫辉
技术主编：孟岩
责任编辑：欧阳瑾 罗景文 韩磊
熊建国 刘如鸿 刘婧
华东记者：熊节(xiongjie@csdn.net)
华南记者：张里(zhangli@csdn.net)
美术总监：张浩祥
美术编辑：吴志民 王敬瑜
电话：010-84540265
投稿邮箱：editor@csdn.net

发行总监：陈春柳
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

上海联络站：孙峻峰
电话：021-63619663-801
传真：021-63619457
信箱：shoffice@csdn.net

西南联络站：谭皓
电话：023-63810152
信箱：tanhao@csdn.net

读者服务部(邮购)
网上订购：www.dearbook.com
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
邮政编码：100028
电话：010-84540262
传真：010-84540263

法律顾问：正见永申律师事务所

印刷：北京乾津印刷有限公司

出版日期：每月1日
零售价：人民币10.00元

本刊文章版权所有，未经许可不得转载。

发现原刊有缺页，请寄回原刊本社读者服务部，
此刊可补寄。

只有不断交流沟通，知识才能分享，技术才能进步
作者、编辑，热切等待你的讯息
沟通、评论、意见



请随时联系

杂志频道
新版杂志频道，编辑完全 Blog，联系作者，
发表评论，选题讨论
<http://mag.csdn.net>
热线电话
编辑部热线 010-84540161，任何杂志问题，
欢迎随时拨打

人物 & 报道

8 编读往来

9 名人堂

盘点 Rob Pike

10 程序天下事

14 封面书摘 & CSDN TOP10

16 网站动态

17 软件开发大事记

感悟

18 软件是门艺术

3G 的 Killer Application?

企业专访

20 3721 程序员揭密

22 超越梦想，一起飞

——专访亚太和大中华地区技术社区
暨最有价值专家项目总经理柯淑芬

23 制造商的软策略

——诺基亚论坛开发伙伴大会

24 Action! 开源竞赛

25 工作流引擎设计正在进行时

特别策划

26 迫在眉睫的职业规划

面对层出不穷的新技术，激增的就业压力，
不断分化的开发角色，做好准备吧，这是
个更加需要规划的时代。

对话

32 做全球最火的软件开发中心

与微软亚洲工程院技术总监林斌对话，了
解这个面向全球软件产品开发团队。

营销

34 软件营销与程序员

在中关村闯荡多年的腾星高级副总裁毛一
丁畅谈软件营销奥秘以及与软件开发关系。

创业

36 独行五年开发交互图形控件

一个程序员埋头苦干五年，开发出通用
的交互图形控件产品并得到了市场的很
好反馈。我们还邀请到了用友华表总经
理，Cell 表格控件的创始人唐爱平做出精
彩评点。

开源文化

39 跨平台的 BT 开源项目

41 声音 & 幽默

中间件, 尽在其间。

您看到了吗?



4

2

1

3

IBM

Tivoli

注解

1. 买家下载有竞争力的报价
2. 经理安全取得发票
3. 司机获得详细的运送信息
4. 以往的供应商无法侵入内部网络
5. 客户身份受到保护, 免于失窃

中间件就是IBM软件。借助单点登录(Single Sign-on)技术, IBM身份管理软件将确保把适当的访问权限赋予适当的用户。开放、模块化的Tivoli软件, 能使员工、合作伙伴、客户和供应商之间的流程自动化, 同时大幅降低成本。这样, 每个人就能方便地获取信息, 随时、随需。

了解更多随需应变世界的中间件, 请登录ibm.com/cn/middleware/automate

ON 随需应变的业务

IBM, Tivoli和ON Brand Business logo为IBM在美国和其他国家的注册商标。©2004 IBM版权所有。

管 理

软件工程

42 迎接 UML 2.0

UML 2.0 标准已经进入了倒计时阶段,作为 OMG 推荐的新标准,你做好准备了吗?

测试专栏

47 性能测试用例

本文以设计性能测试用例为示范,讲解在实际工作中,如何有效划分测试种类和编写对应的测试用例。

理论实践

51 拥抱变化:敏捷设计从理论到实践

如何应付软件开发中的“变化”,一直是近年来备受软件企业关注的问题。本文从理论和实践两方面,和大家分享笔者在敏捷设计方面的心得。

交互设计

55 什么是交互设计

在每天的生活中,我们都要和许多工业产品进行交互,交互设计是一种能影响到所有人的技术。请看本文对交互设计的简要介绍。

CMM 专栏

58 需求管理的思辨

要想掌握某个东西,就必须深入、扎实地理解它。而要做好需求管理,就需要知道需求到底意味着什么,也就必须知道它由哪些工作组成。请看本文作者对需求管理的思辨。

非程序员

64 需求问题根源——需求启发技术

提取需求的时候,我们发现涉众往往存在着不同的启发障碍,因此需要我们根据不同的情况,采用不同的启发技术来探索涉众的真正需求。

技 术

技术专题

68 专题导读: C++/CLI, 凤凰的涅槃

70 “C++/CLI 全景体验”专栏开篇语

我们很荣幸的邀请到 C++ 大师 Stan Lippman 先生和 .NET 技术专家李建忠先生,开辟了一个专论 C++/CLI 的新专栏“C++/CLI 全景体验”,本期将刊登这个专栏的第一部分。

76 聆听未来——Stan Lippman 谈 C++/CLI

Stan Lippman 是最早的 C++ 先锋之一,许多年来他一直致力于 C++ 的改进工作。2001 年加入微软后,他开始着手 C++ 在 .NET 平台上的改进——C++/CLI。本文是本刊记者与他的谈话笔录。

晟唐®

保护无极限

引领最前端...

Word 文档访问控制新技术

——电子文件保险柜

全球第一家光盘软件成功防盗版

——加密龙

电话: 020-85640979

网址: www.tongzheng.com.cn

广州同正网络科技有限公司

78 CSDN C++/CLI 调查报告

为了了解大家对于 C++/CLI 的看法和意见, CSDN 网站上组织了本次调查。

79 C++/CLI 会冲击 C# 吗?

C++/CLI 声势浩大,但它真的会冲击 .NET 上的首选语言 C# 吗? 请看本文作者的观点。

80 C++/CLI: 鼎新革故

C++/CLI 的出现,对于 Managed C++ 无疑是一个巨大的冲击。两者之间究竟有什么巨大的差别呢? 请看本文的介绍!

83 山雨欲来风满楼

——标准 C++ 及 C++/CLI 发展综述

互联网的迅速发展给了 Java 和 C# 这样的新生代语言蓬勃的生机和发展空间,它们也出尽了风头。然而在喧嚣的背后,有一门语言仍然在冷静而理智地发展着——它就是 C++。

85 非典型 C++/CLI 教程

C++/CLI 来了,它带来了什么优势? 我们为什么要选择它? 我们能得到什么? 作为传统的 C++ 程序员,我们该如何转变呢?

中间件, 尽在其间。

您看到了吗?

2

5

4

3

1

IBM

WebSphere.

注解

1. 客人以无线方式登记入住
2. 大堂经理查询客人的偏好
3. 紧密整合供应商的服务
4. 供应自动满足需求
5. 回头客使盈利不断增加

中间件就是IBM软件。强大的WebSphere软件, 就像一条有力的纽带, 将您的业务、供应商、合作伙伴和顾客紧密整合在一起。这一动态的链接, 能使整个企业运作更高效、更灵活、响应更及时, 真正随需应变。WebSphere通过开放的标准, 将所有流程连接起来, 而且易于管理。一切就绪, 您当然安枕无忧。

了解更多随需应变世界的中间件, 请登录 ibm.com/cn/middleware/integrate

ON 随需应变的业务

IBM, WebSphere 和 ON Demand Business logo 为 IBM 在美国和其他国家的注册商标。©2004 IBM 版权所有

蔡学镛专栏

89 微软手机游戏开发经验谈

SmartPhone 这个名词在开发领域已经不再陌生了，而在其平台上的游戏应用开发更是越来越多。窥其门径，作为 .NET 程序员，要跨过这个门槛，其实不难……

熊节专栏

92 了解 Java 规则引擎

每逢看到长串的 IF...ELSE...就让程序员格外心烦，你是否也曾面对一堆这样的逻辑愁眉不展？是否也希望能够清楚地知道逻辑背后到底是什么？

Java & .NET

97 探讨与比较 Java 和 .NET 的事件处理框架

开发事件处理程序时，您是否也曾遇到这样那样的问题？当您对事件处理框架有一个清楚的认识时，将不再困惑！

Java 编程

101 Java 多线程编程实例

——优化 Cache 并发访问性能

缓存设计是一种有效提高系统性能的方法。Java 下多线程的应用当数广泛，然而，如何协调这些进程，让它们尽可能少地访问数据库？缓存来给您答案。

编程新思维

104 趣谈 Functional Programming (上)

Functional Programming 这个概念用目前的技术发展衡量，应该算是一个非常古老的编程方法了。经历了技术一代一代洗礼，它逐渐走向成熟。本文用实例风趣地讨论 Functional Programming 方法。

Ineta 专栏

109 他山之石，可以攻玉

——软件产品设计实践

任何软件产品都有最擅长解决的问题，如何围绕核心问题安排软件功能，减少用户在软件使用中遇到的麻烦是软件产品最重要的设计原则。

.NET 编程

112 深入 .NET 控件开发

如果从 Windows 窗口管理的角度来看，所有控件都不过是一个或多个子窗口。都是围绕子窗口进行用户接口处理的构件程序，.NET 控件也不例外……

产品 & 应用

117 用英特尔软件开发工具释放 IA 架构上程序的最佳性能

本篇是介绍如何运用英特尔软件开发工具提高软件性能的最后一篇，介绍如何利用高度优化的函数库提高代码开发效率和性能。

120 性能——企业级应用开发的最后一环（下）

继上篇介绍了 APM 的概念和特点之后，让我们来看看主流 APM 厂商的产品和它们的解决方案。

124 软件配置管理的最佳实践经验

基于任务的软件配置管理相比基于文件的软件配置管理在许多方面更加具有优势，是一种全新的软件配置管理理念。

工具点评

128 NUnit .NET 项目测试点评

在 XUnit 系列测试软件中，NUnit 作为 JUnit 的 C# 移植版，一定会让你为它的精巧易用而感叹。

书评

131 再读《ASP.NET 揭秘》

这本书的第一版名叫《ASP.NET 技术内幕》，这次本书第二版卷土重来得到了微软 ASP.NET 项目组长的推荐，非常值得拥有。

133 新书上架

CSDN 每月精选上市新书，由编辑和业界专家亲自点评。

程序员 Magalog

135 读书的快乐

广告索引 (2004.11)

广告内容	公司名称	页码
<input type="checkbox"/> 形象广告	维尔软件	封二
<input type="checkbox"/> 2004 中国软件技术大会	中科软、CSDN	扉页
<input type="checkbox"/> 中间件广告	IBM	3、5、7
<input type="checkbox"/> 形象广告	广州同正网络科技有限公司	4
<input type="checkbox"/> 培训招生	长城培训学校	19
<input type="checkbox"/> 培训招生	科胜培训中心	67
<input type="checkbox"/> 正阳软件	正阳软件（深圳）有限公司	88
<input type="checkbox"/> 培训招生	新东方培训中心	116
<input type="checkbox"/> 杂志征订	互联网周刊	123
<input type="checkbox"/> 图书促销	CSDN 第二书店	127
<input type="checkbox"/> 杂志征订	《程序员》杂志社	130
<input type="checkbox"/> 书讯	人民邮电出版社	132
<input type="checkbox"/> 书讯	博文视点资讯有限公司	134
<input type="checkbox"/> 书讯	清华大学出版社	136
<input type="checkbox"/> IBM developerWork Live! 2004 大会	IBM	封三
<input type="checkbox"/> 软件开发工具	英特尔软件事业部	封底

中间件,尽在其间。

您看到了吗?



WebSphere.

注解

1. 销售助理在线查询库存
2. 经理上传盈利指标
3. 主管了解员工的加班情况
4. 店员获取市区的店铺位置信息
5. 通过一个门户,每个人都能访问所需信息

中间件就是IBM软件。作为IBM Workplace家族的一员,WebSphere门户软件能将遍布全球的合作伙伴、员工和顾客连接起来。这样,大家就能借助几乎任何设备,通过一个共同的界面,访问各种不同的应用。正是这个端到端的解决方案,在帮您提高效率、降低成本,使业务转向按需应变。如此良伴,您怎能或缺?

了解更多按需应变世界的中间件,请登录ibm.com/cn/middleware/workplace

ON 按需应变的业务

IBM, WebSphere, IBM Demand Business logo为IBM公司和其他国家的注册商标。©2008 IBM Corporation



编辑老师，你们好：

《会战 2005 J2SE 5.0 vs. .NET 2.0》专题十分精彩。在这个系列的专题中解决了不少原来曾经非常困惑我的问题。作为担当项目执行角色的程序员，往往在技术的选择面前不断徘徊，很多时候，在这些无法完全衡量的技术面前，我都能在《程序员》中找到明确的线索。随着技术的更迭，业务逻辑也在不断复杂化，能否采用最好的利器来解决这些问题成了我目前最为关注的焦点。感谢《程序员》给程序员们指明了方向！

读者：孙凯

读者来信

编辑老师，你们好：

读你们的杂志已经有一年多了，从去年的那次改版之后，最让人震撼的莫过于这次。作为一个忠实读者看到杂志这样的变化，让人有一种欢欣鼓舞的感觉。平心而论，今年早些时候杂志的风格，真让人感觉味如嚼蜡。其实每期《程序员》杂志都有非常充实的内容，非常丰富的信息量，但是如果没有很好的表现形式，让读者觉得一成不变，那么杂志最终无法赢得读者的心。这有些像《谁动了我的奶酪》这本书中讲述的内容，做任何经营都需要不断找到自己的出路。《程序员》的出路在哪里？136个页码的内容，如何让其达到最优化的效果？这些事情，都应该是各位编辑老师经常考虑的。尽管我只是一个程序员，有时候负责一个或者是几个项目，但是我仍然固执地认为，做好一本杂志，需要不断的进步和变化。

这期的杂志在内容和形式上就有了很好的变化。其中最重要的变化当数封面了，新的封面设计让人拿到手

上就有一种耳目

在

也

打开

物报

设计过

来也觉得

不再拘泥

版式，让阅

更加轻松和

的感觉；上期

加的栏目，包括

动态、软件开发大

Magalog 等，都充实

了杂志的信息量更多；另

外一个非常显著的变化则是技术专题在形式上的突飞猛进，增加的文章篇数、更多的专家在这个栏目中各抒己见、极具前瞻性的内容、方便阅读的版式等各个方面都有了长足的进步；但美中不足的地方，亦是技术专题，尽管内容出类拔萃，但是看起来却显得太素，如果能够在版式上有些改变，读起来会更加轻松。

其实杂志就是需要做出一种感觉来，技术媒体能够发出的声音才是读者最为关注的。在中国，几十万开发者如果都能从这种声音得出自己的见解才是最重要的。当他们在工作和学习的过程中遇到问题都能想到《程序员》，那么我想，这就是杂志最成功的地方！衷心祝愿《程序员》能早日看到那天的到来！

读者：戴世杰



封面设计让人拿到手

目一新的感觉，摆

报亭和书报摊上

显得格外显眼；

封面之后的人

道板块，重新

的页面看起

赏心悦目，

于陈旧的

读有了

愉快

新增

网站

事记，以及

了杂志的内容，让整本杂志

盘点 Rob Pike

■ 文 / 方茜

名人堂
Hall of Fame



Rob Pike 何许人也？

作家？不错，Rob和贝尔实验室的Kernighan合作出版了《Unix 编程环境》(The Unix Programming Environment)和《程序设计实践》(The Practice of Programming)。这两本书都有中文译本，为初学者研习和提高编程技术提供了很好的方法和理念。

天文学家？这是Rob的副业。牛烘烘的他曾在70年代得到加拿大皇家天文协会多伦多中心的资助，对光污染进行研究，写有相关论文若干篇。更厉害的是，他设计的伽马望远镜差一点被用在航天飞机上（如果他没有开玩笑的话）。

运动员？网上盛传他是1980年奥运会射箭银牌得主。呵呵，别中他招儿。Rob喜欢开玩笑，谁都知道他是在美国工作过数十年的加拿大公民，而这两个国家都没有参加那一年的奥运会。

Rob Pike就是这样一个人，有着多面的人生。不过，对程序员来说，研发操作系统才是他最经典的人生。

1980年，Rob开始在贝尔实验室的Unix小组中工作。他写出了第一个Unix下的位图视窗系统，著名的Blit终端。这个Blit为Unix系统提供了一套异步多窗口环境，从而突破传统终端的限制，提高了Unix进行多重程序设计的能力。

此后，Rob设计并编写了Plan 9、Inferno等新分布式操作系统。这里，最值得一提的是Plan 9，它是Rob号称锐意革新的操作系统，尽管它并没有引起太多人的注意。操作系统在初期有很多个发展方向：多重处理与并行运算、计算机网络化、以线程闻名的“子程序”概念以及后来诞生的“微内核”概念等。Plan 9承袭了“微内核”概念，让内核提供操作系统的核心功能；同时，Plan 9在融入Unix系统特点的基础上，重新设计元素，包括：用户端的终端、存储固定文件的服务器、更快的CPU计算服务器以及提供用户验证和网关特性。目前，Plan 9的文件系统可以支持所有的系统服务进程，也就是说任何用户可以使用的资源在文件系统中都可以找到惟一的命名。它通过网络层的协议9P来访问所有的资源，支持异构的网络。

Plan 9是Rob的得意之作，可是反响不大。想想这也不奇怪。

当今操作系统开发周期越来越长，代价越来越大，而受使用习惯的影响，普通用户越来越集中到少数几个系统上，因此快速成长期后出现的操作系统难免被冷落，而那些对操作系统的创新变革之举也只能被老系统的测试和修补而掩盖。这很无奈，不过历史就是如此，总是技术创新与市场选择互相博弈的结果。

在贝尔实验室期间，Rob也同Ken Thompson (Unix的发明者之一)合作开发了UTF-8，即：8位Unicode转换格式。这是一种无损耗、长度可改变的Unicode字符编码，主要用在Unix、Linux等类似的POSIX系统上。别以为这是个平常物，它提供了一种简便而向后兼容的方法，使得那种完全围绕ASCII设计的操作系统，比如Unix、Linux，也可以使用Unicode，让基于各种各样ASCII扩展的文件交换变得不再困难。

坊间曾流传是IBM设计的UTF-8，然后由Plan 9来实现。然而，据Pike回忆，事实上是Ken和他设计的。1992年，他们尝试使用取自ISO 10646的原始UTF让Plan 9支持16位字符。在邻近封装到系统上运行时，IBM的FSS/UTF研发人员给Rob打来电话，说有个X/Open委员会的会议。这激发了Rob的灵感，他和Ken决定将这个UTF-8的雏形变为一种标准，并借X/Open委员会推广出去。于是，他俩只用了一个晚上就写完了所有的代码，并在一天后完全实现了在Plan 9上成功运行。

2002年，Google大肆招揽各种操作系统人才，Pike也被成功挖角。在Google里，Rob继续着他的操作系统之旅，目光主要集中到分布式计算系统的容错能力和负载均衡能力。但愿Rob这一跳槽能带来更多更棒的系统。☞



微软技术



主持人: kaneboy

博客堂成员, MS MVP, 专注于 .NET 开发, 在 SharePoint 2003 和 ASP.NET 方面比较擅长。

金

秋九月, 微软在中国已有九年历史的Tech·Ed盛事又一次在全国三个主要城市广州、上海和北京依次拉开帷幕。

对于每一位热衷于微软技术的IT人来说, 每年一度的微软技术教育大会Tech·Ed无疑是最值得期待的了。有意思的是, 今年微软首次将“微软技术教育大会”更名为了“微软技术大会”, 强调了Tech·Ed未来的发展将更注重与整个产业进行技术共享与研讨。从会议形式上也体现出了这一点: Hands-on Labs、BOF(同类人)、Chalk Talk等新颖、有趣的活动形式加强了主办者与参会者更加深入的沟通与广泛的互动。

据一项对包括中国1500家企业在内的全球47万家大型企业的调查显示, 兼容、创新、信息互通共享是当今企业IT最大的需求, 这也成为本届微软技术大会的主题——“集成·创新”。因此这次大会吸引了包括Intel、Dell、HP以及TCL等国内外33家合作伙伴参与展示它

们在微软平台上的各种创新的集成解决方案。

此外, 包括C++语言大师Stanley Lippman及近百位国内外权威技术专家、数千名IT专业与管理人士在内的嘉宾与会者共同分享了产业趋势、企业项目管理、CRM体系建设、智能客户端架构、行业解决方案等热点技术话题。

微软ISA Server 2004简体中文版也在会场同期发布, 选择在Tech·Ed第一天发布该产品突出了微软意在吸引企业级客户的目的。对于中国用户来说, ISA Server 2004简体中文版是微软在国内第一款简体中文的防火墙服务器软件产品。微软方面介绍说, ISA Server 2004简体中文版是高级应用程序层防火墙解决方案, 具有针对应用层防护、图形化的安全配置、整合Web缓存技术三大特点, 能够实现深度防护、简单易用和高速网络访问, 保护企业的网络资源。产品发布之时, 已拥有清华同方、中外运空运、上海市长宁区等

近十家先期用户。

此外, 微软还公布了Windows Server System服务器家族通用工程技术规范, 并承诺在2005年后, Windows Server System旗下所有的服务器软件产品都会按照这一规范进行研发。微软亚太区技术总监张瑞昌表示, 该技术规范的出台, 是微软在集成创新目标下工程开发领域的一次飞跃, 也是微软进军企业级市场的有力保障。

就在微软技术大会召开前夕, 令人瞩目的9月微软MVP(最有价值专家奖)评选也评出结果。微软最有价值专家(简称MVP)称号是微软公司一年一度的重要奖项, 登上领奖台的MVP虽然来自不同工作领域, 却拥有两个共同点: 其一, 精通某一项微软产品或技术; 其二, 乐于与他人分享技术和经验。这一次全国共有58位技术专家经过非常严格审核从700余名候选人中脱颖而出, 获得此项殊荣。拥有了MVP称号之后, 他们能分享微软最前沿的技术资源; 参与微软最新技术最新产品的审核与测试; 与微软更密切地交流、联系。罗莉·摩尔在颁奖大会上致辞, 向获奖的技术专家致以衷心的感谢。

不仅如此, 微软还经常利用技术社区和相关的商业项目为整个产业中的技术人员创造越来越多的沟通与交流的机会, 这对于每个技术人员而言都是值得珍惜和善用的良机, 而微软公司作为业界的重要一员也将从中获得难以评估的价值回报。



9月29日PalmSource宣布推出Palm OS Developer Suite 1.0。该产

品将针对基于ARM及68K基础之上并用于Palm Powered设备的新一代多媒体及无线应用软件的开发工作提供相应支持。同时, 还正式发布了PalmSource Web Browser 3.0, 从而使Palm Powered智能移动设备用户感受到丰富的浏览使用体验, 并同时实现直观且易于使

用的导航能力。PalmSource Web Browser 3.0针对智能手机和无线移动设备进行最优化处理。

10月10日傍晚, 国内软件业代表人物、金山公司董事长求伯君和总裁兼CEO雷军再度身披盛装, 酷扮成神仙模样, 和北京的数千玩家一起揭开了中国

连串引人瞩目的事件使得过去的这个月成为 Java 世界注定不平凡的一段时间。

JBoss EJB3.0 Preview Release 的发布; Sun 启动 JDO2.0 和 EJB3.0 的持久层框架合并计划; J2SE1.5 正式版的发布等等重大事件为我们粗略地勾画出 Java 企业应用的未来发展方向。

10月8日, JBoss发布了EJB 3.0 Early Draft 的实现产品 JBoss EJB3.0 Preview Release。自从 JSR-220 (EJB 3.0) 7月24日发布之后, 业界围绕EJB3.0话题的热度就再也没有降过温, 何也? 因为EJB3.0规范彻底颠覆了传统的EJB模型。

众所周知, EJB2 做为一个重量级企业应用框架从来没有实现过它应有的承诺, 业界的批判之声从未中止。在充分借鉴了 Hibernate, TopLink 等轻量级持久层框架产品, 以及参考了 IoC 容器的依赖注入机制之后, 一个全新的轻量级框架 EJB3 的规范被发布出来。

虽然得到 Java 社区的欢迎, 但围绕 EJB 3.0 的置疑声音也并不弱。最大的置疑就是 EJB3.0 规范能否如期在 2005 年夏天发布。很多人认为, EJB3.0 将有可能拖延到 2006 年发布, 而被广泛采用则将在 2007 年。

不过, JBoss EJB3.0 PR 的发布鼓舞了社区对 EJB3.0 的信心。JBoss 声称, 将每 6 周发布一个新版本, 直到 EJB3.0 规范的正式出台。

有了 JBoss 的强力推动, EJB3.0 规范有望在 2005 年发布并得到一定范围的应用。此外, JBoss EJB3.0 PR 的意义还在

于为 EJB 3.0 提供了一个有实用价值的参考实现产品。在 EJB3.0 规范制定过程中, 可以不断根据产品的应用状况和开发人员的反馈来修改规范, 使 EJB3.0 更加贴近开发人员和企业应用的需要, 避免 EJB2 规范的种种弊病。

JBoss EJB3.0 项目的领导者 Gavin King 和 Bill Burke 都是 EJB3.0 专家组成员。其中, Gavin King 是著名的开源持久层框架软件——Hibernate 的作者, 也是 EJB 3.0 规范中持久层规范的主要制定者之一, 因此 EJB3.0 持久层框架的主要理念都来自 Hibernate。同时, Hibernate 也是 JBoss EJB3.0 中持久层的实现。Hibernate 承诺既可以 inside EJB 容器, 作为 EJB3.0 的持久层实现, 也可以 outside EJB 容器, 作为一个单独的持久层框架软件来使用。

此时, 不得不提 Java 世界的另一个主流的持久层框架 JDO。

JSR-220 (EJB3.0) 和 JSR-243 (JDO2) 在今年夏天的先后发布, 也引发了 Java 持久层框架主流规范的分裂。EJB 厂商 IBM, BEA, Oracle 三巨头, 以 JDO2 规范和 EJB3 规范在持久层框架重叠为

由反对 JDO2, 而代表 JBoss 利益, 同时又是 JDO2 竞争产品 Hibernate 作者的 Gavin King 则毫不掩饰的表达了对 JDO2 充满厌恶的个人感情。

Java 持久层框架主流规范的分裂从某种意义上来说, 对厂商, 对开发人员, 对最终用户都有不同程度的负面影响。

因此, JSR-220 的领导者 Linda DeMichiel 和 JSR-243 的领导者 Craig Russell 联名发表公开信, 宣布了一个合并 EJB3 和 JDO2 持久层规范的计划。新的持久层规范将以 JSR-220 (EJB3.0) 持久层规范为基础, 融合 JDO2 的部分特性。新的持久层规范将进入 J2EE1.5 之中, 独立于 EJB 存在, 既可以 inside J2EE 容器来使用, 也可以脱离 J2EE 容器, 独立运行。

这个统一的, 基于 POJO 轻量级的持久层框架意义非凡。

就短期影响来说, JDO2 作为一个不再进行后续发展, 注定将过期的规范来说, 将遭受比较大的冲击。然而, 从长远影响来看, 无论是 Hibernate, JDO, 还是 Entity Bean, 终将统一到这个新持久层规范中来。☞

Java技术



主持人: JavaEye

“Java 视线”网站站长, 资深 J2EE 架构师。

首款大型神话网游《封神榜》发布序幕。据悉, 《封神榜》是金山公司继《剑侠情缘网络版》之后由其北京“烈火工作室”自主研发的第二部网络游戏作品。

10月19日, 微软(中国)有限公司宣布推出一项名为“个人生产力挑战(Personal Productivity Challenge—

PPC)”的 Office 在线服务, 帮助信息工作人员借助时间管理理论和 Microsoft Office 来提升个人生产力。该项 Office 在线服务与 Microsoft Office 2003、Microsoft Office OneNote, 2003、基于 Microsoft Mobile 的设备、Windows, SharePoint,

以及提供最新服务信息的 Office Online 站点紧密集成。该服务通过在线的生产力的评估, 帮助信息工作者了解自身的时间管理和生产力状况, 并提出相应的加强时间管理、提高生产力的建议。该服务具有多种语言版本, 将在包括中国在内的 25 个国家内推广。

软件工程与项目管理



主持人：潘加宇
UML China 首席专家，
潜心研究和实践
UML/UP 相关技术的
应用。

9月17日，对象管理组织 (OMG) 和 Health Level Seven 组织 (HL7) 宣布进行战略合作，开发医疗保健行业的软件标准。HL7 目前是国际医疗界公认的医疗信息交换标准，中国已经在 2000 年 1 月以 HL7 中国协作中心 (HL7 CHINA) 的名义成为其国际会员。

OMG 主席 Richard Mark Soley 说：“这是医疗标准的里程碑。” HL7 联席主席 Mark Shafarman 也称：“HL7 很高兴能和 OMG 一起创建标准，希望将来 UML 能够在 HL7 的方法学和工具上起到关键作用，HL7 已经而且将来也会继续使用 UML 作为我们方法学的基本元素，用它来建立信息模型。”

在应用生命周期管理 (ALM) 上，Sun 希望在今年底发布代号为“弓 (Bow)”的 Java Studio 企业版 7.0。Bow 在 Embarcadero (今年 5 月被 Sun 收购的公司) 将 UML 和 Java 代码的重构及配置集成起

来，并增强对分布式组织基于团队的开发支持。但是，Bow 面对的是一个已经被各工具割据山头的战场：Borland、IBM Rational、……在明年，随着微软最终发布 VSTS，这个领域将会发生地震级的变化。微软的 VSTS 主管 Rick LaPlante 在 9 月中旬指出，微软希望在未来 20 年内开拓一个庞大的 ALM 市场。

9 月中旬，Popkin Software 发布 System Architect v10，值得注意的是它增加了两种针对管理层的图形作为原有图形的补充：Enterprise Explorer Diagram 用于帮助企业的全貌，Enterprise Direction Diagram 帮助形成企业的目标和战略。CEO Jan Popkin 说，“对技术人员，看 UML 图形是合适的，但对 CIO 来说，他们还需要一种战略图形。如果数据中心瘫痪了怎么办，会影响哪些地方？新图形就负责回答这些问题。”

9 月 28 日，Metamill 软件公司宣布

了其 UML 工具 Metamill 4.0 版，该工具支持 C++、C、Java 和 C# 双向代码工程，并在这一新的版本中支持 UML 2.0。Metamill 采取低价策略，每 license 仅售 \$125，如果大量购买和升级还有折扣。

10 月 1 日，Telelogic 在多方供应商的竞争中胜出，赢得一家亚太地区著名银行的 200 万美元订单。该银行将部署 DOORS、TAU、SYNERGY 等 Telelogic 工具。CEO Anders Lidbeck 说，“我们在激烈的竞争中获胜了，再次证明我们的全面解决方案有过人之处，需求驱动的开发方法与应用生命周期管理的结合，能帮助很多客户解决许多头痛的问题。”

9 月 14 日，PolySpace 和 Embedded Plus 在波士顿的嵌入式系统大会上共同发布了基于 UML 项目服务的集成测试环境 Developer-EP。Developer-EP 可以从各种流行的开发环境中读取 UML 信息，可以自动生成测试用例和脚本，以检查 UML 图中设计的功能是否被正确实现。

另外，Lockheed Martin 公司的开发团队成功引入了敏捷原则，将其与原来的计划驱动流程结合在一起，用来开发 Atlas V 火箭的导航软件。此项目负责人 Michael Bethancourt 说，“对那些说敏捷方法不能用于政府和军事项目的人们，我们只想说：‘切！我们可是在发射火箭。’”

本月移动开发领域的头等大事莫过于诺基亚的中国开发伙伴大会了，联想到六月份微软的移动开发者大会，移动设备领域的大腕们在京城的舞台上真是“你方唱罢，我登场”。

智能手机未来的发展方向逐渐明朗起来，而手机操作系统的标准之争也是群雄逐鹿。Symbian 和 Windows Mobile 现在是这场比赛的领跑者，在他们身后不远处还有一位 Palm，虽然家道中落，但是其在 PDA 领域的影响力仍不容小觑。

一些手机市场的后来者，希望在市场变迁中获得更多的市场份额，所以坚定地站在了微软的阵营之中，比如中国最大的手机生产商 TCL。拥有庞大营销网络的 TCL 却常常被缺乏核心技术所困扰，微软的 Windows Mobile 就成为最佳的选择。而其他

嵌入式移动开发



主持人：马宁
微软最有价值专家，
Windows Mobile 开
发者。

手机厂商选择 Windows Mobile 的理由大多是 微软不生产手机。本来大家创办 Symbian 的本意就是防止一家独大，而诺基亚既是竞争的参与者，又成了标准的制定者，这不能不让其他厂商心存疑虑。

提起Intel,大家首先想到是它生产的CPU,不过它可不止生产硬件,它还涉足编译器、性能分析工具、高性能库等软件领域。当然这些工具是针对Intel的CPU进行开发和优化的。在x86的架构中,Intel是目前已知的编译代码质量最好的编译器,相比较以支持尽可能多的平台而著名的GCC,它能够提升20-30%的性能。

不久前,Intel发布新一版的C/C++编译器8.1(简称ICC)。一如既往,它支持windows和linux两个平台。虽然我们不能再指望ICC会象GCC那样开放源代码,不过,Intel在Linux平台上的授权形式比以前宽松了不少,采取了一个非商业授权(抱歉,windows上没有这个授权)。只要你不牟取商业利益,就可以随意使用

Intel的编译器,比如进行科学研究、编译开源软件。

可是Intel为什么提供一个完全免费并任意使用的编译器,难道真的会大幅度降低Intel的股票价值?怎么看,Intel的软件部门都不像一个盈利部门。这样做说不定还能让Intel的CPU多卖出几块。不过你要知道,AMD的64位之火已经快烧到Intel的屁股了。等到以后Linux都跑到AMD的CPU上,ICC就只能被扫进垃圾场了。


和以前的版本相比较,8.1的改进可用一句话来形容——比GCC更强的兼容性以及更加仔细的代码优化。ICC现在默认的就是连接GCC的动态库,并且定义多个GCC的宏,对GCC编译参数的支持也更加完善。在优化方面开始支持-O3

级别的优化,能够提供循环和内存操作方面的优化。

再来实际测试看看。这个简单测试是在一台双Xeon(TM) CPU 3.0GHz服务器上完成的,操作系统安装的是Debian Linux,并升级至最新版。先使用Apache和PHP运行自己写的几个测试程序,然后换用ICC编译Apache和PHP相同版本的源码,再次运行程序。结果出来了,性能提升了15%。测试并不严格,但仍能说明问题。

综上分析,Linux的用户可以把ICC看成又一根进行性能优化的救命稻草。设想,假如你要进行系统性能调优,而当所有常规方法类如增大缓存、修改内核参数、调整磁盘……已经用尽却仍未满足你的要求时,你就可以尝试一下这最后的优化机会——利用ICC重新编译Apache、PHP、MySQL以及其它重要的库,甚至重新编译一个新内核,你将获得额外的20%以上的性能提升。此时CPU的资源已经被你榨至了极致,而这一切不会花费你一分钱,何乐而不为?

说完了ICC,再看看Python吧。

Python的下个重要版本2.4日前发布了第一个测试版,新版Python并未在语言上产生跳跃式进步,依然只是小的修订。增加了对set集合类型的内置支持;统一了int与long等;对list的循环操作进行了优化,据称性能提升了1/3。Python从2.3开始,语言上改进的步伐就已放慢,这也从一个侧面反应了Python在语言层面上的日臻成熟。Python今后的发展可能会更加注重构造一个强壮且功能强大的标准库。但另一方面,由于没有使用现代分代式GC技术,再加上本身为了支持强大的动态语言特性,使得Python虚拟机现在几乎成了最慢语言的代名词。是应该重写一个全新的引擎的时候了,就如同Perl 6做的那样! 

Open Source




主持人: 汤韬

《程序员》杂志 & CSDN
网站技术编辑, 开源爱好者。

在胜负尚未明朗之时,诺基亚和微软同时看到了一支决定性的力量移动开发者阵营。随着智能手机越来越标准化,更多的个性化应用逐渐浮出水面,游戏、网络浏览器、即时通讯工具……将来人们选择手机时,丰富的手机软件也许会成为重要标准。而这些应用的开发都要寄希望于移动开发者阵营。

国内的移动软件开发商也采取了和手机厂商类似的策略,同时为Symbian、Windows Mobile、BREW和Plam四种操作系统编写应用。这种情况一方面是因为国内的移动软件处于一个起步阶段,许多的工作还处在技术摸索阶段,另一方面也说明,移动开发领域的未来还存在太多的变数。9月,国内网络游戏大鳄盛大宣布收购北京数位红,这次国内网络游戏与手机游戏的强强联手,也预示着手机游戏很可能成为继短信后的又一个IT焦点。

一场决战紫禁之巅的对决已经拉开了大幕,无论最后结果如何,有一点可以确定:中国市场将成为对决的主战场之一,而中国的移动开发者也必将大有作为。 



MSDN Magazine



用“防卫型”编码技巧保护应用程序和重要信息

如今这个互联的信息世界,任何一个程序都可能成为被攻击的目标。为了保证安全,你必须在程序的防卫能力方面做更大努力。你所学到的关于防卫型编程的一切将有助于你写出更加安全的代码,但这还不够。你必须更加深入地应用程序建立完备的防御体系。

在本文中,我将着重关注用户信息安全,保护他们的个人信息,并为服务器构建防御系统。本文将涉及许多常见的开发场景,探索一些实用的技术,它们可以使你更容易地写出免遭攻击的代码。

增加 Web 应用程序的便利特性

基于Web的商用应用程序在设计时通常需要兼顾安全性和用户使用的便利性。最明显的一个例子或许就是在一个J2EE应用程序里如何处理Session超时的的问题。如果您的公司也在使用流行的Apache的Tomcat服务器,默认情况下您会受到一些限制。在讨论如何突破这些限制之前,我们来复习一下Session的概念以及用法。

Session是一种机制,实现在用户浏览器发出的多次请求之间保持某些状态信息。例如,一家零售网站收到一个“结账”的请求,程序就需要知道顾客此次选择了哪些商品。HTTP本质上是一种无状态的协议,这意味着它不会将这次的“结账”和之前该顾客的“加入商品到购物车”的请求相联系起来。换句话说,HTTP将每个请求都独立看待,认为它和其他请求无关,这个不足就是通过Session来弥补的。

Java Pro



Visual Studio Magazine



装饰一下你的 Windows Forms 程序

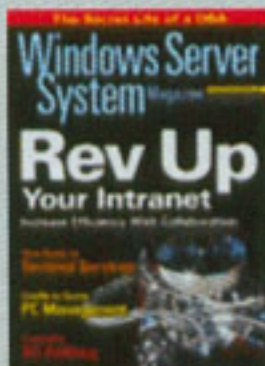
现在的用户要求更高了——他们喜欢Office那样的界面,例如弹出式窗口,不规则外形的窗体以及可折叠和吸边的控件,因此希望其他程序也能做到这样。本文中,我将向你展示创建这三种特色界面的基本技术,它们能装饰(也能破坏)你的下一个Windows Forms程序。

通常Windows Forms都基于一种缺乏个性的外观——矩形。以前在非.NET环境下创建不规则(非矩形)的窗体是比较困难的。幸运的是,.NET框架提供了简单易用的API,可以很好的装点你的窗体。

用协作提高效率

协作(Collaboration)可谓是第三代内联网的游戏规则的代名词。基于微软平台的这类协作系统所采用的核心技术是Microsoft Office SharePoint Portal Server 2003——一种集成化的协作门户技术。SharePoint Portal Server (SPS)构建于Windows SharePoint Services (WSS)之上,WSS是Windows Server 2003自带的服务,用于提供全面的小组站点管理,统一化的门户服务,以及完善的索引和查询功能。然而,一个完整的协作环境不是仅由支持小组站点的技术所组成的,你还需要通信系统,以便雇员可以进行实时和分时的交流。在这方面,可以使用Windows Server系统的另外成员:Microsoft Office Live Communications Server (LCS) 2003和Microsoft Exchange Server 2003。

Windows Server System Magazine



智能系统

对于计算来说,安全性是再重要不过的问题了。在本期DDJ中,我们请到了 Avi Rubin——电子投票,Web表单校验和数字签名方面的权威人士——来展示一个Wi-Fi的安全框架。还将谈及无线计算方面的话题,探索ZeeBee协议并深入研究无线USB设备。此外本期还将探讨Parlay电话规范,二进制XML,Web服务和Apache Axis, .NET, Eclipse, VISA API, 以及更多。祝阅读愉快。

Dr.Dobb's Journal



CSDN 10月文档&论坛TOP

10

帖子地址: <http://community.csdn.net/Expert/TopicView1.asp?id=帖号>
 文章地址: <http://dev.csdn.net/article/ID前两位数/ID.shtml>

技术文档

主题: C语言图形模式下的五子棋游戏, 需要代码的朋友请进

作者: lifan5748

所属社区: C/C++ C

回复次数: 213

帖子 ID: 3417014

简介: C语言图形模式下的五子棋游戏, 使用鼠标控制, 能够实现人机对战, 能够判断胜负, 能够保存棋局和玩家所走的每一步, 并且可以按步骤还原棋局……大概2500行, 提供给需要源码做研究的朋友。

主题: 需要在.NET中实现邮件发送/接收/解码的进来

作者: qzj

所属社区: .NET技术 VB.NET

回复次数: 173

帖子 ID: 3379845

简介: “花了一个多月, 完成了一个类库, 包含4个类, POP3Client, SMTPClient, Mail, MailAttachment, 分别实现了接收、发送、编解码邮件和邮件附件的编解码。”

主题: ASP能做什么, 不能做什么??

作者: liuxiaoyi666

所属社区: Web 开发 ASP

回复次数: 121

帖子 ID: 3358876

简介: 在新的语言不断涌现的今天,

我们还有没有时间仔细想一想, 一门语言究竟能做什么, 不能做什么?

主题: linux 真的好用吗? 我不见得!

作者: bigghostboy

所属社区: Linux/Unix 社区 程序开发区

回复次数: 116

帖子 ID: 3372952

简介: Linux与Windows孰优孰劣, 一直是用户们争论的焦点, 至今没有结果。

主题: 专题讨论 Asp.net 到底有什么用

作者: huwei001982

所属社区: .NET技术 ASP.NET

回复次数: 100

帖子 ID: 3417344

简介: 我们应该选择 asp 还是 asp.net?

论坛

主题: J2EE WEB 应用架构分析

作者: JavcLinIBM

类别: Java

人气: 5248

ID: 42217

摘要: J2EE体系中众多技术的出现给电子商务时代的WEB应用程序的开发提供了一个非常有竞争力的选择。

主题: funny.exe 木马病毒手动删除方式

作者: liguangyi

类别: 其他

人气: 5226

ID: 41901

摘要: 该病毒运行时, 将自动在 qq/msn 等聊天工具上发送/传染。

主题: 在 Asp.Net Web 应用程序中长时间装载页面时显示进度条

作者: chenweionline

类别: 网站制作技术

人气: 4170

ID: 41613

摘要: 虽然是不能实时反映装载进度的假进度条, 但可以避免用户误以为系统故障或死机。

主题: Visual C# .NET2003 语言的改变

作者: s98

类别: .NET

人气: 4118

ID: 42224

摘要: Microsoft Corporation对C# 编译器的实现进行了几处改动。

主题: Delphi 消息机制引入的一个副作用

作者: Spacesoft

类别: Delphi

人气: 4118

ID: 41676

摘要: Delphi 在处理进程的消息时引入了一个进行消息分发的隐藏窗体 Application, 但最近我发现这个机制也引入了一个副作用, 会在某些情况下影响程序的界面交互行为……

网站动态

TheServerSide.COM
Your Enterprise Java Community

<http://www.TheServerSide.com>

TSS一如既往地热闹非凡。这个月，TSS上关于EJB的讨论怕要数最多了。一些人看好EJB 3.0的美好前景，并将它和JBoss进行完美组合；另外一些人却在高呼：EJB应该走向灭亡。新版本到底如何，相信单靠简单的争论是无济于事的，只有等待实践的检验。

同时，《Mastering Enterprise Java Beans Second Edition》这经典书籍的英文版已经正式提供给Java学习者下载了。Java除了它本身的开放性以外，学习资源的开放性也非常让人信服。一般的开发者如果需要学习Java技术，恐怕随时都能找到一大堆资料阅读，而且全部免费！相较MS来说，学习Java的物质成本确实经济得多。

artima developer
A community of programmers who care about their craft

<http://www.Artima.com>

微软的SQL Server 2005，也就是代号为Yukon的下一代数据库产品延期了，估计正式版本可能需要等到明年夏天才能看到。

最近，一个关于契约式设计的讨论在Artima上争论得不可开交。关于契约式设计真正含义和概念的争论其实已经不是头一回，不过这次大家都很有礼貌，各自都在不断和对方获得更进一步的沟通，其实程序员本身渴望自由、渴望沟通的心是不变的，如果能够在软件技术上产生一些碰撞又何乐而不为呢？

其实Gmail这个东东进行内测已经有好一段时间了，csdn网站上都曾经有无数这样的帖子出现：“发放Gmail帐号！”可是Artima网站上，仍然有这样一个帖子如此受欢迎，国内的各大门户网站都推出了1G容量的超大免费邮箱，看来硬件资源，我们确实不比人家差了！

informit.com

<http://www.informit.com>

本期的技术专题是C++/CLI，而最近在Informit上，跟C++/CLI相关的话题也比较火。10月19日，C++标准委员会在微软总部雷德蒙召开第34次会议，这是一个具有象征性意味的表示。Informit给予了及时的报道。此外，最近反病毒的话题也很热火，有人在给病毒制造者提善意的建议，希望他们不要把病毒做的太凶猛。不知道是不是“与虎谋皮”。Microsoft Office已经成了功能完善的办公及办公系统开发平台，Informit也在加大这方面内容的建设力度。

Slashdot
News for Nerds. Stuff that matters.

<http://slashdot.org>

Slashdot永远是那么热闹。那个著名的timothy孜孜不倦的发帖子。最近一篇比较JVM和.NET性能的帖子，一如往常地引起热烈的讨论。不过，相比对于两本书的讨论，上面的那篇帖子还是小巫见大巫。这两本书分别是High Performance MySQL和Programming Ruby第二版。其中后者被认为是Ruby领域里的圣经。本期《程序员》的名人堂介绍了Rob Pike，正好Slashdot对他进行了一个专访，他就Unix的历史和发展介绍了不少内幕故事。

 **O'REILLY
NETWORK**

<http://www.oreillynet.com>

O'Reilly Network已经发展成为一个综合的开发人员技术网站群，很值得关注。特别是O'Reilly一向是开源文化的大本营之一，所以其上有有关微软技术的内容就有特别独到之处。比如最近在其下属站点OnDotnet上，Justin Gehtland在一篇blog里猛烈抨击了.NET程序员社群对于开源文化的拒斥，并称这是一中令人羞愧的情况。最近闭幕的2004 O'Reilly开源大会上，Phaseit的副总裁Cameron Laird从Tim O'Reilly和Guido van Rossum手中接过了首届Frank Willison奖。Cameron Laird则是Python社群里的名人，为Python技术的发展和推广做出了巨大的贡献。Frank Willison奖是为了纪念今年七月份死于心脏病的O'Reilly前总编Frank Willison而设立的，将给对开源技术做出卓越贡献的人士。

软件开发大事记

16

Apache 发布了 APR 1.0 版。APR 是一个庞大的、完备的 Apache 的官方 C 语言基础运行时库。APR 发布并成熟之后，Apache 今后的主要软件都将建立在基础之上。

20

◆ Eclipse Visual Editor 1.0 版本发布，引领 Eclipse 进入可视化阶段。
◆ Popkin 软件发布了用于办公管理的企业架构工具。

23

关注在 Linux 平台和跨平台软件的 Trolltech 公司发布了 Qt 4。

28

Orinda 软件发布了最新的 JDBC 开发工具，这一 Eclipse 插件可以提高 Java 连接 Oracle 数据库开发的效率。

29

Red Hat 公司发布了最新的 Linux 企业版 The Red Hat Enterprise Linux 4 beta。

30

用来开发 Linux 平台 PDA 并与 MacOS X 开发环境整合的项目 myPDA-Zaurus-Edition 发布，其中包括了库、运行组件、PDA 程序和编译器。

1

◆ Programming Ruby 第二版发行，这个标志性的著作被当成 Ruby 发展的风向标，第二版的出版表明，Ruby 正在快速健康地发展。
◆ Eclipse 荣获 Software Developer 读者选择大奖。

3

Sun 发布 Java 2 Standard Edition 5.0 正式版，大家企盼已久的“猛虎”终于下山了。

4

Sybase 公司发布了两款移动开发辅助工具 Sybase Unwired Accelerator 和 Sybase Unwired Orchestrator。

6

IBM 为 WebSphere 增加了自我修复功能，最新版本的应用服务器承诺可以实现“检测和修复”的特性。

7

8

9

◆ 微软发布 Visual C++ 2005 Tools Refresh，显示微软正在为 Visual C++ 2005 正式版的发布而紧锣密鼓地努力。

11

MySQL 的开发者在 4.1 版本中使用了微软的开源代码，正式的发布日期将推迟至 10 月底。

12

Borland 公司对外宣布了 Borland Delphi 2005，即产品代号 DiamondBack 的 Delphi 第九版。

13

Delta 软件公司发布了最新的 SCOUT 开发平台，这一开发平台可以简化 IT 过程管理。
NTT Data 和 VA Linux 日本公司共同发布了 Mini Kernel Dump，这是一个基于开源协议的错误处理工具。

14

IBM 发布新的开发平台，通过开源的 Eclipse 开发框架，很多商业开发可以接入 Rational 的软件开发平台。
KOE 项目发布了 3.3.1 版本，其中包含了简化的桌面和十八个软件包，还发布了 50 种语言包。

15

◆ GoAhead 软件发布了系统管理和分布式通信软件 SelfReliant 的新版本。
◆ Python 2.4 Beta 1 发布，该版本提高了性能，微调了一些语言特性。

软件开发

大事记

软件是门艺术

■文 / 王征

有人把开发软件当作苦工，有人觉得开发软件很单调，也有人觉得软件是一种很神秘而摸不着头脑的玩意，而我的感受是做一个软件缘由一种冲动，尤其是一种想创作的冲动。

不管软件如何神秘如何不可捉摸，但是软件最终是一种工具，就象人们热了想吹空调，累了想看电视一样。而一个软件的成功除了满足功能需要以外也要考虑是否易用。因此做软件便成了集技术与艺术一身的充满挑战和乐趣的活了。

我们曾经开发过Dynaweb IPS这样的大型网站的发布与管理系统，当简单的HTML语言已经无法满足快速灵活地实现各种网络应用的时候，我们知道需要一


种软件工具来帮助更多的人去掌握网站的发布和部署各种应用。如何能让用户不编写一行代码便能快速地定制出他心目中的网页，去定制生成用户需要的功能与服务？于是首先考虑到页面的布局、导航、搜索、导读、图片显示，进而要考虑到用户如何登陆服务器进行可视化的管理，在保障功能实现的前提下，设计师和程序员需要一起研究用户的界面和交互如何处理，这可是件有趣的事，就象要照顾盲人的义工一样，要假想能够“循循善诱”，“亦步亦趋”地帮助无法见面的用户顺利地完成任务，那么就要让软件会说话，让界面会表达。

成功的软件应该是件艺术品，“不胖

不瘦，坚固耐用，顺畅合理，得心应手。

而成为一个优秀的软件工作人员，不但需要掌握和了解开发语言和工具，还应该懂得心理学、美学、人体工程学、机械原理等等，甚至还需要了解哲学、宗教、历史等基础知识，对完美的渴望成了软件人员不断超越自身的动力。

一幅著名的油画一曲动人的乐章可能是来自一个大师的手笔，而一个优秀的软件却往往无法独立完成。也正因为软件开发涉及到众多领域，因此好的软件常常由一个团队完成。软件团队的协作也是一门艺术，才不致于南辕北辙盲人摸象般昏天倒地，要将不同专业的技术人员组织成有效的工作体系，既要象现代化工厂一样严谨有序富有效率，又要能够激发众人的激情和创作灵感，理性和感性的冲突交融就如野马般随时可能脱缰，这非要有大智慧大毅力不可。


软件的设计开发是艺术，软件团队的管理更是一门艺术。 

早年宣传3G的时候一直把可以在手机上看电视电影作为主要卖点之一（当然，还有可视电话等）。但最近，美国一家公司对1300名手机用户作了一个调查，希望了解被调查者是否对将来在3G手机上看电视或者看电影有兴趣。调查结果是大部分用户对在手机或者手持设备上观看电视看电影不感兴趣（The majority were not interested in watching TV on a cell phone or other portable device）。这也难怪，手机那么小屏幕，谁会喜欢在那上面看电视看电影呢？看看天气预报也就算了，看电影？算了吧，在手机上最多看看那种在网上流传的广告Video了，谁会在手机上去看《魔戒》或者《哈利波特》？再说，谁也没有闲到有整段的两个小时时间来看电影，大家都是整天忙忙碌碌跑来跑去的。要是边走边看电

视要撞电线杆的，一边等出租一边看的话出租都被别人抢走了，忙里偷闲分成一段段支离破碎的看，感觉都没有了。

总之，3G手机上的应用问题暂时还将继续存在下去。所以，大公司们还都在寻找3G上的Killer Application，包括DoCoMo借助developer community来寻找3G的杀手应用的做法——“While many operators have been promoting 3G and looking for that “killer app,” NTT DoCoMo is still focused on enabling others to build the killer app”。我相信将来吸引用户转到3G的一

■文 / 郑子颖

定不是在手机上看电影看电视。这就好像DOPOD在宣传686的时候的卖点是“可以看电影的手机”，但恐怕大部分用户用得最多的是686的GPRS、手写输入和强大的PIM功能。至于3G的杀手应用会是什么，我也不知道，但我想Mobile开发这块，要么自己搞出来开公司，要么卖给DoCoMo、沃达丰或者微软，将来肯定是能够产生新的富翁，就像门户成就了张朝阳，网络游戏成就了陈天桥，电子商务成就了马云、邵亦波。 

3G 的 Killer Application?





因为专业|所以职业

The Great Wall Education

[招生专业]

软件开发类

定向对日软件工程师专修班
JAVA软件工程师专修班

网 络 类

网络工程师专修班
网站工程师专修班

与长城计算机学校签约委培用人单位

中国移动通信
北京用友软件工程有限公司
北京华日爱思比科技有限公司(日企)
北京搜狐互动软件有限公司
北京北方新宇信息技术有限公司
北京首都在线科技发展有限公司
中国普天集团北京润汇科技有限公司
中国铁通
北京畅捷科技有限公司
北京金峰星电讯设备有限公司
等2000多家知名企业

长城教育
www.ccjsj.com

The Great Wall Education

地址: 北京市海淀区海淀图书城昊海楼七层
电话: 010-62534487 62534497 62616433 82620105
网址: www.ccjsj.com E-mail: ccjcj@263.net



3721 程序员揭密

■ 文 / Henry

不一样的开发团队 一样精彩的开发文化

10月18号至22日，和乔大厦里多了很多进进出出的外国人，这里是雅虎中国公司的办公室。距离这里不远的嘉里中心在举办一场场雅虎公司全球技术大会的讲座。PHP的创始人Rasmus Lerdorf专程从美国总部赶来，和搜索开发团队Leader和雅虎首席数据官组成了强大的技术团队。Rasmus Lerdorf加入雅虎后，对PHP进行了很多修改和定制，尤其在安全性方面，这次就是专门为雅虎中国的工程师讲述这些方面的内容。

技术开发总监谭晓声也见到了很多以前只在雅虎通与之沟通的全球同事，交换名片时，他拿出两套不同的名片，一套是3721，一种是雅虎中国。在这里，很多人都有两套名片，这从一个断面上展示3721在并入雅虎中国之后的过渡。

合并后的雅虎中国有400多人，其中研发工程师100多人，从事各种开发方向。谭晓声说：“每个方向由于使用的技术和开发的项目存在很大不同，都有自己独特的开发文化。”个人软件事业部主要开发个人客户端软件为主。而20多人搜索开发团队，看似用的是美国的核心搜索技术，实际上做了非常多的外围工作，

一搜网站的很多功能包括图片搜索到前台的蜘蛛功能都是这些人完成的。雅虎中国还有一个10多人的移动业务开发团队，移动业务收入最高的时候每个月有一千万。谭晓声表示：“他们与其它部门的开发方法完全不同，要求反应速度很快，甚至连测试都不需要，因为任何一点时间都非常宝贵，算是一个快速开发的极端。”此外，他们还自主开发了业务支撑系统，这个系统要支撑数千家代理商，这属于典型的企业级开发。

除传统的网页开发外，还出现了一些新的项目，比如雅虎通的本地化产品Mini雅虎通就准备将雅虎邮件和即时通讯功能整合在一起。这是因为现在中国各种IM软件的市场竞争非常大，但因为雅虎通主要在美国开发，各种功能的更新协调起来比较复杂。比如，在中国地区有一个崩溃的问题，但在美国当地带宽的情况下就不出事，其实只要把美国的工程师拉到这里，两天就能修改好这个Bug，但因为在美国很难重现这种情况，修改起来也很复杂的。于是他们准备开发一款独立的产品。

3721并入雅虎中国后，雅虎各国家和地区公司之间的合作也非常频繁。谭晓声说：“雅虎奇摩浏览器Bar是我们开发的，韩国雅虎的bar虽然是他们自己开发，但我们提供了很多技术方面的帮助。”各地的业务也相互补充，日本雅虎在引入网络实名体系后，在市场上取得了巨大的成功。

在雅虎中国，程序员很受重视，不过工作也具有相当的挑战性。谭晓声表示，去年三月份竞价排名上线全员连续工作了37个小时，中文邮上线时连续工作了34个小时，今年年初的整个系统大切换也是30多个小时连续奋战。

直击上网助手开发团队

傅盛还清晰记得当他通过面试后，主管给他分配工作时的场景。当被告知去做上网助手的时候，与很



上网助手的项目经理和雅虎总部的工程师合影

多人一样，他皱了一下眉头，心里想：“怎么是这么简单的一个东西啊。”不过，他现在再也不会发出这样的感慨了。作为“上网助手”的产品经理，他正在不断的从“上网助手”的开发中充实自己，从用户的喜爱中收获欢乐。

“上网助手”纯粹是原来3721一位程序员无心插柳的成果。当时，众多个人网站站长发现了利用IE浏览器的一个漏洞在网页中嵌入一些代码可以修改用户首页。于是，网民在浏览某些网站时，首页经常被偷偷篡改。很多人也写信给3721反映问题。内部开会讨论时，一位程序员提出，不如做一个修补程序发布到网站上让用户可以方便地修正这些问题。这个非正式的项目在发布之后立刻得到了众多网民的欢迎，后来又陆续增加了修改注册表，清理广告等功能。现在，每天的安装量已经达到了20多万次，为3721贡献了16%的浏览率，已经成为公司战略性的产品。傅盛说：“在3721，只要你说话，说的有道理，就有人听，别太在乎你的职位和级别。”

曾经有过这样的故事：经常有熟悉的记者对3721总裁周鸿祎说，你们的产品没有什么技术含量，个性十足的周鸿祎便立刻解释，不过后来因为这方面的问题太多，周鸿祎便懒得解释了。谭晓声表示，只有3721的开发者才知道他们的技术含量到底有多深。

道高一尺，魔高一丈。一些利欲熏心的网站为了获得流量，除了修改注册表外，还从网页上给用户下载木马。这时，单纯的修复远远不够，还需要能够防护。恶意代码拦截，在Norton等软件中也有类似功能，但主要是通过特征库，对代码做分析，一旦遇到加密的脚本就不行了。徐鸣一直从事上网助手的软件开发，他从杀毒软件技术中获得了灵感，提出了软驱动的保护理念。徐鸣说：“我们找到了另外一种途径，首先在系统中注册一个驱动，在每一个脚本要

真正运行的一刹那，我们得到通知，然后看对方做的是什么动作，如果是写注册表和运行木马，我们就可以给它停掉。这样加密脚本的问题就解决了，而且不需要做特征库。”

弹出广告的过滤也证明了这点，最初过滤网页弹出广告一般采用黑名单制，这使用户要不断点击很麻烦，于是试图根据窗口大小进行判断，但误判率很高。后来，他们发现其实程序可以检测到用户点击，只要用户点击，无论窗口是什么大小都可以显示，如果没有点击就出一个窗口，那肯定就是广告了。在他们推出这一功能之后，Google的Bar才出来，发现双方的拦截机制是一样的。

“上网助手”的开发团队还抓住了很多容易被技术人员忽略的功能。比如地址栏下拉列表的选择性清理。很多初级用户连收藏夹都不会用，只能通过下拉列表访问网站，但时间长之后列表就很长，需要清理但又不能完全清除。上网助手把握住了这个需求，这个看似简单的功能在用户使用量一直排名第一。

由于程序涉及到很多底层的开发，而且用户群巨大，作用一个企业推出的产品，他们对兼容性和稳定性的非常重视。一个驱动程序往往需要面对各种环境，要都通过测试，绝对需要时间。驱动



3721 技术开发总监谭晓声



刚开始发布的时候，QA人员会要求所有人安装，并逐个检查公司的每一台机器。出现的问题也多种多样，某些错误甚至连微软中国的工程师也搞不清楚。徐鸣说：“我们还曾经遇到过与某家网络防火墙产品兼容性的问题，其实是对方产品的一些问题，但为了用户，我们需要修改我们的产品。”傅盛认为：“从外面看，一些功能和超级魔法兔子等软件比较类似，但在内部实现方面是完全不同的。很多功能一些共享软件也有，但对于他们来说只是一个卖点，功能做的好不好对作者来说是无所谓的，出现问题大家也能够理解。但对于公司开发这样的软件是绝对不行的。”

“上网助手”尽管属于客户端软件，但是通过网页展示。这主要有两点因素，除了基本的商业目的——利用这个程序为用户的地址栏中加入了网络实名的弱解析功能外，网页版对初级用户来说更加小巧，上网助手只有200k左右，用户可以迅速下载，而且当后台更新时用户也能比较快速地享受到更新的功能。为了让软件尽可能缩小，所有的代码都从SDK写起，没有使用任何控件，甚至包括托栏图标的显示。傅盛说：“技术体现在对程序的控制上。”

现在，互联网使得安全问题无处不在，杀毒厂商主要在病毒防治方面做得比较好，但是在恶意插件和木马方面包括恶意网站关注度不够。基于这种现状，上网助手被定位在安全类的浏览辅助工具。谭晓声还表示：“我们正在做一个比木马克星更好的免费木马防治软件，为网民的上网提供更好的护航。”

2004年9月29日在上海香格里拉酒店,98位微软中国区2004年度MVP齐聚一堂,有58位技术专家从微软公司全球副总裁Lori Moore手中接过了“微软最有价值专家”的荣誉奖牌,这是属于英雄的时刻,正如其中的MVP广告片所提到的,金秋8月,中国体育健儿在雅典书写了中国体育历史上最辉煌的时刻,而9月29日,这些热心助人、技术高超的网络“大侠”们终于得到微软的认同,来到上海,一同分享那一刻的激情与荣耀。

Cally Ko (柯淑芬)从某种意义上来说是这个项目的掌门人,相对于其他微软高层,Cally更加擅长于调动与会者的情绪,总体感觉来说,是一个非常风趣而优雅的女士。在颁奖结束之后,我们专访了Cally,也了解了MVP计划更多的内幕。

问:hi,Cally,看来您还没有从颁奖的激动中恢复过来?

Cally:是的,虽然不止一次经历过这样的场面,但是面对中国MVP的时候,我一想到能够和这些充满激情的人在一块分享,见证他们的激情与荣耀,就忍不住激动。(笑)

问:《程序员》杂志作为国内非常有影响力的技术期刊,那些MVP是否经常阅读呢?

Cally:我个人也经常和MVP们交流,在他们阅读的技术期刊中,《程序员》杂志是必不可少的,因此我也希望通过你们的杂志让更多人了解MVP,从而加入到这个大家庭来。

问:有人说MVP是一群热心而且技术擅长的人群,那么在评选的过程中如何界定技术和社区活跃程度,作为一个MVP最需要的是前者还是后者,或者两者兼备,有没有一个标准?

Cally:如果真让我给你一个数字方面的标准,那么确实有点难,但是有一点可以肯定,我们选出的MVP在各个领域里面都是Top One的,MVP计划也需要一个逐渐成熟的过程,从2年多以前引入中国至今,已经从当初纯粹的社区来源扩大到其他领域,比如以传播微软技术为主的作家、讲师和培训师,当然了也包含一些微软技术书籍的作者。MVP都有两个共同点:

- ◆ 精通某一个微软产品或者技术领域
- ◆ 乐于和他人分享技术和他的经验

从今年的获选的情况来看,我们是更加注重技术,更加注重他们在分享这些技术的过程中对于微软产品和技术的推动作用,

同样的,我们也高度肯定社区那些活跃的朋友,因为没有他们的热忱,微软产品和技术也不会如此普及。你可以看到陈秀峰、郭安定、杨大川还有邹建等人出现在这次的入选名单上,这也反映了我们MVP来源越来越广泛。

问:MVP对于微软的意义是什么?

Cally:微软视所有的MVP为“最有价值的合作伙伴”,目前全球仅有3000多位MVP,在MVP的颁奖会上,我们经常能够见到Bill Gates, Steve Ballmer, Lori Moore, 李开复等微软最高层领导出现在大会上,我相信在刚才的会上你也看到了Lori Moore。我们的MVP是按照产品领域甄选的,那么下面两个因素是我们非常重视的:

◆ 作为个人在某个微软产品或者技术的推动作用。我们很乐意看到越来越多的人使用微软的产品,使用微软的技术,我想MVP在这方面都做了许多努力。

◆ 和产品组的沟通。MVP作为某一个产品或者技术的专家,美国总部产品组非常重视他们对产品或者技术的反馈。

问:可以透露一下微软在未来的几年内,为了宣传和促进MVP品牌,将有哪些举动吗?

Cally:这个问题很好,既然MVP是一个品牌,那么单单通过微软的努力是肯定不够的,也需要这些MVP,需要媒体的宣传,让更多的人了解MVP,更多的人了解MVP的选举过程。当然了,也需要进一步提升获选MVP给个人带来的回馈,我相信不仅仅是物质上的,更加重要的是潜力与支持,还有那种荣誉。中国MVP是一个很优秀的团体,他们在推动微软技术上做出了相当多的努力,比如在技术社区解答他人的问题,为微软产品组提供了宝贵的反馈意见,当然也出了相当多的书籍。微软下一步将会更加紧密地和MVP社区合作,当然也会提供更多的支持。这些在我们社区网站大家都可以看到相关的资料。衷心希望越来越多的人加入到这个团队,一起分享今日的激情与荣耀,这个时刻属于MVP,属于微软,也属于我。☞

■ 文 / 刘如鸿



超越梦想,一起飞

——专访亚太和大中华地区技术社区暨最有价值专家项目总经理柯淑芬

制造商的软策略

诺基亚论坛开发伙伴大会

■ 文 / 才子英



移动通信巨头诺基亚最近有许多“软”的举动，今年2月份成为收购智能手机操作系统厂商 Symbian 公司最大股东，9月份又在“将世界装进手机”的发布上公司最大的股东隆重推出了 Series 60 移动开发平台。可以说诺基亚在移动开发这个产业链条上“万事俱备只欠东风”了。于是继此之后，借着10月15日在北京举办的“汇聚成功 牵手未来”诺基亚论坛中国开发伙伴大会 2004，为诺基亚吹起了一股强劲的东风。会议上400多名移动增值服务业界人士到场，10余家诺基亚论坛的 PRO 会员介绍了他们获得诺基亚论坛技术和业务支持的经历，以及在移动应用开发方面的成功经验。

大会的每一位参会者都可以得到一本《Symbian OS C++ 手机应用开

发》，该书是第一本中文版的 Symbian 开发工具图书。这不得不提到此次大会的举办方诺基亚论坛，他其实担负着专业化和本地化的技术支持和市场服务的任务，包括实验室、中文技术讨论区、开发工具软件包、中文技术和市场资讯，以及市场推广和销售渠道合作等。中文资料的缺乏曾是开发者的巨大障碍，PRO 计划开始推出后诺基亚论坛的技术支持逐渐在技术的本土化方面加大了力度。

大会上的第二个主角是诺基亚论坛的开发伙伴。为了有针对性地向企业级开发伙伴提供支持，诺基亚论坛推出了 PRO 计划。PRO 的企业级会员向诺基亚缴纳一定的费用，获得经整合的商务资源和更加系统完善的技术支持。此次大会为 PRO 会员提供了一个展示移动开发成功经验的平台，大会上掌中万维、新空气软件等 PRO 会员展示了他们所获得的成果。正如诺基亚论坛大中国区总监苏亚瑞先生所说：“诺基亚一直致力于营造开放的应用开发环境，促进移动增值服务的市场化进程，帮助我们的开发伙伴以最有效的成本跟踪新技术、发现新市场并拓展已有市场，在满足用户需求的同时获得预期的市场回报。这才是移动增值市场健康、长远发展的关键。”

北京数码超智信息技术有限公司加入 PRO 计划不久，他们表示：诺基亚论坛能够通过 SDK 和技术文档、BBS 功能、出色的仿真器和开发者社区为用户提供强大的技术支持，对于开发过程中出现的问题，

甚至能够帮助提供完整的解决方案，技术支持更加切合实际了。

诺基亚论坛还为合作伙伴提供市场推广和营销渠道支持，帮助开发商将他们的应用推向市场，并获得赢利。数位红在展会上表示：诺基亚论坛对 PRO 会员进行技术培训和更高层次的技术引导，并沟通国内外软件市场平台，为用户提供商务发展渠道，公司就曾接受过诺基



诺基亚论坛大中国区总监苏亚瑞

亚论坛从国外引进的软件开发订单。

诺基亚论坛得到了应有的回报。到目前为止，诺基亚论坛已拥有160万注册开发商，在中国，诺基亚已有超过10万的注册开发商，并与他们合作开发了数百款基于 Series 60 平台的应用。

移动应用产业链潜藏的巨大市场和财富吸引着众多厂商，与微软、Palm Source 等相比，诺基亚的不同之处在于，他首先是以移动设备制造商的身份出现的。握紧了移动设备，铺好了智能终端操作系统、移动开发平台、开发者这一整条产业之路，诺基亚才能直面强大的竞争对手，才能从移动价值链的所有环节中获利到底。☐



诺基亚（中国）投资有限公司总裁何庆源

“终于开始行动了！”论坛中热爱开源的网友如此评论道。我们可以从这句话里看出很多情感：急切地期待、恨铁不成钢、满怀希望……或许只有开源这个充满激情的世界才能够引发它的技术爱好者们如此复杂的情绪！

首届中国开源软件竞赛已于9月17日拉开帷幕。此次竞赛由国家科技部、863计划发起，主办方中国软件行业协会共创软件分会（通常称：共创软件联盟）事实上就是SourceForge的中文版。

大赛预计有40到50万元用于奖项设置。其中，除了联盟的直接资金投入，还有很多国内开源软件企业的资助。大赛组委会也在和一些外企谈合作，希望多募集资金用于设置大量的个人奖项，鼓励优秀的热爱开源事业的个人。

此次竞赛分两个阶段进行。

第一阶段，从9月17日报名开始到11月中旬，在全国十五个中心城市协助单位的共同运作下，大赛将出现一部分优胜的项目成果，大赛通过这十五个中心城市的软件源挖掘优秀开源软件项目。



刘澎教授：共创软件联盟秘书长，国家863计划计算机软硬件主题专家组专家、副理事长

第二阶段，从11月中旬开始到12月，联盟将推动两岸四地的大中华区竞赛在香港举办。这一阶段中，联盟是大赛的一个参与方，将把第一阶段的优胜项目推荐过来。参赛的开源项目汇集两岸四地的优秀项目，四地之间是平等合作的关系。

目前，台湾地区的参赛项目已经出现九月的京台科技论坛上，两地做了交流，台湾将把近期一个民间的非政治性竞赛胜出的优秀项目推荐到这个合作的大赛里；香港地区的竞赛正在开展中，也将在11月中旬出来成果；澳门方面因为资源不很丰富，只有2家大学和4个企业有相关项目，所以最终的参赛方式由港澳两地商量后自行决定。联盟秘书长刘澎教授说：“大赛第二阶段的最初设想是希望过渡到中日韩范围，但国家间赛事合作必须上半年就有一些计划上的交流，因为不清楚相关规定，所以这一次错过了合作的机会。不过，大赛会一直举办下去，国际合作也一定会开展。”

本次大赛的参赛方主要是科研院所和大学以及社会各界的开源爱好者，因为大赛主要目标就是让这些群体了解开源，参与开源。另外，企业如果有优秀的开源项目也可参加。联盟副秘书长姚郑博士说：“当然，我们同时也希望产业界有共同攻关、共同前进的机制，也就是所谓的智力汇聚。”

一些可能成为评委会成员的专家表示，目前对参加竞赛的项目还不好做任何判断，因为除非让这些项目真正运转起来才能够证明它是否实现了所描述的功能。

■ 文 / 刘婧

姚郑博士就参赛项目的现状发表了自己的看法：“联盟网站上每个项目的进展情况都可以通过它的代码发布、文档发布等表现出来。在已经参赛的900多个开源项目里，进展比较活跃的有100多个，其中有十几个项目发展得相当不错。”

此次参赛的项目都必须开源，大赛监察委员会专门受理各方投诉，维护大赛项目的知识产权。比赛结束后，如果是原创作品，项目可以按照自己的意愿自由处理版权；如果是基于已有软件的二次开发作品，就必须在遵守已有软件的版权基础上，进行其他操作。据刘澎博士透露：“本次大赛的监察委员会正在策划于明年举办一个‘知识产权论文竞赛’，希望大家来探讨开源产品特殊的知识产权模式。”

本次国内首届开源软件竞赛虽然未免千呼万唤了太久，但终究出来了。它最



姚郑博士：共创软件联盟常务副秘书长，中国OSS推进联盟副秘书长，北京共创开源软件有限公司副总裁

大的作用或许就是呼吁，邀请更多的人加入开源。开源世界不仅为开发人员提供了自由开放的环境，磨练技术实力、积累人际资源；而且展开了一幅更广阔更丰富的技术图卷，这里几乎无所不有，同仁间切磋技艺，寻前人的足迹二次开发，与闭门造车、重头开始之间的区别不言而喻。

如果说开源其实是一种生活方式，那么参与开源对于每个开发人员来说或许能够用换一种活法来形容。☐

Action!

开源竞赛



workflow 引擎设计

正在进行时

■ 文 / 沈康



中科软总经理左春

2004年10月19日下午，在中科院研究生院的阶梯教室举办了一场别开生面的技术辩论会，与我们以往在电视上看到的大专辩论会的唇枪舌战不同，这是一次以软件开发技术交流为主的辩论会。

本次辩论会的主题是“workflow 引擎设计与实现”。workflow 从1991年提出到现在已经整整13年，workflow 概念的提出，解决了复杂系统不断演化和开放性的需求。其最大的优点是，实现了应用逻辑与过程逻辑的分离，可以在不修改具体功能实现的情况下，通过修改过程模型来改变业务流程。workflow 联盟 WFMC 是在1993年成立的，经过多年的发展，其标准没有大的变动，理论虽然还没有成熟，但已得到广泛应用。

如何更快更好地掌握先进的软件技术并恰当合理地运用于应用系统开发中，这是当代软件企业技术开发面临的重要课题。中科软科技股份有限公司针对这一主题，结合公司技术开发的实际情况，

选择“workflow 引擎谁与实现”作为辩题，举办这次技术交流活动，旨在增强技术交流，提供 workflow 技术在各个行业内的应用水平。此次辩论会得到了业界的支持，上海普元信息科技有限公司还派出了代表队参赛，而国家863共创软件联盟负责人刘澎、Borland 大中华区总经理刘珍妮、《程序员》杂志社社长蒋涛以及众多专家的到场，为此次辩论会增色不少，并且在会后作了精彩的点评。

比赛第一场虽然两队各有特点，一队是对 WFMC 标准的理解比较到位，一队是对技术方面的实现比较到位，但是作为开发人员不善于表达的这个特点仍然比较明显，双方在沟通和辩论的技巧方面相对不够，但是在技术层面上还是做了比较细致的解说。第二场则是旗鼓相当，双方讨论了 workflow 引擎设计中的关键点技术，交易回退和补偿、系统高性能及其和业务系统的协同工作方面进行了深入的探讨，让与会者学习到不少知识。

比赛结束之后的评委点评也是本次辩论会的精彩之处。刘澎从辩论内容及其辩论技巧方面提出了许多中肯的意见，而创维科技有限公司总经理刘建明从商业化角度对于参赛各队提出了一些建议，伟创软件的总经理则认为 workflow 设计应该放眼全球，借鉴一些国外先进技术和思想。思维加速的总工宋兴烈从技术的角度阐述了 WFMC 的参考模型，也说明了 XPD 需要实际业务去验证，目前的工作流正处于发展的初级阶段。

这次大会的发起人，也是中科软的掌舵人左春总经理接受采访，他认为此次辩论会更是一个技术交流会，也希望将中科软在 workflow 开发方面的经验与业界分享，因此此次大会采用了开放式报名，他也希望通过本次大赛促进和业界的沟通，也为即将举行的第二届软件技术大会做一个铺垫。当问到这次辩论会的主要目标时，他提到了如下几点：

- ◆ 可以展示应用 workflow 引擎技术所取得的成果，希望中科软在 workflow 方面取得的一些成功能够和业界分享；
- ◆ 增进行业技术交流，提高 workflow 技术应用水平；
- ◆ 促进团队荣誉感，增强团队的合作能力；
- ◆ 在首届软件技术大会上，技术人员的交流显得欠充分，用辩论的方式可以提高交互性，吸引听众；
- ◆ 产品提供商往往扩大自己产品的功能与性能，通过辩论的方式可以挤出水份；



◆ 通过比赛和专家点评，可以了解技术发展的现状和方向，往往现实和理论之间存在一定差距。



■ 特别策划 / 本刊编辑部
■ 文 / 闫辉

迫在眉睫的**职业规划**

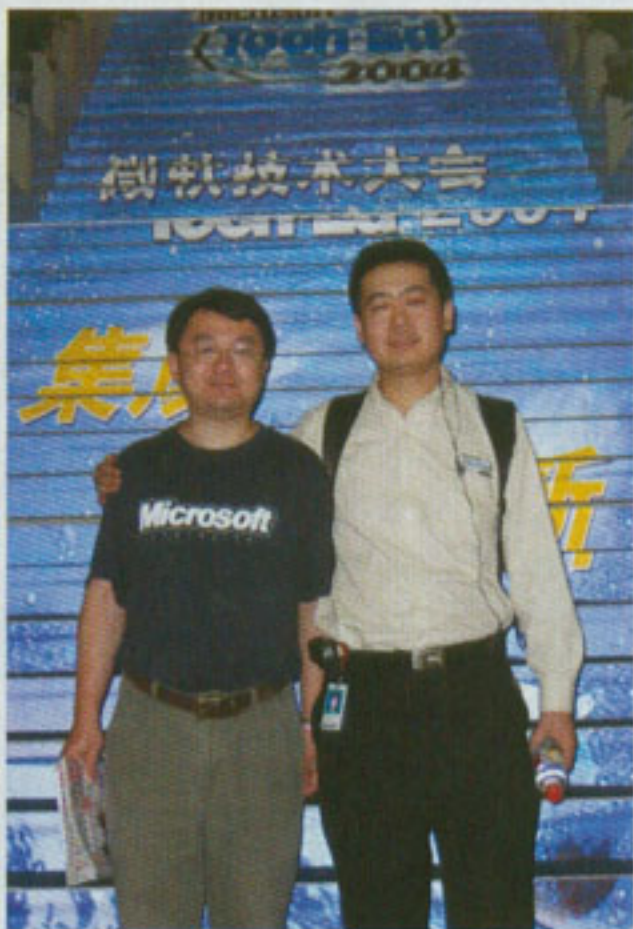
全新的技术、激增的就业压力以及分化的开发角色——做好准备吧，这是个更加需要规划的时代

对于大多数程序员来说，微软是一家值得崇敬的公司，能够加入微软，也是很多程序员的愿望。在付出足够的努力后，一旦进入了微软，也就意味着可以和最先进的技术终日为伍，一直沿着技术这条路线走下去了。对吗？错。今年九月份刚刚加入微软开发合作部的王洪超就为自己规划了一条技术管理的路线。除了在技术方面继续学习之外，他还希望在未来的时间里有意识的提升项目管理的能力。王洪超说：“微软为员工的职业发展规划提供了足够的学习机会。”

对更多的程序员来说，进入微软仍旧是一个梦想。然而，与以往任何一个时候相比，做出职业规划的必要性更加迫切。面对层出不穷的新技术，激增的就业压力，不断分化的开发角色，再加上IT发展的不明确，做出职业规划既是一种挑战，也是必须之举。

以前，学校的学生只要考取一个认证就很容易开始自己的职业生涯，已经工作几年的程序员更是成为公司抢夺的对象，而互联网的出现使得，高级程序员更多想做的是自己出去创业。

现在一切都改变了，混乱的认证市



王洪超（图右）和来自微软总部的讲师孙展波

场让毕业的学生失去了一块招牌，企业更注重其技能和做项目的经验，而少有工作经验的学生和企业需求之间形成了无法弥补的裂痕。已经工作的程序员又面临着学习软件工程规范

和技术更新换代的压力，不明朗的软件行业前景让他们在走向而立之年的路途上遭遇困惑。对于已经具备相当水平的资深技术专家或者技术领导者来说，风险投资对软件项目的谨慎使得创业变得更加困难。即便如此，仍然有很多非软件专业的人员源源不断的加入到这个大队伍中，更为市场增加了很多竞争的对象。

与企业需求接轨是学生的职业规划的第一步

很久以前，企业要承担起学生的培训工作，毕业的学生要在公司经过一段时间后，才能融入整个开发团队。而现在，很少有公司愿意承担这样的培训费用了，在激烈的市场竞争下，拿着工资却无法带来利润的职员是大部分企业无法容忍的。

大学教育是普适教育，教给学生的只是知识，而企业对学生的需求是技能。这之间就是一个很难弥补的差距。中科天博总经理王健华表示：“大学生学习完之后，只是知道是什么，根本不会用。学习了C、Delphi，学习了Java，只能够按照书本的案例照着做下来。但哪怕让他做一个最小的项目，例如包括一个带后端数据库的小网站，都很难独立承担。”前IBM软件部高级软件工程师李



中科天博总经理王健华表示企业需求的是技能，学生在这方面存在较大距离

巨锋现在担任着科瑞尔思培训中心专职教师，他也表达了同样的看法，“企业更关心你学习到了什么技能，做过什么，如何把学到的技术应用到实际中去。”

应届毕业生很难克服这个困难，因为学校不具备这样的环境。IT大环境没

特别策划

有解决的情况下，就需要大学生自己想办法。如果在学校通过某些方式已经积累到了经验，当然是最好的。但如果没有积累到这些知识，就必须寻找机会通过其他途径弥补了。

曾担任亚信公司软件开发技术总监，现在创办达内培训的韩少云也有切身的体会。“原来我在亚信做人才管理，需要不断从社会上招聘好的软件工程师，组建开发团队。我发现很难找到合适的人选，但是有些被淘汰的人是很可惜的，他的基本素质，包括计算机专业的相关背景非常不错，但具备的技能和企业不能很好地匹配。”为此，韩少云萌发了要做IT培训的念头，希望将企业需要而学员缺乏的知识和技能通过这种培训进行

弥补。创办达内科技以来也取得了巨大的成功。

对学生来说，职业生涯就是进入正常的轨道中。寻找到适合的入口，提高适应企业需求的技能也是为了寻找到适合自己的公司。面临毕业的学生要谨慎地做出自己的选择。如果找不好入口，起点太低，对未来的发展也没有好处，甚至导致以后的工作习惯都不好。金山公司负责人力资源的副总裁王春伟说：“《哈利波特3》中有一句话让我印象非常深刻：选择比能力更重要。一个人年轻也只有很短暂的几年，如果能够选择一个精彩有活力的团队，会使他的职业生涯充满精彩。如果在选择职业的时候，因为一些失误造成两至三年没有学到什么，自己也会非常惭愧的。”王健华也表示，企业是员工社会价值的附着。“大公司的工作规范和工作习惯会更加正规，其经理的素质会更好。很多人由于没有对未来进行规划，没有想清楚，工作之后对自己的岗位不尊重，经理也看不上他。”

企业对软件工程和团队合作能力越来越重视，这是企业做事的方式。要适应企业的需要，首先要学会规范文档，

然后才是技术，这样企业知道你受过正规的训练。王健华说：“我们要求学员注重4方面能力：眼界、学习的方法、技能和规范。他要学习如何与别人合作，比如代码风格要统一。虽然只是一名程序员，但仍需要站在项目经理的层面上看自己的工作，这样才能更好地合作，融入到团队中。”王春伟认为团队合作能力也是通用软件厂商非常看中的一点，“现在不是凸现个人英雄主义的年代，在金山公司，一个人如果很孤僻，很冰冷，沟通艰涩就不行。我们要求程序员心态非常端正，有良好的配合意识，个性特别乐观。”

一些学生也意识到了这点，正在软件学院读大四的刘未鹏说：“我想当杂志社编辑、从事教育工作或进入研究机构，总之得是一个能够静下心来来的地方。如果非要进公司工作，也得像趋势、金山或微软这样具有开放工作环境的公司。现在国内很多公司还是被市场所奴役，不能建立自己的个性，个人觉得不要在一个终日赶工的小公司工作。”

因此，对现在的学生来说，不断增加和企业能够顺利接轨的砝码就是职业规划的第一步。

工作后程序员的职业规划是要将技术提升与行业结合

已经工作一段时间的程序员更要注重职业规划。很多人刚毕业时充满活力，然而几年的摸爬滚打后，冲劲就会慢慢减弱甚至消失。再加上IT大环境的起伏不定，一旦无法跟上技术或者行业发展的步伐，便会迷茫。王洪超说：“以前晚上睡觉之前想事情，经常都不知道自己以后做什么。”

这并不奇怪，工作的新鲜感褪色后，就需要设法通过一个目标为自己输送动力。这便是职业规划。

程序员首先需要提高系统设计能力。

从2000年以后，软件业的编程思维和模式、方法发生了翻天覆地的转变，与九十年代、甚至2000年之前是完全不同的。但是，很多在企业工作的程序员自身的知识架构并没有跟上。中科天博谢新华老师直率地指出：“现在很多程序员不是按照应该如何设计最好做，而是按照我会什么来做的，最终造成设计思想落后。现在做项目需要的是新的、巧的设计思想。我们可以将从无数次失败中得到的理念告诉他们，包括如何学习，如何思考，技术变化的原因在哪里，其核心在哪里。这样程序员的理解力就会完全不同。现在，对程序员的要求降低了，但对系统设计的要求提高了。所以程序员必须要提高系统设计的能力，否则永远出不了头。”

而曾经从事宏观经济研究工作的李巨锋更喜欢从行业走向来谈问题。他说：“现在中国纯软件公司不多，但是，中国的经济非常好，从整个社会的经济和行业角度可以得出很多有价值的内容。很多人不了解行业的需求情况，因此做不了判断。”

跟随一个快速发展的行业，就容易取得发展，在一个发展缓慢的行业中成功则比较困难。当然，有了市场容量和机会，并不代表一定会成功，但没有这个因素成功的几率会更低。工作一两年的人，最关键的是要有一个方向感，不要太盲目，这就需要程序员有一定的判断力。

李巨锋认为，电信行业新增的市场容量每年有1万亿，而且由于是新增市场，人员也比较容易进入。此外，中国的手机市场非常大，由于无线应用的迅速发展，可以预料，基于手机和电信结合的软件方向就业前景会比较好，事实证明，很多人也在向这个方面转变。达内的成功与行业需求也不无关系，韩少云说：“从金融和电信行业角度看，对软件人才的需求每年以30—50%的比例增



金山副总裁王春伟告诉学生选择比能力更重要



达内科技 CEO 韩少云认为专业人员也需要职业培训

长。熟悉这两个行业所需要的IT技术和背景的人才，需求量是最大，缺口也是最大的。”

中国的信息管理软件市场也是容易就业的市场。现在市场上有数不胜数的小管理软件公司，这是同中国社会走转型之路相关的。李巨锋说，“基于商流、资金流、信息流、物流的行业都有相当的发展前景。通过产权交易的商流，带动资金流和信息流的发展。物流是新兴的行业，这些都需要主流的IT技术。中国的物流成本现在是20.9%，美国是9.8%，差距非常很大，这其中有1万亿的市场容量，如果软件和信息化服务占20%的份额，也有几千亿的市场，行业需求很大了。事实也证明供应链等类型的软件市场发展非常迅猛。”

而对于游戏行业，李巨锋认为现在还不能称之为一个产业，“因为它不像电信那样带动很大的一个产业链，现在取得成功的只有几个游戏，而且还主要偏重销售，不是一个上下游的产业。尽管比较热，但相比来说，至少一两年内可能不会有很大的市场容量，这就需要判断。”

李巨锋建议，已经工作两年以上的

程序员可以有几种基本的职业规方向：技术专家、软件架构师、实施顾问或销售。其中程序员最主要的发展方向是资深技术专家，无论是Java还是.NET，还是数据库领域，都要首先成为专家，然后才可能继续发展为架构师。“尽管架构师的职位可以工作一辈子，待遇也非常好，对于科班出身的程序员最为适合，但这种工作职位非常有限。”一位毕业的学员到IBM软件开发中心后，过了一年左右，开始请教其未来的发展，结合对方的情况，建议他先深入技术，因为在中国架构师需要的条件比较复杂，而且需求量也比较少。

实施顾问和销售就比较偏向市场了，除了一定的技术能力外，需要掌握很多IT以外的知识。这些发展方向对于从其他专业转入IT的人员更为适合。比如用友就培养了很多实施顾问，这些人加上行业背景，收入也很高。要做行业专家，就要比自己的行业客户还了解这个产业的发展现状。李巨锋说：“在烟草行业做，一定要了解大的趋势，比如中国最多的烟民在哪里，烟草行业的产业政策是什么，用户的需求是什么，这些信息对于职业发展很重要。IBM的一些顾问都是行业内的资深专家，他们的发展就非常具有代表性。”

对大多数人来说，首先是要专，在技术上做的比较深入，然后进行工作调整，把自己转变为某个领域的专家，第三步，根据自己的情况，决定自己做软件架构师还是高级的顾问销售，另外一部分人可能就会走向行政管理，这与个人性格能力有关。

要获得职业成长，培训也很有帮

助。韩少云说：“在北美，职业培训是一种高端的培训，即便是专业的人员也需要职业培训。一个人在一生中需要经过很多职业培训才能成为一个资深或者专业的人员。而在国内大家的观念中，职业培训还没有被大家广泛的认同。”

对于更高级的技术人员，他们所关注的就不是解决基本的生活问题了。他们所关注的是发展和成就感。从现在看，他们更为缺乏的是交流，尤其是和水平差不多或者更高的人进行交流。李巨锋说：“我建议他们做一些国产的产

特别策划

品，做一些自主知识产权的技术。比如，我们还有产品开发的部门就在做国产数据库设计。”

职业规划带动更好的成长

从中国的软件开发人员的层次看，工作几年以内处于初级水平的程序员占据最大比例，至少在50%之上，高级的人员最多也就10%左右。但无论处在哪个层面，一定要有规划，按照自己的个性



科瑞尔思讲师李巨锋认为职业规划一定要和行业相结合



微软高级副总裁 李开复

李开复谈职业规划

当我们追求理想时，当然不能忽略了实际的问题。

最完美的是能将理想和实际结合，找一份你最爱的工作。

当理想和实际有分歧时，你有三种做法。(1)是追逐理想。只要能得温饱，你从中得到的享受将超过物质的享受。(2)是先为了实际，做你不是最爱的工作。把不爱的工作做的足够好。然后用你其他的时间去追逐你的理想。(3)是先为了实际，做你不是最爱的工作，而在你不爱的工作上拚命，赚够了钱，解决了吃饭的问题，再去做你最爱的工作。

比如说你的理想是做一个诗人，但是实际上你为了温饱，你可以：(1)在文学杂志工作，虽然待遇不高，但可得温饱，也可做你爱做的工作，(2)你可以到报社工作，待遇也许较好，然后把每一分秒多余的时间（跑新闻路上等）做你的理想工作，(3)你可以作待遇最好的写作工作（如公关、市场、技术、小说），甚至找和写作无关的工作（做生意、编程……），拚命赚钱，赚够了钱再做你的最爱的工作。

你的阶段性目标在三个例子里会有显著的不同：在(1)、(2)里可能是和写作、文学造诣有关的，而在(3)里可能是和金钱有关的（每月存多少钱）。

我有一个好朋友，他也是非常有才气的作家。他在25岁时决定先赚够钱，再继续他的最爱。于是，他开了一家咨询公司，在做咨询时，他也尽力地找写作的机会（例如，我的网站上有一些报道就是他写的）。十年后，他的公司所得很出色，卖给了另一家公司。现在，他正准备作他理想的事情。

理想与实际是可以兼得的，但是你必须有计划，必须付出，必须执著。

和优势做一些规划。

一个程序员的成长，学习过程中首先要读到好书，然后是交到好的学友，找到好的老师，在这三个外围条件下，找到适合的工作环境，结合自己的特点，然后在一些重要的环节上遇到适合的人和合适的项目，这样才能成功。有些职位可能在开始的时候不适合，但工作一段时间后，就可能走上更高的职位。最近，IBM软件学院现在启动了一个“软件人才库”的项目，为的就是跟踪技术人员的成长，给他们在合适的时机提供合适的职位。业内人士说：“IT行业良性的人员流动也是必要的。”


有了一定的职业规划，就需要补充自己缺乏的经验，只有经历过足够的项目，才有可能不断积累。对行业的判断有

一定的理解之后，对一般的企业适应起来是没有问题的，但高级的人才需要长时间的积累。

在中国，除了个人的基本技能，还需要具备一定的社会资源，行业资源和资本。学生要学会在提高智商的情况下，提高自己的情商。因为，在人和人交往中情商起很大的作用。技术人员需要慢慢改变自己的一些思维方式。个人成长要有意识地积累社会资源，认识相关的人，了解相关的政策和行业发展的过程和规则，这些东西是非技术的，非智商的，只要你关注就可以得到。

机遇和经历对于职业的发展有相当的影响力。某些人或者某些事情都可能会影响到未来的发展。王洪超也认为成为微软MVP是自己的一个转折

点，由于成为MVP，便有机会接触更多的技术朋友和微软人，为自己的发展奠定了基础。

总而言之，每个人至少三年要点评一下自己：是环境的原因，还是自己个人的原因，如果是环境的原因，可以考虑是否需要换地方，如果是自己的原因，可以调整工作或者将目标设的现实一些。比如，从一个普通的程序员做起，3年时间至少能够可以做独立需求分析和设计。 

我与新浪

■ 文 / 新浪研发副总裁 李嵩波

我在新浪工作八年了，到新浪之前，我曾在一家公司写过出租车上小打印机的驱动程序，那是我的第一份工作。

1995年的时候，我就在网站上认识了汪延。大家都觉得做网络一定会有机会，当时还用TCP协议写的Socket工具来聊天，每当有一些收获都会非常兴奋，觉得在做一些改变我们生活的事情。

1995年底，我通过汪延认识了王志东。当时开发一个可以浏览各种内码网页的工具，放在网站上对软件进行宣传和支持，效果不错，当时就感觉网络技术一定有发展空间，企业会愿意把互联网当成一个推广的平台和与用户沟通的平台。

1996年我到了专门做互联网的部门，只有四个同事。我们当时做一个原始的“门户网站”，就是做一个窗口，在里面做一个天气、黄页、分类站点索引。后来我们还汉化了一套程序做了“谈天说地”的论坛。很可惜我们当时不知道如何把互联网当作一个行业来创业，只是以部门的形式在软件公司里面操作，所有的业务都要帮助和配合公司的主流业务。

1997年，国外各种各样的模式都起来了。尽管看到了各种各样成功的案例，总感觉很多产品已经做到了我们没有能力去做的地步了，类似浏览器等产品，可能明明知道这里有市场机会，但自己抓不住，因为我们不可能做出这样级别的产品出来。我们部门当时只剩下了两个人，很庆幸竟然在那样的环境下坚持下来了。

1998年，我们借助网络上直播十强赛和98世界杯获得了前所未有的体验，也给了我们激励。由于国外已经有网络公司上市或合并的积极消息。公司认为互联网本身就是可以发展的行业，所以对这部门也有更多的投入。那时我们学到了垂直网站设计流程上的经验，不同的栏目用什么架构来做，如何确定自动和手动的规范。

后面的事情就是大家在新闻报道里都知道的，包括新浪的成立、与华渊的合并……

2000年的时候，我到北美新浪工作。当时北美是新浪的总部，因为有上市背景，市场发展也很迅速。

我最初从事的业务还是非常技术型的。在硅谷的公司工作，不管是创业的小公司还是大公司，里面员工都把责任心看得很重。他们在接手一个工作之后，会先做规划，过程中要各处沟通，最后要做汇报。这不是一个公司要求或是流程的东西，不是硬性的，而是一种很自然的行为，大家都是这样工作的，把整个事情都当作自己的责任。

在那里，技术领导会给你很多空间，但在整个过程中会和你做很多沟通，他会很客观地评价你能不能做这件事情，而且到最后如果事情失败了，他不会说都是你自己的责任，他也是直接要担责任的。

到北美工作对我是一个重新的开始，而且时机也很好，北京新浪在业务成长上很快速，在资源互通和项目协调方面，由于我有语言方面的优势，位置很重要，自己也在不断成长。当然，这也同之前的积累密不可分。

四个月前我回到新浪北京，负责新浪的开发管理工作。现在，我们最看重的有两个方面：一个是项目，一个是有潜力的新人。因为承担着非常重要的责任，希望能够提供更好的平台和工具，通过引入新的东西，让每个人觉得自己的工作更轻松、合理、有成就感，同时

特别策划

也更有激情地工作。比如，为了把工程师的工作从重复劳动中抽身出来，我们在做一些将功能标准化的工作。

网站技术领域也没有什么不同，我觉得所有技术领域都一样。没有什么技术不是互通的，但是要找精通所有技术的人是不可能的，而且你想要找到这样的人非常难，重要的是培养综合能力，每个人也要有某领域的专长。对于每个人的职业规划，因人而异，但是前提是的三个原则：专业、尽职、敬业。大家都是规划的，做开发几年之后，很自然都会流落出一些对将来的规划。

我希望用好的方法和结果，使一个技术人员成长为资深工程师，以后可以做产品的主导或者关键技术的主导，让他们来证明新浪是一个开发人员很好的成长平台。甚至要让员工明白，只要成为某一技术领域的专家，技术可以做到五十岁。

我是与新浪共同成长起来的，现在也必须帮助我们的开发人员考虑三十岁以后要做什么，这是对员工负责的事情。

做全球最火的 软件开发中心

微软亚洲工程院已经成立了将近一年，外面的报道虽然非常少但是工程院的成长速度远远超出了想象，最近，他们又在大学中召开了新一轮的招聘工作。为了探究微软亚洲工程院，我们联系到了工程院技术总监林斌，他在给员工做“完美工程”培训的紧张工作中抽出时间与我们进行了愉快的交流。

■ 文 / 闫辉 刘婧



微软亚洲工程院技术总监林斌

《程序员》：微软亚洲工程院成立之后，您的工作职责包括哪些范围呢？

林斌：我的工作职责总体上比以前增多了，之前，我带领一个新技术开发组继续负责研究技术到产品，新技术开发组比以前规模要大，到去年年底已经有20多人，每个人员的能力都比以前要好很多。工程院成立之后，我负责工程院所有项目的开发和测试的管理工作。

微软产品开发流程的三只脚就是开

发、测试、项目管理。在工程院也是分这三个部分：开发部分包括编程人员、测试部分包括测试工程师、测试主管、测试经理、项目经理。

副院长张益肇和我两个人有分工。我负责开发和测试，张益肇负责项目管理。并不是每个开发人员和测试人员都向我汇报，他们会以项目为单位，项目经理针对实际项目的进展向我汇报。

工程院开展了一系列可以让所有开发和测试人员都受益的事情，比如说培训，本周我们正在做一个从早到晚，八点到五点的内部培训，是美国总部的专业培训讲师过来做的专门针对新开发人员的培训。

还有就是开发流程里面，每天都要有很多代码要共享，如果有好的积极的事情可以互相分享的，我会在中间做很多工作。比如，如果有某个小组在设计、安全性等方面做得非常成功，值得别的组借鉴，我们会促进小组之间的交流，让开发和测试人员共享新的资源和知识。这部分工作就是从工程院成立以后就来做了。相比以前，就是除了研究到技术的部分，还有推进工程院的各个工程的工作。

《程序员》：现在有了工程院，微软

亚洲研究院的技术在产品转换方面有了哪些不同？

林斌：其实，和原来一样，最终目的都是希望把技术做成产品。不同的是，工程院可以加速研究技术更好地转化成产品。

例如，如果一个项目原来在研究院的时候只有开发人员，那么，我们能够做的就是受限制的，当产品组有需要这个技术时，我们会投入开发人员进去，但是可能到做出来之后却没有相应的测试人员，或者是产品组有三个研究人员，只有一个开发人员，那就没有办法做产品了。

对于产品组来说，他们最希望看到的当然最主要的就是新技术。然而，从技术到产品还有很大的工作量在里面。一个技术完成后，代码是不是符合规格要求，接口如何与以前的几千万行的代码结合，以及最终的测试，都需要很多的工作要做。这些在以前我们是无法提供的，原来的工程师都是在写代码，但不负责做测试，项目管理以前也是没有的。

工程院现在可以提供从开发到测试甚至到项目管理的人员，以前可能只是项目里一个很小的功能给我们来做，而



现在可能把整个组件和很多个功能都放在工程院做了。这样就形成了一个良性循环,一方面对产品组的支持力度加大了,而且做完之后直接形成产品,能够做的技术转移比以前更多,而且我们不仅仅是开发,还包括了测试和项目管理。有了这个机制之后,研究院可以把更多的技术变成产品。产品组也很高兴,因为现在有了足够的资源来帮助他们实现一些产品的想法。

《程序员》:微软亚洲工程院开发的产品会是什么样的形态?是一个功能模块还是一个软件产品?

林斌:这两类都有。第一类,是从研究变成产品,占差不多一半的数量。这一类每个项目的转化形式也有不同。比如已经发布的微软多媒体中心就包含了研究院开发出的红眼识别及修复技术,还有相片自动剪辑技术。在整个开发中,工程院一直与整个产品开发组的进程保持同步。开发组在总部,一共有400~500人,我们进行平行的开发,利用微软的源代码管理工具共享开发的进程。第二类是一个组件,区别以前的技术组件,这个是一个功能组件。我们现在可以接受这样一些技术已经很成熟的整个功能组件的开发;另外就是整个产品的开发完全由微软亚洲工程院主导,涵盖了从无到有的过程。

《程序员》:项目的成长情况如何?一般一个项目平均的开发周期有多久?

林斌:现在很多大的项目基本上比刚接手的时候在人员和技术上成长了一倍,还有三四个项目成长了50%。对于开发周期,每个项目都不同,少则几个月,多则一年甚至三年的时间。

《程序员》:我们了解到,现在有很多是从美国微软总部回来的工程师,他们主要从事哪些方面的工作?

林斌:去年年底刚成立时,从总部回来的有我和王健(开发主管)。到今天,差不多有十多个人了,刚才我还在面试一个准备回国的华人工程师。他们当中有开发人员、项目经理、测试主管都有,而且在微软的经历都很长了。已经回来的和正在回来的工程师中,他们的背景都不一样,但他们都对在国内做一些事情非常感兴趣。

但是有一点,不管是从国外回来还是国内大学出来的,只要是一样有实力,那么待遇是一样的,我们基本上没有招海外刚毕业的,都是招收有工作经验的人员,有一个从国内毕业的例子,大概有6~7年的开发经验,在公司也是做到主管级的层次,所以,我觉得国内的学生不用太担心这方面,只要有实力就行。

《程序员》:我们看到在国外回来的人员有很多测试方面的人才,是不是国内很难寻找这样的测试人员?

林斌:其实在技术上是没有什么问题的,主要是思想上的。这和国内的教育有关系,因为国内的教育没有这部分内容,可以说是一片空白。有很多老师只是把教材上面的内容说一说,没有实际操作的机会。很多学生对测试没有概念,以为只是用鼠标随便点击,而事实上很多测试是需要做测试案例,写测试工具。包括WEB的大容量测试都是需要一种思维的。

但是,这个问题并不是很难解决的。一些新员工经过我们的培训之后,开发人员可以很快上手这部分的工作,也证明了这是一个思想上的问题,而不是技术上的。

《程序员》:您认为微软亚洲研究院和亚洲工程院还有哪些不同的地方?

林斌:这里不像研究院可以鼓励大家去创新、冒险,在研究院,假如冒100个风险最后出来一个成功的产品,那么整个项目就是成功的。但是工程院不是这样,做一个工程投入很大,公司不会轻易

启动项目,要做很多调研之后才会实施。所以项目到了工程院是不允许失败的,必须成功。每个项目过程中是不允许出现任何严重失误的,比如什么时间能够发布,质量上每个设计和每个代码是不是符合要求。当然,市场上的成功是另外一个方面了。

《程序员》:您认为微软亚洲工程院目前员工身上具备的突出特质是哪些?


林斌:首先,是每个成员都很有激情,能感觉得出来,有两个方面的原因,一是每个人都很有责任心,二是他们对项目都很热爱。国庆节的时候,很多人自发地继续工作。

其次,就是追求完美,如果一个项目,工程院能够胜任,我们不仅要做,还要保证做得更好,才能吸引他们把项目交给工程院来做。所以我们无论在高层的理念上和很具体的工作上,都要力求完美,这是我们追求的方向。

在工程上这个体现得很特别的,就象我们现在的培训,原本是脱产的,没有要求开发人员继续工作,但是很多人为了能够不影响项目的进程,就在参加培训之后抽出自己的私人时间来继续项目的研发。

《程序员》:我们看到有一篇国外的报道说研究院是“全球最火的研究院”,是不是工程院也希望做成“全球最火的软件开发中心”呢?

林斌:当然,这也是我们的希望。虽然这个机构是在中国,但我们面向的是全球的产品研发;比如多媒体中心的产品,他是面向全世界7个国家语言版本的,这对全球的用户都有帮助。当然,我们也会做很多专注中国市场的产品,因为在中国的市场环境里面,所以我们来做这部分很适合。

本刊将继续深入采访微软亚洲工程院的软件开发管理和技术主管,敬请关注。 



毛一丁

北京瑞星科技股份有限公司副总裁，负责公司市场推广工作。设计瑞星全线产品及公司战略、品牌等相关推广工作。之前曾在北京金山软件公司、长城集团、连邦软件、8848、中文之星等公司任职。

软件营销 与程序员

■ 文 / 毛一丁

老百姓买软件买的是产品和后续服务，是对厂商的信心，买的是一个安全感

在中国，科特勒的6P营销理论还是被认为是做市场最基本的原则。其中包括product(产品)、price(价格)、promotion(促销)、place(渠道)、政治权力(Politicalpower)和公共关系(Publicrelation)。不过，软件也有它自身的很多特点。比如，软件更像音像制品，它不是一个实体，在用户接触到之前只有一个名声而已。所以，这也决定了它不能像硬件那样销售，比如硬件可以用大量的广告费来砸这个市场。软件厂商一般都没多少钱，因此需要考虑其他方式来影响用户。

2001年9月份，我着手创建一个瑞星病毒预报系统。这是我做连邦软件排行榜时形成的思路，当然也要付出代价，这就是要能忍受一两年花钱但效果不明显。不过现在我们可以看到，全国有100多家媒体提供我们的病毒安全预报，已经成为了一个相当有影响力的品牌。其实，很多人可能都看不懂病毒播报，但会知道这是一个善意的提醒，一旦发生问题，首先会想到瑞星。

软件营销是积累的过程，但首先要确定策略。我到瑞星之后，公司重新定义成最具价值的信息安全产

品和服务提供商，形象有了很大的调整。这样，产品线也在不断扩充，2002年发布了自主的防火墙产品，现在还有了漏洞检测、VPN等网关级的产品。

除了产品线的扩充，渠道的梳理也是软件营销的重要环节。单机版软件作为消费类软件，需要打造品牌，需要有非常方便的渠道、强大的铺货能力，相对较高的性价比，让消费者喜闻乐见，像肥皂、洗发水那样易于采购。因此，除了连邦、赛乐氏等软件连锁企业，我们还与大的软件批发商结成了很好的关系。这些批发商每家手中都有上百上千家小店铺，他们从厂家吃货，然后分给非连邦的小户。现在的大批发商有40多个，每家都可以进上千上万套的量，大大拓展了渠道结构。

我们还发现，很多时候大家不买正版是因为采购渠道不通畅。反病毒产品有一种类似药的特性，只有在生病之后才买，而且会倾向买真药。于是我们率先开辟了网上直销，打个电话，或者发电子邮件，就可以送货上门。去年5月18日，瑞星专门推出了一个下载版，把配送变成下载方式，并开通了各种各样的付费渠道，比如声讯电话、短信服务等。我们用网上销售的电子商务方式从盗版手中抢占了非常大的一块蛋糕，现在这些服务占到了我们营业额的10%。我总结出一个字：我们挣的就是人的“懒”。不过，做电子商

务也要耗费很大的精力。首先研发部要开发一套定制的产品和电子商务系统。而且要进行很多支付系统的商务谈判。

现在优秀的软件营销人才越来越多，网络游戏的商业模式已经锻炼了很多新人。很多做游戏的经验也可以借鉴。比如仿照游戏点卡销售的方式，我们也推出了用户可以买一个月服务的举措。

用户购买软件产品是个很简单的流程，但你有没有想过其核心是什么？其实老百姓买的是产品和后续的服务，是对厂商的信心，买的是一个安全感。知道了这点，作为厂商来说需要关注什么呢？那就是服务，我们为了将服务做好，购买了无数的服务器，而且提高快速反应能力，将原来的升级服务从一周一次扩展到现在一周五次升级，这样用户对厂商的感觉会好很多。

此外，我们还积极开辟大陆之外的市场，除了做一个英文的共享版提交到Download外，还推出了香港版以及日文版，利用当地的渠道商拓展渠道。日本代理设计了漫画风格的产品包装，销售价为1800日元（合人民币500元）。现在我一边总结经验，一边在和加拿大、罗马尼亚、澳大利亚的商家探讨合作。

给我100万，我只敢用30—40万投入到开发中

从一个好的主意，到组织开发、测试完成，这是产品化过程，但这只是产品周期中很小的部分。而后的商品化过程更需要优秀的人、好的创意和商业化操作、好的渠道和后续的服务，这样才能将产品导入到市场中，完成整个生命周期。很多公司认为反正有连邦，做完之后连邦就会销售我的产品，我就有钱了，因此在产品阶段就花光了所有的钱。问题是连邦会不会来销售你的产品，即便进货，如果老百姓不认，回头还会退给你的。所以说，如果给我100万做一个产品，我真的只敢用30—40万来做开发。

以前我和雷军还商量过能不能做一家软件出版公司，帮国内的一些软件开发团队包装他们的产品，但后来放弃了。其中除了知识产权保护的原因外，开发者往往觉得自己抱着了大金蛋，舍不得交给别人，在这点上和开发者的沟通很困难。而且上市之后还有可能赔钱，双方的责任无法划清。

现在软件市场中还经常出现这样的情况：有了一个好主意，没有想好就做了，而且把所有的精力和财力

全部投入，但由于产品定位不准确，与用户需求不贴近，做出的产品也不适合市场做销售。有个典型的例子：早期王码字形的输入法取得成功后，市场上出现了无数的输入法，这些输入法仅仅是一个好主意，但没有意识到老百姓只能把拼音认全。现在市场基本上还是以拼音和五笔为主，其他的输入法都是昙花一现。


前些天，有个朋友请我帮忙：他们投入几十万开发了一个互联网黄色信息过滤的软件，也生产了几万套，但不知道如何进行销售。我告诉他们犯了几个错误：首先，软件一定要有订单才能生产；其次，产品定位就有问题，这是一个窄众市场。反黄软件是限制用户，降低效率的产品，潜在的受众是家长，但关键是现在的孩子比家长电脑水平还高。

应该说，做出一个软件挺不容易的，但营销对于公司运作的重要性更大。有人更极端的说过：软件卖出去就是钱，卖不出去就是垃圾。因此，软件公司还是要以市场为导向，挣了钱才能继续对研发进行大的投入。

程序员的数量并不大，但这些人的影响力非常大

现在，有些程序员志向比较大，也想成为软件英雄。不过，以前经常是一个人做一个很普及的通用软件，那是创造个人英雄的时代，但现在不是了，大家要学会相互之间的协作。中国人做软件都很聪明，但我们差在合作精神上，每个人都想做老大，开发管理就会有问题。原来取得成功的个人并且现在也成功的，都是后来带出团队的。

程序员要清楚，在公司做不仅仅是挣工资，而是要清楚你所开发的庞大软件能够实现怎样的价值，否则就没有方向感。程序员还要对程序有追求，以前在DOS系统上程序都做得非常精巧，比如PCtools几乎完成了DOS所有能够完成功能，但小到一张软盘就可以安装好几份。现在的程序员更多借助快速开发工具，软件庞大不堪。做通用软件实现功能不是本事，而如何高效率地实现功能才是本事。

我想，程序员主要分两类，一种是有思路，有干劲，这种人能力很强，不妨去做一个共享软件出来，互联网的销售支付越来越发达，可以一个人面向全世界市场。还有一类可以在企业中工作，他非常擅长做自己喜欢的工作，将其当作一个职业一直做下去。 



独行五年

开发交互图形控件

■文 / 张广军

毫无疑问，本人非常适合软件开发！

我1993年才开始学计算机。那时刚接触Windows，行业里正流行可视化开发和多媒体生成系统，于是我尝试着做一个类似VB的开发环境。由于是初学，做了三个月才稍微出来一点眉目，可是教授看了之后，说我这方面的智商不够，不用做了。但因为喜欢，我后来也一直利用业余时间做软件开发。

1994年，我来到北京，希望找一份编程工作，但当时的打工环境非常差。

因为没有北京的身份证，找工作也很受影响。曾经去联想公司应聘，但当时他们要求有北京户口，这个门槛也很难突破。还曾经参加长城软件公司的面试。他们招聘人员做日本软件的汉化，当问到我对薪金的要求时，我提出1200，他们非常惊讶地说：在我们这里，连博士生都才800。

在北京生存得十分艰难，八个月后，我无奈地“转战”广州。

恰巧，巨人集团招聘编程人员。我在自己的简历封面写下了这样一句话：“毫无疑问，本人非常适合软件开发！”

在巨人工作的头三个月里，我被安排做其他工作，可是我非常希望能够做编程工作。在努力争取之后，我得到了两个机会，也通过这两个任务为公司创收四万元。可是在公司仍然一直得不到重视，于是，工作了8个月后，我走出了巨人。其后，又经历了一家台湾公司，也依旧不是自己理想中的软件公司。

总之，我经历了差不多十几个软件公司，几乎都是经营不景气，于是很失望，对进入一家软件公司做开发工作再也提不起兴趣。

真正开始做 Visual Graph 是在 1999 年

过了几年出外打工的生活，我于1999年回到了家乡山西，经朋友介绍，进入一个大学教授创办的软件开发公司。

这家公司曾经从北京请到一个院士级的编程高手做软件开发，但是开发进行了三个月却没有任何结果。所以和我谈合作之前，他们给了我一个开发项目，我用15天完成了。于是签约合作，按照合约，做出合格的产品之后，我会有25万元的回报。

可就在我做了四个月的开发之后，



公司因产品运作不善而倒闭。我最后也只收益了5万元。但是因为不甘心做出来的产品就这样荒废，我立刻花了2万元买来一个笔记本电脑，亲自去找客户，向他们演示我的产品，到过珠海、深圳等地，却始终没有结果。

其实，想走上创业的道路，会有很多的干扰，比如家人的期望，生存的压力。到了2001年，我才开始决心摆脱一些干扰因素，专注于实现自己的想法。

Visual Graph 技术含量比较高，开发它花费了我很多时间和精力。它是一个很综合的产品，要实现的功能很多。坚持开发这样的一个软件，对于我来说，几经挫折。虽然想法很早就有，但一直因为种种因素没能付诸实施。2000年之前它一直是我的业余爱好，直到2001年我才抱

着破釜沉舟的心情开始专注于开发这个软件。这三年期间我专心研发、完善这个软件，非常辛苦。偶尔会接一些零碎的开发工作，保证自己的生存。

算起来，从99年那个雏形开始，Visual Graph的开发、完善历经5年。

我就是要做通用软件

我一直觉得做软件就是要有创新精神，不能走在人家的后面，不要总是跟着别人的脚步。要做就做别人没做过的，另辟蹊径！

“交互图形控件”非常适合工业自动化监控，在电脑屏幕上通过鼠标操作图形，让它和数据库发生关联，有点象FLASH一样的界面，就是一种“交互”。

但是，我想做的是关注行业里的具体应用，有针对性地给公司用户做产品，而不是做大众休闲类的软件。而且，我要做通用软件！

很多人不相信我们能做出这样的通用软件，但在我看来，这完全能够实现。

但是大软件公司不会去做这样耗时的产品，就象很多大的软件公司不可能做中间件，比如微软。它们不会把它当成是核心技术，即使做这一类的软件也很少做得彻底，只可能是做出一个专用软件。但是从专用软件到通用软件，还有很多因素要考虑。如果要做一个专用软件，可能四五个月就可以完成了，但要做通用软件，就必须考虑标准化、接口统一、稳定性、兼容性、文档的全面、测试等方面。

但是我有信心沉下去做好它，也相信一旦做出来，它的用处很大。因为我自己有图形处理、编程语言、表格处理等方面的技能，我毅然选择了开发这类软件。

不正规导致销售的艰难

Visual Graph从去年到今年升级了

0.3，已经更新到4.5版本了，技术上可以说已经非常成熟。

但我们遇到的难处是不够正规，所以我不方便做大的宣传，宣传不够，就只能主要靠老客户的支持，新客户也多是通過老客户介绍的。直到去年在CSDN网站做了一个广告才开始慢慢有了市场回报，所以我的目标就是要走正规化的道路。希望近期在北京注册，但我并不追求快速做大做强，而是想稳扎稳打，不好大喜功，一点点地与客户建立信任关系。

当时产品开发出来之后，我就曾在CSDN上发帖，但是没有人相信我的产品是正规的。只是有很多人在留言要和我商量合作事宜，并没有人购买我的产品。

不过如今我的软件已经销售出了20到30套，总共收入四五十万元，其中有一大半是老客户，其中包括老客户给我们做宣传，影响到的那部分企业。

在中国销售软件非常难，因为国内客户最初就抱着不信任的态度和你交流，这也是中国软件产品总是出现各种质量问题以及售后服务不到位等现状导致的。

我一直都是单干，曾经有人希望合作，但最终都没有实现

Visual Graph的开发和销售，一直都只是我一个人。不过有一位朋友不能不提，他就是宋清海，在电力系统工作。他99年的时候开始用Visual Graph，2002年时主动联系到我，告诉我他真的特别喜欢这个软件，非常希望见到我本人。于是非常热情地邀请我去他家里，给我提供了很好的调试环境，并提出了很多完善软件的建议，一直在帮助我完善Visual Graph。我一直非常感激！

但是，我本人并不知道到底怎样去做一个公司，所以由于性格和认识等等方面的原因，一直没有真正去做个软件公司。以前也只想技术，对市场和政府政策完全没有概念，虽然很多人想投资或

者合作，但是几次不愉快的经历让我最终放弃。或者是技术上观点不统一，或者遇到些很急功近利的人，希望做游戏一类的所谓“赚钱快一点”的软件开发……

如果再年轻一些，或许我还是会选择打工

如果再年轻一些，或许我还是会选择打工，创业其实是被逼上梁山的。

国内的大环境确实不好，如果是我做软件公司的老板，我可能会很尊重程序员，因为他们确实很不容易，我会在人为上关怀、尊重他们；但是从经济方面来看，待遇也很难提高，因为软件公司也有自己的难处，生存下去也非常不易。

可是，我也是一直到30岁以后，才突然觉得自己根本就无法再打工了。一来心态上不能接受，因为做软件这么多年了，一直没有什么象样的产品出来，觉得很遗憾；二来年龄渐渐大了会越来越难以找到编程工作。

最终决心把精力关注在Visual Graph的开发上。

一定要有技术实力，而且要稳得住，沉下心去做技术

对于程序员，一定要磨练出技术实力，并且沉下心去做事情。然后，我建议你把你开发出的软件买得贵一点，一定买高价！Visual Graph有2兆大小，我不希望买得多，但是买出一件产品我就一定会对客户负责。

Visual Graph售出时不同时出售源代码，但是我售价几万元，我的目标就是做好每一单，对客户负责到底，建立彼此的信任。这样你的软件产品才能健康地发展，走得更远！

评论嘉宾



北京用友华表软件公司总经理唐爱平

曾开发了自主知识产权的电子表格 CELL 组件, 并成功推广到全球各地, 在中国拥有巨大的市场份额, 最终以一千多万元的价格被用友软件公司收购。

创业

嘉宾评论

第一桶金的掘取需要的是激情与专注

国内软件业的发展经过了两个时期, 一是 90 年代初期, 诸如 WPS、自然码、UCDOS、CCED 等都是个人英雄的成功; 二是 90 年代末期, 互联网作为一种新生事务, 成为了个人英雄的摇篮, 网易、FOXMAIL、网络蚂蚁等也成为互联网时代的弄潮儿。

随着产品的日趋成熟与市场推广手段的不断完善, 个人英雄式的成功越来越少了, 但只要新技术出现, 个人英雄就不会消亡。

这种针对图形界面的产品有一定的市场空间。因为对于应用软件来说, 最重要的是业务, 界面设置这部分一直是个难点, 也是容易被忽视的地方, 一般企业级应用都讲究把需求放在第一位, 但实际上外观和操作易用性对于用户来说同样非常重要。而且, 从程序设计上看, 这部分编程也是最费精力的。但是, 这部分市场空间相对于整个软件行业来说其实很小, 不过对于程序员个人挖掘第一桶金还是足够的。


程序员的优势在于他对技术的把握, 要打破目前的市场平衡体系, 新产品是最好的武器。我觉得程序员创业不要考虑太多的市场因素, 而是应该沉下心来把产品做好做完美。

我也接触了很多创业的程序员, 常常沉不住气, 在产品做到一半的时候就做不下去了, 忙着到处找投资, 考虑各种市场因素想要推广产品, 却经常无功而返。虽然靠一个 idea 成功的例子不是

没有, 但是我不建议创业者有类似买彩票中大奖的心态。

经营了三年市场之后, 我有一种深切的体会, 那就是实际上对于一个优秀的程序员来说, 挖到第一桶金是不难的, 只要静下心来做。然而, 从程序员到产品专家的转变却比较困难, 需要经历过一定时间的经营才能形成对市场的准确判断。

我现在已经不再写程序了, 经过市场经营之后才明了编程和市场推广是完全不同的两件事, 而且一个人不可能同时做好这两件事情。似乎程序员与市场天才似乎总是格格不入, 从程序员到产品专家的转变总是很难实现, 只能看市场的机遇和程序员对自我的认知程度。但也有成功的例子, 例如盖茨, 例如丁磊, 如果能够完成这个转变, 程序员就可以在**第一桶金**的基础上攫取到**第二桶金**。

华表和用友合并以后, 最大的想法就是怎么把已有的产品做大。我们大概用了一年的时间, 尝试过很多市场手段, 但都没有成功, 最终才意识到一个基础的工具产品是很难做大的, 它实际的市场空间就很小, 但是我们决定继续保持它的优势, 因为它有很好的现金流。在和用友讨论如何更多地体现华表在技术方面的优势时, 我们决定进入国内的企业级应用软件领域, 所以华表这两年一直在做一个企业级应用产品, 叫做“用友销售通”, 它可以帮助企业改进销售过程, 提高销售人员的管理能力, 类似于 CRM。 

跨平台的 **BT** 开源项目

—— Source Forge 最佳推荐

■ 译 / 才子英

【作者介绍】

项目领导者资料:



Chalouhi Olivier

年龄: 25

职业或经历: CEO

教育经历: Diplome d' Ingenieur Supaero, 法国 (硕士 equiv.)

住所: 法国



Paul Gardner

年龄: 38

职业或经历: 系统架构师

教育经历: BSc 计算机科学, Warwick 大学, 英国

住所: 英国

主要开发者:



Goussard Olivier

年龄: 26

职业或经历: 待业

教育经历: Diplome d' Ingenieur Supaero, 法国 (Masters equiv.)

住所: 法国



Alon Rohter

年龄: 24

职业或经历: 开源程序员

教育经历: 计算机科学 BS

住所: 明尼阿波利斯, 明尼苏达州 (美国)

【项目描述】

Azureus 是一个用 Java 写的优秀跨平台 BitTorrent, 包括一个

种子定位器和种子制作器, 因此它能够提供运行 BitTorrent 所需的全套解决方案。BitTorrent 是一个使用点对点技术的文件分配协议, 它能够减少在中央服务器上的直接负载。使用 BitTorrent 协议, 客户端不再是仅从一个单独超负荷的服务器上下载, 而是可以同时从其他上百 (甚至上千) 个同等的客户端下载, 进行高效高速的内容传输。在 new-from-the-ground-up 协议的执行上, Azureus 与其它 BitTorrent 客户端软件有所不同, 它还更容易使用多下载的图形界面, 并且支持更多的外挂设备。

【作者访谈】

◆ 为什么开始写这个软件, 是如何开始的?

Olivier: 我们开始开发 Azureus 有两个原因: 在 Eclipse 平台上试验标准窗口小部件工具箱 (Standard Widget Toolkit), 和提供一个比其他 BitTorrent 客户端更好的 GUI。随着这个项目获得更多的成功, 最初的目标也发生了变化, 现在我们在尝试做出最好的 BitTorrent 客户端。

Alon: 我一年前首次安装了 Azureus, 那时它还处在幼年阶段。由于软件有一些困扰我的错误并且一些功能不能满足我的需要, 于是我带着对 Java 程序设计的渴望, 下载了 Azureus 的源代码并开始研究它。我对这个项目所做的第一个真正的贡献是安装每晚 (nightly) CVS 网页快照, 这个功能现在还保留着。我对代码更加熟悉以后, 便开始对项目做一些基础上的修补, 从那时起我实际上成为了一名核心开发者。像许多人一样, 我最初作开源项目是因为心里痒痒的想写些东西。

◆ 有哪些特定的用户?

通常是些终端用户, 但我们正在为想要使用这种技术的公司建立完整的解决方案。Azureus 能够管理供应链 (distribution chain) 中的每一个步骤——处理原始资料发布, 数据服务和终端用户下载。因此, 如果人们想要下载最新的 Linux ISO CD 映像 (image), 给客户传输演示 (demo) 和软件产品补丁, 在公司内部发布和同步信息, 这种技术将是非常有用的。

◆ 你认为多少人在用你的软件?

我们并不知道确切的数字,但我们相信,每天大概都有20万人在用Azureus。据SF.net近期的统计,Azureus有超过1000万的下载量,但这个下载量中包括对软件的升级次数。我们还相信,超过1000万的用户至少使用过这个软件一次。

◆ 能不能举几个典型的例子说明人们如何使用你们的软件?

当最新的虚幻竞技场(Unreal Tournament)演示发布的时候,我们利用Azureus给成百上千的狂热爱好者分发这个演示,改善了过量下载造成下载服务器超出极限带宽负荷的情况,使情况更易控制,所有的人都在更短的时间片内把文件下载到了硬盘上。在局域网应用中,Azureus被用来在机房中作大学课程多媒体课件的镜像,也能够为成功的LANParty分发所需要的文件。当然,Azureus也使用BitTorrent技术来自动分发自己的升级文件,这样也能减少SF.net自己下载服务器的负荷。

◆ 是什么让你们意识到项目变得越来越成功?

当论坛的帖子,用户的错误报告和功能需求开始显著增长时,我们意识到Azureus成功了,因为这意味着人们兴趣的增长和用户基础的快速扩张。当我们开始扩展API插件时,我们看到了用户在捐赠模块时的热情澎湃(像IRC文件发布bot、PeerGuardian IP过滤和基于RSS的组合内容订阅)。我们也不能忘了那些骗人钱的站点随后雨后春笋般地冒出来,就是打着Azureus的名号,诱惑那些毫不怀疑的用户感染特洛伊木马或者为软件的拷贝付钱。你知道,如果你的软件被人用来耍手段,那证明你的软件正走向流行。

◆ 最意外的惊喜是什么?

Olivier:我开始做这个项目时,我对开源世界一点都不了解,至少不是作为一个开发者去了解的。我最大的惊喜是看到自己可以得到多大的支持,看到人们是如何潜心于自己的项目。

Alon:除了核心开发者,还有很多人贡献他们宝贵的时间来提交补丁、测试代码、写文档、做翻译、提供技术支持、表达他们的思想和建议,甚至是捐赠。人们喜爱参与进来。Azureus是我第一次经历开源开发,整个过程已经给我留下了深刻的印象。

◆ 曾遇到过什么大的挑战?

Alon:开源运动仿佛带来了一种令人难以置信的社群参与感。管理这样一个有众多不同用户和志愿者的项目,会造成相当大的时间延误。当我们开发者自己难以承担时,我们依赖于社群的支持和帮助。我们不懈努力来寻找新的促进与社群交流和社群内部的方式,以期达到非开源商业软件可能达到的质量水平。

◆ 为什么你认为大家已经接受了你的项目?

我们提供了强大的软件,它既有丰富的功能又非常易于使用,而且几乎能在任何操作系统下运行。我们关注用户的阐释、需求和期望,在整个开发过程中我们也尽可能的参与进社群,我们一直在通过这些努力完善Azureus。

◆ 你认为你的项目将会如何发展?

Olivier:还有很多工作要做,还有一些核心的类需要重建。现在的许多功能与我们自己的想象和使用者的需求相差太远了,我们必须建立一些新功能。我们希望更多的人为Azureus开发插件。从我们这方面讲,这意味着开放更多的核心源代码。

Alon:我们希望Azureus成为每一个人最喜爱的BitTorrent客户端,我们还希望能通过持续精炼和提高已经很不错的下载效果,达到我们最终的目标。同时,我们还在努力提高BitTorrent在商业方面的可用性,我们将使用BitTorrent在商业方面分发或传输数据的技术视为未来点对点科技的前沿。

◆ 对你的项目有什么发展计划?

Alon:为了提供更简单可靠的使用,我们希望能对所有东西进行自动设置和自动优化。我们希望建立查找功能,来以更加简单的形式定位可下载的数据。

Olivier:更多的插件,更多的用户。

◆ 最令你自豪的是什么?

我们的开发团队。

◆ 你如何协调这个项目?

我们主要使用IRC来交流,当我们只有四个开发者时,这样比较易于管理。程序的错误会分派给最合适的人来调试,但是一个开发者也可以去纠正其他人程序中的错误。当我们在不同时区工作时,我们也使用email。无论出现什么情况,我们都会互相帮助,例如技术支持和错误调试,我们还为参与到这个开发过程中的所有其他人提供社区论坛和wiki站点。

◆ 你们全职做这个项目,还是同时有其他工作?


Olivier和**Alon:**我们是全职的。

◆ 你们喜欢什么样的开发环境?

Olivier:主要使用Windows。为了给Macintosh做测试,我还已经得到了Mac OS X下的PowerBook。

Alon:我在一个使用多年的运行Windows的dual-933GHz奔腾处理器工作站上进行开发,使用Eclipse IDE作为代码编辑器。

◆ 其他人能帮你们做些什么呢?

我们缺少一个真正的设计师,但我觉得这是开源社区的通病。我们还一直需要一些新的插件。核心团队或多或少被局限住了,因为我们发现当我们引导人们使用Azureus时,他们却常常想做其它的事情。参与一个有25万行程序的项目并不容易,如果有一个简单的插件界面,参与才会变得更容易。 

算命软件

编绘:张绿



声音

我个人的理想就是做最好的软件,对最多的人有用。

——新西兰的奥克兰人 Goodger 在过去的四年中一直在为美国在线公司下属的网景公司和 Mozilla 基金会工作,目前他领导着 Firefox 的开发。

40% 的企业都禁止使用即时通讯,但这并不能保证企业的员工不使用即时通讯。

——在许多企业咨询与使用即时通讯服务相关的问题时,英国奥斯望律师事务所的一名律师马克·史密斯说。

这是我们将开放源代码与传统企业级应用软件整合的一次成功的尝试,它将更好的服务于企业用户和个人用户。

——Novell 公司在推出“开放式企业级服务器”软件的同时,还计划发布 SuSE Linux Enterprise Server 9 的第一个升级服务包,Novell 副总裁大卫·帕特里克 (David Patrick) 这样表示。

我不是去创造一个产品,而是生产一个工具,可以供其他的开发者用来构造他们自己的软件。

——英国广播公司 (BBC) 宣布了一项开放源代码视频压缩技术的计划,开发团队的领袖托马斯·戴维斯 (Thomas Davies) 对此表示。

几乎没有人要硬即时,即使是嵌入式装置。

——MontaVista 软件公司即将赋予 Linux 一项目前由专有软件独占的功能:从自动控制电脑到放影机都需要的快速反应时间保证。而 Linux 的创建者和精神领袖 Linus Torvalds 却在一封电子邮件访问中发表了上述观点。

我们的要求主要包括两点,一是伙伴公司要使用 IBM 的硬件和中间设备作为其软件捆绑供应的一部分,二是至少拥有一个 IBM 的共同客户。

——IBM 发布新的 IBM 高级合作伙伴计划,试图吸引更多的应用软件供应商,这是与微软的关键一仗。

作为庞大文件的通讯协议,eDonkey 显然更好。

——Kazaa 的对手“eDonkey”已在上个月取而代之,成为互联网上使用率最广的点对点 (P2P) 应用程序,受娱乐公司委托监看文件交换网络活动的加州公司 BayTSP 表示。

iSphere 的 EPL 可以将 100 行的 Java 源代码减少到只需 10 或 20 行。

——一家 iSpheres 的公司发布了一种专为处理“事件”(events)而设计的语言,例如金融交易,或是网络入侵企图等等商用软件必需监视的一些活动。

现在我们的开发人员可以集中精力开发软件功能,而不是测试,修正程序错

误不是问题,发现错误和错误的原因才是最困难的。

——Netline 决定以 GPL 许可证 (开放源代码最严格的一种许可证) 发布它的 Open - Xchange 电子邮件服务器的源代码,以便和微软的 Exchange 竞争。

这还并不过分积极,然而却是一个良好的开端。

——微软公布 FlexWiki 源码逐渐扩大开放程序范围,对此 SourceForge 主管 Patrick McGovern 表示,人们对于微软涉足源码开放反应平静。

未来 6 或 7 年,微软应该更积极地投入 Linux 应用软件,因为那个部分将提供比操作系统或数据库更好的增长机会。

——哈佛商学院副教授 Christensen 在 Future Forward 科技会议的演说中表示,开放源代码对微软和整个软件产业而言,是一个明显的颠覆。

我们真的希望它能够受到计算机用户的欢迎。

——在说到“Google 桌面搜索”时,Google 公司负责消费者 Web 产品的主管迈尔说。

幽默

coding and coding,
programing and programing,
boss is seeing,
boss is smiling,
but,
we are sorrowing,
sorrowing?
smiling?
who can know our feelings.

——权彦杰

迎接 UML 2.0

■ 文 / 杜玄

2003年4月Rational公司在上海举行UML讲座,这时的Rational已经成为IBM旗下的一员。这次讲座,Bran Selic作为IBM公司的Thought Leader亲自介绍了UML2.0的标准进展情况和未来发展方向(Bran Selic是IBM Rational加拿大的首席工程师。具有30年实时软件系统的设计开发经验,是《Real-Time Object-Oriented Modeling》的作者。近来,Bran正在领导一个小组为实时应用制订UML profile的OMG标准)。在Selic为我们介绍UML2.0的时候,觉得UML2.0似乎距离我们还很遥远,而今天UML2.0已经来到了我们的面前。UML2.0所具有的清晰、简洁、准确的表现力,使每个人怦然心动。下面让我们一起来感受它的魅力。

现状

从ARTISAN Software公司(www.artisansw.com)的Alan Moore那里,我们获得了UML2.0当前状态的信息:今天的UML2.0标准已经进入了倒计时阶段,FTF(Finalisation Task Forces,OMG标准采纳流程中的一个阶段的任务组,参见www.omg.org/gettingstarted/process4-Finalize.htm#WhenFTFcompletes)计划在2004年年底提交最终报告,也许在你看到本文时,OMG已经正式发布了UML2.0版本。

从OMG的网站上我们可以得到当前UML2.0的相关文档,当然,如果你是OMG的成员,你可以得到更新的关于

UML2.0的信息。

《UML 2.0 Infrastructure Final Adopted Specification(03-09-15)》

《UML 2.0 Superstructure Final Adopted specification(03-08-02)》

《UML 2.0 Diagram Interchange final adopted specification(03-09-01)》

《UML 2.0 OCL Final Adopted specification(03-10-14)》

《UML 2.0 Infrastructure》是关于UML2.0基础的,内部的规范,为新版本的UML提供了坚实的概念基础,精确的语义定义更好的为MDA提供了支持;《UML 2.0 Superstructure》是UML2.0的定义,它是一个用户层面的规范;《UML 2.0 Diagram Interchange》是关于模型交换的标准,为实现模型的交换提供支持,比如关于图的信息的交换,我们知道在UML1.x的年代XMI交换的模型将丢失图的位置坐标。这个标准将改善这个情况,实现不同层次的模型工具可以通过XMI完整的交换模型;《UML 2.0 OCL》是对象约束语言的标准,它也是一个通用的模型查询语言。

《UML 2.0 Superstructure》是最贴近现有的普通软件开发设计人员的标准,这篇文档通过结构和行为两大部分规范了UML语言2.0版本。如果你在下载《UML 2.0 Superstructure》时注意一下这个标准文本的联系人,你会发现他就是Bran Selic。这些文档是Final Adopted Specification,也就是说UML2.0的基本内

容与最终发布的UML2.0不会有大的变动,各个模型工具厂商已经依据这些内容来开发自己的UML2.0工具。比如:

ARTISAN公司的Real-time Studio

I-Logix公司的Rhapsody

Telelogic公司的TAU/Developer和TAU/Architect

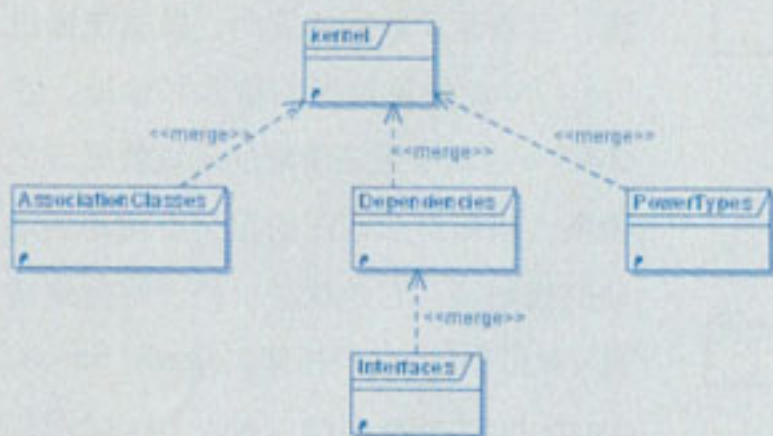
Borland公司的Together Designer Community Edition

这些公司都已经推出了UML2.0兼容的产品,我相信会有越来越多的模型工具兼容UML2.0标准。我们看到,用于实时(Real Time)系统建模的产品一马当先地推出了UML2.0兼容的工具,比如Real-time Studio和Rhapsody就是两个老牌的实时系统建模工具。这个现象是与UML2.0的某些新特征有关系的。UML2.0中增强了结构化的建模能力,比如结构化类(StructuredClasses),端口(Port)等特征。UML2.0的这些新特征就是从UML-RT等现有的嵌入实时建模中继承的内容。而现在把这些特征纳入到了UML2.0标准中。我们看到,《UML 2.0 Superstructure》的领导者Bran Selic本身就是一个实时嵌入系统设计开发的专家。实时系统建模工具最先兼容UML2.0也就不奇怪了,我们也可以看到他们在UML2.0标准中的影响力。当然,没有任何一个人和一个公司可以决定UML2.0的标准定义,UML2.0是由OMG组织成员共同推动和定义的,UML2.0为软件领域的建模提供了新的,更强大的表述能力。

2004年3月Grady Booch (UML创始人之一) 和Bran Selic接受了ComputerWorld的采访, 回答了关于UML2.0的几个问题 (<http://www.computerworld.com/developmenttopics/development/story/0,10801,91325,00.html>)。采访中他们透露, UML2.0可以处理领域中更复杂的问题; UML2.0具有更加精确的语义; UML2.0为MDA (模型驱动) 提供了基础, 可以把问题建立在更高的抽象层次的模型上来处理; UML2.0的新特征可以渐进地引入项目, 也就是说UML2.0是向后兼容的, 你可以渐进的过渡到UML2.0, 而不用丢弃过去的工作。究竟UML2.0为我们带来什么? UML2.0又有哪些新特征可以作为软件设计开发的新手段? 研读UML2.0规范可以为我们提供准确的答案。可以结合一个UML2.0兼容的工具来作为实践的助手, 以加深我们对规范的理解, 提高学习的效率。

语言结构

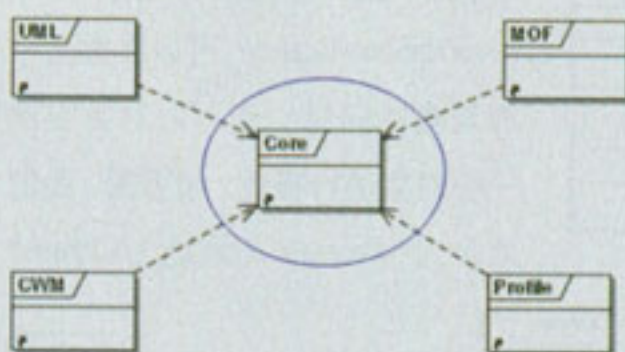
UML2.0在语义的定义上更加清晰明确, 并且显著改善了UML1.x过于复杂的问题。这里不能不说说《UML 2.0 Infrastructure》的Core包和《UML 2.0 Superstructure》中的Kernel包, 以及他们之间的关系。在《UML 2.0 Superstructure》中无论是结构, 还是行为的UML定义都是以Kernel包为基础的。下图是Classes包的内部结构, Kernel子包是整个Classes包的核心, 也是其它包引用和依赖的包。



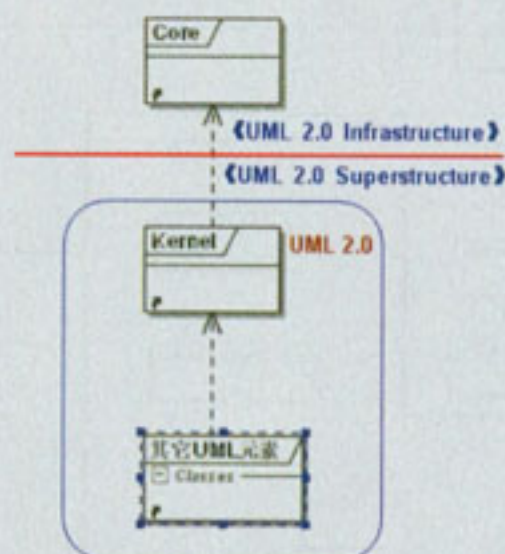
比如, 构件, 动作, 活动, 状态机, 用例等都直接或间接的与kernel包有关

系, 通过引入Kernel包, 也把Kernel中定义的语义直接拿来利用或继承扩展它。这样的好处是非常明显的, Kernel中只定义了少量的基本语言要素, 对Kernel的理解和学习是比较容易的。其它的语言元素都是基于这个基础来搭建的。这样在基本概念定义上就更加集中简洁, 而且具有层次性, 提高了复用性。UML2.0不象UML1.x中那样定义了大量的概念, 而且缺乏基本语义之间的联系, 缺乏语言定义的层次, 最终导致语言的复杂化。

仔细翻阅Kernel包中的内容我们会发现, Kernel包也是对《UML 2.0 Infrastructure》中Core包内容的复用。Core中的内容是更加基础的定义, 它作为一个基础库, 不但为UML服务, 也为MOF和CWM等服务。可以这样说Core是各种建模语言的公共基础库, 而Kernel是UML的核心内容。UML的Kernel中大量的引入的Core中的内容。Core作为一个公共基础库, 它提供了更高级别的抽象, 以适应UML, MOF, CWM等不同需要的引用。



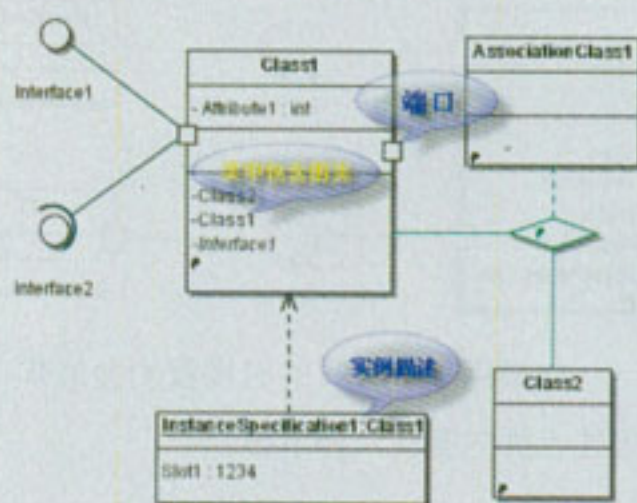
Core包的定义充分体现了重用性, 同时又为不同模型提供了一致性。Core与其它包的关系也体现了一种层级的关系。如下图可以更清晰的



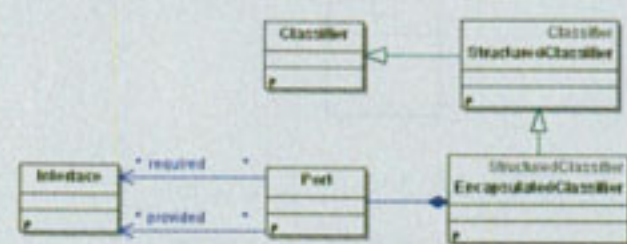
看到UML语言元素的脉络。

结构模型

UML2.0加强了对结构化建模能力的支持, 结构化建模能力的加强对基于构件开发和描述系统运行时架构提供了有力的支持。让我们看看类图, 在UML2.0中增加了哪些元素。如下图, 这是一张具有UML2.0新特征的类图, 其中最明显的特征是类的结构化特征, 也就是端口概念。端口就是在类图标的边线上增加了一个小方块, 这就是UML2.0中所特有的端口 (Port) 的图示表示方法。



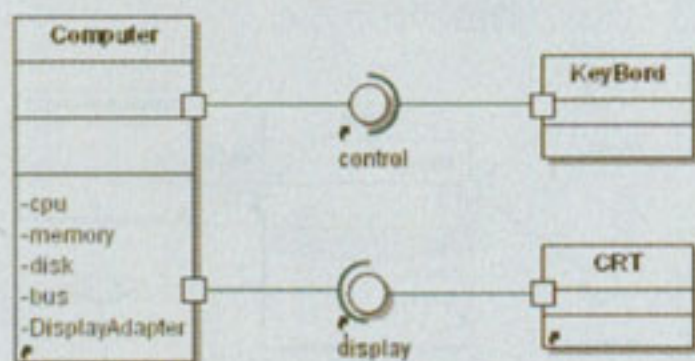
端口概念可以表现更强的封装性, 我们可以看看端口与类关系的元模型 (MOF)。从UML2.0规范中抽取简化Port相关的部分元模型如下图。



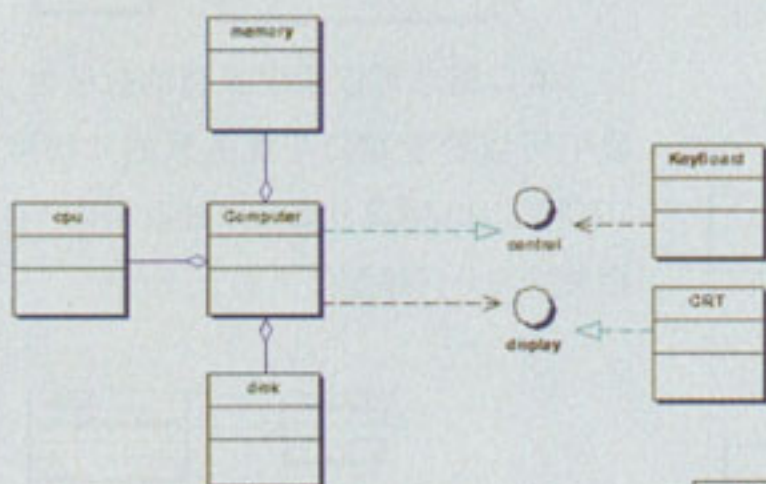
端口明确地提供了类元与其环境之间交互点。端口可以实现对外部提供服务接口和对外部的请求接口。端口概念可以表现更强的封装性。从元模型中可以看到端口在封装元类 (EncapsulatedClassifier) 中引入。这里的类Class, 是结构化类元 (StructuredClassifier) 的子类, 更是胶囊元类的子类, 所以类具有端口属性。

我们可以利用端口来表现一个PC机的结构。如下页左上图所示, Computer提

提供了一个键盘控制接口对键盘的命令作出响应。CRT的端口实现display接口提供图像的显示能力。而Computer的显示端口调用这个接口。端口为Computer, Keyboard和CRT的交互提供了明确的交互界定,严格的约束在端口中实现交互,比单纯的接口实现提供了更显而易见的调用方向性。这些特征使模块化程度的表现更强。类Computer中可以包含其它的类,这样更形象的体现了Computer中各个模块的物理结构特征,比如Computer中包括CPU, Memory, Disk等模块。



如果用UML1.x的时候我们会怎样描述上面内容呢? 如下图:

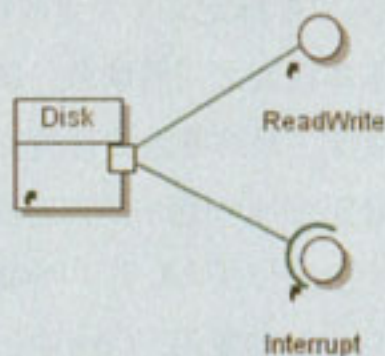


显然UML2.0的结构化的类表现PC的结构更加清晰直观,组件化更好,端口语义的引入,提高了组件化的表现能力;结构化类更好的表现了模块的相关特征。

除了类图以外,UML2.0中可以用组合结构图(Composite Structures Diagram)更好的表现模块的结构特征。我们建立一个组合结构图。如右下图:

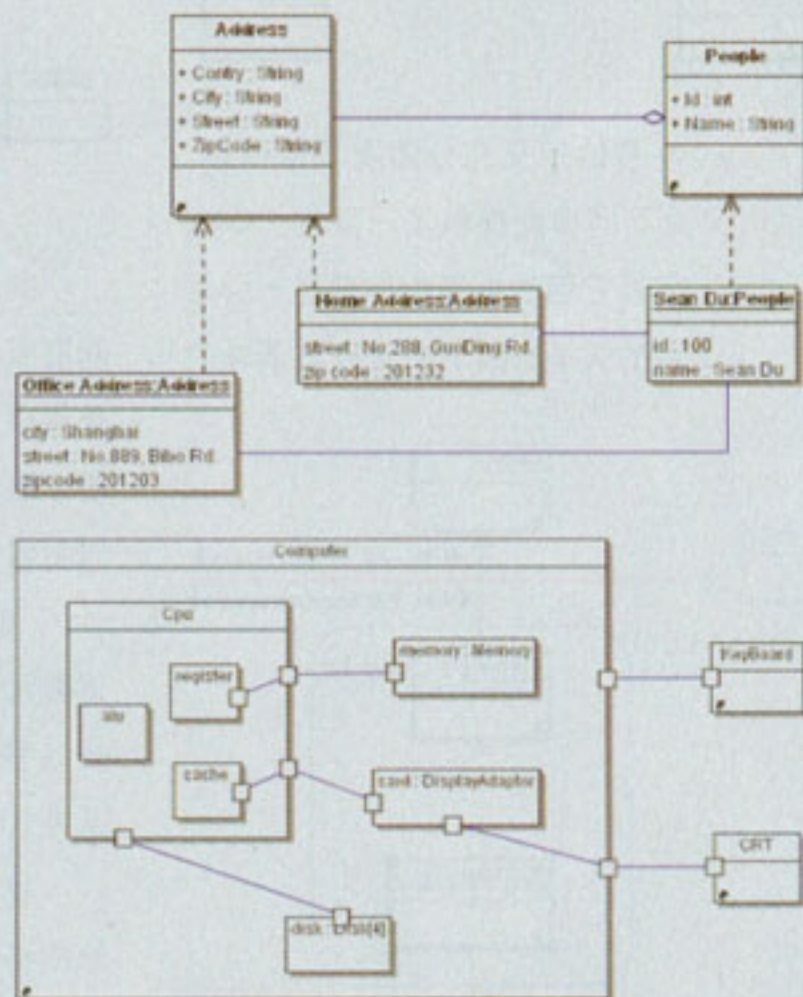
图中的Disk, Memory, DisplayAdaptor都是相应类的实例,叫做角色(Part),他们属

于容器类Computer。角色(Part)的端口之间通过连接器(Connector)表示连接关系。通过类容器,角色,端口,连接器清晰的展现了一个计算机的结构。图中Disk是Disk类的一个实例,[4]表示有4个Disk实例,参考下图Disk的类图,我们可以清楚的知道Disk提供的功能。一个Disk提供了读写服务和触发中断的接口。



类图中还为我们提供了InstanceSpecification和Slot的概念,这也是UML2.0为我们提供的新特征,在《UML 2.0 Infrastructure》中有对它的定义说明。InstanceSpecification是对类元实例的描述,如果是一个类,那么InstanceSpecification就是他的对象

(Object)描述。InstanceSpecification中可以包含Slot, slot中给出了这个实例结构特征的数值。有了InstanceSpecification,可以在类图中表现类的同时,还可以对类实例——对象进行描述。如下图,我们表示人(People)与地址(Address)

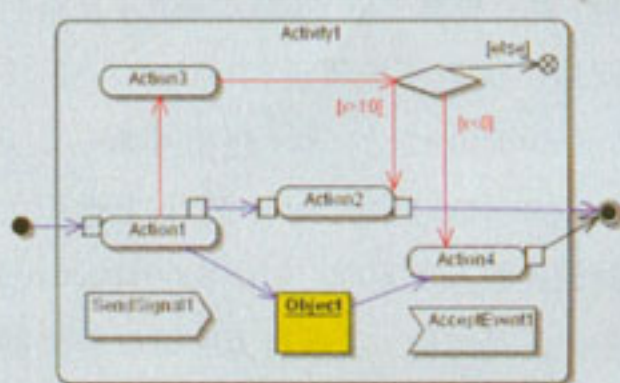


的关系。

这里用InstanceSpecification为People类描述了一个实例,这个实例是Sean Du这个人;又描述了地址的两个实例——办公室地址和家庭住址。Sean Du对象与办公室地址和家庭住址用链(link)标识了实例之间的关系。Sean Du对象中有两个Slot,一个是Id的值100,另一个是Name的值Sean Du。在类图中提供InstanceSpecification可以更直观地,全面精确地表述模型。

行为模型

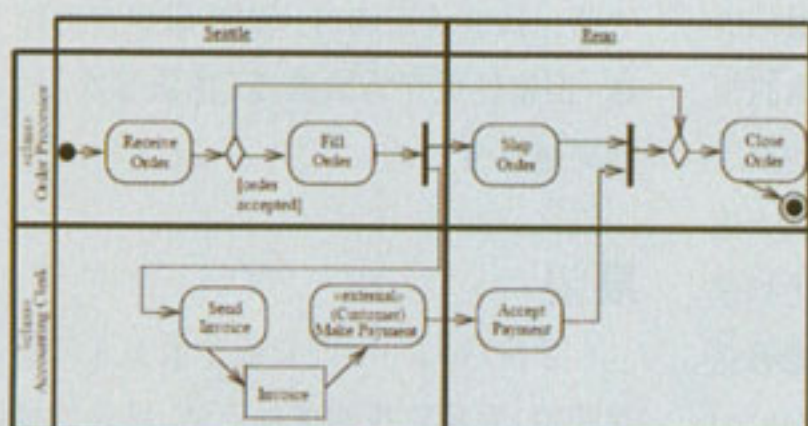
UML2.0为模型的静态结构提供了更强的表现力,提供了更多的,更精确的语言元素。在模型的动态方面,UML2.0也进一步增强。比如,UML2.0中的活动图提供了更强的数据流的语义,并发控制方面吸收了Petri网的内容,把活动图从UML1.x标准中基于状态机约束中解放了出来。可以用UML2.0的活动图描述商务流程。如下图就是活动图(Activity Diagram),我们来看看UML2.0的新特征。



上图中红色的线条是控制流程,它可以实现分支和并发;蓝色的线是数据流,它表示了数据的流向。数据流通过Pin, Pout连接数据的输入和输出。在UML1.x版本中无法清晰的表现数据流的模型,以前对活动图的定义是活动到活动的控制流,它是状态机的一种特殊情况(参见《UML用户指南》,Grady Booch, James Rumbaugh, Ivar Jacobson, p177)。这样的定义也限制了活动图的表述能力。上图中,活动(Activity)包含动作(Action),动作是原子的,活动不是原子

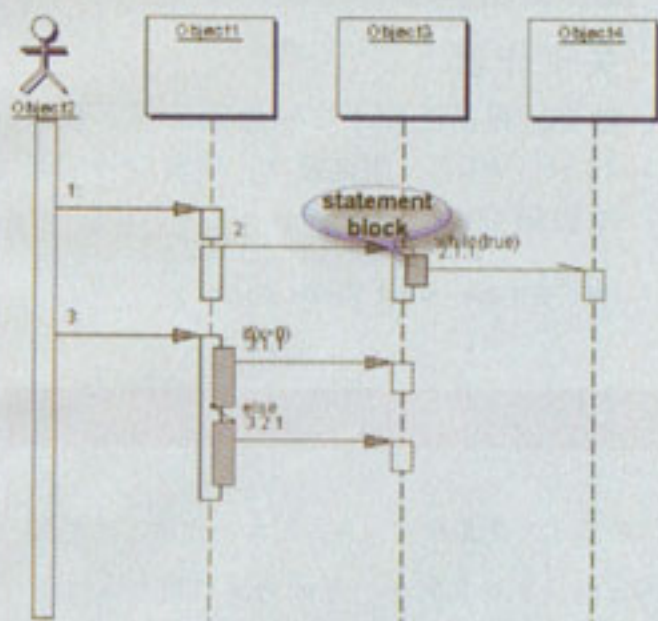
的过程，是可以被打断的。

UML2.0中的泳道 (swimlane) 能力也有所增强，希望可以看到模型模型工具厂商可以支持这个功能。下面是从UML2.0标准中摘取的多维泳道的例子：



多维泳道可以处理更复杂的系统模型，用泳道对系统进行横向与纵向的分割，再表示他们之间的调用关系。在现有的多层、多模块的体系构架中，多维泳道尤其有用。

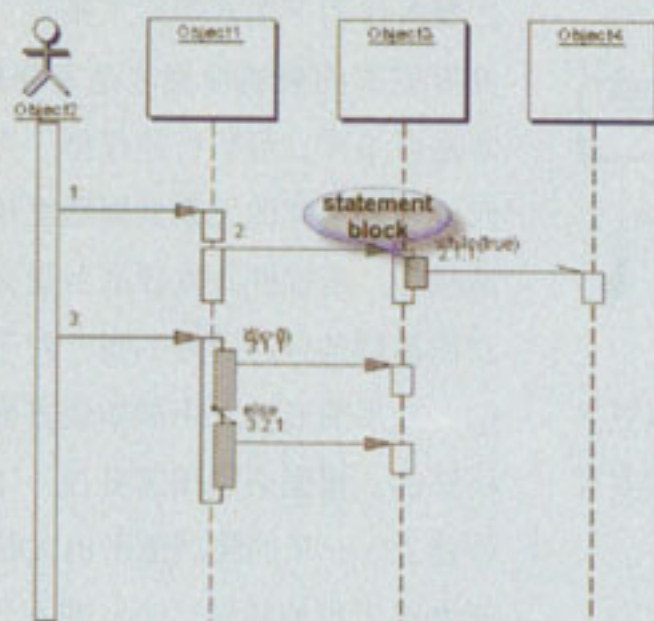
调查表明，类图，时序图，用例是最常用的UML图形。我们先回忆一下UML1.x中是怎样表现对象的时序关系的。



如上图，是用Together 6.1绘制的时序图，我们看到Together中为了处理分支和循环等流程情况，引入了语句块 (Statement Block) 的概念，用语句块来表现时序图的分支。这是UML1.x的一个变体。在《UML用户指南》中170页，作者建议一个单独的时序图只表示控制流，对于可选择的控制流或例外，用不同的时序图表示，再用包来组织这些时序图的集合。但是在实际应用时，这样的组织表示方法会造成时序图过多，也不利于导航。Together 6.1中提供的语句块很好的解决了

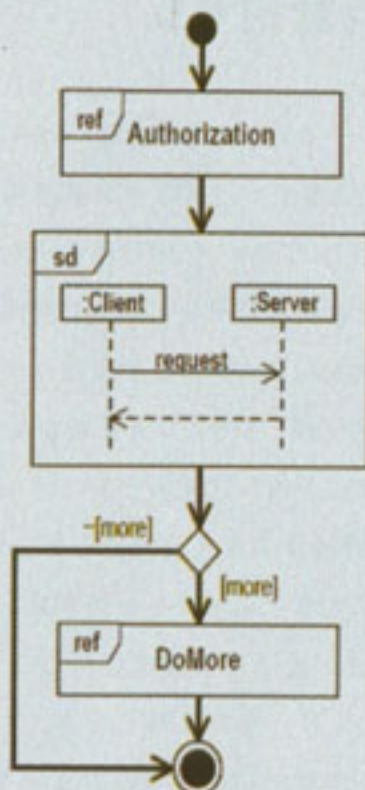
这个问题，是处理分支问题更加方便。

在UML2.0的时序图中全新提供了解决这个问题的标准定义。如下图：



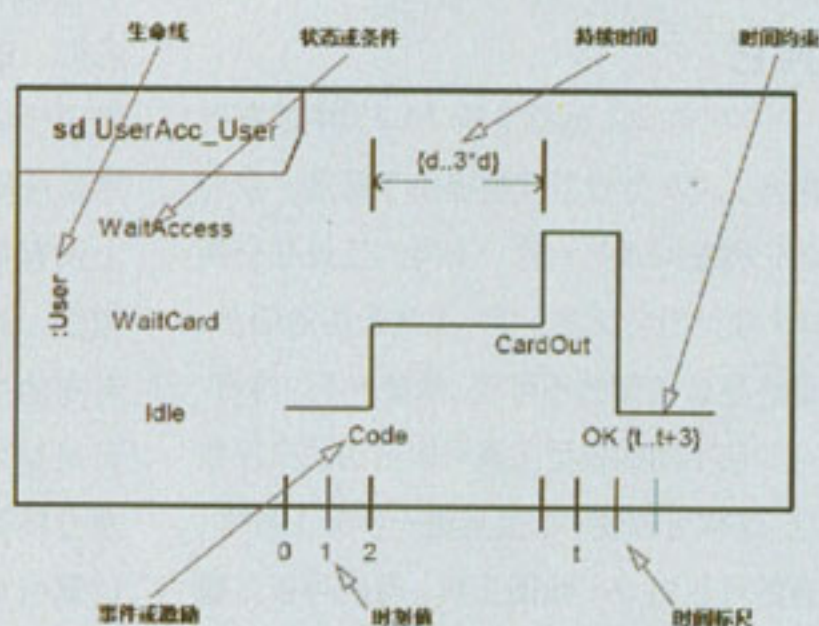
图中的alt表示出了条件分支的区域，流程可以根据不同的条件进入不同的分支，比如 $x > 0$ 调用3.1过程， $x \leq 0$ 调用3.2过程。在图中还看到循环调用的标识loop。UML2.0还定义了其它一些类似的标识，可以处理并行，case，break等情况。应用参考 (ref) 可以使时序图模块化，利用这个机制可以建立更加复杂的时序图。比如图中的“搜索资源”的引用，就可以指向具体的时序图，而这里不用在绘制“搜索资源”的过程，是现有的时序过程更加清晰明了。

下图是时序图与流程图结合的图，表现了一个时序的流程，在UML2.0标准



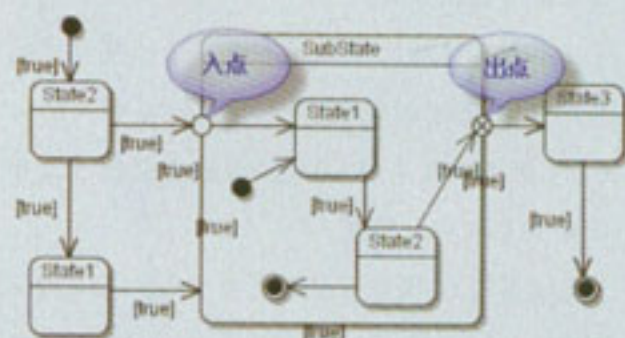
文档中有相似的例子。这个图把时序关系串连了起来。希望可以在某些工具中看到对它的支持。

对于动态模型中，在UML2.0中增加了一个全新的图，就是定时图 (Timing Diagram)。在Bruce Powel Douglas的《实时UML——开发嵌入式系统高效对象》一书中，我们已经见过了定时图，它是在2000年是已经在实时嵌入系统的UML中出现的内容，今天定时图纳入了标准的UML中。如下图



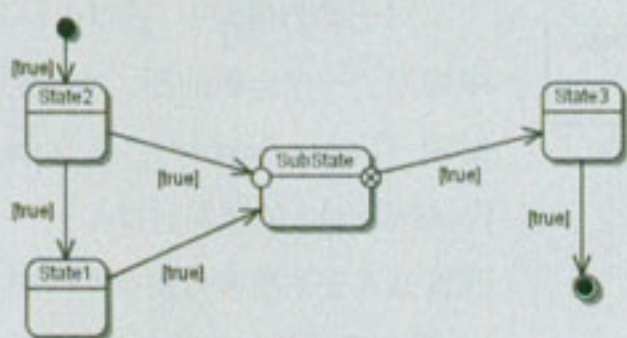
自动控制系统的设计中我们经常可以看到时序图（不是UML的时序图，这里是指信号时序关系的图），控制系统的时序图可以表现器件管脚的信号电压变化在时间轴上展开，清晰的刻画了信号随着时间的变化特性。UML2.0的定时图展现了对对象的状态在时间轴上展开的情况，也可以表现信号的变化情况。

动态模型中状态机是一个重要的内容，UML2.0在状态机的基本内容上没有大的改变，只是提高了状态机建模的封装能力，如下图。



从图中可以看到子状态可以被很好

的封装，新提供了入点（again）和出点（aborted）作为与外部的交互点。下图就是利用子状态机的而简化的状态图。



提高状态机的封装能力，可以更方便的组织大型复杂的状态机模型，实现状态机模型的复用。

其它

XMI和OCL也都是新UML2.0中的重要内容。XMI为模型互换提供了标准。我们还记得在UML1.x时，不同的工具在交换UML模型时会丢失信息，尤其是图的信息，最终导致模型的不可读。希望XMI的推行，可以使不同的模型工具可以自由的交换数据。这样可以使UML工具进一步分工协作，有的可以做UML绘图工具，有的可以做模型编译器，绘图工具绘制的模型可以直接导入模型编译器进行编译器编译执行。热切地盼望统一的数据交换的到来。

OCL提供了UML图示语言无法清晰表达的东西，OCL可以使模型更加精确。比如通过OCL可以为类标识前置条件，后置条件和不变式等约束条件。应该说具有完整约束的模型才是完整的模型，才是可编译、可执行的模型。可视化的元素与脚本化的元素共同建立精确的领域模型，是软件开发提高到更高抽象层次的基础条件。通俗的说，对于一个模型，只用图形和只用脚本语言表述都是不够的。模型必须图文并茂，才能够完整建立。一味的强调图形的人是UML初学者经常犯的错误，UML并不只意味着图形，这个情况到是容易理解：一味的强调代码就是模型的人却常常是资深程序员所犯的错误的。代码是模型，在逻辑上没有错，但是代码的模型是低层次的模型，对于解决复杂问题是笨拙的，没有高的抽象层次很难处理复杂问题。这正是MDA和产生式编程所要做的事情，建立精确的，高抽象层次的模型，通过计算机自动生成代码。可以这样说，也许未来“建模就是编码”。这里的编码已经不是现有编程语言（如C，Java等）级别的编码，而是用UML图示元素和OCL

这样的语言来编码，也就是定义精确的，形式化的模型。UML2.0已经为未来的“建模就是编码”做好了准备，让我们拭目以待，软件开发何时可以上升到更高的抽象层次，这是一个漫长和曲折的路，就像从汇编时代到高级语言的跨越，但软件生产方式提高“抽象层次”的方向是不会错的。

展望

迎接UML2.0，把它逐步地引入开发实践中，利用它搭建更加简洁、精确的模型，去解决更复杂的问题。

欣喜地听说，UML创始人三剑客中的Rumbaugh将来到中国，亲自为我们讲解UML2.0的新特征。Rumbaugh的到来使我们能够接近大师，亲耳聆听他传经布道，这样的机会确实是一件令人兴奋的事情。

期待中……

关于作者

杜玄，目前工作于中兴通讯公司。从事J2EE、AOP、测试技术、网管软件的开发和研究。

■ 责任编辑：罗景文 (ljw@csdn.net)

铺就IT 新型人才之路

——北航软件学院高级IT项目管理与营销专业

随着我国IT企业的成熟，现代项目管理技术和项目经理的作用日趋重要，项目管理人才群正在形成，它将成为未来的一种新的重要职业。最近，北航软件学院新开设了《高级IT项目管理与营销专业》。

“为我国的IT企业培养具有国际化水平的高级项目管理和现代营销人才是本专业的目标。”软件学院高级IT项目管理与营销专业项目主任王新河这样说。

《高级IT项目管理与营销专业》是与国际顶尖名校——美国卡内基-梅隆大学（CMU）海兹学院以及美国专门培养项目管理人才的顶尖培

训咨询机构——美国项目管理学院（PMC）联合开办的。本专业首次引入《IT项目管理硕士认证证书》课程是CMU/PMC于2002年首次联合开发的新课程，并由美方教师赴中国亲自授课。该课程已于今年纳入到CMU海兹学院的公共管理硕士MPA的必修课程。另外，本专业结合北大、清华、北航在管理与营销和IT技术方面的精品课程和优秀师资，共同培养高层次的项目管理与营销人才，该专业是学院唯一一只招收硕士的专业。

据调查，大家对目前中国IT企业最急需的人才描述为：有一定专业知识背景，掌握与国际化先进水平同步的IT项目管理知识与营销理念，

能熟练应用其相关知识、工具和方法，能为企业的业务建立完善、有效的盈利能力运作模式，熟练掌握一门外语的综合型、国际化的运营层面管理者，这正是高级IT项目管理与营销专业要培养的人才类型。

自阿波罗登月工程首次运用项目管理的方法至今已经近半个世纪，经过几代人的努力，无数项目实践，已形成了一套完整的知识体系，ISO在美国项目管理协会PMI的PMBOK体系基础上，建立了ISO10006标准，成为全球项目管理经理共同遵从的规范。

项目经理已形成了一个独立的职业阶层，优秀的项目经理就像作战部队的指挥官一样将是保证企业有效执行力的关键因素。

一个新兴的“黄金职业”——项目经理正在悄然显现，企业和个人都要把握机遇去创造成功。

性能测试用例

文 / 陈绍英 金成姬

目前国内，测试工程师时常要面对“已经延期几倍计划时间的项目”，测试用例如何发挥更大的作用，是一个迫切需要解决的问题。事实上，完全可以把测试用例看成是测试工程师编写的程序：这个“程序”是为了辅助测试工作的进行而开发的，目的是为了发现软件问题，同时“顺便”证明软件功能是否符合要求。

本文针对上面的问题，以设计性能测试用例为示范，讲解在企业实际工作中，如何有效划分测试种类和编写对应的测试用例，使测试工作更加合理、高效率地开展。

测试种类和阶段

测试种类

对于测试种类的说法多种多样，最多的能达到30多种测试类型。而实际工作中很多测试是互相包含的。按照企业中实际工作需要，通常主要进行下面几种类型的测试：功能测试、健壮性测试、接口测试、强度测试、压力测试、性能测试、用户界面测试、可靠性测试、安装/反安装测试、文档测试。

测试种类的划分不要拘泥于上面的形式，总体来说应该服从于测试策略，可以根据具体工作的特点进行安排，为了工作更容易开展，完全可以把一些测试合在一起进行。在后面的性能测试用例的编写上，充分体现了这一思想。

测试阶段和开发过程相对应，测试

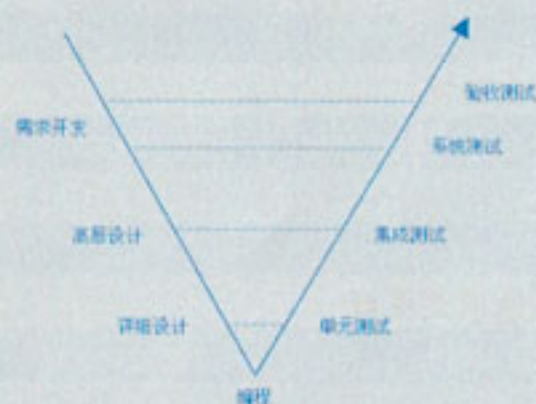


图1 开发与测试的“V”型关系

过程会依次经历单元测试、集成测试、系统测试、验收测试四个主要阶段。对应关系如图1所示。

单元测试：单元测试是针对软件设计的最小单位——程序模块甚至代码段进行正确性检验的测试工作，通常由开发人员进行。

集成测试：集成测试是将模块按照设计要求组装起来进行测试，主要目的是发现与接口有关的问题。由于在产品提交到测试部门前，产品开发小组都要进行联合调试，因此在大部分企业中集成测试是由开发人员来完成的。

系统测试：系统测试是在集成测试通过后进行的，目的是充分运行系统，验证各子系统是否都能正常工作并完成任务的要求。它主要由测试部门进行，是测试部门最大最重要的一个测试，对产品的质量有重大的影响。

验收测试：验收测试以需求阶段的《需求规格说明书》为验收标准，测试时要求模拟实际用户的运行环境。对于实际项目可以和客户共同进行，对于产品来说就是最后一次的系统测试。测试内容为对功能模块的全面测试，尤其要进

行文档测试。

尽管测试阶段的划分十分明确，但是在具体的项目和产品的测试中，尤其在执行测试时，会根据实际需要来开展。

测试种类、阶段和用例的关系

为了便于在实际工作中提高效率，同时方便测试用例的编写和执行，可以把上面提到的各个测试类型与对应的测试用例合并。合并后的测试用例主要有以下几种：

1. 功能测试用例：包含功能测试、健壮性测试、可靠性测试
2. 性能测试用例：包含性能测试、压力测试、强度测试
3. 集成测试用例：包含接口测试、健壮性测试、可靠性测试
4. 安全测试用例：安全测试用例
5. 用户界面测试用例：包含用户界面测试用例、少量功能测试用例
6. 安装/反安装测试用例：安装/反安装测试用例

综合上面的分析，测试种类、测试阶段以及执行人员具体的关系如表1所示。

表1 测试的种类、阶段和执行人员的关系

测试阶段	测试类型	执行者
单元测试	模块功能测试, 包含部分接口测试、路径测试	开发工程师
集成测试	接口测试、路径测试、含部分功能测试	开发工程师 (如果测试人员水平较高, 可以由测试人员执行)
系统测试	功能测试、健壮性测试、性能测试、用户界面测试、安全性测试、压力测试、可靠性测试、安装 / 反安装测试	测试工程师
验收测试	对于实际项目来说基本同上, 并包含文档测试; 对于软件产品, 主要测试相关的技术文档。	测试工程师 (根据实际需要, 可能包含用户)

总之, 测试的种类应该尽量的少, 这样每次都可以执行更多的测试内容。例如在进行功能测试的同时, 完全可以进行健壮性的测试。(当然如果产品健壮性方面要求较高, 就可以把健壮性测试作为独立的测试。)

性能用例编写方案

性能测试在软件测试中占有重要的地位, 而性能测试又关联很多内容。例如压力和强度测试就与性能测试密切相关: 针对一个网站进行测试, 模拟 10 到 50 个用户就是在进行常规性能测试, 用户增加到 1000 乃至上万就变成了压力 / 负载测试, 如果同时对系统进行大量的数据查询操作, 就包含了强度测试。

为了便于性能测试工作的实施, 这里的性能测试综合了性能、强度、压力、负载等多方面的测试内容, 主要包含的内容有: 预期性能指标测试、用户并发性能测试、疲劳强度测试、大数据量测试和速度测试、网络、服务器等方面的内容。

性能测试针对不同的系统有不同的要求, 编写方法要根据实际要求进行编写, 本文提出一个常见的参考方案, 在实际工作中, 可以根据需要加入其它例如内存泄露等和性能相关的测试用例。

下面介绍各个部分性能测试用例应包含的内容:

预期性能指标测试用例

通常系统在设计前都会提出一些性能指标, 这些指标是性能测试要完成的首要工作之一。针对每个指标都要编写

多个测试用例来验证是否达到要求, 并根据测试结果来改进系统的性能。

这类通常以单用户为主, 如果遇到并发用户的情况, 可以归到并发用户测试用例中。这类用例通常都是可以通过手工来执行的用例, 例如示例中的上传一份文件, 期望的性能为 2M/S, 完全可以手动上传文件, 同时用秒表计时。这些内容通常在需求说明书中可以显而易见的查到。不过当看到如支持并发用户 300 人, 就应该放到后面进行。测试结果也是直接记录是否达到要求, 如果系统没有达到要求则进行改善。

用户并发性能测试用例

用户并发测试是性能测试的最主要部分, 包含了负载测试和压力测试的过程。主要是逐渐增加用户数量来加重系

统负担, 直到出现不能接收的性能点或者瓶颈。一般要测试正常数量的用户并发和极限数量下用户并发的情况。

并发用户测试主要是对系统的核心功能和重要业务进行测试, 要以真实的业务数据作为输入, 选择有代表性和关键的业务操作来设计测试用例。主要编写以下两个方面的用例:

核心模块的测试(可以理解为“单元性能测试”): 对核心功能模块进行并发用户测试, 测试系统是否能够稳定运行。例如对于互联网的公用邮件系统, 每天早上 9 点左右可能是收发邮件的高峰, 这时候上千的用户都要在上班后进入邮件系统, 系统这个时候需要接收和发送大量的邮件。所以邮件系统这一功能模块要进行并发测试。通过测试可以知道数据库服务器、操作系统、网络设备等是否能够承受住考验, 同时可以对瓶颈进行分析。

表 2 列出来一些常见的参数(表格中的数据为示例的测试用例和测试结果), 可以根据实际需要进行增加和删除, 其中磁盘 I/O、数据库相关测试参数要根据实际情况进行选择, 因此没有列出。

在编写这类用例时, 要进行综合分析, 选出系统中的各个核心模块, 分别设

表 2 核心模块的性能测试用例

功能	在线用户达到高峰时, 发送和接收普通邮件正常, 保证 200 个以内用户可以同时访问邮件系统, 能够正常发送和接收邮件。					
目的	测试系统 200 个以内的用户同时在线能否正常发送邮件。					
方法	采用 LoadRunner 的录制工具录制一个邮件发送过程, 然后利用其完成测试, 要监视数据库服务器和 web 服务器的性能。其中发送的邮件为普通的邮件, 附件大小不超过 1M。					
并发用户数与事务执行情况						
并发用户数	事务平均 响应时间	事务最大 响应时间	平均每秒处 理事务数	事务成功率	每秒点击率	平均流量 (字节/秒)
100	1.344	2.078	5	100%	102	5177
...
并发用户数与数据库主机						
并发用户数	CPU 利用率	MEM 利用率	磁盘 I/O 参数	DB 参数 1	其它参数	
100	23%	11%	
...	
并发用户数与应用服务器的关系表						
并发用户数	CPU 利用率		MEM 利用率		磁盘 I/O 参数	
100	32%		27%		...	
...	

计每个模块的测试用例：把模块划分成小的“事务”进行测试，这样在测试分析中便于定位问题究竟出现在哪里。例如邮件系统可以划分成：接收邮件、发送邮件、打开邮件等小的事务进行测试用例的编写，每个操作作为一个用例来执行。

业务组合性能测试（可以理解为“集成性能测试”）：所有的用户不会只使用核心模块，通常每个功能都可能被使用到，所有既要模拟多用户的“相同”操作，又要模拟多用户的不同操作，对多个业务进行组合性能测试。

业务组合测试是更接近用户实际操作系统的测试，因此用例编写要充分考虑实际情况，选择最接近实际的场景进行设计。这里的业务组成单位以不同模块中的“子操作事务”为单位，进行各个模块的不同业务的组合。例如在办公自动化系统中就可以选择“公文模块中的发送公文、电子公告模块中的查看公告信息、网上论坛模块中的上传文件”等事务作为一组组合业务进行测试，用例设计信息如下：

功能：在线用户达到高峰时，用户可以正常使用系统，保证500个以内用户可以同时在线使用系统。

目的：测试系统500个以内的用户同时在线能否使用比较常见的模块：公文系统、电子公告、网上论坛。

方法：采用LoadRunner的录制工具录制三个业务：

业务1——在公文系统内，进行打开、修改等操作；

业务2——在电子公告系统内，查看、发布公告；

业务3——在网上论坛系统内发布帖子，查看文章。每个业务分配一定数目的用户，利用LoadRunner来完成相关参数的测试。

其它部分设计可以参考表2。执行时要分别记录各个事务的执行情况。

多用户并发性能测试是性能测试的

核心内容，包含了全部与多用户相关的测试。因此设计时要全面考虑，不要有遗漏。在测试执行时，本部分通常是采用性能测试工具例如LoadRunner来进行测试的，因此更容易执行和提高效率。

疲劳强度与大数据量测试

疲劳强度测试是在系统稳定运行下模拟最大用户数量、并长时间运行系统，通过综合分析执行指标和资源监控来确

目的	测试系统运行网络在不同并发用户条件下的使用情况		
方法	在不同的广域网带宽下（例如256K）使用LoadRunner录制邮件系统的相关事务操作脚本，以不同的并发用户数进行测试，记录各种用户连接数下，不同并发请求的性能变化；同时记录路由器端口的流量和其他数据。		
运行时间	10小时		
用户并发数	事务平均响应时间	服务器端口流量	丢包率
100	2.816	50.2M/S	0.001%
500	3.876	98.2M/S	0.002%
...

定系统处理最大业务量时的性能。

疲劳强度测试的目的就是检验系统长时间运行后的性能，因此设计用例时，需要编写不同参数或者负载条件下的多个测试用例，对服务器、软件、网络进行不同条件下的综合测试分析，测试时要记录系统发生故障的信息作为测试结果。疲劳强度测试也是采用测试工具进行的。

大数据量测试分为两种：一个是针对某些系统存储、传输、统计查询等业务进行大数据量的测试；另一个是与前面并发测试相结合的综合数据测试。编写用例时主要编写前一部分，后一部分尽量放在并发测试中。

大数据量测试一般是针对那些对数据库有特殊要求的系统进行测试，例如电信业务系统的手机短信息表，由于有的用户关机或者不在服务区，每秒钟需要有大量的短信息保存，同时在用户联机后还要及时发送，因此对数据库性能有极高的要求，需要专门测试。

本部分用例设计表格可以参考用户

并发性能测试部分。

网络性能测试

网络性能测试主要是为了准确展示带宽、延迟、负载和端口的变化是如何影响用户的响应时间的。在实际的软件项目中，主要是测试用户数目与网络带宽的关系。

编写用例的格式如表3（表格中的数据为示例数据）。

表3 网络性能测试

本部分可以独立测试，也可以和用户并发性能测试、疲劳强度与大数据量性能测试结合起来，在原有的基础上采用工具来调整网络设置，从而达到监视网络性能的目的。通常网络性能都是采用工具进行性能评估，由系统集成工程师来进行。

服务器性能测试

本部分的测试用例不必独立编写，或者根据实际需要编写少量的测试用例，建议这部分的用例编写和前两部分结合起来，在用户并发性能测试、疲劳强度与大数据量性能测试时完成对服务器性能的监控，完成对服务器性能的评估。

性能分析基本策略

在上面的用例执行完成后，接下来要进行性能分析。性能分析是性能测试的最终目的，否则测试出的指标就不会有实际意义，这里主要介绍一下性能分析的基本思路。性能分析通常要围绕三个方面进行：软件、服务器、网络。

软件主要是分析具体事务执行时间,尤其并发用户部分,根据测试工具测试出的结果分析那些事务执行的慢,然后可以分析执行较慢的代码,针对网页可以分析具体的页面元素执行情况。

服务器的分析要结合软件的运行情况进行分析,着重分析硬件的执行参数,CPU、硬盘、内存、中断、内存等情况,分析尤其要注意对这些参数进行综合分析,往往各个参数之间会互相影响,最后在调查、分析整体系统的基础上,找出影响服务器整体性能的瓶颈,确定相应的升级需求:

1. 服务器硬盘负载较重,需增加硬盘。
2. CPU 整体性能偏低,需增加或更新 CPU。
3. 网卡性能偏低,需更换光纤网卡。
4. 硬盘 I/O 负载任务繁重,需使用高转速硬盘或采用 RAID 卡。
5. 内存资源短缺,需增大内存。
6. 其他方面,需要升级软件系统、合理进行子网划分、加强管理等。

网络性能分析要结合服务器和测试目标软件,通常网络传输慢会有软件和服务器方面的原因,甚至有时候会有客户端方面的原因。不过目前网络的环境普遍可以,不管是局域网还是广域网,网络的环境越来越好。

用例管理

测试用例的管理我们可以借鉴开发过程中对程序的管理方法,我们可以把测试用例看成程序——测试工程师编写的程序,这个程序也要经过“设计”、“开发”、“测试”、“版本管理”、“发布”、“维护”等一系列操作,然后按照管理软件程序的方法来管理测试用例。

用例管理主要包含评审、修改、执行用例、用例版本维护、用例升级方面的内容。

用例评审

测试用例评审是测试用例不可缺少

的一个环节,这是对“测试部门开发出的产品”进行的“测试”。基本思路是对测试准备阶段的成果进行分期评审,依次评审系统/验收测试用例、集成测试用例、单元测试用例。

评审用例在比较正规的公司更容易实施,要求相应的软件开发团队必须在实际工作中对测试给予足够的重视,才可以把这项工作做好,否则只是走走形式。有效的用例评审通常由下面两种形式组成:

测试部门外部评审——主要是由开发部、项目实施部、甚至销售人员参加的评审,目的主要是查找测试工程师编写的用例是否缺少内容。外部评审主要工作方式是用文档直接记录评审结果,测试人员根据评审结果对用例进行升级修改。

测试部门内部评审——部门内部同行对测试策略的评审,评审的核心内容是测试策略和用例编制思路是否正确,以此来保证测试用例的有效性。可以组织正式的评审,由用例的设计人员进行讲解,然后大家共同评审;也可以把文档发给部门的同事进行评审。内部评审有些象开发人员在单元测试中的交叉测试。

内部评审的主要工作方式是项目会议,大家可以进行激烈的讨论,共同探讨用例编写、交流经验,这样用例的编写水平才能提高,同时可以进行一些创新。

评审方式中的外部评审最为重要。因为开发人员很容易发现用例中遗漏了什么内容,同时还可以发现错误的用例——因为存在对需求理解的偏差。用例外部评审可以理解为开发人员在查找测试人员编写的“程序”缺陷。

通常情况下先执行内部评审,然后执行外部评审。很多时候,内部评审会被忽略,建议要进行内部评审。这样至少有两个好处:集思广益和提高测试小组输出文档的质量。

管理方案

国内大多数 IT 公司在测试用例的发展经历了以下几个阶段:

◆ 无用例而执行测试:测试的初级阶段,完全手工测试,测试执行工程中没有测试用例作为执行依据,可能会按照需求进行测试;


◆ 有用例而不使用用例执行测试:已经编写测试用例,但是受各种环境的影响,例如需求变动频繁、编写的用例过于简单等,测试用例编写后在实际工作中不能使用;

◆ 按照部分用例执行测试:随着用例编写水平的提高,部分测试用例可以在测试中发挥作用;

◆ 完全按照用例执行测试:组织建立了规范的项目管理过程,对测试用例进行规范的管理,执行测试用例以用例为准则来执行测试。

完全按照测试用例执行测试是一个公司测试水平的体现,测试用例管理成为这一阶段的主要内容。测试用例管理的核心内容是版本管理。如果测试用例是采用文字编辑软件例如 Word 编写,建议采用工具(例如 Visual SourceSafe)对用例进行控制。

在测试用例管理规范化并成为测试的执行准则后,管理测试用例带来的巨大好处开始逐渐显现出来,测试用例成为评估测试和改进测试工作的主要依据,可以给工具带来巨大的方便。

总之,不管是性能测试还是其它测试都要本着“一切从实际出发”的原则,根据不同产品的特性进行用例编写,最后按照要求完成测试,达到提高产品质量的目的。在测试用例的编写过程中,尤其要记得“创新”,如果长期依靠某一测试用例编写模式,采用某些固定的模板,测试用例编写工作肯定会停滞在某一层次上不再发展,一定要跟着测试对象的不断变化来调整策略,在具体的工作中改进和提高,才能“开发”出优秀的测试用例! 

■ 责任编辑:罗景文 (ljw@csdn.net)

未来将越来越不可预测,这是新经济最具挑战性的方面之一。商务和技术上的瞬息万变会产生变化,这既可以看作要防范的威胁,也可以看作应该欢迎的机遇。

——Martin Fowler & Jim Highsmith,《敏捷宣言》

拥抱变化

敏捷设计从理论到实践

■ 文 / 温昱

如何应付软件开发中的“变化”,一直是近年来备受软件企业关注的问题。敏捷方法的兴起,更是为“随需应变”带来了一股强劲的浪潮。

本文从理论和实践两方面,和大家分享笔者在敏捷设计方面的心得。首先,以一种全新的角度考察耦合,并将其表述为良性依赖原则;然后通过应用实例,说明该原则如何和著名的“面向对象设计五大原则”结合,来“务实地应付变化”;最后从应付变化的角度,对各条原则做综合总结。

需要说明的是,本文采用“良性依赖原则”的叫法,是出于和依赖倒置原则(Dependency-Inversion Principle)的叫法保持一致的目的。由于“耦合”和“依赖”是一对使用都非常广泛的同义词,所以叫做“良性耦合原则”也是可以的。

理论篇

1. 换个角度考察依赖

1) 依赖的概念

依赖(Dependency): 两个元素之间的一种关系,其中一个元素变化,导致另一个元素变化。

依赖的同义词: 耦合(Coupling), 共生(Connascence)。

依赖的危害: 如果被依赖元素发生变化,可能引起另一个元素不得不变化。

2) 从会不会造成“实际危害”的角度考察依赖

关于依赖,已经研究得很多了:从依赖程度的大小角度来考察,有了耦合度相关理论;从依赖产生的原因角度考察,有了静态共生性、动态共生性和差异共生性的相关理论;不一而足。

是的,如果被依赖元素发生变化,可能引起另一个元素不得不变化,这就是依赖的危害。但是,如果被依赖元素不发生变化呢?答案是不会造成危害!于是,“冤案”产生了:由于需求分析上的偏差,设计中“在理论上”很稳定的耦合度低的依赖,可能“在实际中”恰恰是给我们造成危害的家伙;相反,“在理论上”声名狼藉的耦合度高的依赖,“在实际中”也可能并不给我们造成任何危害。

于是,我们很自然地想到,区分依赖的“实际危害”和“理论危害”是有实践意义的。下面,从会不会造成“实际危害”的角度考察依赖,将其分为良性依赖和恶性依赖两种类型:

◆ **恶性依赖:** 被依赖元素“在实际中”,而不是“在理论上”,是“易变的”。

◆ **良性依赖:** 被依赖元素“在实际

中”,而不是“在理论上”,是“不易变的”。

2. 良性依赖原则

不会“在实际中”造成危害的依赖关系,都是良性依赖;依赖的“理论危害”不一定成为“实际危害”,反之亦然。这就是良性依赖原则。

依赖是不可避免的,重要的是如何务实地应付变化。这就是良性依赖原则要做的。

1) 依赖是不可避免的

OOD的实质,简而言之,就是妥善地为多个类进行职责分配,使这些类相互协作而构造起完成特定功能的系统。在软件设计中,依赖是不可避免的,就象人类社会不可能没有人与人之间的协作与依赖一样,这一点其实是不言自明的。

2) 重要的是如何务实地应付变化

需求改变时常发生,而良性依赖是那些不会“在实际中”造成危害的依赖关系,所以,良性依赖是相对的——需求改变可能使先前不易变的元素变得易变起来,从而良性依赖也变成了恶性依赖。

Robert C. Martin在《敏捷软件开发》中提供的“只受一次愚弄”的策略很精辟。在我们最初编写代码时,假设变化不会发生;这时的设计很简洁,但对当时的

需求,却是有效的、“不多不少的”。当变化发生时,我们就通过创建良性依赖,来隔离以后发生的同类变化;一般认为要通过创建抽象来隔离变化,而本文务实地认为只要是“不易变的”元素就可以。

实践篇

下面,通过几个应用实例,说明良性依赖原则如何和著名的“面向对象设计五大原则”结合,来“务实地应付变化”。

1、第一个例子——需求改变引起良性依赖变成恶性依赖

需求改变了,原先的良性依赖变成了恶性依赖,但我们“只受一次愚弄”。

比如,在开发一个需求跟踪工具的时候,起初可能仅需要支持保存为专有格式的“项目”文件,但后来又需要支持导出为HTML格式的网页。

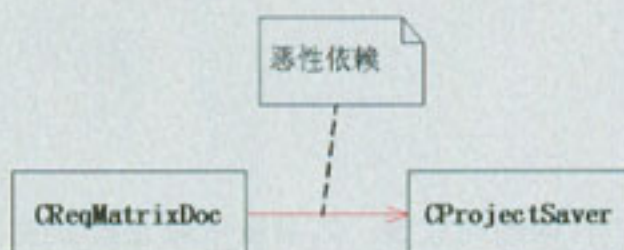
让我按照敏捷软件开发过程,来讲述这个故事:

最开始的设计如下图所示,CReqMatrixDoc调用CProjectSaver来保存自己。此时,所有需求就是支持“保存为专有格式的项目文件”,而且我们并没有预见到将来还需要以更多的形式保存,所以类CProjectSaver此时是“不易变的”,CReqMatrixDoc对CProjectSaver的依赖是良性依赖,整个设计也是个“稳定的”设计。顺便说明,按照开放-封闭原则(Open-Closed Principle),这并不是一个好的设计;但按照当前的需求,这个设计却“不多不少”刚刚好,因为当前它是满足良性依赖原则的。

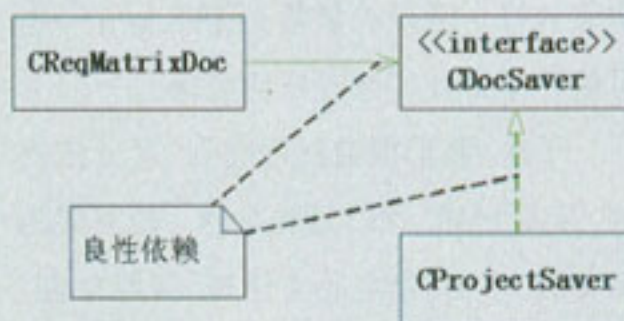


后来需求发生了变化,这个工具需要支持“导出为HTML格式的网页”的特性。是的,这个需求不管是客户新提出来

的,还是设计人员在上一个迭代有意忽略的,总之在这个迭代周期需求发生了变化。于是,设计人员意识到,需求跟踪工具可能需要支持多种保存策略;如果不改变原来的设计,那么CProjectSaver就是“易变的”,因为它要支持可能不只一种新的保存策略。好了,如下图所示,设计虽然没有改变,但由于需求的变化,原来设计中的良性依赖,现在变成了恶性依赖,这意味着CReqMatrixDoc可能也要随着CProjectSaver的改变而改变,这不是一个灵活的设计。

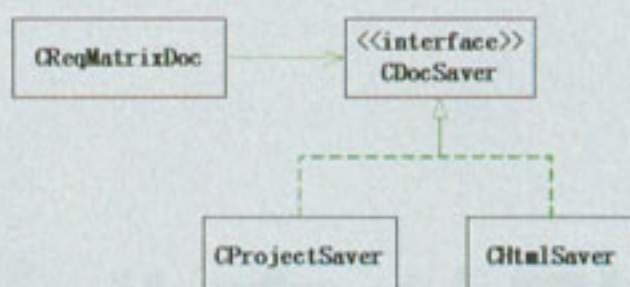


是的,代码出现了臭味(Bad Smell),需要重构(Refactoring)。让我们谨遵Martin Fowler的教诲——不要将重构和添加新功能同时进行——这一步我们仅进行重构。我们要做的就是去除这个恶性依赖,采用依赖倒置原则(Dependency-Inversion Principle)惯用的“用两个抽象依赖代替一个具体依赖”策略,重构之后的设计如下图所示。我们引入了一个接口CDocSaver,然后让CProjectSaver实现这个接口。一个设计良好的接口无疑是“不易变的”,所以不管是CReqMatrixDoc对CDocSaver的调用,还是CProjectSaver对CDocSaver的实现,都是良性依赖。重构完毕。



哈,新的设计非常易于扩充,我们只需新写一个CHtmlSaver来实现接口CDocSaver,就离支持“导出为HTML格式的网页”不远了,如右栏上图所示。噢,

原来是策略模式。



策略(Strategy)模式

关键字: 算法族。

支持变化: 它使得算法可以独立于使用它的客户而变化,多个算法之间也可以相互替换。

2、第二个例子——隔离第三方SDK可能造成的冲击

恶性依赖“作恶多端”。当恶性依赖中的被依赖元素变化时,依赖它的元素也可能要跟着变化;如果后者元素又在其他依赖关系中担当“被依赖元素”的角色,可能还会引起别的元素变化;这样,影响就会传播到很大的范围。

有时候,去除恶性依赖的代价比较大;还有时候,恶性依赖在所难免;我们应当如何?答案是,不去追求完美,而是务实地用良性依赖隔离恶性依赖造成的危害。

就让我来举个极端的例子吧——第三方SDK。

我们要开发的是压缩工具,我们不可能漠视现有的第三方SDK的存在,它们的诱惑实在太大了。在第一个迭代周期,要支持Zip压缩格式,我们决定采用著名的Info Zip开发包。Info Zip开发包并不在我们的控制之下——它的接口可能发生改变,当我们要使用更新的包时,我们可能面临不得不改动分散在很多类中的使用Info Zip代码的问题。所以我们要隔离这个我们控制不了的变化——对,就用Adapter模式——引入一个CInfoZipAdapter类来包装Info Zip,如下页左上图所示。这样,在Info Zip包升级时,我们仅需改动

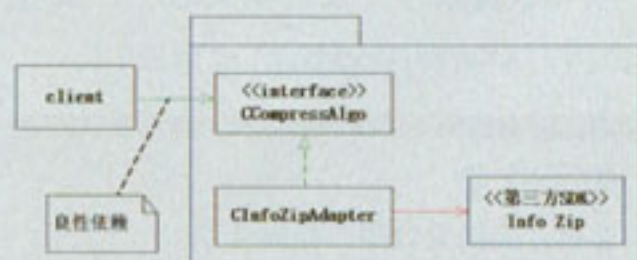
CInfoZipAdapter 的实现就可以了。这个 CInfoZipAdapter 并不是一个抽象类, 所以当前的设计并不满足依赖倒置原则 (Dependency-Inversion Principle) 推崇的“依赖于抽象”的要求; 但 client 对 CInfoZipAdapter 的依赖关系是稳定的良性依赖, 我们完全可以安心地远离过度设计 (Over-engineering)。



适配器 (Adapter) 模式

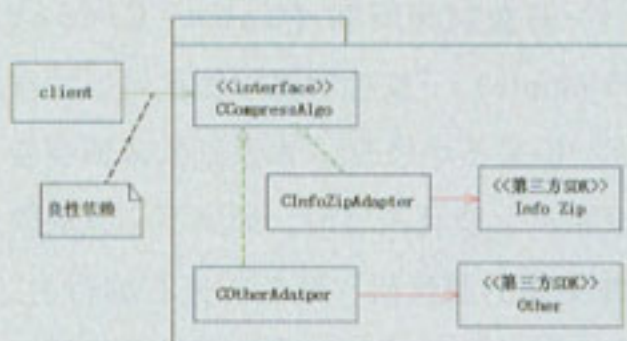
关键字: 已存在/不可预见, 复用。
支持变化: 由于 Adapter 提供了一层间接, 使得我们可以复用
一个接口不符合我们需求的已存在的类, 也可以使一个类 (Adaptee) 在发生不可预见的变化时, 仅仅影响 Adapter 而不影响 Adapter 的客户类。

谨遵敏捷宣言“经常性地交付可以工作的软件”的教诲, 第一个迭代周期过后, 我们发布了压缩工具的一个可以工作的版本。第二个迭代周期, 我们需要使用另外一个第三方的开发包来支持新的压缩格式。显然, 我们不当让 CInfoZipAdapter 承担多于一个的职责, 还是需要先重构。引入了接口 CCompressAlgo 供“外部”调用, 如下图所示。



重构完毕, 可以添加新功能了。从 CCompressAlgo “接口继承” 出来一个 COtherAdapter 来封装另一个第三方 SDK,

如右栏上图所示。哈, 基本满意: (多个) client 对 CCompressAlgo 的良性依赖, 使 client 的代码相当稳定; 所有第三方 SDK 的不可控制的变化因素, 都被妥善隔离。



3. 第三个例子

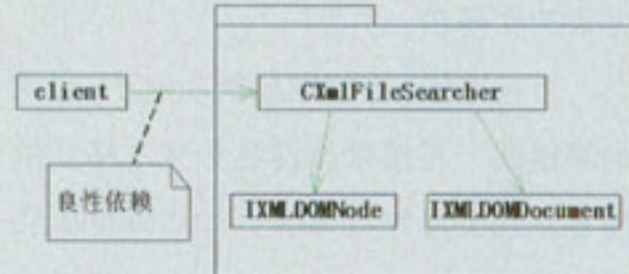
——对具体类的良性依赖

良性依赖可以是对抽象基类的依赖, 也可以是对具体类的依赖。其实, 这种对具体类良性依赖的例子是很多的, 比如《设计模式》和《敏捷软件开发》中 Facade 模式的相关例子。下面, 笔者举一个基于组件开发的例子。在“组件重用”这种“黑盒重用”日益盛行的今天, 也许更具现实意义。

好的组件库, 都恪守接口隔离原则 (Interface-Segregation Principle), 以达到“不应该强迫客户依赖于它们不用的方法”的目的。这其中包含了基于角色的设计 (Role-based Design) 的思想: 协作被定义为“多个对象为了完成某种目标而进行的交互”; 角色被定义为“特定协作中的对象的抽象”, 它“仅定义了对象特征的一个对某协作有意义的子集”; 协作和角色的概念和现实世界很接近, 我们很容易通过已有角色的组合来构造新的协作, 以完成新的功能。

好了, 看我们的需求: 在多个 XML 文件中, 查找那些包含特定名字的元素的文件。比如, 我们可能希望知道哪些 XML 文件中包含名为 book 的元素, 这在 XML 网页越来越多的今天, 对搜索引擎无疑是很有用。我们使用微软的 DOM 组件来实现: CXmlFileSearcher 代表一个高层模块, 它的职责就是上面描述的需求; 而具体的打开 XML 文件的工作, 它交给

IXMLDOMDocument 来做; 具体的读取 XML 文件中的每个元素的名字的工作, 交给 IXMLDOMNode 来做; 它是个典型的 Facade 模式, 如下图所示。

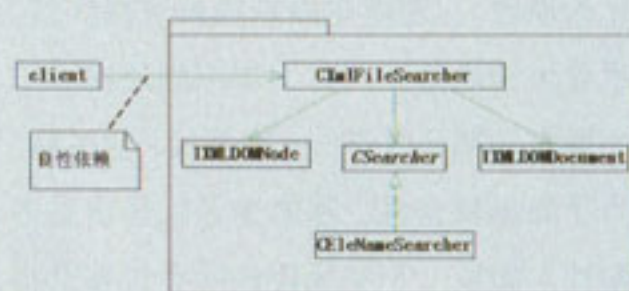


外观 (Facade) 模式

关键字: 子系统, 高层接口。
支持变化: 它实现了子系统与客户之间的松耦合关系, 而子系统内部的功能部件往往是紧耦合的。松耦合关系使得子系统的组件变化不会影响到它的客户。

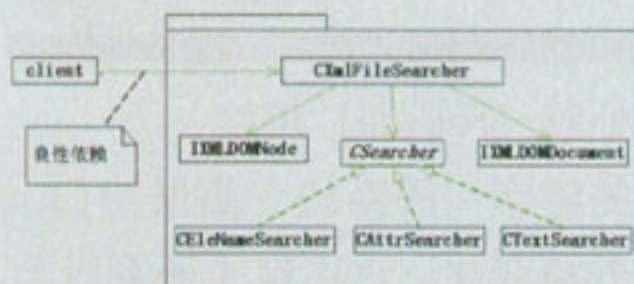
值得说明的是, 按照依赖倒置原则 (Dependency-Inversion Principle), 应当做到“高层模块不应该依赖于低层模块, 二者都应该依赖于抽象”。就是说, 我们应当为 CXmlFileSearcher 提供一个抽象基类, 供外部的高层模块 client 调用, 这在当前的需求之下, 未免有过度设计 (Over-engineering) 之嫌。这也正好体现了良性依赖原则的“务实”优点。

当然, 我们并不是一味地回避使用抽象基类的“抽象耦合”。这不, 没过多久, 新需求出现了, 不仅要搜索元素名, 还要搜索属性名和文本格式的内容。还是首先仅做重构, 引入抽象基类 CSearcher, 原先在 CXmlFileSearcher 中实现的搜索策略移到具体类 CEleNameSearcher 中, 如下图所示。需要特别说明的是, 现



在引入抽象和上一步引入抽象，是完全不同的两回事：现在引入抽象是基于实实在在的需求，而上一步引入抽象是基于猜测，其接口很可能不能满足刚刚提出来的新需求。

重构完毕，可以增加新功能了。运用策略模式，将搜索属性名和文本格式的内容分别实现作具体类 CAttrSearcher 和 CTextSearcher，如下图所示。



总结篇

“依赖”是和“变化”紧密联系在一起的概念。由于依赖关系的存在，变化在某处发生时，影响会波及开去，造成很多修改工作，这就是依赖的危害。可以说，变化是始作俑者，依赖是助纣为虐。

我们可以不去拥抱变化吗？不可以。未来将越来越不可预测，这是新经济最具挑战性的方面之一。商务和技术上的瞬息万变会产生变化，这既可以看作要防范的威胁，也可以看作应该欢迎的机遇。

既然变化不可避免，我们所能做的就是处理好依赖关系，将变化造成的影响波及范围尽量减小。

下面总结一下“面向对象设计五大原则”和良性依赖原则在应付变化方面的作用。

单一职责原则(Single-Responsibility Principle)。“对一个类而言，应该仅有一个引起它变化的原因。”本原则是我们非常熟悉地“高内聚性原则”的引申，但是通过将“职责”极具创意地定义为“变化的原因”，使得本原则极具可操作性，尽显大师风范。同时，本原则还揭示了内聚性和耦合性是“一物两面”的关系，为了降低耦合性，基本途径就是提高内聚性；如果一个类承担的职责过多，那

么这些职责就会相互依赖，一个职责的变化可能会影响另一个职责的履行。其实OOD的实质，就是合理地进行类的职责分配。

开放封闭原则(Open-Closed Principle)。“软件实体应该是可以扩展的，但是不可修改。”本原则紧紧围绕变化展开，变化来临时，如果不必改动软件实体的源代码，就能扩充它的行为，那么这个软件实体的设计就是满足开放封闭原则的。如果我们预测到某种变化，或者某种变化发生了，我们应当创建抽象来隔离以后发生的同类变化。在Java中，这种抽象指抽象基类或接口；在C++中，这种抽象是指抽象基类或纯抽象基类。当然，没有对所有情况都贴切的模型，我们必须对软件实体应该面对的变化做出选择。

Liskov替换原则(Liskov-Substitution Principle)。“子类型必须能够替换掉它们的基类型。”本原则和开放封闭原则关系密切，正是子类型的可替换性，才使得使用基类型的模块无需修改就可扩充。Liskov替换原则从基于契约的设计演化而来，契约通过为每个方法声明“先验条件”和“后验条件”；定义子类时，必须遵守这些“先验条件”和“后验条件”。当前，基于契约的设计发展势头正劲，对实现“软件工厂”的“组装生产”梦想是一个有力的支持。

依赖倒置原则(Dependency-Inversion Principle)。“抽象不应依赖于细节，细节应该依赖于抽象。”本原则几乎就是软件设计的正本清源之道。因为人解决问题的思考过程是先抽象后具体，从笼统到细节，所以我们先生产出势必是抽象程度比较高的实体，而后才是更加细节化的实体。于是，“细节依赖于抽象”就意味着后来的依赖于先前的，这是自然而然的重用之道。而且，抽象的实体代表着笼而统之的认识，人们总是比较容易正确认识它们，而且它们本身也是不易

变的，依赖于它们是安全的。依赖倒置原则适应了人类认识过程的规律，是面向对象设计的标志所在。

接口隔离原则(Interface-Segregation Principle)。“多个专用接口优于一个单一的通用接口。”本原则是单一职责原则用于接口设计的自然结果。一个接口应该保证，实现该接口的实例对象可以只呈现为单一的角色；这样，当某个客户程序的要求发生变化，而迫使接口发生改变时，影响到其它客户程序的可能性最小。

良性依赖原则。“不会在实际中造成危害的依赖关系，都是良性依赖。”通过分析不难发现，本原则的核心思想是“务实”，很好地揭示了极限编程(Extreme Programming)中“简单设计”和“重构”的理论基础。本原则可以帮助我们抵御“面向对象设计五大原则”以及设计模式的诱惑，以免陷入过度设计(Over-engineering)的尴尬境地，带来不必要的复杂性。

参考文献：

- 《敏捷软件开发：原则、模式与实践》 Robert C. Martin著邓辉译
- 《设计模式：可重用面向对象软件的基础》 Erich Gamma等著李英军等译
- 《Java设计：对象、UML和过程》 Kirk Knoernschild著罗英伟等译
- 《重构——改善既有代码的设计(影印版)》 Martin Fowler
- 《敏捷软件开发——使用Scrum过程(影印版)》 Ken Schwaber, Mike Beedle
- 《敏捷软件开发(英文版+中文注释)》 Alistair Cockburn, 俞涓译

关于作者

温昱，架构设计师，松耦合空间(<http://lcspace.nease.net>)创办人。擅长面向对象、架构和框架设计，对设计模式、UML和软件工程有深入研究。

■ 责任编辑：罗景文(ljw@csdn.net)

什么是交互设计?

■ 文 / Windy

一个交互实例

vivi (薇薇, 26岁, 一位优雅迷人的办公室白领) 打开钱包, 从卡夹层里拿出那张有着金黄葵花的银行卡, 又到了发工资的时候, 不知道今天到帐了没有, 还约好了明天和死党一起 Shopping 呢! 刚才路过银行想查一下余额, 但是排队的人太多了, 不过还有电话银行嘛, vivi 一边想, 一边拿出手机, 拨通了电话银行的号码:

一个温柔礼貌的MM语音提示: “您好, 欢迎使用招商银行电话银行系统, 1 自动语音服务, 2 人工服务:”

vivi 把手机从耳边拿下来, 找到1号键, 按了一下:

“1 个人银行服务, 2 公司银行服务, 3 银证通功能, 4 个人外汇买卖服务, 5 基金服务, 0 退出:”

vivi 又按了1:

“1 存折户, 2 一卡通户, 3 个人信用卡户, 4 新旧卡号查询 0 退出:”

vivi 按下了1旁边的2号键:

“请输入一卡通卡号, 以#号结束:”

“1080 80699”, vivi 连忙输入了卡号, 按#号键:

“请输入查询密码, 以#号结束:”

因为开户不久, 刚设的密码, vivi 稍微想了一下, 才把密码输进去, 输完又看了一眼, 按了#号键:

“1 帐务查询, 2 转账, 3 修改密码, 4

电话挂失, 5 通讯业务, 6 自助贷款, 7 自助缴费及一卡通上网, 8 神州行充值服务, 9 凭证式国债, 0 退出”

汗……这都是些什么呀? vivi 皱了皱眉, 再次按了一下1:

“10 人民币, 21 港币, 32 美元; 35 欧元, 65 日元, 43 英镑, 29 澳大利亚元, 87 瑞士法郎 39 加拿大元, 69 新加坡元, 00 退出:”

“10, 拜托, 我可只有人民币……”

电话里仍然是温柔礼貌而不折不扣的录音提示: “1 活期, 2 整存整取, 3 零存整取, 4 整存零取, 5 存本取息, 6 定活两便, 7 大额定期, 8 通知存款, 9 教育储蓄, 0 退出”

“1” vivi 一边按键一边看着手机屏幕上已经输入的一大堆数字。

“1 余额, 2 当天交易, 3 历史交易, 0 退出”

“1”

“您当前的余额是陆仟九佰五拾陆元伍角玖分”。

谢天谢地, 工资到帐了。vivi 长舒一口气, 查个余额, 真不容易啊!

什么是交互

如果你看到这里, 已经觉得很辛苦的话, 我想提醒一下, 上面可是一次完全真实的招商银行电话银行服务记录, 也就是一次真实的“交互”过程描述。

除了招商银行之外, 其他大大小小的各家电话银行, 交互的过程大抵都是这样。也就是说, 每天那些到电话银行查询余额的人们, 都要经过前前后后这样的折腾, 才能最后得到他们想要的一个数字。(实际上, 招商银行的很多服务都非常好, 仍然是我最喜欢的一家银行。)

交互, 在这里指的是作为服务使用者的用户和作为服务提供者的电话银行系统之间的互动通信并交换信息的过程, 有请求, 有应答, 需要双方参与。

在每一天的生活中, 我们都要和许许多多的产品进行交互, 回想一下, 就在今天, 和你发生交互的产品有多少? 早上叫起床的闹钟 (或提供了闹铃功能的手机和电话机), 热早餐用的微波炉, 电脑, 网站, 各种软件, 手机, 空调和电视机 (通过遥控器), 数码相机, 随身听, 银行服务, 等等等等。

那么再回想一下, 其中有哪些产品的交互很顺利, 并让你毫不费力地完成了各种任务, 甚至让你觉得惊喜? 例如, 在使用Gmail的时候, 我发现它自动帮我将邮件按主题组织在一起, 这样就可以很方便地回溯同一个主题前几次的邮件, 而且, 最近又增加了按联系人组织邮件的贴心功能。然后, 发现又有哪些产品, 总是很“难用”, 甚至还会一次又一次地让你犯错误, 又或者让你厌烦? 例如, 每



次关闭的时候,某些软件总是这样问你
“您确定要关闭××吗?”

什么是交互设计

在使用网站,软件,消费产品,各种服务的时候(实际上是在同它们交互),使用过程中的感觉就是一种交互体验。随着网络和新技术的发展,各种新产品和交互方式越来越多,人们也越来越重视对交互的体验。当大型计算机刚刚研制出来的时候,可能因为当初的使用者本身就是该行业的专家,没有人去关注使用者的感觉;相反,一切都围绕机器的需要来组织,程序员通过打孔卡片来输入机器语言,输出结果也是机器语言,那个时候同计算机交互的重心是机器本身。当计算机系统的用户越来越由普通大众组成的时候,对交互体验的关注也越来越迫切了。

因此交互设计(Interaction Design)作为一门关注交互体验的新学科在二十世纪八十年代产生了,它由IDEO的一位创始人比尔·莫格里奇在1984年一次设计会议上提出,他一开始给它命名为“软面(Soft Face)”,由于这个名字容易让人想起和当时流行的玩具“椰菜娃娃(Cabbage Patch doll)”,他后来把它更名为“Interaction Design”——交互设计。

从用户角度来说,交互设计是一种如何让产品易用,有效而让人愉悦的技术,它致力于了解目标用户和他们的期望,了解用户在同产品交互时彼此的行为,了解“人”本身的心理和行为特点,同时,还包括了解各种有效的交互方式,并对它们进行增强和扩充。交互设计还涉及到多个学科,以及和多领域多背景人员的沟通。

交互设计和界面设计

有很多人会问,交互设计,不就是界面设计吗?尤其是在理解同软件产品的交互时。人们在界面设计方面已经有了

一定的关注,然而,交互设计更加注重产品和使用者行为上的交互以及交互的过程,因此我在前面特意举了一个电话银行系统的例子,在这个例子里,并没有可以触摸的可视界面,而它在交互方面的行为本质却完全表现出来了。

界面是一个静态的词,当进行界面设计的时候,我们关心的是界面本身,界面的组件,布局,风格,看它们是否能支撑有效的交互,但是,交互行为是界面约束的源头,当产品的交互行为清清楚楚地定义出来时,对界面的要求也就更加清楚了,界面上(如果存在可视界面的话)的组件是为交互行为服务的,它可以更美,更抽象,更艺术化,但不可以为了任何理由破坏产品的交互行为。

从广义上来说,也可以认为界面设计包含交互设计,在这样的情况下,它同时还包含另外的部分(例如外观设计或平面设计),这些都是可以单独进行研究的更细分支。

为什么要进行交互设计?

回到前面vivi使用电话银行系统的例子,这是一个“没有任何满意度的交互”,虽然看起来它好像可以满足所有人的所有要求,实际上,每个人的要求都受到了影响,vivi为了查询到自己工资卡里的余额,一共需要九次按键,花费大约2分钟时间,其中大部分时间在各种名目的毫不关心的语音菜单之间选择,何况查询余额又是一个如此基本的操作。

你想说,不就是一个电话银行吗,有多大问题啊?不是这样的,因为:

就在此刻,数以亿计的产品正在和用户交互。这不是夸张,交互设计正是一种能影响到所有人的技术,让人们的感觉更好,或者更差,让产品更受欢迎,或者无人问津。

如果用战场作比喻,那么产品就处在和用户接触的最前线,最深刻,最具体的体验就在产品和用户之间展开。

拙劣的交互一定可以打败用户:

“确实要删除吗?是,否”一个熟练的用户受到了伤害,“我都已经删了几百次了,每次都这样问我”。

“无效的操作”,无辜的用户一头雾水。

“您带的资料不合格,请备齐资料,下次再来。”你好不容易请了半天假,排了两个小时队,把材料递上去,却得到这样的回复,意味着一切都要重新来过,包括手工填写那张有几十个框框的表格,而仅仅因为要换一个日期。

在这种情况下,用户受到的挫折和打击只会让他们越来越逃避和这样的产品交互。

反过来,良好的交互可以赢得用户的喜欢和信赖:

“这个软件真是太好了,现在我的工作完成得又快又好。”

“用Skype打网络电话真方便,还能支持多人会议!”

接下来,你就会看到用户乐滋滋地向朋友们推荐了。根据研究,对产品来说,朋友推荐的影响力远远超过公众场所投放的广告,那么,再接下来,你将看到的是,朋友的朋友,朋友的朋友的朋友,他们也将加入到用户的行列中来。

如果你是产品的提供商,你将乐于看到哪种情景?客服人员已经回来了,而产品留在用户那里,巩固或破坏用户跟品牌之间的感情。出于自身的感受,用户会支持具有良好交互设计的产品。你能忽视这样一个至关重要的前线阵地么?你的竞争对手们,他们会对这一阵地无动于衷么?

Jef Raskin在*The Humane Interface: New Directions for Designing Interactive Systems*(人本界面,交互式系统设计)一书中指出:

“就消费者而言,界面就是产品。”(我猜想在这里界面应该是产品外观加上产品的交互行为。)

“一个出众的界面也是杰出的长期投

资, 它将获得:

顾客更高的生产率
更高的用户满意度
更高的可见价值
更低的客户支持成本
更快、更简单的实现
有竞争力的市场优势
品牌的忠诚度
更简单的用户手册和在线帮助
更安全的产品。”

8月份的商业周刊指出, 通过设计研究来争取客户是全球增长最快的趋势之一。交互设计, 正是这样的研究中最典型的技术。目前, 许多公司, 网站, 新兴的行业都开始意识到交互设计在品牌的创建, 客户回头率, 客户满意度等方面影响很大, 因此, 交互设计也越来越受到重视, 并在近 10 年得到了快速的发展。

交互, 使用, 交互设计和可用性

长期以来我们已经习惯了“使用”这个词, 使用某种工具, 使用某个软件, 人类对产品的使用, 很好理解。

可是, 我们现在为什么要提到“交互”这个词呢? 因为“使用”, 是一种从人类出发的主动语态, 说到使用, 自然就想到以人类为主导, 使用对象为辅助的一种关系。

提到“交互”, 感觉是参与交互的双方更加对等, 实际上, 引入“交互”这个

词的意义, 就在于交互对象地位的提升。来自《超越人机交互》一书Terry Winograd先生说: “……也就是把计算的长处和人的长处相结合, 而不是让计算机模拟人, 正是这个想法把我引入了HCI领域”; Alan Cooper先生在UI Design的访谈中也同样提到: “关键在于, 让人去做他胜任的事情是一件非常好的事。让人做人擅长做的事情, 而让计算机做计算机真正擅长的事情。”对交互对象的重视有利于让人和交互对象的关系更合理, 从而得到人和交互对象相得益彰的相处方式。

可用性是交互设计的基本而且重要的指标, 它是对可用程度的总体评价。也是从用户角度衡量产品是否有效、易学、安全、高效、好记、少错、满意度的质量指标。

同时, 交互设计的目标不止于此, 它还包括要考虑用户的期望 (Donald Norman先生说到期望设计是交互设计的下一站) 和体验, 可用性保证产品可用, 基本功能完备且方便; 而体验在于给用户一些与众不同的或者意想之外的感觉。也就是说, 可用, 是产品应该做到的, 理所应当的, 体验则是额外的惊喜和收获。

(当然, 目前, 可用性方面的专家和著作都已经开始拓展“可用性”本身的内涵, 让它包含用户体验的内容。这说明, 用户体验同样也引起了可用性专家们足够的重视。)

国内软件的交互设计情况

前面提到更多的是产品, 其实, 软件存在着同样的问题, 说实在的, 国内的软件交互设计情况还有很大的改进余地。例如前面说到的电话银行系统, 就是一个典型的IVR (Interactive Voice Response) 软件, 还有在我另一篇文章里提到的“如意路径”问题, 还有长期以来大家认同的“功能越多越好”这样的观点。

还有某些网站时不时弹出一个安装插件的提示, 和某些假装有关闭按钮 (实际上点该关闭还是到广告链接) 的广告条, 这些也许超出了交互设计的范围, 但对用户来说, 实际上是非常非常糟糕的交互体验, 用户对这样的网站认同指数是一1000以上的下跌, 何必呢, 赢了点击输了整个形象。

开发人员们都在尽力用最新的开发技术, 认真地投入, 为用户建造新的计算机系统, 追求更新更高深的技术是没有错的, 也正是他们的专业精神所在, 但是, 对用户和系统的交互进行更多的关注, 可以得到满意度更高的系统, 这可不是秘密了。想要改善目前的交互情况, 可以和用户进行更多的沟通, 进行全面的交互设计, 还可以进行一些用户交互方面的评估, 只要做了, 就不算晚。

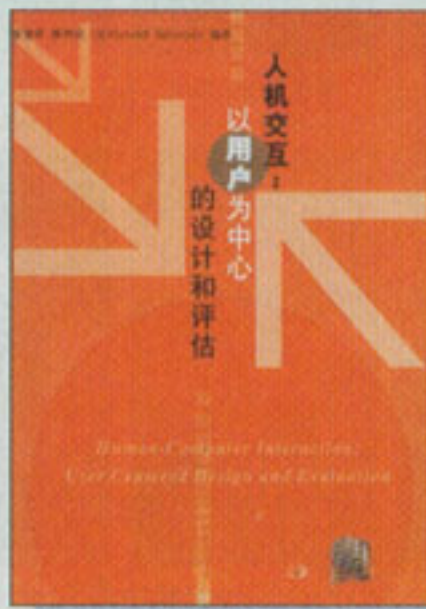
由于篇幅所限, 这里不能继续介绍交互设计的方法, 如果对交互设计方法感兴趣, 请关注笔者这方面后续的文章。

■ 责任编辑: 罗景文 (lijw@csdn.net)

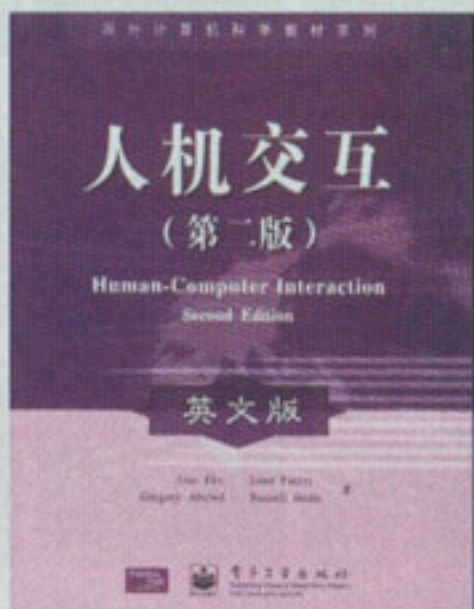
交互设计相关图书推荐



《人本界面》



《交互设计——以用户为中心的设计和评估》



《人机交互》(第二版) 英文版

需求管理的思辨

■ 文 / 徐锋

序

根据Standish Group对23000个项目进行的研究结果表明, 28% 的项目彻底失败, 46%的项目超出经费预算或者超出工期, 只有约26%的项目获得成功。而在于这些高达74%的不成功项目中, 有约60%的失败是源于需求问题。

因此, 作为软件过程改进的权威指南, CMM/CMMI框架将“需求管理”列为了最重要的待改进关键区域(KPA)之一。

产业界的情况又是怎样呢? 太多的挫折使人心灰意冷, 下面这幅有心人泡制的漫画也许可以让我们更真切地体会到他们心中的失望。

每当有人宣扬需求管理方法论, 最佳实践时, 就会引来一大堆反面的声音: “你知道吗? 需求天天都在变化!”, “我们面对的是什么样的客户呀, 他们甚至都不知道自己要什么!”, “这些家伙都像变色龙, 一开始说得好好的, 系统都正常运行了, 他才说不符合他们的需求, 可是早他们干什么去了呀?”

从上面的抱怨中, 都可以发现“所有的问题都出在客户方面”, 难道我们就没有错吗? 别骂我, 这是一个事实。虽然我们承受了许多委屈, 但错不在客户呀。想想你找装修公司帮你美化温馨小筑的场景吧, 你知道你要什么吗? 当设计师问

你要用什么木料, 什么工艺, 什么风格的时候, 你说了什么? 也许地板的颜色也是直到铺好的时候, 你才有想法吧! 瞧瞧你的所做所为和你整天在抱怨的客户又有何异呢?

“最好的防守就是攻击!” 笔者结合自己长期的实践与研究, 撰写此文, 希望对征服需求的这场圣战有所帮助。

明白我们在干什么 需求是什么

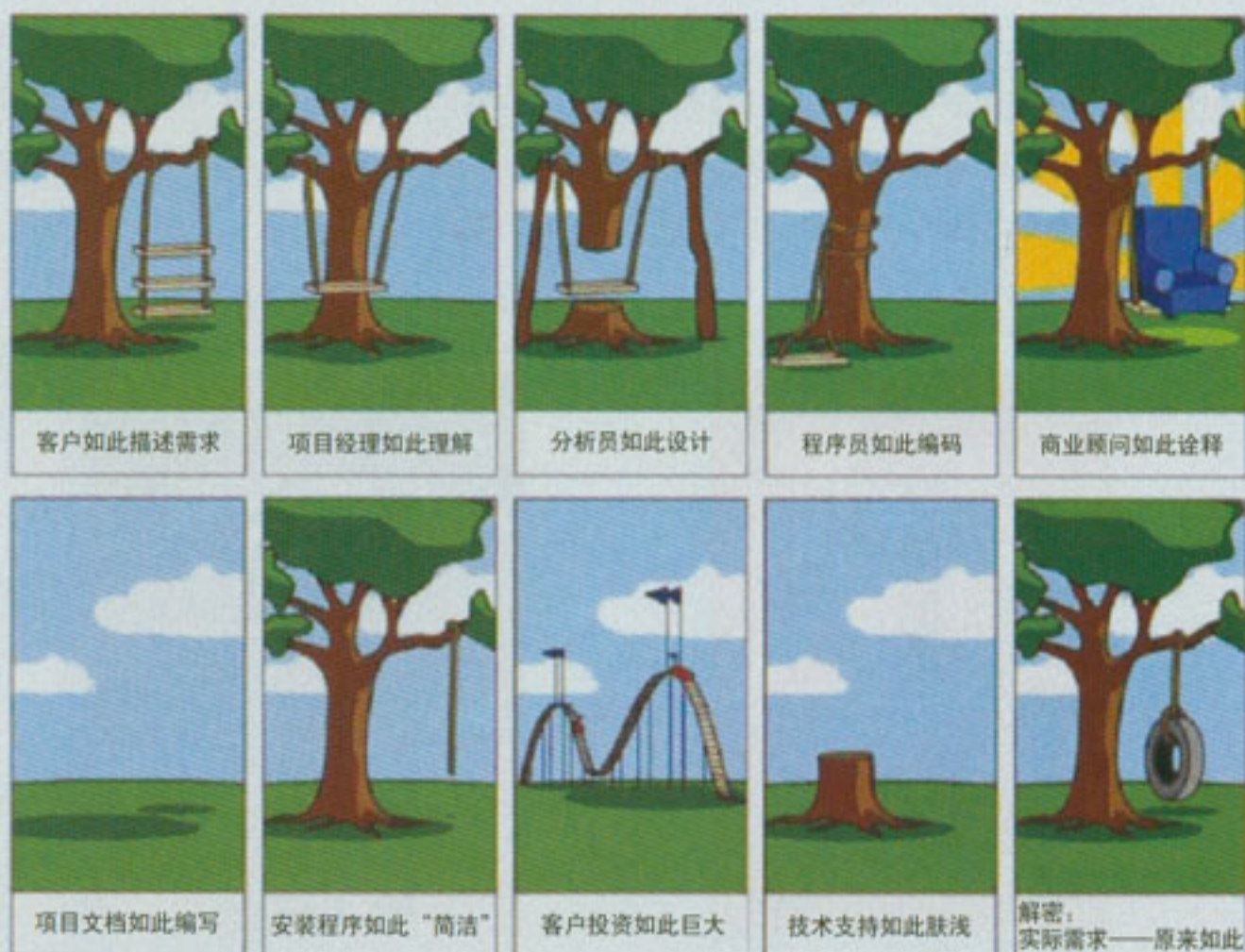
软件需求就是系统必须完成的事以及必须具备的品质。软件需求包括功能需求、非功能需求和设计约束三方面内容。正是这样耳熟能详的定义, 让大家忽视了许多并不应该忽视的东西。图二(见下页)是一个需求组成的全景图。

除了这三种需求之外, 还有业务需求、用户需求、系统需求这三个处于不同层面下的概念, 充分的理解这样的模型才能够使你更加清晰地理清需求的脉络。

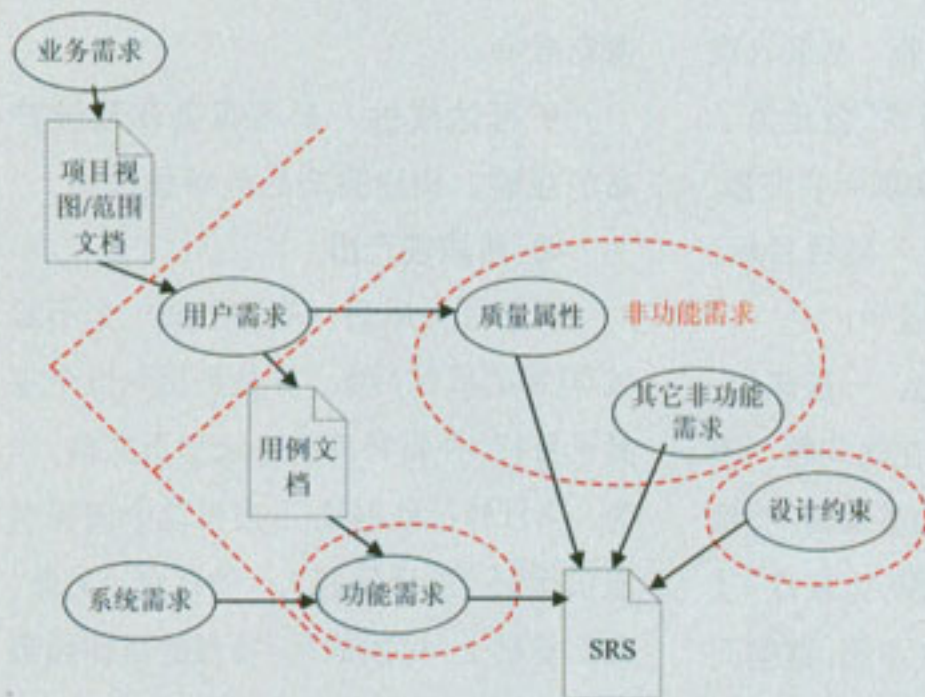
◆ **业务需求(Business Requirement):** 反映组织机构或客户对系统、产品高层次的目标要求。

◆ **用户需求(User Requirement):** 描述用户使用产品必须要完成什么任务, 怎么完成的需求。

◆ **系统需求(System Requirement):** 从系统的角度说明软件的需求, 包括用特性说明的功能需求, 质量属性以及其它非



图一 需求曾让我们如此无地自容



图二 需求的组成

功能需求，还有设计约束。

其次，大多数软件开发组织在需求方面的工作重心只放在功能需求上，对非功能需求和设计约束的理解也不充分，重视度也不高。

◆ **功能需求**：是指系统必须完成的那些事，即为了向它的用户提供有用的功能，产品必须执行的动作。

◆ **非功能需求**：是指产品必须具备的属性或品质，如可靠性、性能、响应时间、容错性、扩展性等等。

◆ **设计约束**：也称为限制条件、补充规约，这通常是对解决方案的一些约束说明，例如必须采用国有自主知识产权的数据库系统，必须运行在UNIX操作系统之下等。

在需求工作中，首先要从企业目标来了解系统的使命，对项目的范围进行划定，这将会给整个开发团队建立一条纲线；然后在这条纲线下，戴上“领域专家”的眼镜，通过各种有效的需求捕获实践对用户进行深入的研究与分析，以真正地了解每个岗位、每个用户的所思所想；最后再从系统实现的角度来考虑具体的功能需求。这应该是一个迭代过程。

需求工程又是什么

需求工程包括创建和维护系统需求文档所必需的一切活动的过程，包括需求开发和需求管理两大工作：

◆ **需求开发**：包括需求捕获、需求分析、编写规格说明书和需求验证四个阶段。在这个阶段需要确定产品所期望的用户类型、获取每种用户类型的需求、了解实际用户任务和目标以及这些任务所支持的业务需求、分析源于用户的信息、对需求进行优先级分类、将所收集的需求编写成为

软件规格说明书和需求分析模型、对需求进行评审等工作。

◆ **需求管理**：包括定义需求基线、处理需求变更、需求跟踪等工作。

这两个方面是相辅相成的，需求开发是主线，是目标；需求管理是支持，是保障。

需求开发

需求开发主要是四种活动，也是一个迭代的过程：有计划地进行需求的捕获；然后将捕获到的零散信息进行综合分析与研究，并利用软件建模技术构建系统蓝图；再将其用合适的形式（根据项目的规模，可以是很正式的文档，也可以是比较简单的文档；可以是正式提交的文件，也可以是一个Web模型，甚至是一个扫描件）将需求规格化；最后与客户一起进行验证（有时还需要通过制作原型的方法来进一步加强验证的效果）。这样的循环并非一次性完成的，而是周而复始地进行。

不过这种迭代也不应该是盲目的，整个过程应该先建框架，再填血肉；先宏观，后细节的有序过程。

在需求实践中，存在许多认识上的误区，主要体现在以下方面：

◆ **需求捕获**：这方面大家都意识到其必要性，但普遍存在缺乏计划性、科学性，“走过场”的情况很常见：很多人只是简单地和客户闲聊，然后收集一些相关的文档，表格就结束了。

◆ **需求分析**：很多人喜欢争论采用OOA（面向对象分析）还是SA（结构化分析），但这个重要过程经常被忽视，没有予以系统化的考虑，很多只是开发组偶而的会议，然后就轻率地开始编码了。

◆ **编写需求规格说明书**：比较“正规”的开发组织都会重视这个活动，甚至可以说是“重视过度”，而且产生出来的文档经常是与实际的开发脱离，完成之后就束之高阁，再也不使用，不更新。这是一个需求崩溃的信号。

◆ **需求验证**：大多数组织都不重视这个工作，导致直到交付系统时才真正被履行，这也就是为什么客户拿到系统后才提出许多这样那样的需求变更，甚至认为整个系统都不是他所需要的。

需求管理

需求管理的主要活动包括需求的基线管理、需求变更管理、需求跟踪三方面。这一块在实际的项目中，大多显得更加薄弱，甚至呈现“真空状态”。

◆ **需求的基线管理**：基线管理的核心思想是将所有现在的、将来的需求进行优先级评估，然后分解成为不同的组，每次迭代都选择其中优先级最高的部分进行开发，然后在迭代完成之前，开发工作不响应变更，这些划入的需求项就是需求基线的组成部分。

◆ **需求变更管理**：需求变更管理并不是要逃避变更，而是控制变更。一方面，基线内的需求不响应变更，为开发人员提供安静的工作时间状态；另一方面，专门的需求变更管理对所有的需求变更进行响应，了解需求变更的关键意图，新产生的工作量，从而良好地进行重新计划，以便有效地解决其对整个开发带来的麻烦。

◆ **需求跟踪**：需求的跟踪是指对需求的完成情况、变更影响进行系统化的跟踪与处理。也就是解决以下的一些问题：“需求是不是已经被实现？”、“需求的变化将需要修改哪些设计元素？会影



响谁的工作？对已经完成的部分是否有影响？”等等。

我们应该如何做

下面，我们主要针对需求捕获、需求分析和建模、需求基线与变更管理三个方面来探讨如何使“方法论”软着陆。

需求捕获

需求捕获工作在日常实践之中，基本上所有的软件开发组织都有投入人力、物力，但往往效果不佳。笔者认为，真正有效的需求捕获过程应该是更系统化、更针对性的有序活动。具体来说，应该从“明确业务需求”、“理解业务流程”、“明确用户需求”三个主要的步骤。

明确业务需求

业务需求是整个系统最宏观层面的东西，也就是“项目的目标”，通常其内容不会太多，是整个需求过程乃至整个开发过程中最本质、最重要的航标性问题。但却常常没有引起足够的重视，致使“上梁不正下梁歪”。

1) 业务需求在谁那里？

通常是在“项目发起人”的脑子里，也就是谁提出项目，谁就拥有对“业务需求”的最清晰的理解。经常我们接到项目时，最先接触的都是普通的办事人员，因此，在需求捕获活动伊始，我们首先要了解该项目在客户方整个企业内部处于什么样的位置，谁是主要的负责人，是谁最初提出该项目的，这十分重要。

了解了这样的信息之后，需求捕获的第一个工作就准备好了：那就是与这个关键人物座谈，充分地了解“业务需求”。那么应该如何来做好这项工作呢？从哪些地方着手，问些什么样的问题呢？什么时候标志着我们已经成功地完成了这个任务了呢？

2) 业务需求包括什么？

通常来说，“业务需求”可以分为“产品/项目目标”和“子目标描述”两个方

面的内容。“产品/项目目标”是最宏观层面的描述，而“子目标描述”就是为了完成这个目标而具体需要实现哪些内容。

要想充分地定义产品/项目目标，我们应该从以下几个问题着手：

◆ **产品/项目要做什么：**一般来说，这个本质的问题应该从存在的问题、不足着手，预先了解该企业的基本情况，然后在此基础上设计一些问题，以获取“项目发起人”发起项目的真实原因。这些问题应类似于：你觉得在企业运作中存在什么问题？这些问题主要体现在哪些方面？这些问题对企业造成了什么影响？你认为可以怎么解决？希望达到什么样的效果？

◆ **产品/项目提供了什么业务优势：**这也就是用户对产品/项目的期望值。当了解了问题之后，我们就应该转到更进一步理解“项目发起人”的真实用意以及其期望的目标。换句话说，也就是了解客户希望我们把问题解决到什么程度！针对这个内容，我们在访谈时要分而治之，将大任务分解成为小目标，并且引导客户良好地定义，这也是我们形成“项目子目标描述”的关键基础。例如，客户希望“降低库存率”，那么在这个部分，就应该继续分解成为“提高库存周转率”、“销售系统能够实现预警”等等。

◆ **如何衡量这些优势：**了解了以上两个层面的信息之后，紧接下来，还需要继续明确如何来评判是否达成了目标，并且提供一个定性的描述。例如“降低库存率”，具体降低到什么程度，到多少才算成功。

有了这些信息之后，我们还进行综合的思考，以避免从根源下埋下失败的种子。具体来说，需要考虑以下几个方面：

◆ **合理性：**是否能够通过信息系统来解决他的问题？是否能够找到一个投入与产出合理的解决方案？

◆ **可行性：**是否能够解决到他想要的程度？我们的解决方案能够达到度

量标准吗？

◆ **可达成性：**是否具备实现该产品的技能？构建完成后能够操作吗？

3) 阶段性产出

本阶段完成后，应该形成一个不超过50字的项目目标，并且列出5-9个主要子目标，并且将其制作成一页文档，作为“项目的行动纲领”，当然这个内容也应该得到“项目发起人”的认可。另外，在此基础上，可以编写“项目的目标和范围文档”，对于产品而言，我们还可以构建一个“愿景”文档（参见RUP框架，Vision文档）的初稿，从市场角度来综合分析。

4) 注意点

要注意的是，这个部分的内容可以通过“迭代”方式来获得，不一定是在一次访谈中就全部解决。这是因为：

◆ 我们需要时间来综合的思考；

◆ 项目的发起人、决策人不止一个，我们需要多番征求意见。

总的来说，这个部分工作处于“需求过程”的金字塔尖，多花费一些时间和精力是值得的，也是必要的。

理解业务流程

对业务需求充分理解之后，接下来就需要进一步细化业务流程了。这也就是在“业务需求”这个框的基础上填血肉。

1) 要做什么？

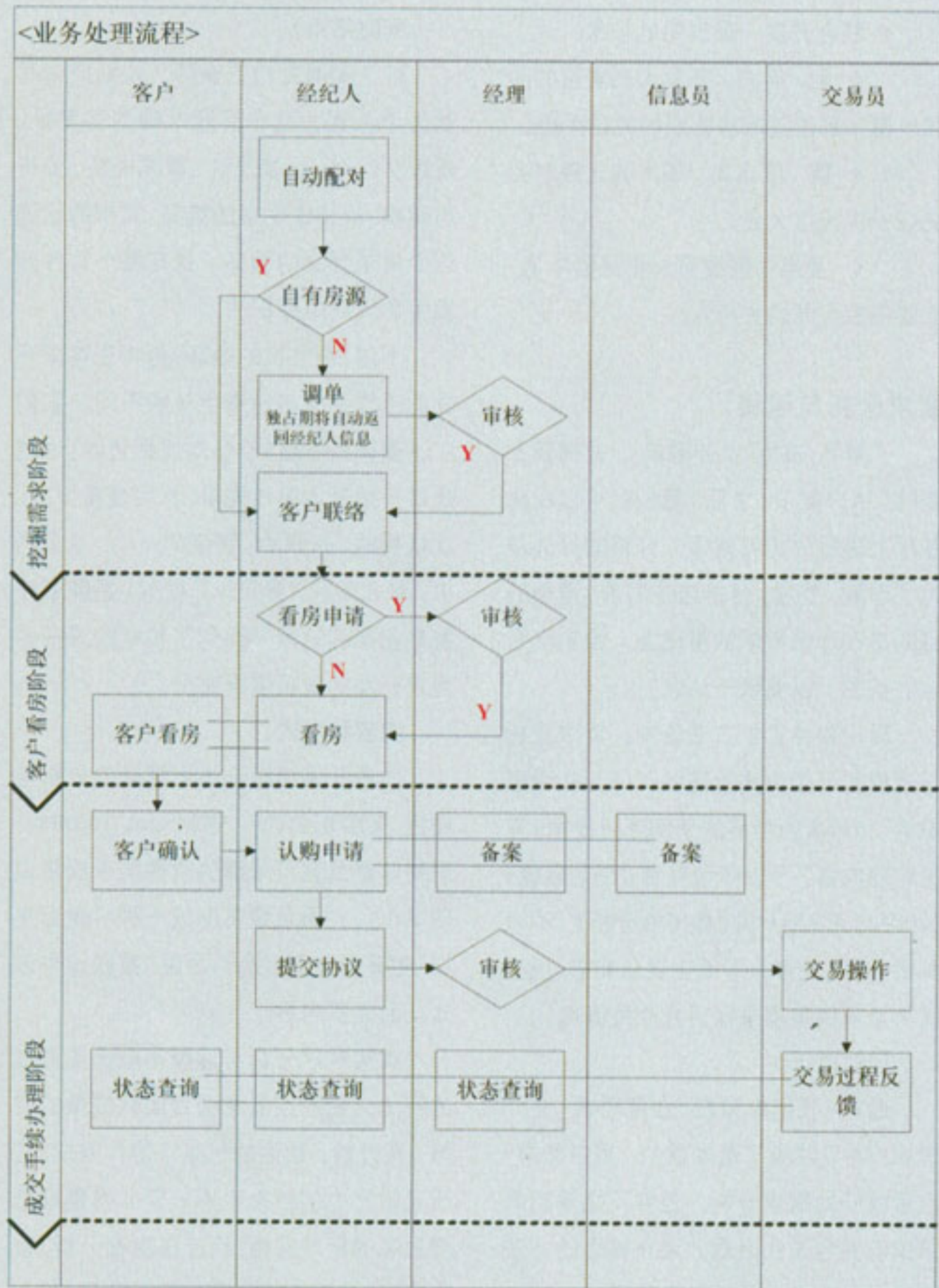
这个部分的工作如何进行，取决于项目的规模、团队的特点等诸多方面因素。但综合而言，可以遵从以下指导思想：

◆ 如果项目较大，或者业务较陌生：我们应该进行业务建模；

◆ 如果业务较陌生：应聘请领域专家，并且对项目组队进行领域培训；

◆ 如果术语较多，易于混淆：应该构建业务术语表，也就是在需求捕获的过程中，将遇到的业务术语收集下来，使用数据词典方法构建一个共享的名词解释库，以帮助项目团队达成共识；

◆ 无论如何，我们都应该建立跨部门职能流程图。



说明：矩形代表活动，菱形代表判断、平行线代表并行操作、带箭头线代表活动流、每个大列代表一个职能部门、最左边的每个分块代表业务阶段

图三 跨部门职能流程图

2) 怎么做?

笔者认为比较合适和可行的方法是

◆ 索取客户方的“组织结构与岗位设置图”，对其管理层次与线条建立全局了解；

◆ 根据系统的需求，梳理出与系统实现相关的实际业务流程；

◆ 针对这些业务流程，绘制出跨部门职能流程图，以帮助开发人员对客户方的业务流程建立宏观、清晰的认识，以便更

加有的放矢地做进一步的需求捕获工作。

图三是一个跨部门职能流程图的实例，这取材于笔者曾经完成的一个物业中介行业的内部信息管理系统项目。

明确用户需求

知道了项目目标，了解了业务流程之后，接下来就需要进行更细化、深入的需求调查了。要做好需求调查，必须清楚地了解三个问题：What(收集什么信息)、Where(从哪收集)以及How(如何收集)。

1) 应该搜集什么信息?

针对这个问题，应该细化流程图，看看是否对每个环节、每个步骤都清楚地认识了。再根据自己的理解首先对每个流程的工作进行定义，写出事件流，并且标识出疑问点，这些都将使我们明白“应该收集什么信息”。

2) 从什么地方搜集这些信息

通常情况下，你需要的信息会藏于客户、原有系统、原有系统用户、新系统的潜在用户、原有产品、竞争对手的产品、领域专家、技术法规与标准里，在具体的实践中该从何下手呢？有了上面的流程图，流程涉及到什么部门、岗位，答案就应该从谁身上找。

经常看到这样的现象：需求调研的时候经常遇到有的人热心或刚好空闲；有的则很忙，总是没有时间。结果比较闲的A部门员工会坐下来和你聊一大堆信息，甚至会告诉你很多关于B部门的事情，而许多需求调研小组都会将这些话当作宝，结果也犯下了错误。这是因为A部门对于B部门业务的描述很可能是猜测、侧重了解，更有甚者是基于他自己心目中的设想。这一切都将会把你带到弯路上。这也是一个很值得引以为戒的例子。

具体来说，首先从人的角度来说，应该首先对涉众(也就是风险承担人、项目干系人)进行分类，然后从每一类涉众中找到1-2名代表；而对于文档、产品而言，则更容易有选择地查阅。在制定需求捕获计划的时候，不妨列出一个表格，左边写上你想了解的信息，右边跟上你认为可能的来源，这样就能够建立起一一对应的关系，使得需求捕获工作更加有的放矢，也更加高效。

3) 用什么机制或者技术来搜集这些信息

需求捕获的技术有许多种，但每种技术都有它的适用性和局限性，应该结合实际的项情况有针对性地选择。最常见的有以下几种：

◆ 用户访谈：最基本、最常见的技术

❖ 利：直接有效、形式灵活、交流深入，应该作为主要的需求捕获技术。

❖ 弊：占用时间长（特别当客户忙时更显示出其不足）、面窄而容易造成信息的片面性。

❖ 要点：首先要有准备：通常包括说明对流程的理解，并征得客户的意见；预先根据流程中的不明确点设计要询问的问题，并将客户的反馈记录下来，应留有一些即兴的空间，根据实际情况应变，以确保信息的完善。第二是要有计划性：计划好时间、计划好人员、计划好策略。

◆ 用户调查：调查面最广的技术

❖ 利：面广，能够获得更多的人的反馈。这点是对用户访谈技术不足之处的最好补充。

❖ 弊：不够深入，容易形而上学。而这点正是用户访谈技术所能够解决的。

❖ 要点：先设计问题，制作成为用户调查表，下发填写完后，进行仔细的分组、整理、分析，以获得基础信息，然后再针对这个结果进行小范围的用户访谈，作为补充。

◆ 现场观摩：最生动的技术

❖ 利：百闻不如一见，能够对需求与业务流程建立直观的认识。

❖ 弊：消耗时间长，而且由于“被观摩”的微妙心理变化，会使得“观摩”失真。

❖ 适用性：要对于复杂流程的更加深入的理解时。

❖ 要点：悄悄地进行，明确要强化理解的具体流程环节。

◆ 文档考古：最贴近实现的技术

❖ 利：能够详细、直观地对数据流细节进行了解与分析。

❖ 弊：容易陷入文山书海之中不可自拔，甚至经常引起误导。

❖ 要点：应该尽量让客户提供写有真实数据的文档。

◆ 联合开发：最理想的技术

❖ 利：客户、开发人员直接的头脑风暴，是击破需求盲点的关键手段。

❖ 弊：成本高，如果缺乏控制会变成一次闹扯大会。

❖ 要点：需要有一个经验丰富、能够把控大局的主持人。

需求分析与建模

了解客户的“业务需求”，并捕获主要的“用户需求”之后，我们就可以在此基础上进行分析与建模。分析的目的是为了理解、整理、合并这些需求；建模的目的是在理解需求的基础上，绘制出系统的蓝图，以便统一认识。

现代软件工程方法论中，需求分析与建模的最佳实践就是以OOA为指导思想的，以UML为标准的用例分析技术。有关这些内容，可以参考笔者《用例建模》（2004年第3期）和《鲁棒性分析》（2004年第4期）文章。下面主要从需求过程、需求管理的角度来探讨几个关键点。

何时进行？

通常，我们应该在“业务需求”充分理解，并且收集了最本质的“用户需求”之后就开始需求分析，但并不是等到需求捕获完全做完之后。瀑布模型的方法论现在早已经显得捉襟见肘了。

我们应该采用的是交替进行的方式，首先把握用户需求的主要部分，然后就开始分析，在分析的基础上引入系统级的需求（也就是从系统的设计与实现角度进行考虑），并且根据分析的结果建立分析模型，这个模型将成为伴随整个需求过程的关键内容，并且成为开发人员之间、开发人员与客户之间达成共识的一个平台。

然后在分析的基础上，会发现更多的不明确项，更多待捕获的信息，这时就可以生成第二次的需求调研的计划、问题、素材，并且更有针对性地进行二次捕获，然后再来完善分析模型，周而复始。

何时结束？

这个周而复始的循环，走到什么时候是个头呢？这个问题准确的答案是，系统交付时。也就是说，需求捕获、分析与建模、规格说明书的编写、需求的验证这个需求开发的循环，是在整个软件开发生命周期中存在的。

不过，每一次的循环，都将在需求开发的工作要点与份量上有所不同，它们应该遵循以下趋势：从本质到边缘，也就是最开始是本质性需求，然后是重要的、次重要的、一般的、镶金的……；从多到少，细化阶段（参见RUP框架）是需求开发最密集的阶段，而到了构建阶段开始需求开发的量将慢慢变少。

内容和形式

许多开发组织一谈到需求的分析与建模，就会想到UML，想到Rose、Together，想到厚重且显得呆滞的软件需求规格说明书……，而且经常形成一种“表面重视”的形而上学。换句话说，重视这种形式，而非其内容。

其实需求分析与建模不应该是孤立的行为，它应该汇集项目团队成员的智慧，在开放、融洽的气氛下协作完成的。而且其产生的结果也不一定非得是规范度很高的标准文档，而应该重在分析、重在方法、重在交流、重在解决问题。笔者认为：

◆ 团队聚在一起，利用白板甚至是纸张，在充分的合作下进行分析与初步建模是成本最低、效率最高、实用性最强的方法。

◆ 对于这些活动所产生的结果，可以利用数码相机、扫描仪进行文档化，并置入共享、版本控制，正如Robert Martin说过的“直到你一定要用时，再写文档”，文档是为了帮助开发工作而创作的，而不是为了文档而创作的。

◆ 对于比较重要、核心的内容，我们再采用Rose、Together这样的工具进行文档化，以便加强管理。

需求基线与变更管理

需求基线管理

需求的变化会破坏开发的节奏,我们要控制这种事件对开发的影响,要注意是控制影响,而不是控制其发生。

如果进行控制呢?那就是“需求基线”,在每次迭代或每几次迭代中,建立一个刚性的需求基线,在开发完成之前,开发组(注意是开发组)不响应变更,而是一门心思想如何完成基线中的需求,以保持良好的开发节奏。

如何构建这个基线呢?我们应该在分析的基础上,将需求整合成为用例或功能项,然后对其优先级、依赖性进行综合性评估,再将一部分优先级高的需求放入迭代的目标,周而复始,直到所有的需求都已经被实现为止。

那么如何判断优先级,根据“业务人员确定业务决定,技术人员确定技术决策”的原则,由客户进行判断。而帮助用户判断的好办法就是“满意度/不满意度”模型。也就是说,用以下两个指标来衡量一个需求的优先级:

◆ **满意度**: 取值1-5,表示当需求被实现时用户的满意程度。

◆ **不满意度**: 取值1-5,表示当需求未被实现时用户的不满意程度。

将它们取值相乘,就得到一个参考优先级,并且在客户的确定下进行微调,以作为基线设置的一个重要依据与标准。

而依赖性是指对于某些功能,在实现上有必须的依赖关系,即当某些功能没有实现时,另外的功能无法开始,这就需要对其进行调整。

变更管理

要做好变更管理工作,需要从两个方面着手:

1) 变更管理流程

需求变更应该正式化,如果具体的开发团队直接响应变更,将对项目造成很大麻烦。采用合理的流程是十分重要的。

◆ **提出变更**: 要求客户以正式的方

式提交变更是很重要的,这可以作为今后合同履行的依据,也可以为双方提供一个合约式的沟通平台。

◆ **变更评估**: 当接到需求变更的请求时,首先应该对其进行评估。评估包括三个方面:合理性评估、工作量影响、影响面分析评估。合理性评估的意思是进一步了解其变更的主要原因,认清其是否是因为沟通上的误会与不理解而造成的不必要的变更;工作量评估则是评估

其对进度的影响;影响面分析则是评估该变更会对哪部分工作产生影响,具体地说会对哪些人的工作产生影响。

◆ **分级响应评估**: 如果变更不影响相关模块开发进度的,可直接响应;如果影响本模块开发进度但不影响项目总体进度的,可由项目经理协调后直接响应;如果影响项目进度的,则应该交由市场人员与客户协商响应方式,这也是避免“丧权辱国”式合同执行的重要措施。

2) 需求跟踪

要想正确地评估变更的影响面,以及其所带来的工作量变化,很重要的是对需求进行跟踪管理。具体来说,就是要建立一张映射表,将需求项与分析项、设计项、实现项建立清晰完整的对应关系,以便一目了然地获得所需的信息。

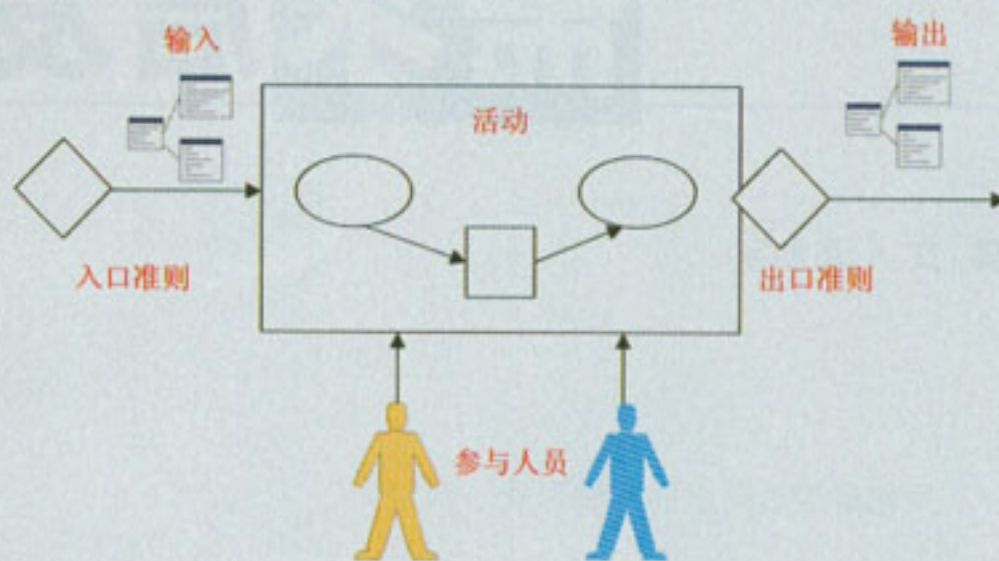
这项工作可以借助 RequirsitePro、Doors来解决,也可以通过Excel创建对应表格,其关键还在于经验的积累。

如何制定过程规范

通过上面的描述,将对整个需求工程的关键组成部分和流程环节有了完整的认识。因此,可以在这个基础上制定组织的需求管理过程规范。过程规范的制定包括过

程描述、过程指南、过程模板三个方面。

过程描述是最核心的内容,通常包括:参与人员、入口准则、出口准则、输入、输出、活动等部分,它们之间的关系如下图所示:



图四 过程规范结构示意图

然后编写相应的活动指南和文档规范,以供在执行软件过程规范中使用。另外,还要注意的,在改进初期切忌大而全,软件过程规范写在纸上是没有用的,只有写在团队成员的脑子里才能够体现出作用。因此,一开始尽可能地简单,让团队适应后,再根据实际情况修改、增加,不断地进行改进。

结语

最后借用温伯格大师的智慧,对需求进行一个总结,那就是“需求是需要探索的”,在我们的开发实践中,首先应该采用迭代的思路,不断地探索需求,分析需求,实现需求,响应需求的变更。

在需求开发的整个过程中,应该紧抓主旋律,也就是从本质开始,边捕获、边分析、边实现,通过不断的迭代交付来响应需求变化,以需求基线来保证开发的节奏,通过对变更的管理来更好地响应变化。总而言之,需求管理是整个开发生命周期中的重中之重,有效地根据团队、项目的实际情况,大胆尝试各种新的需求方法论,并且通过先局部试点再全局推广的循序渐进过程,必将能够有效地提高需求能力。

■ 责任编辑:罗景文 (ljw@csdn.net)

需求问题根源

需求启发技术



■ 文 / 潘加宇

用例是表达需求的一种形式，例如：

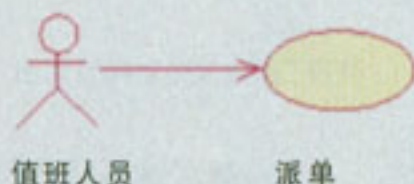


图1 一个系统用例

表达了这样一个要开发的系统的需求：“值班人员”需要使用系统来达到“派单”的目的，或者说，“派单”是系统应该为值班人员提供的一种价值。但问题来了：我们怎么知道“值班人员”使用系统来“派单”是一个合适的需求？答案只能来自涉众，和“派单”这个事情利益相关的人们。

需求只能从涉众中来，涉众包括最终用户、客户、政府、法律、文化、开发人员、管理人员、竞争对手……但是，需求不能直接从涉众中来。需求工程师往往不能通过询问涉众“你的需求是什么”来得到需求。经常有人抱怨“客户什么都不懂”，或者纳闷“客户需要懂得UML或者用例吗？”。其实需求工程师应该把启发的责任揽过来，而不应该过分强求客户。想想有这么一类无知的涉众：婴儿。婴儿只会哭和笑，不会直接表达需求，但出色的公司仍然可以从婴儿身上探索出“天线宝宝”这样的需求。通过各种手段探索到客户“想要的是什么”，是需求工程师最重要的能力。



图2 天线宝宝

启发障碍

涉众往往存在以下启发障碍：

1、涉众无法陈述自己的需要。即使说出来了，也可能是解决方案不是需求。涉众说“我要基于Web的三层架构系统”（从报纸上看的或从别的地方听来的），其实他真正想要的很可能是“想让分布在各个地方的员工能一起工作”。

2、对于某些新产品的研发，开发者往往是第一个想到的人，涉众不会直接构想出新的工作方法。这种情况，更不可能从涉众那里得到直接的需求。

3、涉众的利益矛盾。开发一个系统，往往会涉及很多人的利益。有的人会获得权力，有的人会丧失权力。不同立场的涉众之间必然会存在矛盾，如果启发工作安排不当，这种矛盾会成为一种启发障碍。更不用说有些系统的开发可能会使某一类涉众下岗或者使得头脑里面的经验不再重要，导致这类涉众抵制变更。

所以，要根据不同的情况，采用不同的启发技术来探索涉众的真正需求。

常用启发技术

文档研究

文档研究往往是需求调研的第一步，主要是为了获取业务领域的初步知识，为下一步的启发工作作准备。研究的文档资料可以是工作中的表格、文件、便函、工作报告、作业日志……等等。

文档研究的时候要注意：文档资料往往会比较多，有价值内容的相对比例较少，如果碰到有价值的信息，随时做笔记，或者把该页复印、扫描甚至撕下来。

问卷调查

当需要调查的人群分布较广时，随意挑选几个人来访谈或观察是不够的。例如，要去调研一个即时消息软件的需求，如QQ，如果在自己公司随便找几个人问，事实上这几个人很可能都属于一类涉众“软件开发人员”，他们的利益诉求不能代表即时消息软件其他类型的涉众，例如“中学生”、“新闻工作者”，“网络管理部门”。如果涉众的分类尚不明确时，就需要做一些问卷调查（电子的或者纸张的），根据问卷调查的结果来给涉众

分出类别,然后再从各类中选取样本,进行下一步的启发工作。

访谈

访谈是最重要也最常用的需求启发技术。需求工程师和涉众直接交流以收集信息。我们可以分几部分来说这种技术。

1、需求工程师

需求工程师的态度要让涉众觉得自己被尊重。首先是言语上的礼貌:例如,不能表露出“你不懂软件!我才是专家”的意思,“懂得软件”不是涉众必须的素质,涉众只需要清楚自己的利益和关注点。另外,访谈的涉众往往处在一个从业人员平均学历和能力都比软件业低得多的行业,涉众可能在接受访谈时潜意识中有一种自卑感,如果需求工程师的态度不礼貌,更容易引起抵触心理。另外,不可忽略这个事实:系统的出现可能对受访者不利。要么是头脑里的经验不再那么重要,甚至职位还可能会取消。所以在言语中暗示“我们在这里是为了让你下岗”的意思是不适当的。即使是表示“我们在这里是为了帮助您把工作做得更好”也不合适。他的工作做得很好!并不需要我们帮助。而应该把自己摆在一个低姿态的位置上“我们在这里是为了帮助您更方便地完成工作”——方便,想方便的时候就方便一下。

其次在行动上,访谈的时候身体应前倾,不时点头,发出声音“嗯—嗯”,手上做一些笔记(表明重视),适当的时候作两句总结。这样,涉众的心里多半会大为受用,向你倾出心中所想。

访谈的时候光听肯定是不行的,要想办法记录。可选的方案有:

自己做笔记——手上肯定会有做笔记的动作,但记录的速度是比不上说话的,而且还会因为问答,节奏经常被打断,不能因为低头专注记笔记而影响交流。所以,边问做笔记只能记住一些关键点和做出尊重涉众的姿态。

同事做笔记——如果有两个人去访谈,可以一人主问,一人主记,然后再掉换角色,最后将笔记合并。

录音——现在录音设备越来越小了,放在桌面上基本不会对访谈造成影响,是一种非常好的记录方法,如果说有什么不足之处,就是只记录了声音,回去后无法研究涉众的表情来确定涉众的真实意思。

录像——可以看到肢体语言,但在有一个镜头对准的情况下,受访者往往受到影响。

有可能的话,记录手段应该采取双备份。录音笔会出故障的,因为去一趟不容易,保险一点为好。但要把录音录像当作好像不存在,该倾听倾听,该记笔记记笔记。

2、涉众

涉众代表必须名副其实。“涉众代表”不等于“部门主管”。

要访谈车间的操作工,就不能用车间主任来做代表。操作工岗位的酸甜苦辣,只有操作工自己才知道。应该把“车间主任”看作是另外一类涉众。实际工作中常见的是把甲方的“信息中心主任”头衔的涉众作为主要的需求来源,认为他们既懂电脑,又懂业务,其实大谬。

不同类型的涉众,应该尽量单独访谈,如果图省事把有利益冲突的两类涉众集中到一起访谈,受访者言辞之间可能就会有顾忌。

3、访谈问题

内容上,访谈就是要收集用例的素材,所以用例思维在这个地方就可以起到指导作用。例如,问“为了完成这项工作,您需要哪些材料?”,可能就对应到“用例的前置条件”,问“这项任务要是做得不好,会影响到谁?”可能就对应到“涉众利益”,问“如果您不做这项工作,公司会受到什么损失?”可能就能追索到“业务用例”。

形式上,和新闻记者提问一样的:5W+1H。谁(Who)、什么(What)、什么时候(When)、什么地点(Where)、为什么(Why)、怎么进行(How)。提问的时候尽量采用业务词汇,不要采用技术词汇。

4、环境

访谈的环境。要在涉众的工作环境里进行。涉众在自己的工作环境中会想起许多工作中的喜怒哀乐,如果把他搬到软件公司或者宾馆的会议室,环境发生变化,一些本来深有感触的东西,因为环境的因素,一时之间会想不起来。



图3 访谈要在涉众的地盘进行

观察

观察就是需求工程师亲身去体验涉众的工作。这是最直接的技术,也是最费时间的技术。

要挑选经验丰富的涉众来观察。因为系统的开发就是为了分担他的经验。经验丰富的业务工人在长期工作中归纳出一套行之有效的方法,这套做法也许就是系统应该借鉴的。

观察时,重点关心:完成一项工作所需时间、操作次数、出现的错误和混乱。所需时间越多、操作次数越多、出现的错误越多,系统如果能在这项工作有所帮助,必将会给涉众带来强烈的改进感觉。

观察在今天的产品研发越来越重要。在市场竞争激烈的体验经济时代，客户的口味提高了，光是有个东西给他用是不行的，市场上有十几二十种同类的产品。对精益求精的产品开发者来说，观察是不得不花代价去做的启发技术。

开会

开会就是把不同类型的涉众集中到一起，往往用于验证需求和解决冲突和妥协。不同涉众有不同的利益诉求，归纳成需求文档后，可能就会发现利益之间的冲突导致需求无法确定。这时，把大家召集在一起，验证写好的需求文档和集中商讨其中的冲突。

开会时值得注意的是小心少数人主宰讨论。如果高级别的与会者不断插话，身为下属的与会者可能就不敢发表意见。所以，事先可以就此提醒高级别的与会者。

研究竞争对手

这是做产品型软件的厂商最重要的考虑。如果说以前产品的需求是开发人员导向的，现在软件公司也逐渐意识到需要以客户为导向，所谓“客户是上帝”。问题在于，这只是必要条件，而不是充分条件。所有公司都知道要“面向客户”，这个时候客户如何才能从这么多家中选择你？

研发就是战争，战场是客户的头脑。必须研究竞争对手，做出对应的策略，而这些策略直接影响了需求。市场经济里，一个领域可能都会有以下战局。市场领先者负责防御，扎紧自己的篱笆，不断更新自己。几家跟随者，它们负责进攻领头羊。其他的众多公司想尽办法占据一小块细分市场。只有研究清楚对手和战局，才能了解自己的位置和应该采取的战略。

可以借用齐白石的话，“学我者生，似我者死”。不是“更好”，而是“不同”。研究竞争对手时，思路就是：研究对手的优点，但不是模仿，而是在关键的地方反其道而行之。例如在中国，MSN的“不和陌生人说话”进攻QQ的“爱陌生人”。

从素材到需求

得到素材之后，需求工程师需要结合软件的愿景，把各种素材碎片归纳成需求。并不是所有涉众的需要都会变成软件需求。象不符合愿景的要求，例如：目标是改进进货流程的效率，访谈到供应商时，提出了自己库存管理上的一些要求，这和愿景就不相关了，所以这样的要求不能变成软件的需求。还有不是计算机所擅长的要求。计算机最擅长做什么？一是信息的自动流转，如：患者做检查后检查图像自动流到医生桌面；二是演绎复杂业务逻辑，如：比较价格寻找最优价；三是访问和操作业务实体，如：把病历电子化以方便管理和检索……对一些

需要人去判断的地方，计算机实现是很困难的——对这一点，涉众有时并不了解，经常提出过分的要求。

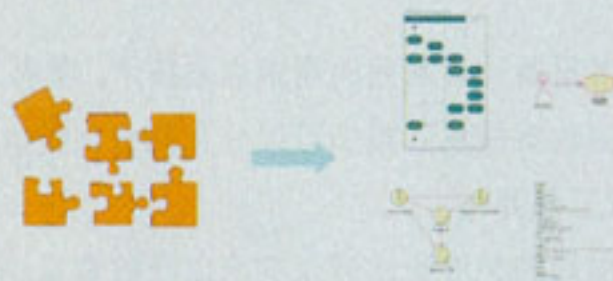


图4 把素材归纳成需求

素材本身是跳跃多个边界的。要在不同边界来回思考，需求工程师有强的归纳合成能力，并且掌握一些思考的工具。需求过程本身主要是和人打交道的工作，软件工具帮助有限，目前需求工具的帮助局限在“管理需求”，而在“启发需求”和“定义需求”上难以起到作用。温伯格的《探索需求》介绍了一些思考的工具，此书是目前少数几本专注于“启发需求”方面的书。

结语

软件开发中的需求过程可以通过活动图简要表示如下：

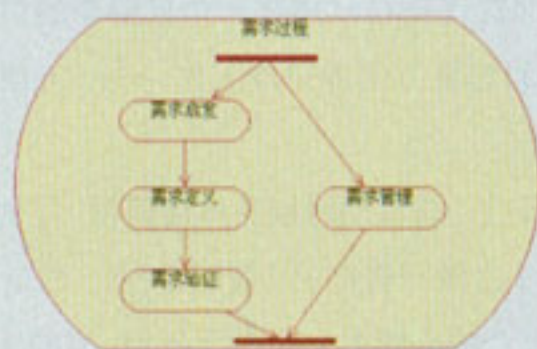


图5 需求过程

需求启发：从涉众那里得到需求的素材。

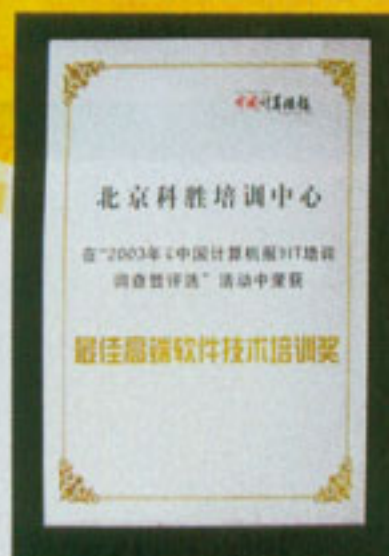
需求定义：把有价值的素材组织成需求文档（用例文档）。

需求验证：让涉众验证需求文档。

需求管理：跟踪需求，管理变更……

这四个环节里面，除了需求管理环节目前已经有一些工具如Borland的CaliberRM，Rational的RequisitePro……来辅助之外，其他环节上，工具很难起到大的帮助。需求启发和需求验证的绝大部分工作是在人和人（需求工程师和涉众）之间进行，强调的是人际交流的能力，需求定义需要的则是思考的能力。很难想像有这样的工具：客户只需呆呆坐着，把它套在客户头上5分钟，就可以知道客户“想要的是什么”；也很难想像这样的工具：它能够从一大堆乱七八糟的笔记和录音里，自动整理出一份整齐的需求文档。

当我们关起门来“编写有效用例”的时候，当我们激烈争辩一个用例的“粒度”的时候，一定要记住，答案根本不在我们这里，在涉众那里。到第一线去，用我们上面提到的各种技术，探索涉众心中所想，才是需求的正道。



北京科胜培训中心 软件技术专业培训

The success you want and the confidence you need—guaranteed here!

北京科胜培训中心是中国最早从事计算机技术培训的企业之一，是著名的技术培训和 IT 图书出版、电子商务等软件技术服务企业。中心依托北京大学、清华大学等高校和中科院坚强的技术后盾、以及中关村知名软件企业的骨干开发力量，自身丰富的管理经验和优良的教学水平，以培训软件开发人才为主线，面向计算机用户和专业人士、团体、企业提供各层次计算机技术培训服务，先后为众多的知名企业和单位提供培训和咨询服务，深得用户的认可和好评。中心摒弃纸上谈兵，强调项目实战，培养软件精英人才，被学员赞誉为“程序员的摇篮，高级软件人才成功的阶梯”，荣获 2003 年度“最佳高端软件技术培训奖”，是软件人才继续教育和进修的理想选择！

类别	培训课程班次名称	学时	培训费	
			公费	自费
IT 认证 电子商务	专业网站建设/网页制作与 Macromedia 认证 (专业多媒体网页制作与提高)	64	1260	980
	ASP 交互式电子商务网站设计与实现	56	1280	990
	Authorware 多媒体制作软件应用	40	1280	1050
.NET 系列	VB.NET 高级编程与 .NET 架构	56	1560	1200
	ASP.NET 高级网站设计与实施	56	1580	1200
	Visual C# 语言及 .NET 开发技术	40	1380	990
JAVA 系列	JAVA 2 语言高级编程	48	1260	980
	JSP 网站编程和远程管理系统	48	1280	990
	J2EE 企业级应用系统开发	56	1680	1260
	J2EE 应用系统设计和 Weblogic 实战训练	20	1280	960
	IBM WebSphere 高级培训	56	3850	3050
高级 程序 开发	VC++ 6.0 高 C/C++ 程序设计	48	980	800
	级应用编程 VC++ 深入编程 (含 COM)	88	1580	1260
	COM、DCOM、COM+ 面向组件编程	32	1280	1000
	Win2000 系统编程与驱动程序设计	40	1500	1200
	Visual Basic 6.0 应用程序设计	56	1260	980
	Delphi 7.0 高级程序开发	56	1480	1200
	Power Builder 及 MIS 系统开发	56	1280	960
数据 库	SQL Server2000 系统管理和开发	64	1350	990
	ORACLE 8i/9i 管理、开发和性能优化	80	2990	2400
	IBM DB2 数据库系统管理与开发	56	3600	2880
	Sybase 系统管理和开发	56	2600	2080
UNIX 系列	Linux /UNIX 使用、系统管理和网络建设	56	1680	1280
	SUN Solaris 操作、维护与管理	56	2500	2000
	Unix/Linux 环境下 C 程序开发	48	1500	1200
	Linux 驱动程序设计及嵌入式开发	48	1500	1200
硬件 与 网络	电脑组装、维护与维修进阶班 (提供实习)	40	980	800
	局域网组建与维护工程师班 (提供实习)	40	1200	960
	Windows 服务器网络工程师高级班	48	1480	990
优惠 套餐	第七期“百日造就软件人才”工程 程序员转行就业长期班 (从零起步)	4 月	9800	8000
	Web 网站与电子商务开发专家	280	4600	3800
	VC++ 项目实战进阶	200	4800	4050
	JAVA 高级软件工程师	232	6800	5500
信息 安全	操作系统安全 (Windows/UNIX/Linux)	40	4200	3500
	黑客攻击与防御	40	4800	4000
	安全产品实用技术	40	6000	5000
	安全管理理论与实践	40	9600	8000
	CISSP/BS7799/NCSE 安全认证考试辅导	详见网站		
	信息产业部《数据恢复职业认证》	56	2800	2500

为方便大家了解最新技术，科胜
长期开办免费公开课，敬请关注！

团体培训，专家咨询，
定制内容，签定合同，上门服务

团体/上门培训典型客户案例

中国银行监督委员会，北京市商业银行 115 家支行，国家税务总局扬州进修学院，北京市 500 所中小学骨干教师培训，新疆自治区税务系统，安阳钢铁集团公司，北方计算中心，平安保险公司北京分公司，中科院遥感应用研究所，铁道部信息中心，山东行政学院，济南钢铁集团自动化部，天津大港油田，邯郸钢铁集团，大庆油田，东营胜利油田石油管理局，总参某部网络中心，北京普纳科技集团，公安部信息所，中油股份有限公司吉林油田分公司，华北计算技术研究所，鄂尔多斯市电信分公司，国家海洋标准计量中心，唐山启新水泥有限公司，重庆西南铝业（集团）有限责任公司，安徽亳州古井贡酒集团，新疆库尔勒塞龙计算机培训中心，北海舰队指挥所，北京市国家税务局燕山分局，中国人民银行国有资产监事会，大庆油田职工医学院，江西南昌电业局职工培训中心，北京科胜博达技术开发有限公司，等等……

咨询电话：010-62610417 62610418 62610419

团体培训：010-62610415 51668758

网 址：<http://www.ecoms.cn>

E-Mail：reg@netcenter.net.cn

地 址：北京大学南门对面科城大厦三层

邮政编码：100080

计算机软件高级技术培训，颁发国家人事部监制的《中国继续教育联合学院》钢印证书
ECOMS training fits your needs, especially for IT Professionals and Computer Users.



- P70 “C++/CLI 全景体验” 专栏开篇语
- P76 聆听未来——Stan Lippman 谈 C++/CLI
- P78 CSDN C++/CLI 调查报告
- P79 C++/CLI 会冲击 C# 吗？
- P80 C++/CLI：鼎新革故
- P83 山雨欲来风满楼——标准 C++ 及 C++/CLI 发展综述
- P85 非典型 C++/CLI 教程

C++/CLI

凤凰的涅槃

自上世纪八十年代末以来,C++就成为程序员中长盛不衰的话题。对于九十年代成长起来的中国程序员来说,C++更有特殊意义。他们是从Borland C++那里体验了编程的乐趣,从C++教程里摸索面向对象的真义,从Visual C++入手体验Windows编程的奥秘,从MFC中体会框架的宏大和精致,从STL步入泛型编程的大门。可以说,C++伴随了整整一代程序员的成长与成熟。不但如此,极盛时代的C++在工业界据有至高的地位,在九十年代的最初几年,从前端到后端,从系统层到应用层,C++全面渗透。

“四十年来家国,三千里地山河。凤阁龙楼连霄汉,玉树琼枝作烟萝。几曾识干戈!”

但是,进入九十年代末以来,形势急转直下,几年之内,风云变幻。C++因为其固有的弱点,在竞争中渐显颓势。先是Visual Basic和Delphi等可视化开发工具在Windows客户端编程领域抢占了C++的支配地位,随后是Web的兴起给了C++又一个沉重打击;Java,尤其是J2EE的大行其道,在服务器端给C++带来了严重的威胁;.NET的发布之后,C#成了聚光灯的焦点,而Managed C++ Extension则因为它的丑陋沦为二等公民,一切似乎不可逆转,C++即将失去最重要的堡垒。

难道真的是“落花流水春去也”?

2001年冬,Stan Lippman加入了微软Visual C++开发组,次年春,Herb Sutter以社区联络官的身份入职微软,转而升任Visual C++首席架构师。仅仅一年之后,在PDC 2003大会上,Stan Lippman和Herb Sutter就对外公布了“新的C++ .NET”,也就是我们今天所知的C++/CLI。紧接着,微软开始在各种场合介绍C++/CLI的特性,并在前后不到4个月的时间里,发布了两个Visual C++ 2005的测试版。而且从这两版的特性比较来说,我们可以推测,微软正在紧锣密鼓地改进C++/CLI。如果说C++在企业应用领域内的迅速衰落,让人有“逝者如斯夫”的惋惜,那么C++/CLI重新崛起的速度则是令人惊诧。

为了帮助我们的读者了解C++/CLI的具体内容,本期技术专题隆重推出的C++/CLI专栏。我们很荣幸的邀请到C++/CLI创造者之一、世界著名的C++大师Stan Lippman先生和国内著名的.NET技术专家李建忠先生,为我们的读者开辟了一个专论C++/CLI的专栏“C++/CLI全景体验”,本期将刊登这个专栏的第一部分。此外,我们还组织了几篇文章,帮助读者抢先了解C++/CLI的特性、意图、特点和发展思路。

■ 策划编辑: 孟岩 罗景文

Stan Lippman

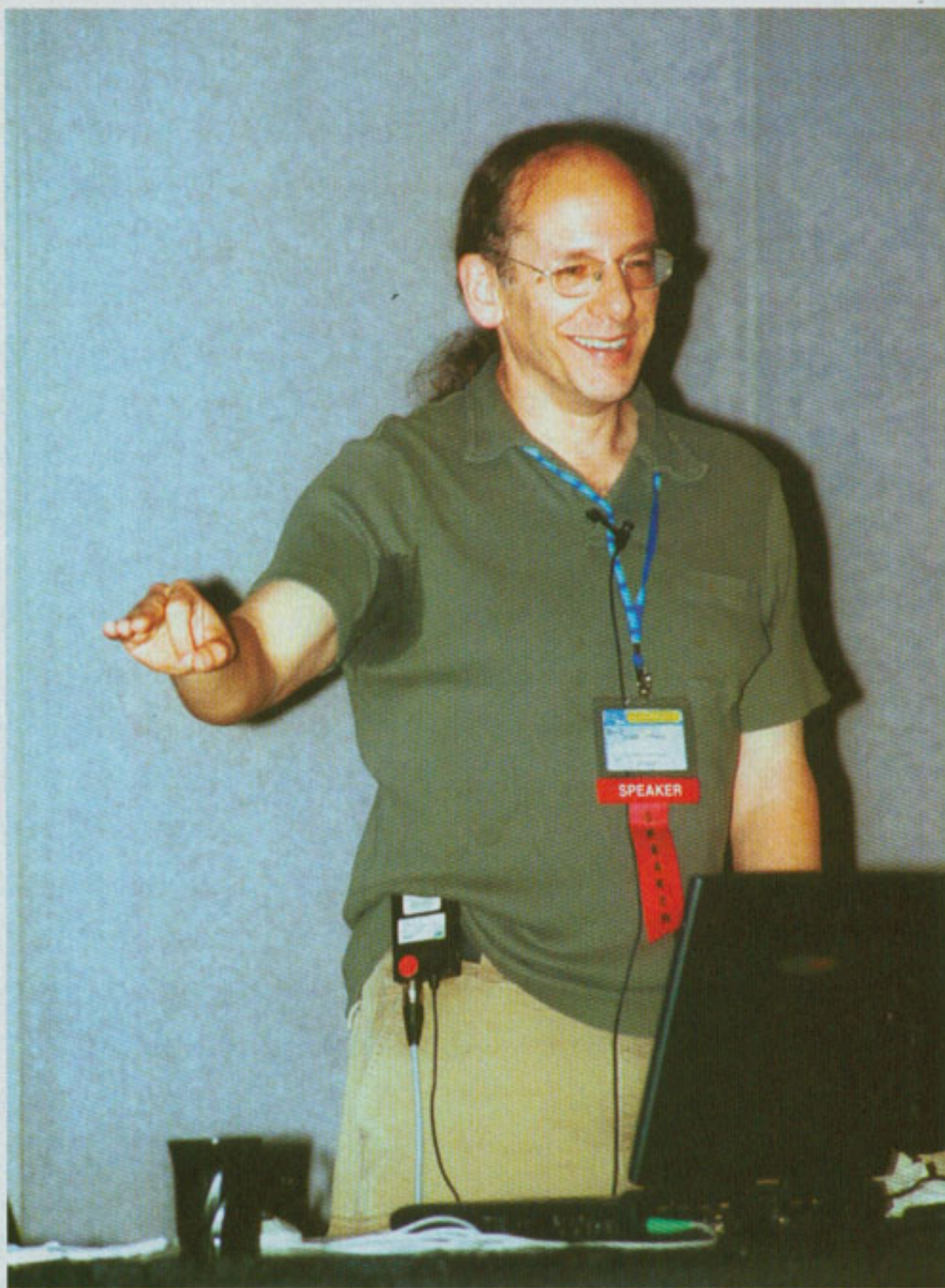
著名的 C++ 语言先驱，从 1984 年开始一直从事 C++ 方面的工作。曾在 Bell 实验室工作 10 余年，是与 Bjarne Stroustrup 一起工作的早期成员之一。还曾参与开发最初的 C++ 编译器，致力于 Cfront 的各种实现。而后又参与了 Foundation 研究项目中关于程序设计环境的对象模型部分。现为 Microsoft Visual C++ 开发组的系统架构师。著有《C++ Primer》等多本经典著作。

▶▶▶ Stan Lippman

最近我访问了中国的上海和北京，参加在两地举办的微软 Tech Ed 技术大会，在那里我非常荣幸地向大家介绍了我们在 C++/CLI 方面的工作。大家的反馈非常之好，特别是中国年轻一代程序员对 C++/CLI 的热爱和理解给我留下了深刻的印象。在那里，我还认识了来自上海的一位开发人员，同时也是一位技术作者，李建忠先生。我们经过讨论之后决定合作撰写一系列 C++/CLI 方面的文章，并以“C++/CLI 全景体验”专栏的形式独家授权于中国《程序员》杂志发表。此篇短文旨在为大家简单介绍一下我们写作这个专栏的一些背景——有点电影中“定场镜头”的味道。

面对 C++/CLI，很多人的第一个问题自然是“什么是 C++/CLI”，我个人喜欢将其看作是位于静态程序设计和动态程序设计之间的一座桥梁。C++/CLI 这个名称本身就包含着一组术语——而其中最重要的术语却是最不明显的那一个。

首先来看第一个术语“C++”，这当然指的是由 Bjarne Stroustrup 在 Bell 实验室时发明的 C++ 编程语言。它所支持的是一种为代码执行速度和执行体



所占空间所高度优化的静态对象模型。除了堆内存分配以外，它不支持在运行时对应用程序进行任何的更改。它允许我们对底层机器进行无限的访问，但对于正在运行的程序中的活动类型、以及相关的程序基础构造，它的访问能力却非常有限、或者根本就不可能。它是一门非常成功的编程语言，但是它却不能适应目前的 Web 编程环境以及相关的安全问题——这已经成为目前程序设计中一个越来越重要的考量。

再来看第三个术语“CLI”，即通用语言基础构造 (Common Language Infrastructure)，这是一个支持动态组件编程模型的多层架构。在许多方面，它所表示的对象模型和 C++ 的完全相反。在 CLI 中，存在一个运行时软件层（即虚拟执行环境）运行在应用程序和底层操作系统之间，应用程序代码对底层机器的访问会受到相当严格的限制；事实上，CLI 根本不允许安全环境中的代码进行这样的访问。但另一方面，CLI 却允许我们对正在运行的程序中的活动类型、以及相关的程序基础构造进行完全的访问，甚至允许我们动态构造额外的类型和程序基础构造。这些灵活性的获得当然伴随有相当的空间（执行体所占空间）和时间（程序执行效率）代价，但是它却解决了日益增长的基于连接的计算环境中所面临的问题和需要。

最后，再来看第二个术语，即中间的斜线“/”，它往往为人们所忽略。其表示对 C++ 和 CLI 的一种绑定 (binding)，它正是 C++/CLI 设计的焦点所在。据此，对于“什么是 C++/CLI”这一问题可能的一种答案便是“它是对静态 C++ 对象模型和动态 CLI 组件模型的一种绑定”。

对于 C++/CLI，C++ 程序员只需要将其添加到已有的编程工具箱中即可。要成为一个 C++/CLI 程序员，你无需放弃任何已有的东西，虽然你要步入一个新的技术世界，你仍需要学习它——但

愿你能享受这一过程，至少我知道我是这样的。由此观之，我们还可以将 C++/CLI 看作一扇通往另一个世界的大门。

C++/CLI 将动态的、基于组件的编程模型和 ISO-C++ 集成在了一起，这种集成非常类似于我们当年在 Bell 实验室对使用模板的泛型编程和当时的 C++ 所做的集成。在两种情况下，你已有的代码投资和编码经验都将得到保留。这是我们设计 C++/CLI 时一个基本的需求。

通用语言基础构造 (CLI) 是一个多层的体系架构，它为所有 CLI 语言提供了各种各样的服务。例如 CLI 中定义了一个通用类型系统 (Common Type System，简称 CTS)，而各个 CLI 语言都提供了自己对 CTS 的一个映射。该类型系统由一个根基类开始被组织为一个完整的类继承体系。实际上，每一个 CLI 类型都是一个类——不仅包括像 integer、double 这样的数值类型，而且也包括字面常量 (literal constant)。每一个 CLI 类型（或者值）都表示一种 Object（所有 CLI 类型的根基类），比如数值 3.14159，比如字符串常量“Homer Simpson”。

单一的根基类为运行时类型查询和代码生成（通常被称为反射）提供了支持机制，这是 ISO-C++ 所缺乏的。我们将在今后一系列文章中详细讨论它们给 CLI 带来的动态编程特性。

除此之外，CLI 还支持一种被称作特性元数据 (attribute metadata) 的构造，它允许我们定义一些特性类，然后将其关联在 CLI 类型和当前正在运行的程序构造上——这有效地扩展了内建于 CLI 中的类型和程序构造。这些用户定义特性也可以通过反射机制来获得，应用程序则可以根据它们的值来进行条件逻辑判断。这也是 C++/CLI 为 C++ 带来的动态组件编程的一部分。再次强调一遍，类型反射和特性将在我们的专栏中得到深入的讨论。

那么，对于大家来说怎样学习 C++/CLI 呢？学习 C++/CLI 的其中一个要点



便是学习底层的通用类型系统 (CTS)，它包括以下三种类型：

1. 多态引用类型，其用于所有的类继承。我们将在早期的一些专栏文章中讨论它们。

2. 非多态值类型，其用于实现一些类似于数值类型那样的、对运行时效率要求比较高的类型。我们将其放在引用类型之后讨论。

3. 抽象接口类型，其用于定义一组供引用类型或者值类型实现的操作。接口为多继承提供了一种别样的设计模式。我们也将有一系列专栏文章来讨论它们。

将CTS映射为一组语言内置类型对于所有的CLI语言都适用，虽然各种语言所使用的语法各不相同。这也是一门CLI语言所要面对的第一个设计层面。例如，在C#中，我们可以用以下代码来定义一个抽象基类型Shape（一些具体的几何对象将继承自它）。

```
public abstract class Shape { ... }
```

而在C++/CLI中，我们用下面的代码来定义同样的类型。

```
public ref class Shape abstract { ... };
```

除了语法差异之外，两种声明的实际表示完全相同。类似地，在C#中，我们可以用下面的代码来定义一个具体类Point2D。

```
public struct Point2D { ... }
```

而在C++/CLI中，我们用下面的代码来定义同样的类型。

```
public value class Point2D { ... };
```

我们对语法的选择基于如下的出发点：以一种直观的设计视角将CLI类

型和ISO-C++类型紧密地集成在一起。

因此，简单地说一种语言比另一种语言更接近底层CLI并不正确。相反，每一门CLI语言都只是表达了自己对底层CLI对象模型的一种视图。

学习C++/CLI的第二个要点是学习我们选择直接提供给程序员操作的那些底层CLI元素。例如，CLI为所有语言都提供了垃圾收集服务。一门语言不能选择是否支持垃圾收集，而只能选择如何更好地提供该服务。

在CLI中，一个引用类型的所有对象都只能被分配在CLI托管堆上。这意味着C++/CLI支持两种动态堆——本地堆（没有任何形式的自动内存回收机制），和CLI托管堆。对于这两种动态堆，开发人员通常要用某种形式的新操作符来分配对象：如果操作成功，对象在堆中初始位置的地址将被返回。但是两者又有所区别，这是因为CLI托管堆中对象的位置有可能在垃圾收集器的清除以及随后的压缩中被重新调整。如果一个对象的位置被重新调整，那么CLI运行时中所含的其中一项服务会透明地更新所有引用该对象的指代品（thingee）。

这就使得我们面临一种困难的选择：是将这些指代品称为指针，并继续用指针的语法来表示呢？还是引入一种新的类似的语法来表示它们需要特殊处理？我们最后决定采用后者，看下面的代码：

```
N *pn = new N;  
R ^rn = gcnew R;
```

这里，N表示一个本地类型，而R表示一个CLI引用类型，帽子状的符号（^）表示相关的地址是一个托管堆上的追踪句柄（tracking handle）——也就是说，对象位置的任何重新调整都会被CLI所追踪，相应的句柄也会被透明地更新。其中关键字gcnew在这里被用作与CLI托管堆打交道的new表达式。

值类型事实上也可以位于托管堆上，

虽然这并非必须。当它们作为一个引用类型的成员时，就会出现这种情况。如果我们允许获取一个引用类型内部成员的地址，那么本地指针也是不合适的，因为这些成员的位置也需要被追踪。一种解决方法是简单地禁止该项功能。这样语言当然会变得更加简单，但是同时语言也会变得更弱——例如我们将不能通过增长元素的地址值来遍历CLI数组，这是因为CLI数组是一个引用类型，其内的元素都位于托管堆上。不提供这样的功能意味着CLI数组将不能适用于标准模板库（STL）中的iterator模式以及泛型算法。对于一个C++程序员来说，这是不可接受的。

支持获取可能位于托管堆中的值类型的地址同样需要引入一种追踪指针，我们称之为追踪内部指针（tracking interior pointer）。另外，我们还支持追踪引用（tracking reference）这样的概念——它具有类似本地引用的别名语义，但是它会在必要时被CLI透明地更新。最后，我们还支持一种固定指针（pinning pointer）的概念，它可以在该指针的作用范围内阻止垃圾收集器移动其所引用的对象。

这些新的符号及其表示的复杂的间接类型是在我们对托管堆反复学习和认识之后产生的。面对生存期短暂的托管堆对象，我们需要某种精巧的方式来认识和使用它们，我们相信这些额外的间接类型可以给大家很多帮助。我们将在今后的专栏文章中详细讨论它们。

我们在此对一门CLI语言所选择的第二个设计层面表示了其对底层CLI实现模型的一层映射。选择什么样的映射取决于该编程语言定位于什么样的程序及程序员模型。当你选择一门CLI语言进行编程的时候，你实际上也是在选择遵从一种程序员模型。我们对于C++/CLI程序员的定位是那些历练较深的系统程序员，这些程序员通常所面对的任务是为高层的商业逻辑提供基础性的构造和关键性的应用，这时候她就必须要同时考

虑系统的扩展性和性能,因此必须对底层 CLI 有一个系统级的视角。

学习 C++/CLI 的第三个要点是学习那些非 CLI 本身所直接提供的功能特性。这也是每一门面向 CLI 的语言所要面对的设计选择,也是各种 CLI 语言之间相互区分的一种体现。

例如,CLI 本身并不支持多类继承 (multiple class inheritance, MCI),而只支持多接口继承和单类继承。但 Eiffel 在设计其面向 CLI 的实现时就选择了支持源代码级的多类继承。这需要一种巧妙,甚至是复杂的设计将源代码级的多类继承映射为底层 CLI 的单类继承模型。Eiffel 语言的设计人员认为这种映射对于 CLI 平台上的 Eiffel 程序员是一个利好的元素。

在此 C++/CLI 的第三个设计层面上,我们没有采用多类继承的方案。其中一个原因是我们不能说服自己多接口继承模型有任何不够简单或者优雅的地方。我们没有足够的经验来确定哪种方案绝对的优秀,但是我的直觉告诉我多类继承 (MCI) 是一个死胡同。我们在此设计层面上的主要关注点在于为那些 CLI 本身所欠缺的地方提供一些额外的解决方案,我们主要集中在以下三个方面:

1. 为某些 CLI 要求手动干预的地方提供一种自动化的解决方案,例如确定性终止化操作 (deterministic finalization) 和稀有资源释放。
2. 提供一些特殊的类成员函数——例如拷贝构造器和拷贝赋值操作符,以及在 CLI 直接支持的操作符的基础上再为一些操作符提供一些扩展支持——例如用来支持函数对象 (function object) 设计模式的调用操作符 “()”。
3. 提供一种静态的参数化机制来支持设计适用于 CLI 类型的标准模板库 (STL),这是因为 CLI 中的泛型机制在我们来看对于当代的参数化设计是不够的——虽然我们也支持它们。

以上几点在我们的系列专栏中都将有相关的讨论。特别地,我们将会详细阐释 C++/CLI 中的模板和泛型机制。

C++/CLI 的第四个设计层面在于它选择了“集成”而非“替换”的策略,这是 C++ 以及一些语言所独有的,而其他一些语言则没有这样做,例如 Visual Basic 采取的就是“替换”的策略。一个合法的 C++ 程序是可以顺利通过 C++/CLI 编译,并且可以正常运行的。我们认为这对于我们的程序员是一项基本的需求。

谈到 C++/CLI 的第四个设计层面,这究竟是什么意思呢?它表示我们对 C++/CLI 语言规范和 ISO-C++ 所做的深入的集成。例如,除了我们扩展支持集合使其也适用于统一的 CLI 类型系统,表达式评估的标准转换集合与重载函数的辨析都和 ISO-C++ 的相同。当我们引入模板和多继承机制时,我们也应用了同样的扩展策略。这些都是在语言中稍显抽象的部分,在某种程度上我们已经使它们的行为变得更加直观,免除了程序员深入算法细节的需要。但我们仍会在系列专栏中花费笔墨关注一些主要的变化,例如对字面常量 (literal) 字符串的处理。

在 C++/CLI 未来的版本中,我们希望为本地类型和 CLI 类型提供更为无缝的集成。在目前的实现中,仍然存在许多不能跨越的壁垒。例如,我们现在还不能直接在一个 CLI 类中声明一个本地类的实例对象;相反,我们必须声明一个指向那个本地对象的指针,然后在 CLI 类的构造器/析构器中对处理对它的内存分配与释放。我们希望将来能够透明地处理它们。类似地,如果可以方便地编写下面的代码就更好了:

```
N n = gcnew N;  
R* pn = new R;
```

即将一个本地类透明地放在垃圾收

集控制的托管堆中,以及将一个 CLI 引用类型透明地放在本地堆中,并使它们正常运行。这些是我们对于 C++/CLI 未来的一些设想和愿景。随着这些设想的实现,我们也会在我们的专栏中讨论它们。

最后,再回答一个大家经常问到的问题,“我为什么要学习 C++/CLI”?首要的原因是 C++/CLI 将会为你进入 CLI 所表示的动态组件编程模型领域提供一张第一等的入口签证。如果你像我一样认为这将成为越来越重要的一种编程模型,并且如果你是一个历练较深的程序员,那么 C++/CLI 就是你想要的一个语言工具。如果你不喜欢某些地方,或者发现某些东西很难表达,那么请告诉我们。我们代表着一个动态编程社区, C++/CLI 也会持续不断地前进。

在 C++/CLI 之前,如果我们希望或者需要在 CLI 所表示的动态编程领域工作,那么我们只能放弃使用 C++,这意味着我们同时放弃了我们现存的代码库和编码经验。有了 C++/CLI 之后,我们就拥有了一条沿着 C++ 向上的移植路径。这是学习 C++/CLI 的第一个原因。

学习 C++/CLI 的第二个原因在于它允许我们访问整个 CLI 框架类库,包括用户界面、线程、网络、XML、ADO、NET、ASP.NET,以及 Web 服务这个宽广诱人的世界。另外,在即将推出的 WinFX 中,一个封装了整个操作系统的类库体系(包括应用程序及其执行空间)也会被收编在 CLI 门下。

Steve Lippman
Beijing 2004
Thank You!



李建忠

国内著名.NET专家, 著名译作者, 是《Microsoft .NET 框架程序设计》、《Microsoft .NET 框架程序设计——Visual Basic .NET 语言描述》等书的译者。

李建忠



撰

写这组专栏文章是一个偶然的机会, 虽然在我看过C++/CLI标准草案并试用过beta版的C++/CLI编译器之后, 有过写一组C++/CLI文章的打算。但是将这一想法这么快变成现实却要归功于Stan Lippman先生。我于今年9月份在上海的Tech Ed技术大会上有幸聆听Stan Lippman先生的C++/CLI讲座, 并在会后和Stan Lippman先生有过一段私下的交流。我

不仅被C++/CLI这门技术本身的魅力所深深吸引, 也为Stan Lippman先生对C++/CLI的杰出贡献和深刻理解而折服。在C++/CLI对我们的共同感召下, Stan Lippman先生和我非常自然地达成了撰写C++/CLI专栏文章的合作意向。按照我们的合作计划, 我们会首先一起合作撰写英文的文章, 然后由我来负责再将文章翻译成中文。基本上来讲, 专栏的每一篇文章都会在英语社区和华语社区同期发表。我们期望这种跨越太平洋的合作能够同时

为整个英语世界和华语世界的C++和CLI程序员社区带来些许价值, 同时也为C++/CLI的发展尽绵薄之力。

我们知道, C++长期在程序设计领域扮演着重要的角色, 不仅仅因为它在各种软件系统中广泛成功的应用, 同时因为它一路走来为整个编程语言的发展所注入持续不断的活力。但是自从.NET(微软的CLI实现平台)推出之后, C++所扮演的角色便倍显尴尬, 同时也为很

多C++程序员所困惑。一方面, 以C#和Java为代表的新一代托管编程语言以突飞猛进的速度挤压着C++的生存空间, 另一方面Managed C++这种仅仅着眼于用扩展语法来跟进CLI的发展走向让许多C++程序员感到失望和不安, 这也使得很多人怀疑是不是C++只能适合以静态对象模型为主要构造的传统软件系统, 而不适合以动态组件对象模型为主要构造的新一代软件环境——说实话, 我也曾经是这些怀疑者中间的一员。

幸运的是, 以Stan Lippman和Herb Sutter为首的微软Visual C++开发组及时地调整了C++在CLI上的发展策略, 提出了深层的范式绑定, 而非简单的语法扩展的发展方向, 为C++的发展注入了新的活力。我要特别指出的是这样的幸运者不仅仅是C++程序员, 同时也包括很多CLI程序员。为什么这么说呢? 这个问题可以换一种说法来回答: CLI为C++带来了什么? C++又为CLI带来了什么?

从C++的角度来看, CLI为C++打开了通往动态编程的一扇大门。这种动态编程特别适合于当前计算环境下的现代组件程序设计, 从它出发还可以导向或者加速许多有趣的发展方向, 比如产生式编程、声明性编程、面向方面编程(AOP), 等等。从本质上来看, CLI引入的这种动态编程范式赖于其中无处不在

的元数据。当然, CLI并不是这一概念的创始者, 比如Smalltalk、Java等语言也非常地倚重元数据, 但CLI却是将这一概念发挥得淋漓尽致的一个, 特别是其在设计之初就为多种语言集成CLI提供了先天的支持, 并且更为重要的是CLI已经成为ECMA和ISO国际标准, 而C++/CLI也有望在不远的将来纳入这些标准之中。这意味着C++/CLI将有可能像标准C++一样得到各种平台的支持, 当然前提是这些平台支持CLI。

从CLI的角度来看, C++不仅只是为CLI提供了更多一门语言的选择, 将广大C++程序员带到CLI上来。而且更为重要的是C++为CLI语言大家族带来的是一门高效、强大的系统级编程语言, 这个在C++/CLI诞生之前堪称空白。C++/CLI在所有CLI语言中占据着相当特殊的地位, 担当着很多其他语言所难以担当的重任。事实上, 从我个人的感觉来看C++为CLI语言的发展注入了很多富有活力的元素, 比如内部指针(`interior_ptr`), 比如确定性终止化操作(`deterministic finalization`), 比如对装箱值类型实例的强类型引用, 比如适用于托管类型的静态模板技术(STL.NET就是这一技术的产物。这个会让很多C#程序员垂涎三尺。不过别急, STL.NET通过ICollection、IEnumerator、IList等这些集合接口为其他CLI语言提供了访问的可能)。比如在托管对象中内嵌本地对象以及在本地对象中内嵌托管对象, 在托管堆上分配本地对象以及在本地堆上分配托管对象(这些将在C++/CLI后续的版本中得到支持), 等等。这些将会给大家带来一种非常别致的系统级视角, 这也是很多传统C++程序员引以为荣的地方——C++带给广大程序员的这一精神特质在C++/CLI中被完全继承下来了。从这个意义上来说不仅C++需要CLI, CLI也需要C++。

以上两个角度是大家理解C++/CLI两个不错的角度, 也是我们撰写专栏文

章时所着眼的两个基点。我们希望专栏的读者具有一些起码的C++和CLI基础, 但有些地方也并非必须——专栏文章会尽可能地做到循序渐进, 深入浅出。当然, 大家的功底越深越好。)


需要特别指出的是C++/CLI的目标并不是鼓励大家放弃传统C++而去拥抱CLI。相反, C++/CLI的理念是将以CLI为代表的动态组件编程范式 and 传统C++所拥有的各种其他静态编程范式有机地结合在一起, C++/CLI的发展方向也暗含着这一趋势。也就是说, 在C++/CLI中, 我们提倡混合编程范式, 即过程式编程、面向对象编程、泛型编程和动态编程, 这将是C++/CLI极具魅力的地方, 也是我们在专栏中着力探讨的一个方向。事实上, 这更符合许多软件系统的实际情况。这就像我们要到达一个目的地, 在这样的行程中我们通常并不是单单仰赖一种交通工具, 而往往是一会坐飞机, 一会又要乘汽车, 有时候甚至还可能要搭一段轮船。

C++/CLI固然为编程语言注入了新的活力, 在很多方面都有不小的进步, 但这并不意味着它适合每个人, C++/CLI有其清晰的定位。我想至少对于所有Windows平台上的C++程序员来说, C++/CLI立马就将成为毋庸置疑的选择, 尤其是在考虑微软下一版本的重量级操作系统Longhorn将以CLI作为基础支撑平台的事实下。随着CLI这一标准在其他操作系统平台上的推广, 相信C++/CLI会成为更多程序员的选择。我个人并不认为C++/CLI会对C#、Visual Basic.NET、Delphi.NET等语言造成多大的挤压, C++/CLI只是找回了自己应有的位置——想象一下当年多少C++程序员面对“丑陋”的Managed C++只能被迫改道C#的悲壮场面吧。不过, C++/CLI也并非C++程序员的专利, 任何喜欢“系统级体验”的程序员都是C++/CLI的目标程序员。当然由于C++/CLI

对传统C++是一种集成, 而非替换, 传统C++在C++/CLI中的烙印非常之重, 所以学习C++/CLI的同时, 也需要学习ISO-C++。

C++/CLI的确强大, 但是伴随着强大的同时是它的学习难度也很大。我喜欢用一组简单的数字来说明各门编程语言的学习难度。如果说Java的学习难度为10, 那么C#的学习难度将为15, 而C++的学习难度将为100。换言之, Java和C#的学习难度数量级相当, 而C++要高出一个数量级。那么现在如果再将C++/CLI放诸这个比较系统中, 我想其难度应该位于150。是的, C++/CLI不是C++和C#简单的相加, 而是复杂的绑定。即便只是针对CLI这一块, 它的难度都要比C#高一截, 更匡论C++/CLI中本地类型系统和托管类型系统各种形态的集成。不过, 技术领域通常有一个规律, 难度大往往意味着威力也大, 套用一句不恰当的话来说就是高风险意味着高回报。如果大家有志于此, 那么请从以下三件事情做起: 第一, 下载C++/CLI Standard (草案), 简单浏览一下; 第二, 下载beta版的C++/CLI编译器, 简单把玩一下; 第三, 阅读我们的专栏以及blog。

在这个专栏中, Stan Lippman先生将和我从最基础的地方开始入手, 层层解剖, 逐步深入, 对C++/CLI做全景式的探讨。同时这个长期的专栏也会随着C++/CLI的飞速发展融入一些新的东西。我们希望那些热爱C++和CLI的人从专栏的一开始就和我们一起来体验这一美妙的编程范式。

最后, 我要特别感谢Stan Lippman先生给予我的信任、指导和鼓励。希望广大C++和CLI程序员能够喜欢这个长期的专栏。如果您对我们的专栏有任何问题或者反馈, 请通过email或者blog和我们交流。 

聆听未来

——Stan Lippman 谈 C++/CLI

■ 文 / 孟岩

Stan Lippman 是最早的 C++ 先锋之一，许多年来他一直致力于 C++ 的改进工作。2001 年加入微软后，他开始着手 C++ 在 .NET 平台上的改进——C++/CLI。本文是《程序员》记者与他的访谈录。



《程序员》：两年前您加盟微软的时候，我还是个一线的 C++ 程序员，听到这个消息，我们都很高兴，觉得 C++ 又有希望了。

Stan：过奖了，呵呵。

《程序员》：那么您在微软的这两年感觉如何？快乐吗？

Stan：感觉不错，棒极了！这两年我过的很快活，微软管理层让我回归自己，所以我感觉很自在。

《程序员》：你们是什么时候开始设计新的 C++/CLI 的呢？目的是什么？我的

意思是说，你们觉得 Managed C++ Extension 失败了，是吗？

Stan：微软聘请我的时候，我对管理层的每一个人说，Managed C++ 把我惹火了，我明确告诉他们我不会是一个听话的士兵。当年我跟 Bjarne Stroustrup 一起工作的时候就是这样，对一些事情我有不同意见，我发表了 my 意见，而且也做了必要的努力。当然最后可能是别人赢了，我的意见可能被搁在一边，但我还是会不断提出我的意见。

说正经的，如果说 Managed C++ 有一点成功之处，那就是它还能用，而且是把现有代码移植到新平台上的唯一途径。这真的很了不起，我确实感到惊讶。但是从编程语言的体验来讲，它实在是不怎么样。我觉得一部分原因是，MC++ 的设计者（对于标准 C++）过分虔诚了。我比他们更敢于得罪 Bjarne，而后来公司的 Herb Sutter 比我还勇敢。他们应该更坚定一些，他们太缩手缩脚，不够大胆。我在贝尔实验室学到一件事情，那就是你应该对你认为正确的东西保持坚定的信心和热忱。

我想说，MC++ 确实能够工作，而且是把 C++ 放到 .NET 中并且包装起来的惟一办法。但是，它不是 C++，它的实现就语言而言是不自然的，这一点很多人

需要理解。

我之前说过，CLI 并不仅仅是微软的平台，它是目前唯一有前景的技术。Java 已经基本到头了，我是说它只是一种技术而已，它没有更远的目标了，它已经竭尽所能了。而 .NET 是当前唯一一种前景远大的技术。就算这一代的编程语言发展到尽头，也不意味着这个平台本身发展到了尽头。所以，对一种语言来说，如果打算超越原生代码，就必须移植到 .NET 上。这就是我对 C++ 的义务。（设计 C++/CLI）的工作开始以后，很多人参与其中，尤其值得称道的是 Herb Sutter，他非常非常重要，C++ 社群里的很多人都贡献了自己的力量。我很高兴。但是你知道，这并不能归功于我。

C++ 的路线是 Bjarne 确定的，我当然觉得这条路线是很成功的，我对于 ISO C++ 的发展方向感到很振奋。但是，我确实不认为 ISO C++ 有多么有趣，我说这话不是要冒犯谁，只是谈我的真实看法。ISO C++ 很了不起，但是它跟 C99 一样，只是对语言机制的一些修补和加强。它没有给我提供线程机制，也没有网络的支持。这事放在 21 世纪，简直是一种犯罪。而 .NET 提供了完备的支持库。在我看来，这是 C++ 早就应该做的事情。

你知道，整件事情很令人惊喜，我离开了 C++ 的发源地，贝尔实验室，也没打算再回去，然而我居然还能够参与 C++ 的改进工作，我很高兴。其实我并不认为现在的 C++/CLI 就很完美了，不过我们已经有了些非常棒的想法，你们会在下一个版本里看到。

这对 C++ 社群当然是一个挑战。我个人感到 ISO C++ 社群已经变成一个闭关自守的社群了，我们要把它打开。这当然不是一个能取悦所有人的事情。有些人对我们的做法很反感。这跟当年发生过的事情是一样的。当年 Bjarne 创造 C++ 的时候，很多用 C 的人也很反感，C++ 刚出来的时候，有些人觉得它是有史以来

最烂的语言。他们要抛弃那些导致性能下降的机制，把虚函数戏称为“破烂函数”。我是说，在实验室内部就存在着这样对立的两派。现在我看到这一幕在重演，有些人警告说，微软正在搞阴谋。我们尽了很大的努力，虚心听取社群的意见，邀请他们，倾听他们的声音，这一切正如 Bjarne 在 1980 年代中期所做的一样，当时他获得了管理层的许可，直接询问 IBM 和 Apple 对新语言的看法，其范围和深入程度超过了管理层事前的预期。我的观点是，我们是在发扬 Bjarne 的精神。所以我想，就算他不是绝对支持我们的做法，也不会是完全反对。

《程序员》：Bjarne 并不完全支持 C++/CLI，是吗？

Stan：他根本不需要如此。他本人就参与了这项工作。我和 Herb Sutter 在牛津的一次 C/C++ 使用者会议上跟他交谈过，我们向他展示我们的设计，Herb 花了很大的力气深入细节，Herb 真是一个强人，他非常善于深入细节，相比之下我并不是一个很喜欢深入细节的人。我们每星期都跟他开会。Bjarne 会建议我们采纳不同的思路，更重要的是，我们都能听到他的想法，他也知道我们的目的，所以，我觉得我们搞出来的这个语言还是不错的。

《程序员》：也就是说，您觉得 C++/CLI 已经成功了？我是说，这次革新已经成功了？

Stan：要我说，是一次进化，而不是革新，跟 Bjarne 当年往 C 中加入抽象数据类型和面向对象一样。在这些过程中，在不同的设计范式（paradigms）之间，总是会出现一些不和谐。跟 CLI 的整合是很困难的，因为 CLI 是最高统治者。举个例子，你试图在基类构造函数中调用派生类的虚函数，这应该是非法的，因为那个时候派生类对象还不存在，但是因为 CLI

的统治，我们对这种非法行为无能为力。再比如，如果你想绕过值类型的缺省构造函数，在 CLI 的统治下，也是不可能的。我是说，我们确实不得不在很多地方向 CLI 让步。那么是不是说我认为 CLI 问题多多呢，完全不是。像值类型构造函数这种问题很复杂，我无法在这里说清楚。总而言之，CLI 还是不错的。

在微软内部存在很多争论，我并没有赢得每一场争论，即使我认为我应该赢，也不一定能赢。没人能百战百胜。但是 Herb Sutter 有能力使大家达成共识，从而确保 C++/CLI 在一个动态编程环境中拥有美好的未来。我认为 C++/CLI 是 C++ 面对 Java 和 C# 挑战的一个合理的反应和完善。

《程序员》：C++/CLI 在未来的 .NET 平台上将扮演怎样的角色？

Stan：我们希望它能变成 .NET 平台上的系统编程语言。就是说，它可以用来开发那些驱动一切的程序。你不可能用 C# 来写 .NET 驱动程序，也不能用 C# 来写 C# 编译器，起码现在不行，但你可以用 C++/CLI 做到。你可以在 C++/CLI 上达到最高的效率和最大的能力，因为我们对于 CLI 模型的整合更为完整深入。C# 的编程模型跟 C++/CLI 相比是有所不及的，没有我们这么“先进”。所以我觉得，如果你以前是一个 C++ 程序员，现在打算学习 .NET，C++/CLI 是一个很好的选择。.NET 编程是让人兴奋的，而 C++/CLI 让你能够发掘 .NET 的全部能力，这真的是很让人兴奋的事。

坦率的说，我对传统的编程已经感到厌倦了。我以前在梦工场的时候，在那里构建了一个 Linux 开发平台，在其上进行软件开发。这本身确实没什么不对，但是我的感觉是一下子回到了 20 年前。我被绑在了 GCC 上，这让我感到不舒服。我找到了更好的编译器，更快，产生的代码质量更好，但是我不能用，我们要确保与

GCC 的二进制兼容。这些事情让我烦躁，我觉得我花了 20 年时间改进计算平台，却一下子一无所得。

相比之下，.NET 真的是一个有趣而又富于创新的技术。我经常跟人说，并不是因为我在微软，我才说 .NET 好，而是因为我感觉 .NET 好，才决定加入微软。

对于一个 C++ 程序员，你可能跟我一样觉得 .NET 很有趣，C# 当然是一个不错的选择，事实上我就写过一本 C# 的书。但是说实话，C# 有点“傻瓜”。对于很多人来说这是好事，我们并不希望所有的人都被拖到机器层次上。但是如果你想要那种最终的能力，而且你有把握运用好，那么 C++/CLI 就是你的选择，此外别无他选。我觉得我们的工作赋予你们这样的能力，我自己就渴望这样的能力。

《程序员》：最近几年 Java 和 C# 兴起，很多原来的 C++ 程序员都转向了 Java 和 C#。现在 C++/CLI 来了，你认为他们会回来吗？

Stan：我们希望如此，要不我们干嘛做这些工作呢？我说过，.NET 就是未来。那些拥抱 Java 和 Linux 的人们，我理解他们，他们觉得自己是在英勇地对抗一个庞大的帝国。但是这种想法不对。这事跟 20 年前很相似，1980 年代的时候，Bjarne 和我也把 IBM 视为一个邪恶的帝国，但是事实是，情况立刻发生了变化。如今我们放眼四顾，贝尔实验室的辉煌成为了历史，IBM 不再搞研究了，Apple 也不再搞研究了，只有少数几个人在努力进行技术创新，努力向人们提供更具创新性的计算环境，让人们更加自由自在。微软就是这少数人中的一个。我在微软工作，那气氛让我似乎回到了十多年前。

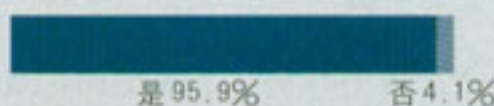
我告诉过你，我搞过 Linux，因为商业的原因。很多人都因为商业的原因搞 Linux，其目的是为了赚钱。赚钱没错，完全正确，但这完全是另外一回事。你应该明白。但是从技术角度来说，你不能说微

CSDN C++/CLI 调查报告

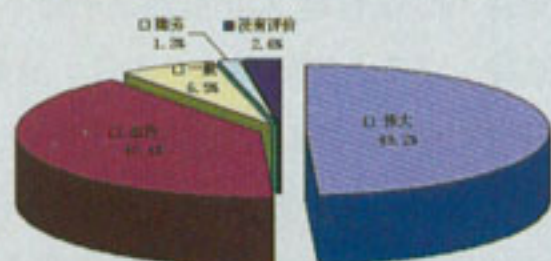
调查时间：2004 年 10 月 8 日——10 月 14 日

本刊编辑部

问题 1：你是否学习和使用过 C++？

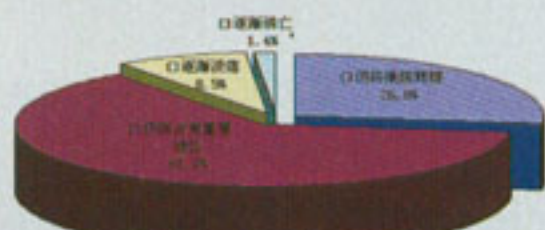


问题 2：你对传统 C++ 的评价如何？



参加本次投票的网友，96% 的人曾学习或使用过 C++；近九成的人认为 C++ “伟大”或“出色”。我们可以得出的结论是——传统 C++ 在人们心目中的地位极高。

问题 3：.NET 推出后，你对 C++ 前景的看法是：

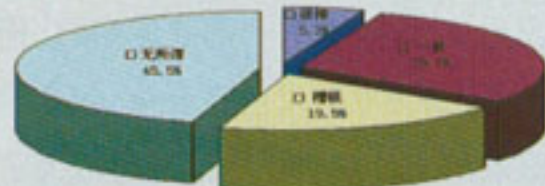


本题表明 C++ 传统支持者对 C++ 的地位相当有信心，这与一般观点给人的印象不符合。

问题 4：你是否知道微软的 MC++？

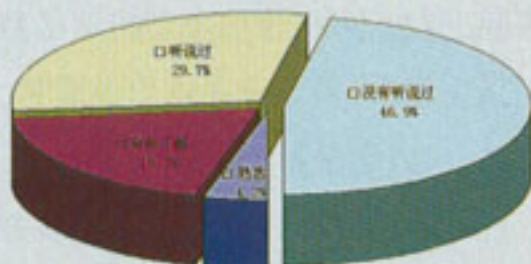


问题 5：你对 MC++ 评价如何？

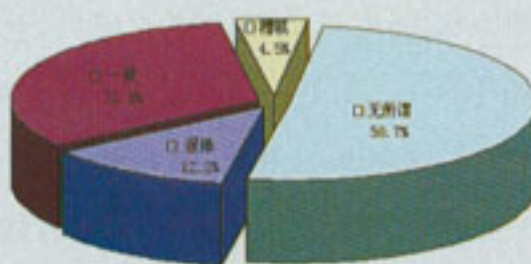


第 4、5 题反应了 MC++ 的实际情况，评价中等以下占压倒多数。可以给 MC++ 盖棺定论——败笔！

问题 6：你是否知道 C++/CLI？

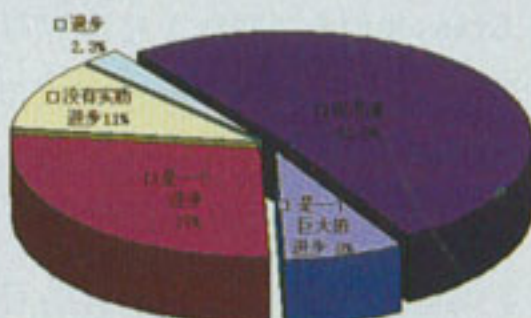


问题 7：你对于 C++/CLI 印象如何？

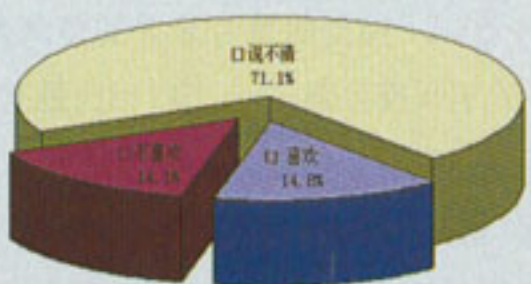


第 6、7 题显示 C++/CLI 还未正式推出就已有超过半数人有所了解，对此微软应当高兴。

问题 8：你认为 C++/CLI 相比 MC++ 是个进步吗？

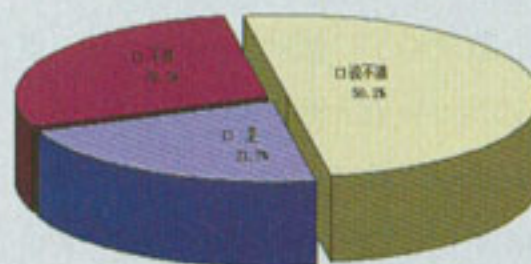


问题 9：你喜欢 C++/CLI 的语法吗？

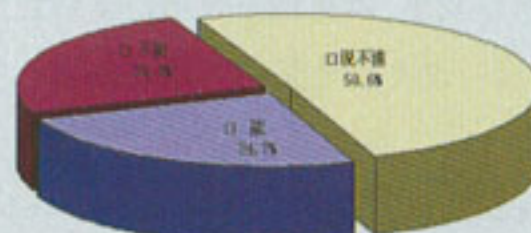


注意，结合第 6 题的结果，了解 C++/CLI 的人，喜欢其语法并认为有进步的人占很大比例。

问题 10：你相信 C++/CLI 是 .NET 平台上最强大的语言吗？

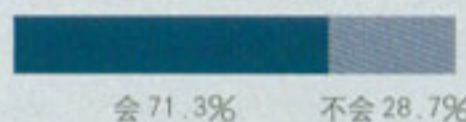


问题 11：你认为 C++/CLI 能够改变 C++ 在 .NET 平台上的弱势地位吗？



同 MC++ 结果相似，说明对于其前景和定位还存在很多的争议。

问题 12：你会考虑学习 C++/CLI 吗？



问题 13：你会考虑在项目中使 C++/CLI 吗？



问题 14：你认为 C++/CLI 的未来是？



第 13 到第 14 题反映了相当多的人（7 成）考虑学习 C++/CLI，愿意使用者略少（6 成）。这两个结果一方面表明大多数人对 C++/CLI 感兴趣；另一方面，亦有一些人怀疑其是否适用于实际的项目开发。对 C++/CLI 前途的看法，多数的人是持审慎的乐观。

软是恶魔，就是这样。

我在微软过的很开心，你知道我们这一代是跟着 UNIX 成长起来的，我热爱软件，但我还是要对你们说，希望你们能成为 .NET 和 C++/CLI 的一代人。

《程序员》：非常感谢！这是您第一次来中国吧，感觉怎么样？

Stan：棒极了，真的，我太喜欢这里了。下次有机会，我还会过来。我去过很多国家，见过很多程序员，中国的年轻人给我留下了很深的印象。你们热情，而且也不缺乏理解力。你知道，那种热情，在美国的青年程序员身上已经找不到了。我不知道我的判断是否正确，我只是觉得这里充满了热情和痴迷的气氛，能够

跟你们交流真的很愉快。你们让我觉得自己处于宇宙的中心，哈哈。

《程序员》：请回去告诉 Herb Sutter，下次我们希望看到他跟您一起访问中国。

Stan：好的，我一定转告他。

《程序员》：感谢您接受采访。

C++/CLI 会冲击 C# 吗

■ 文 / 刘如鸿

本来我是打算写一篇文章来比较 C++/CLI 和 C# 的语言特性,为此我认真阅读了一些 C++/CLI 相关的文章和部分 C++/CLI 语言规范文本,结果让我感到沮丧。我原本对 C++/CLI 语言的热情,被其远远超出我原本想象的复杂性给浇灭了一大半。

编辑部里有一个很好的风气,就是对于一些问题在理解上有不同看法的时候,我们都会拿出来讨论。讨论的焦点集中在托管代码和非托管代码的交互上,我想这是 C++/CLI 的独特优势,也是传统 C++ 程序员关注的重点。而恰恰是这次讨论让我完全改变了看法。对于 `pin_ptr` 和 `interior_ptr`,我们还无法透彻了解其背后的奥秘,比如 `pin_ptr` 在托管堆中如何指向固定对象,如何防止移动对象,在此技术上托管堆的垃圾回收如何进行, `interior_ptr` 和 `pin_ptr` 的关系如何等等。网络上相关的文章都是相对泛泛的介绍 C++/CLI 带来的激动人心的特性,对于这些方面的剖析,似乎没有见到。虽然不知者不为罪,但是不知道却胡说就是一种罪过了。显然,我们所了解的内容还不足以让我有资格去评判两个语言的优劣。在放弃当初的想法之后,我下面的文字更多的是从个人的观点去阐述对于这两个语言的看法。

C++/CLI 会冲击 C# 在 .NET 中的地位吗? 我的观点: 不会。正如 Stan Lippman 所言的,每一种语言都有自己的定位。C# 作为一个没有任何历史包袱的

语言,其设计的重点是开发效率和语言威力的平衡。从目前来说,C# 是 .NET 语言家族中最均衡、适用面最广的语言,并且在 VS.NET 中提供了最好的 IDE 支持。在 C# 2.0 版本中加入的泛型、匿名函数和匿名内部类,也是从开发的角度去考虑的。传统 C++ 适合做系统层面上的开发,但是在开发效率方面劣势明显。Managed C++ 在这个方面没有做太多的改进,相反加入的一些语法规则让许多开发人员接受不了。虽然 C++/CLI 在开发者体验方面已经做了大幅度的改进,从而使开发人员能够更加快速的进行应用开发,也提供了更加优雅的代码,但是 C++ 天然的复杂性让其在业务应用范畴捉襟见肘,依旧无法替代 C# 在整个 .NET 开发语言中的地位。这是两个针对不同定位设计的语言,因此也是在不同的领域内发挥自己的特长。

C++/CLI 对于 .NET 意味着什么? 我的观点: 大大拓展 .NET 平台的威力。当年的 Managed C++ 只是完成了一个使命: 保证现有的代码迁移到 .NET 平台上来。在其他方面因为其设计上过于保守,不但要完成自己的使命,也希望保持和 ISO C++ 的高度兼容。这两个方面的考虑让设计者缩手缩脚。从某种意义上来说,正是因为对于 Bjarne Stroustrup 或者说 ISO C++ 的虔诚造成了 Managed C++ 在语言革新方面的扭曲。而到了 C++/CLI 时代,这一切将不复存在。Stan Lippman 和 Herb Sutter 主导设计的 C++/CLI 除了保持和

C++ 的兼容之外,自然地引入了 CLI,使开发人员能够以一种更加自然的方式去编写代码,从而找回了在 .NET 平台中应有的位置。按照 C++/CLI 设计组的说法,C++/CLI 将成为 .NET 之上最强大、最底层的语言——其能力甚至超越 MSIL 本身。优雅的语法,强大的功能,极限的效率,自然能够吸引更多传统 C/C++ 程序员投入 C++/CLI 的怀抱。同时,C++/CLI 对于 .NET CLR 提供了更加灵活的控制方式,也为开发人员掌握和扩展 .NET 的能力提供了前所未有的强大武器。这还是一个开始,据 Stan Lippman 说,一些更具震撼性的思想将融入 C++/CLI 的下一个版本中,我只举一个例子就可以启发大家的遐想: 下一版的 C++/CLI,可以在 CLR 托管堆之外,于本地堆之中建立新的托管堆,将托管对象分配在本地堆中。

C++/CLI 对于 C# 开发人员的诱惑在于何处? 我的观点: 配合 C# 拓展其功能。一些从 C++ 转到 C# 的开发人员应该会有些抱怨,在日常的业务应用开发中,C# 是最合适不过的语言,但是在于某些方面,比如需要密集运算、特殊算法逻辑方面并没有展现出优势,习惯于使用指针去解决的 C++ 开发人员这个时候就认为 C# 是一个比较笨拙的设计,虽然 C# 也通过 `unsafe` 关键字提供了部分指针的支持,但是在一些对象操作方面,远远不如 C++ 那样随心所欲。C++/CLI 为 C# 开发人员提供了一个新的选择: 利用 C++/CLI 去实现高性能部分,使用 C# 实现整体业务。这样的配合与使用 VC 写组件 VB 写应用逻辑的配合比较接近,有一点不同的就是都基于 .NET Framework,较之当年的 VB 与 VC,明显容易许多。

综述如上的观点,C++/CLI 的推出只是填补了 .NET 开发领域内的一块空白,对于 C# 不会造成本质的冲击,同时其大大拓展了平台的威力,对于 C# 开发人员,是朋友,不是对手。☞

C++/CLI: 鼎新革故

■ 文 / 董颖涛

If you believe the additional language support simply represents yet another domain abstraction, you will choose to remain within the existing language. If you see the additional language support as representing a shift in programming paradigm, you will extend the language.

— Stanley B. Lippman

.NET 的出现无疑给C++的发展带来的新的契机和挑战。为了让C++的程序员能够充分利用CLR所带来的强大功能, Visual C++ .NET 推出了 Managed C++ Extension。它充分地保留了原有C++的语法, 不仅让C++程序员能成功地编写.NET程序, 同时能让原有的C++代码很好地移植到.NET平台上来。表面上这是个巨大的成功, 但事实上广大C++程序员并不乐于使用这个新生事物。

Visual C++开发团队经过长期的努力和实践, 并结合广大.NET和C++用户的意见, 创造性地提出在Visual C++ 2005中重新设计其对CLR的支持。这个新的设计就是C++/CLI。它将在C++中提供更为自然的语法来支持CLR。它是ISO C++语言针对动态程序设计范式的扩展。

追根溯源

托管C++自从出现以来, 可谓是“骂”声不断。其主要不足之处有:

- ◆ C++非常复杂, 而托管C++更加复杂, 其语法实在丑陋之极, 那些加了下划线的关键字让人看了就不爽。
- ◆ 二流的.NET支持, 相对于C#、VB.NET, 托管C++对.NET的支持是不完全而且蹩脚的。
- ◆ 容易混淆, 尤其是指针的用法, 不管是在概念上还是语法上。

这么多的不足, 原因何在? 大家知道, .NET上的一等公民是C#和VB.NET。同时, .NET也没有忘记其他众多的编程语言, CLI规范正是为此而生。而C++由于其相对底层的地位, 不太符合CLI规范的要求。那么要让C++程序员享受到.NET的巨大好处, 面前有两条路: 一是使用原有语言的元素实现对.NET的支持, 另一条是扩展原有语言。后者将违反经过这么多年好不容易建立起来的C++标准。

在这样的岔路口, 托管C++选择了前者, 它致力于保留传统C++语言的概念、语法, 通过加入符合C++标准的下划线式关键字来实现.NET带来的丰富功能。也正是这一点导致了以上的不足。而C++/CLI则选择了第二条路, 下面来看看C++/CLI做了怎样的改进。

鼎新革故

新的语法元素和表述方法使C++/CLI能更自然的表达语义。使C++/CLI程序员更像C++程序员。

上下文相关关键字替代“__”

C++/CLI带来的第一个变化——去掉“__”式关键字。在托管C++中, 引入“__”式关键字主要出于尽量少的影响原有C++语言的考虑, 但它显得复杂而且容易带来语义上的混淆。而C++/CLI则简单的去掉了这些蹩脚的关键字, 转而引入了“上下文相关关键字”。

“上下文相关关键字”和它所在的上下文密切相关。例如: 在一个普通的程序中, “sealed”被认为是一个普

通的标识符；但是当它出现在托管引用类型的声明中时，它就会被编译器视作这个上下文中的关键字。这样做对与现有代码的兼容性是非常好的，同时也给使用新功能的程序员带来了直观的体验。下表列出了C++/CLI的上下文相关关键字是如何替代托管C++中的“__”式关键字的。

CLI 类型	托管 C++	C++/CLI
引用类	__gc class R	ref class R
数值类	__value class V	value class V
抽象类	__gc __abstract class R	ref class R abstract
封闭类	__gc __sealed class R	ref class R sealed
接口类	__gc __interface class I	interface class I
CLI 枚举类型	__value enum E	enum class E
代理类型	__delegate void Callback()	delegate void Callback()

追踪句柄替代托管指针

在托管C++中，声明一个引用类的对象就是用的指针的语法。而在C++/CLI中，声明CLI引用类对象使用“追踪句柄”来实现，语言中用符号“^”来表示。这里的“追踪”隐含了CLI托管堆对CLI引用类对象透明的内部管理。同时C++/CLI还引入了“追踪引用”的概念，语言中用符号“%”来表示。这里“%”和“^”的关系与ISO-C++中“*”和“&”的关系非常类似。下面的例子更明显的说明了托管C++和C++/CLI的区别：

```
// 托管C++
String* ps = S "a string literal";
String& rs = *ps;
// C++/CLI
String^ ps = "a string literal";
String% rs = *ps;
```

由于“^”的引入，使CLI句柄和传统的C++指针有了区别，这给语言带来了很大的好处。例如：

```
double pi = 3.1415;
// 托管C++
__box double* br = __box( pi );
// C++/CLI
double^ br = pi;
```

可以看出，在托管C++中繁琐的装箱操作，在C++/CLI中显得既简洁又自然。

既然托管对象声明从“*”走向了“^”，那么托管对象又是如何分配的呢？为了更加明确的区分在CRT堆和在CLI堆上的分配，C++/CLI引入了关键字“gcnew”。看下面的例子：

```
// 托管C++
StreamReader *ifile = new StreamReader(filename);
NativeClass *pnc = new NativeClass(args);
// C++/CLI
StreamReader ^ifile = gcnew StreamReader(filename);
NativeClass *pnc = new NativeClass(args);
```

表面上C++/CLI比托管C++多了一个gcnew，但实际上，C++/CLI的程序变得更加清晰。

不论是指针还是句柄，都必须有一个值来表示空值。C++/CLI又引入了关键字“nullptr”作为指针和句柄的空值。

托管数组

在托管C++中，托管数组的声明是通过关键字“__gc”来实现的。显得和C++的语法格格不入：

```
// 托管C++
void PrintValues(Object* myArr __gc[]);
void PrintValues(int myArr __gc[...]);
```

C++/CLI放弃了这种蹩脚的语法，转而引入一种类似于模板的“array”：

```
// C++/CLI
void PrintValues(array<Object>* myArr);
void PrintValues(array<int, 3>* myArr);
```

这个array看起来很像STL中的vector。它的第一个参数表示了数组的类型，第二个参数是数组的维数（其默认值为1）。托管数组对象本身是用追踪句柄表示的，所以声明时需要加上“^”，当其元素类型是引用类型时也同样需要加上“^”。

属性

对于属性的表达，托管C++是在相应的set_P和get_P方法前加上关键字“__property”来实现的，程序员必须在命名上将属性和其对应的set、get方法联系起来，这样的语法当然让人困惑。在C++/CLI中，首先去掉了“property”前面的“__”；其次，改变了传统C++类声明中的写法，将get和set方法内置于property的声明之中：

```
// C++/CLI
public ref class Vector sealed {
    float _x;
public:
```



```
property double x {  
    double get() { return _x ; }  
    void set(double newx) { _x = newx ; }  
} // 注意：这里没有分号  
};
```

显式的接口方法重载

在很多情况下,程序员希望提供对某个接口方法的两个不同实现。一个用于实现该对象通过接口句柄访问时的行为,另一个用于实现通过该对象的确切类型访问时的行为。在托管C++中,这可以通过下面的方法实现:

```
// 托管C++  
public __gc class R : public ICloneable {  
    virtual Object* ICloneable::Clone() ; //through ICloneable  
    virtual R* Clone() ; //through R  
};
```

可以看到,在上面的代码中,类R显式的指出实现的是哪个Clone方法。这样的写法很有效,但是两个Clone的存在感觉不太符合C++的习惯。在C++/CLI中,借鉴原来纯虚函数的写法做了改进:

```
// C++/CLI  
public ref class R : public ICloneable {  
    virtual Object* InterfaceClone() = ICloneable::Clone ;  
    virtual R* Clone() new ;  
};
```

改进的语法要求两个不同的方法拥有在类中唯一的名字,然后显式指明InterfaceClone是通过ICloneable调用时采用的方法,而下面的new给出了通过R访问的Clone方法。这样的改进让C++/CLI比托管C++更具有C++的血统。

旧语新意

为了适应新的语言范式,C++/CLI不再像托管C++那样,一味地遵循C++或者是CLI的规范,而是有效地将他们融合起来,大胆地改变了原有的语义。而一切又是那么和谐。

析构函数

在托管C++中,析构函数被映射为Finalize方法,其被调用的时机由GC来决定,和对象的生命周期无关。这种析构方式被称之为“非确定性析构”。

非确定性析构对于内存的管理非常有效,任何类都不用在析构函数中去释放占用的内存,因为垃圾收集器

会非常胜任这个工作。然而,对于非内存资源的管理,譬如文件句柄等,垃圾收集器就无能为力了。当对象的生命周期结束时,程序员如果没有手动释放这些资源,仍然会造成资源的泄漏。


CLI提供的解决方案是让拥有非内存资源的类实现IDisposable接口,在Dispose方法中释放资源。问题在于这个方法是由程序员来手动调用的,托管C++则没有任何自动的语法结构。它将托管类的析构函数在内部直接转换成为类型的Finalize函数,遵循CLI的语义。这和传统C++的析构函数的语义有很大差别,对C++程序员而言,非确定性析构可以说是资源管理上的一个退步。

C++/CLI结合传统C++析构函数的语义和CLI对资源处理的方法,采用了将托管类的析构函数映射到Dispose上的方法,并且让这些类自动地从IDisposable接口继承下来。这使得C++/CLI在和其它CLI语言交互时能正确的释放资源。并且,在对象生命周期结束时,对象的Dispose方法被自动调用。这样,C++/CLI的析构函数语义和传统C++的析构函数语义就一致了,这就是“确定性析构”。

在C++/CLI中,当程序员用gcnew在CLI堆中分配一个对象后,和传统C++一样,通过delete就能结束对象的周期。但是,如果程序中没有delete去显式地结束对象的生命周期,那么对象的析构函数也就是Dispose方法就不会被调用,内存自然会被回收,但这会造成非内存资源的泄漏吗?答案是不会。C++/CLI在引入了“确定性析构函数”的同时,并没有抛弃Finalize方法,而是引入了新的语法表示“!R()”来表示类R的非确定性析构函数。下面给出了例子:

```
// C++/CLI  
ref class R {  
public:  
    R(){} // 确定性析构,被映射成 Dispose  
protected:  
    !R(){} // 不确定性析构,被映射成 Finalize  
};
```

总结

作为新生事物,C++/CLI的确是C++语言的一次革命,可以说正在向正确的方向前进。随着Visual C++ 2005的推出,相信C++/CLI会不断普及,相信读者们也会切身感受到这次革命给C++语言和.NET程序设计带来的激动人心的变化。 

白从Java和.NET诞生的一开始,语言以及平台之间的竞争似乎才真正开始无休无止起来。凭借在Web应用以及企业级应用开发上的先天优势,Java和.NET在这些领域出尽了风头。尽管在优点的背后,弱点也是很明显的,然而光芒毕竟还是掩盖了尘埃,在各大厂商的大力宣传下,开发者们似乎也获得了某种勇气,一往无前的投入到各种新名词新概念的学习中去了……

然而,在喧嚣的背后,有一门语言仍然在冷静而理智地发展着——它就是C++。

标准 C++

从某种意义上说,C++ 仍然是一门很年轻的语言,从98年C++的第一次标准化到现在,虽然语言的核心并没有变化,但是库的发展却未曾有一日停止过:STL, Boost, ACE, LOKI, MTL, Blitz++……这些熟悉的名字正在渐渐深入人心,这些库所带来的思想和效益正逐渐深入工业界,这些库保持着C++长盛不衰的活力。

而现在,C++社区又在酝酿一次新的变革,这就是所谓的C++ 0x标准,也就是C++的下一代标准。当然,我们最为关心的还是,这次变革会给我们带来多少惊喜,简洁的答案是“会有极大的惊喜”。答案就在下文。

Bjarne Stroustrup 的思想——灯塔

这位C++创始者仍然活跃在C++标准化委员会中,毫无疑问,他仍然具有举足轻重的发言权。对于C++ 0x的方向,Bjarne Stroustrup的总体思想是:对语言核心的

改动应该是谨小慎微的,而对库的发展则应该是大胆而激进的。看起来非常简单平淡的两句话,却道出了真谛——保持语言核心的简洁紧凑和一致性,仅作必要的小范围改动,可以避免增加语言的复杂度,并将对现存代码的兼容性提到最高。从另一个方面,库的发展则大可不用顾忌这些,C++社区期待一些优秀的,可以提高生产率的库已经很久了。

语言核心的发展——精雕细琢

总的来说,C++ 0x对语言核心的调整是极其理智和谨慎的——避免大的语言扩充,进一步改善对泛型编程的支持,改善对嵌入式编程的支持。然后就是一些小范围的技术勘误和调整,改善语言的一致性。

总的来说,所有的语言核心的演化,都必须遵从零开销(zero overhead)原则,即用户不必为他们用不着的东西付出代价(特别是效率的代价),这正是C++在系统级开发领域能够保持长久活力的关键。无论如何,系统级开发是C++的立场,千变万变,立场不变。

库的发展——风起云涌

在C++库的发展的历史中,最为瞩目的当属STL,由于GP思想在其中的完美运用,STL兼得了“鱼和熊掌”——效率和优雅。看来大家都看到了GP的强大表达力,特别是在库的构建方面,所以像Boost, ACE, LOKI, Blitz++, MTL……等库都不约而同的使用了GP,并取得了巨大的成功。可以预见GP仍然会在库的构建方面发挥巨大的能力。

发展最为迅疾的库是Boost,因为Boost库是“准”C++标准库,所以其中的很多设施被提议加入下一代C++标准库,如正则表达式库,智能指针库,线程库,泛型函数指针库等,这些都为通用编程提供了极大的支持,另一个重要方面是,这些库的标准化意味着使用它们的代码将是完全可移植的。

之所以C++这两年来“看起来”风头减弱,主要是大家的注意力都被吸引到了J2EE和.NET所擅长的Web开发以及企业应用开发领域里去的缘故,而C++在这方面又如何

山雨欲来风满楼

——标准 C++ 及 C++/CLI 发展综述

■ 文 / 刘未鹏

When programmers are faced with a choice between productivity and control, technologies that make them more productive tend to win out over time.

——Don Box

呢? CORBA 太庞大, ACE 还不够“傻瓜”, ICE 据说是以轻量级为目的, 但愿如此。但最重要的还是, C++ 本身对分布式的支持还不够。

而在 C++ 中, 这样的日子可能也不会太遥远了, Bjarne Stroustrup 正致力于为 C++ 加入分布式计算的能力, 一旦这成为现实, 程序员将可以通过极为简单的几行代码就可以和远在 Web 另一端的对象交流, 远程调用就如同本地调用一样简单直观……这些特性可能会在 C++ 0x 中和大家见面——当然, 以库的形式。我们有理由相信, C++ 在分布式计算以及 Web 开发领域也将有一个美好的未来。

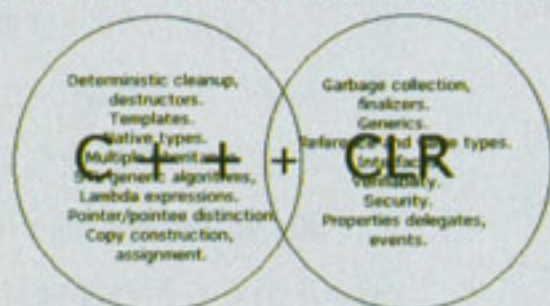
同样, 为程序员熟知的 GUI 编程也曾是 C++ 的“软肋”之一。但是这也即将成为过去。一个泛型的 GUI 库正在悄然兴起。以前的 C++ GUI 库如 wxWindows 以及 MFC 由于只使用了 C++ 的普通特性, 所以表达力有限。而这个兴起的库名为 Win32 GUI Generics, GP 与多继承的完美结合, 结果是简洁而优雅的 GUI 框架。

可以看出来, 虽然 C++ 的主要阵地是系统级开发, 但是无疑在其它更高层的领域也有不可估量的潜力, 我们拭目以待。

C++/CLI——微软的又一张王牌

微软希望 Windows 下的 C++ 开发者既能够做高效的底层系统级开发, 又能够方便的做高层的应用开发。怎么做? .NET? 当然是 .NET! .NET 是微软的王牌, 然而, 微软最钟爱的语言——C++——和 .NET 相处得又如何呢? 如果把时间往前推半年, 答案只能是——差强人意, Managed C++ Extension 只不过是一次失败的尝试。

为了解决这些问题, 并且稳固 C++ 在 Windows (.NET) 平台上的地位, 微软将 Stan Lippman 和 Herb Sutter 两位大牛人请了过来, 重新设计 C++ 在 .NET 下的表现形式, 在完全支持标准 C++ 的前提下, 引入一些新的语法和语义, 从而对 .NET 环境提供第一流 (first class) 的支持, 这就是 C++/CLI。其意义是非常巨大的, C++ 程序员从此可以以一种非常“C++”的方式去利用 .NET 中丰富的类, 同时仍然可以利用标准 C++ 的强大表达力, 可谓左右逢源。C++/CLI 的能力可以用下图来描述:



图一: C++/CLI 的能力

语法

C++/CLI 除了和标准 C++ 相同的部分之外, 还增加了一些新的语法元素, 以便支持托管环境下的编程, 其中最关键的就是加入了托管环境下的“指针”(称为 Handle) 和引用以及用于在托管堆上创建对象的 gcnew, 它们的声明形式分别为:

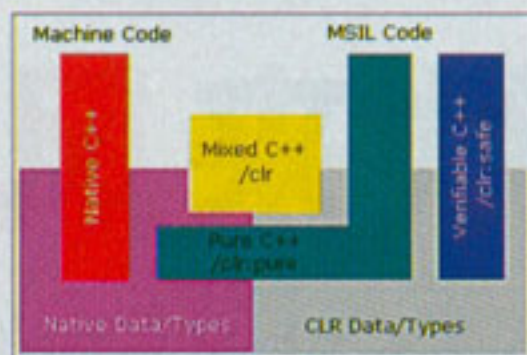
```
// 托管的指针 “*”, 称为 Handle
int* handle = gcnew int(0);
int% managed_ref = *handle; // 托管的引用 “%”
```

值得注意的是, 对 handle 解引用也使用 “*”, 这是为了在模板函数中可以以一致的方式来解引用。

这两个语法形式的加入, 确保了在 C++/CLI 中你可以利用 .NET 中所有的托管类, 并且把托管的世界和 native 世界从语法上区别开来。

语义

加入了托管语义的 C++/CLI 呈现出一种混合 (Mixed) 的语义 (但并不混淆), 你既可以高效地进行 Native 编程, 也可以在必要的时候为了方便而使用 .NET 中现成的类, 两者互补且可以相互沟通, 如果想把托管的对象传给 Native 接口, 只要先把该对象“定”在托管堆上 (通过 pin_ptr<>, 不然 GC 在压缩堆的时候会移动对象, 从而导致指向它的 native 指针统统失效), 然后传递指向该对象的指针给 Native 接口就行了, 反之, 在托管环境里访问非托管对象则一般不用任何辅助动作。这种混合式的编程环境提供了极大的自由度, 也完全兼容了现存的 C++ 代码。



图二: C++/CLI 编程环境示意图

C++&C++/CLI 的未来——坐看云起时

Bjarne Stroustrup 在一次接受采访的过程中曾说: “C++ 拥有极其美好的未来”。他的自信当然不是盲目的。就目前标准 C++ 的发展趋势来看, C++ 正在越强大, 以前主要在系统级开发领域挥洒的 C++ 正在逐渐进入越来越多的领域。☞

非典型 C++/CLI 教程

■ 文 / 刘未鹏

就像我们在作出其它任何选择的时候一样，在选择之前最重要的是先要清楚为什么作出这样或那样的选择——C++/CLI到底提供了哪些优势？为什么我们（标准C++程序员）要选择C++/CLI而不是C#？我们能够得到什么？CLI平台会不会束缚C++的能力？

好消息是：情况比乐观的人所想象的或许还要更好一些。

世界改变了吗？

对于谙于标准C++的程序员来说，最为关心的还是：在C++/CLI中，世界还是他们熟悉的那个世界吗？在标准C++的世界里，他们手里的各种魔棒，还能挥舞出五彩缤纷的火焰吗？是不是标准C++到了.NET环境下就像被拔掉了牙的老虎一样——Managed C++ Extension的阴影是不是还笼罩在他们的心头？

答案是：以前你能所做的，现在仍然能做，需要的只是学习。我们失去的只是限制，得到的是更强大的能力。

C++/CLI——优秀的混血儿

C++/CLI最大的成功在于引入了混合式编程的环境。这是一种非常自由的环境，其中Native和Managed代码可以共存，可以相互沟通，从而完全接纳了标准C++的世界，同时也为另一个世界敞开了大门……

下面就是C++/CLI扩展的几大关键特性：

Handle 和 gcnew ——通往 Managed 世界的钥匙

还记得在Managed C++ Extension世界里是如何访问托管类的吗？丑陋的__gc关键字无处不在——事实上，不仅是“丑陋”而已（MC++为什么会消亡？）。而在C++/CLI里则引入了一个新的语法元素，名为Handle，写作“^”——你可以把它看成Managed世界里的Pointer（不过不能进行指针算术）。

Handle用于持有Managed Heap上的对象，那么如何在Managed Heap上创建对象呢？原来的new显然不

能用，那样会混淆其语义，所以C++/CLI引入了一个对应的gcnew关键字。这两个新的语法元素是操纵Managed世界的关键。现在，使用Handle和gcnew，你就可以和任何托管类进行沟通。另外，既然有了Handle这个Managed指针，当然，基于另外一些重要原因，Managed世界里也要有一个和Native引用类似的语法元素——这就是Managed引用“%”——“^”对应“*”，“%”对应“&”，这样一来，从语法的层面上，指针、引用、以及在堆上创建对象的语法就在两个世界里面对称一致了——哦，等等，还有解引用：对Native Pointer解引用是以“*”，出于模板对形式统一性的要求，对Handle解引用也是用“*”。例如：

```
SomeManagedClass* handle = gcnew SomeManagedClass( ... );
handle->someMethod();
SomeManagedClass% ref = *handle;
```

那么，既然有gcnew，有没有gcdelete呢？答案是没有——虽然它们看起来很对称。理由是对于托管类，根本就不用回收内存。但更为重要的还是，delete的语义不仅仅是回收内存，从广义上说，delete是回收资源的意思，从这个意义上，delete托管类还是Native类的对象都是一个意思。所以，即使你需要delete你的托管类对象，以强制其释放资源，你也应该用delete，这时候托管类的析构函数会被调用——是的，托管类也有析构函数，它的语义和Dispose()一样，但是在C++/CLI里面，你不应该为你的托管类定义Dispose()函数，而总是应该用析构函数来代替它（编译器会根据析构函数自动生成Dispose()函数），因为析构函数有一个最大的优点：

Deterministic Destruction & RAII

——资源管理的利器

正如每一个熟悉标准C++的程序员所清楚的：由C++构造及析构函数的语义保证所支持的RAII（“资源获取即初始化”）技术是资源自动和安全管理利器，这里的资源可以包括内存、文件句柄、mutex、lock等。通过正确的

使用RAII,管理资源的代码可以变得惊人的优雅和简单。相信有经验的C++程序员都熟悉应该类似下面的语句:

```
void f()
{
    ofstream outf("out.txt");
    outf<<"...";
    ...
} //outf 在这里析构!
```

这里,程序员根本不用手动清理outf,在函数结束(outf超出作用域)时,outf会自动析构,并释放其所有资源。即使后续的代码抛出了异常,C++语言也能保证析构函数会被调用。事实上,在异常抛出后,栈开解(stack unwind)的过程中,所有已经正确构造起来的局部对象都会被析构。这就为异常环境中资源的管理提供了一种强大而优雅的方式。

那么,在C++/CLI中,原来的那种优雅的,靠析构函数来确保资源正确释放的手段还存在吗?答案正如你所期望和熟悉的,RAII仍然可以使用,仍然和标准C++中的能力一样强大:

```
ref struct D
{
    D(){System::Console::WriteLine("in D::D()\n");}
    ~D(){System::Console::WriteLine("in D::~D()\n");}
    !D(){System::Console::WriteLine("Finalized!\n");}
};

int main()
{
    D d; // in D::D()
    ...
} //d在这里析构! in D::~D()
```

pin_ptr —— 定身法

千万不要小看了pin_ptr的能力,它是Native世界和Managed世界之间的桥梁。在通常情况下,任何时候,GC都会启动,一旦进行GC,托管堆就会被压缩,对象的位置就会被移动,这时候所有指向对象的Handle都会被更新。但是,往往有时候程序员会希望能够把托管堆上的数据(的地址)传给Native接口,比如,为了复用一个Native的高效算法,或者为了高效的做某些其它事情,这种情况下普通的Native指针显然不能胜任,因为如果允许Native指针指向托管堆上的对象,那么一旦发生了GC,这些得不到更新的Native指针将指向错误的位置,造成严重的后果。办法是先把对象“定”在Managed堆上,然后再把地址传给Native接口,这个“定身法”就是pin_ptr——它

告诉GC:在压缩堆的时候请不要移动该对象!

```
array<char> arr = gcnew array<char>(3); // 托管类
arr[0] = 'C';
arr[1] = '+';
arr[2] = 'C';
pin_ptr<char> p = &arr[0]; // 整个arr都被定在堆上
char* pbegin=p;
std::sort(pbegin,pbegin+3); // 复用Native的算法!
std::cout<<pbegin[0]<<pbegin[1]<<pbegin[2]; // 输出 "++C"
```

在上面的代码中,我们复用了STL里的sort算法。事实上,既然有了pin_ptr,我们可以复用绝大部分的Native算法。这就为我们构建一个紧凑高效的程序内核提供了途径。

interior_ptr —— 托管环境下的Native指针

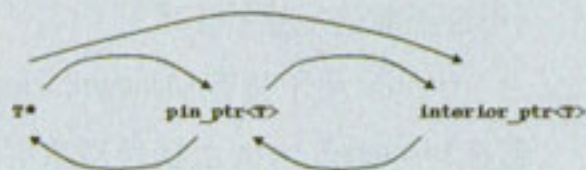
Handle的缺憾是不能进行指针运算,能力较为有限,不如标准C++程序员所熟悉的Native指针那么强大。C++/CLI中引入了一种新的指针形式——interior_ptr。interior_ptr和Native指针的语义几乎完全一样,只不过interior_ptr指向托管堆,在GC时interior_ptr能够得到更新。除此之外,interior_ptr允许你进行指针运算,允许你解引用,一切和Native指针并无二致。interior_ptr为你操纵托管堆上的数据序列(如array)提供了强大而高效的工具,iterator模式因此可以原版照搬到托管环境中,例如:

```
template<typename T>
void sort2(interior_ptr<T> begin,interior_ptr<T> end)
{
    ... // 排序算法
    for(interior_ptr<T> pn=begin;pn!=end;++pn)
    {
        System::Console::WriteLine(*pn);
    }
}

int main()
{
    array<char> arr = gcnew array<char>(3);
    ... // 赋值
    interior_ptr<char> begin = &arr[0]; // 指向头部的指针
    interior_ptr<char> end = begin + 3; // 注意,不能写
    &arr[3], 会下标越界
    sort2(begin,end); // 类似STL的排序方式!
}
```

T*, pin_ptr, interior_ptr —— 把它们放到一起

T*, pin_ptr, interior_ptr是C++/CLI中三种最为重要的指针形式。它们之间的关系像这样:



强大的 Override 机制

C++/CLI 的 Override 机制从 C# 那里借鉴了一些好的经验, 请看下例:

```
class B
{
public:
    virtual void f();
    virtual void g() abstract; // 纯虚函数
    virtual void h() sealed; // 阻止派生类重写该函数
    virtual void i();
}

class D:public B
{
    virtual void f() new; // 新的 f, 并没有重写 B::f。
    virtual void h() override; // 错误! sealed 函数不能重写
    virtual void k() = B::i; // “命名式”重写!
}
```

通过正确的使用这些强大的 override 机制, 你可以获得对类成员函数更强大的描述能力, 避免出乎意料的隐式重写和版本错误。不过需要提醒的是, “命名式”重写是一种强大的能力, 但是需要谨慎使用, 如果使用不当或滥用很可能导致名字错乱。

值类型 & 封箱和拆箱

如果你来自 C#, 我几乎可以听到你的叹息声。的确, 在 .NET 平台上编程, 你无可避免的要面对值类型和引用类型的微妙差别以及“疯狂”的隐式封箱——引用类型 (对应于 ref class) 的实例是第一流的对象, 继承自公共基类 System::Object, 拥有方法表, 对象头等等。但是值类型 (对应于 value class) 却极为简单, 类似于 C++ 中的 POD 类型, 没有方法表和对象头等, 值类型应该被分配在栈上, 而当你用 Handle 来持有值类型实例时, 它就会被隐式的封箱到托管堆上 (因为 Handle 必须持有一个一流的对象), 只有当值类型的实例被封箱到堆上的时候, 它才会拥有第一流的对象特征, 可以被 Object 来引用。

这些都是 .NET 内在的特性, 所有使用 .NET 平台 的语言都必须遵守, 从这个意义上说, .NET 的确是最高统治者。

幸运的是, 情况或许没有你想象的那么糟糕, 或许比在 C# 里面还要好一些——因为 C++/CLI 中的 Handle 的语法特征是如此明显, 所以你几乎可以立即发现什么地方会出现封箱拆箱 (尽管如此, 还是要面对一些微妙的情况), 我们来看一个例子:

```
value class V // value 关键字表示这是个值类型, 值类型
              // 应该分配在栈上
{ int i; }
V v; // 在栈上创建 V 的实例

// 由于 V 必须引用一个“完整”的对象, 也就是具有方法表,
// 元数据以及对象头并继承自 System::Object 公共基类的
// 对象, 所以 v 被隐式封箱到托管堆上。
V* hv1 = v; // 注意, 隐式封箱!
V* hv2 = %v; // 也是封箱!
              // 把“%”用到值类型上会导致一个 Handle,
              // 所以会封箱, 这种形式比较明确!
hv1->i = 10; // 改变的不过是堆上封箱后的对象中的 i, v
              // 的成员 i 的值并未改变
v = *hv1; // unbox, 然后逐位拷贝到栈上, 这时候 v.i 为 10
```

这里你可能意识到了问题——既然用 Handle 来持有值类型总会导致它被封箱到托管堆上, 那么万一我要写一个函数, 接受一个 (栈上的) 值类型实例为实参并改变其成员的值, 该怎么办呢? 如果使用 Handle, 那么你所指向的就不是原来的值而是封箱后的对象, 从而看起来改变了其成员, 其实只不过改变了一个“临时”对象 的值而已! 所以, Handle 在这里应该退居二线, 这里是 “%” (托管的引用, 对应于 Native 引用——“&”) 的用武之地——把一个托管引用绑定到位于栈上的值类型不会引起封箱操作, 我们看一个例子:

```
void adjust(V% ref_v)
{
    ref_v.i = 10; // 改变 ref_v 的成员!
}

int main()
{
    V v;
    adjust(v); // 不会引起封箱操作
    System::Console::WriteLine(v.i); // 打印出 10
}
```

原则是: 要修改栈上的值类型实例, 优先使用 “%”, 而不是 “^”。这样你将获得最好的效率和程序的正确性。

小结

C++/CLI 是一个创举, 它把托管环境和 Native 环境整合在一起, 使开发者同时拥有了“上天入地”的强大能力。面对 C++/CLI, 已经不是争论该不该学习的问题, 而是如何让它发挥更大的能量的问题。👉

免费迁移PB应用至Web

本活动截至11月15日

请访问正阳软件网站: www.appeon.net.cn
进行网上报名。

联系咨询电话: 800-830-1932
0755-26018699 26010514

PB应用发布到Web的几种方法

使用J2EE/.NET重写PB应用

废弃原有PB应用, 从零开始开发新的Web应用**不仅历时漫长,**
而且耗费巨大的人力和财力。

使用PB+Sybase EAServer改写PB应用

重用PB应用的数据窗口(DataWindow)和非可视化对象(NVO), 但
仍需将PB应用的所有窗口逐一重写, 往往**需要数周到几个月的时间。**

Appeon 方法

使用PB+APB将PB应用自动发布

重用PB应用的所有数据窗口、非可视化对象和窗口, 需要修改少量PB源代码以符合
Appeon迁移标准, 之后仅需点击按钮完成到Web的发布, **整个过程仅需几天到数周。**

精密复制原有用户界面
高度用户交互能力



窗口式的C/S应用界面

IE浏览器中的B/S应用界面

正阳软件(中国)有限公司

地址: 深圳市南山区科技园高新南一道创维大厦A座3楼、12楼

总机: 86-755-26743318

网址: www.appeon.net.cn

传真: 86-755-26010074

www.appeon.net

邮编: 518057

微软智能手机游戏 开发经验谈

文 / 蔡学镛

微软在PC平台上具有无可撼动的地位、在服务器平台上也占有相当版图、在PDA平台上更是节节逼近龙头Palm,而现在,微软也开始进入智能型手机平台了。从2004年开始,市场推出多款微软Smartphone手机,包括了华硕、多普达、神达、联想和摩托罗拉等品牌的手机。利用微软的.NET Compact Framework已经不是困难的事情,.NET平台的程序员甚至能够非常迅速地适应这种开发环境。本文就Smartphone手机平台上的游戏开发谈一些作者的个人经验。



.NET Compact Framework

由于Smartphone手机平台已经预先整合了.NET Compact Framework(后面简称为.NET CF),而且.NET CF程序其实就是一般的.NET程序,不像J2ME程序还有一些特殊的规定要遵守(例如,必须遵守MIDP Application Model),所以使用.NET技术来开发手机程序是最方便的选择。任何一个.NET Framework程序员都可以很快地写出第一个.NET CF的程序。利用.NET CF所写出来的程序,因为完全符合Managed PE文件格式,而且所用到的标准库也是.NET Framework完整版的子集合,所以可以在PC上执行。图一的例子是一个利用.NET CF开发出来的Smartphone游戏,也可以直接在PC上执行。(请注意,.NET CF程序兼容于.NET Framework 1.1,但不一定兼容于.NET Framework 1.0)。

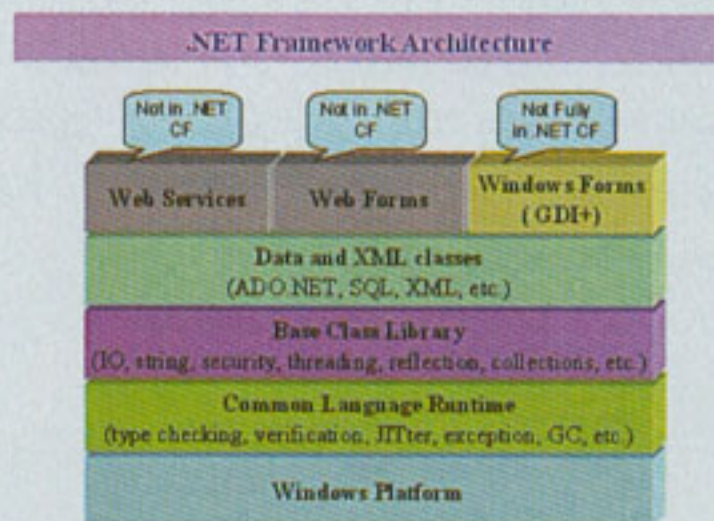


图一

如果你已经习惯用J2SE开发Java程序,然后才开始用J2ME开发程序,你一定会觉得受到许多限制而感到痛苦不已;如果你已经习惯用.NET Framework开发.NET程序,然后才开始用.NET CF开发程序,你一定会因

为没有受到太多限制而感到惊讶不已。大致上说,.NET CF只比完整版的.NET Framework少了ASP.NET(包含Web Services以及Web Forms)。.NET CF不支持ASP.NET是很合理的,因为只有Server才会用到ASP.NET。.NET CF甚至也支持ADO.NET。(如图二所示)

为没有受到太多限制而感到惊讶不已。大致上说,.NET CF只比完整版的.NET Framework少了ASP.NET(包含Web Services以及Web Forms)。.NET CF不支持ASP.NET是很合理的,因为只有Server才会用到ASP.NET。.NET CF甚至也支持ADO.NET。(如图二所示)



图二

下面列出.NET CF所支持的namespace,事实上,除了System.Drawing与System.Windows之外,其它的namespace都在ECMA-335(ISO/IEC 23271)的定义中,我认为以后或许会有微软以外的厂商(例如Esmertec)根据ECMA或ISO的标准来实现出BREW、APOXI、embedded Linux、Symbian等平台上的.NET CF。


```
System.Collections
System.ComponentModel
System.Configuration
System.Data
System.Diagnostics
*System.Drawing (Not in ECMA 335)
System.Globalization
System.IO
System.Net
System.Reflection
System.Resources
System.Runtime
System.Security
System.Text
System.Threading
*System.Windows (Not in ECMA 335)
System.Xml
```

从这些namespace来看，.NET CF的功能似乎相当完备。但是事实上，.NET CF只需要2MB的空间，而完整版的.NET Framework需要超过30MB的空间。.NET CF除了比.NET Framework少了ASP.NET之外，.NET CLR更精简，并删除了许多不实用、少用、累赘、以及耗费运算能力的API。.NET CF 1.0也不支持COM Interop。下面的表格整理出.NET CF与.NET Framework在API个数的差异。

SDK for Windows Mobile 2003

	Classes	Methods
Full .NET Framework	18,700	80,000
.NET Compact Framework	4,700	13,000

为了帮助程序员进行 Smartphone 手机 .NET 程序开发，微软提供了 SDK for Windows Mobile 2003-based Smartphones (俗称 Microsoft Smartphone 2003 SDK)，作为开发工具。你可以到微软的 MSDN 网站免费下载 Microsoft Smartphone 2003 SDK。安装完后，此 SDK 会被整合进 Visual Studio .NET (如图三所示)，并具备 Smartphone Emulator (仿真器)。



图三

有了 .NET CF，Smartphone 的软件开发者，不再需要像早期 Windows CE 的开发者一样使用 eMbedded Visual C++。对于程序员来说，.NET 的语言和 API，更可以从 Server、PC、PDA、到 Smartphone，“一以贯之”，减少重

新学习新技术的精力耗费。这倒不是说 .NET CF 是万能的，尽管 .NET 可以在许多地方取代 eMbedded Visual C++/Win32 API，但是仍有一些程序需要利用 eMbedded Visual C++ 来开发。

如果开发的游戏是益智类 (例如象棋、俄罗斯方块)，那么使用 .NET CF 的 GDI+ 应该足够应付。如果开发的游戏是射击类、动作类，那么就需要执行速度更快的 API 了。关于游戏的 API，我们直接想到的就是 DirectX。DirectX 的确是有提供 Windows CE 版本，不幸的是，.NET CF 程序无法调用 DirectX，因为：

- ◆ Managed DirectX (也就是 .NET 版的 DirectX) 尚未移植到 Windows Mobile 平台上。

- ◆ 微软虽然有提供 Windows CE 平台的 DirectX，但是 DirectX 是 COM 的 API，且 .NET CF 1.0 不支持 COM，所以 .NET 程序无法调用 DirectX。

- ◆ DirectX 没有内建在 Windows Mobile 平台上。

总而言之，在 Windows Mobile 上，不管是用 eMbedded Visual C++ 或 .NET Framework 开发游戏程序，都没办法使用 DirectX。虽然没有 DirectX 可用，微软对于 Windows Mobile 游戏开发者，另外提供了 GAPI (Game API)。GAPI 的文件是 \Windows\gx.dll (只有 9.4KB)，是一组很简单的 API，包含少数几个函数 (function)，用来控制屏幕和键盘，功能远比不上 DirectX 的完整。我们可以在 .NET CF 程序中利用 P/Invoke 来调用 GAPI，所以我们依然可以在 .NET CF 的程序中调用 GAPI 与 DirectX，虽然麻烦了一些。

使用 .NET CF 将遭遇到的挑战

开发手机游戏程序时，必须注意到画面的设计，因为画面的美观与否，会立即影响使用者对此游戏的观感。当游戏画面的信息量多时，需要利用高度的技巧，才能在这有限的空间中放入足够的信息。一般惯常使用滚动条 (Scrollbar) 来增加空间，但是此技巧只适合呈现静态信息，对于 Smartphone 游戏并不太适合。

为了要在小画面上呈现许多信息，可以利用色彩携带一些信息。使用适当的色彩，不但可以增加信息量，也可以让画面更美观，但是太多的色彩会造成视觉上的负担，错误的色彩组合也会造成画面丑化，违反直觉的色彩选择更会造成操作上的错误暗示。关于色彩，如果程序员没有足够的美感，必须请教美工专家给予建议。

在 PC 上的 .NET 程序可以利用 Double Buffering (双缓冲) 的技巧，来消除画面闪烁。但是 Double Buffering

会耗费许多内存空间,所以.NET CF并不支持。为此,必须利用invalid()和update(),来将画面需要更动的小区域进行重绘,而尽量减少大范围的重绘。

Smartphone游戏程序免不了要用到绘图的API,也就是GDI+。然而,.NET CF版本的GDI+ API不支持许多功能,其中包括了Anti-alias(消除锯齿状),和Transform,这使得利用GDI+所绘制的图文看起来不甚美观。Anti-alias与Transform会耗费CPU的计算能力,所以.NET CF不支持,这是我们可以理解的。为了达到Anti-Alias的效果,我的作法是:将图利用Anti-Alias绘制好,程序利用贴图的方式来展现这些图。这么做随之而来的缺点是,程序体积会变大(因为嵌入图档的关系),同时也欠缺使用上的弹性。

Pocket PC vs. Smartphone

Pocket PC和Smartphone虽然很类似,但是仍有不少差异,开发游戏时必须特别注意这些差异。

微软Smartphone的画面大小为176x220,如果扣除标题列与选单列则为176x180;微软Pocket PC的画面大小为240x320,如果扣除标题列与选单列则为240x268,如图四所示。如果想设计一个.NET CF游戏能同时在Pocket PC和Smartphone上执行,必须一开始就把画面处理的原始码独立出来,不要和游戏的主要程序逻辑混杂在一起。



图四

对于Smartphone来说,操作接口指的就是手机上的按键,其感知的是按键事件(key event)。Smartphone游戏必须以一只手能操作为最佳,且必须符合使用惯例。对于Pocket PC来说,主要的操作接口是触控笔(stylus),其感知的是鼠标事件(mouse event)。如果你在.NET CF游戏程序中同时提供按键事件和鼠标事件的event handler,可以让此游戏程序兼顾Smartphone与Pocket PC的需求。

对于Menu的使用,Smartphone有严格的限制。最上层Menu只能有两个Menu Item,且只有右边的Menu Item可以有Submenu(次选单)。不遵守这些规则,将导致程序执行失败(但是编译仍会过关)。

固然Smartphone 2003和Pocket PC 2003都是使用.NET CF 1.0,但是两者的API仍有些许差异,

Smartphone版本的.NET CF是Pocket PC版本的子集合,有一些API被去除。按键的接口适合用选择的方式设计操作接口。(如图五所示)



图五

AI

游戏的AI(人工智能)对手不能太聪明,也不能太笨,否则玩此游戏老是输或老是赢,就一点都不好玩了。AI对手的动作必须有适当的停顿,例如,移动棋子时,一格一格地移动,而不是马上移到定点。关于游戏的AI程序设计方式,可以参考O'reilly出版的《AI for Game Developers》一书(<http://www.oreilly.com/catalog/ai/>)。另外,我的朋友David Weller等人所著的《.NET Game Programming in C#》一书也值得参考(APress出版)。

设计AI时,切忌过于理论。手机游戏的AI应该是以“堪用”为目标,而不是以“击败人脑”为目标。击败人脑是深蓝(deep blue)超级计算机的事,与我们无关。也因此Smartphone游戏软件开发步骤,从我的经验归纳出来的建议是:

先设计画面,因为画面可能会导致游戏规则与操作接口的小幅度变动。再设计游戏引擎,衔接游戏画面,此时必须把游戏规则以及限制条件都加上去。接着设计操作接口,和游戏引擎进行衔接。这个步骤通常很快。然后进行自己和自己对战,对战的过程中,可以找出画面、游戏引擎、以及操作接口的一些瑕疵,同时也可以归纳出一些“赢的手段”。

最后设计AI,这个时候,之前习得的“赢的手段”就可以派上用场,被写成算法。

结语

虽然目前.NET CF的Smartphone才刚开始推出,但是.NET CF的前景相当看好。NET CF程序设计的门坎很低,Microsoft Smartphone和Pocket PC的屏幕规格统一,Visual Studio .NET使用上很方便,这些特点对于程序员来说是很大的吸引力。我相信未来.NET CF的程序会逐渐变多,回过头来推动.NET CF的成长。而游戏与多媒体应用类的程序会在.NET CF的程序中占有相当高的比例。

■ 责任编辑:欧阳璟 (ouyangjing@cscdn.net)

了解 Java 规则引擎

■ 文 / 透明

“规则引擎”，这个词对于很多 Java 程序员来说或许显得有些遥远——那不是总与 CLIPS、人工智能、专家系统之类的“火箭科学”联系在一起的吗？实际上，我们日常工作的问题领域——银行、电信、电子商务、政府信息化，等等——都有着各种各样的业务规则，我们只是“身在此山中”而不自知。为了缓解你的陌生感，就让我们先来玩个游戏吧……



引子

假设你正在给一个机器人编写程序，让它学会开车。不妨再假设别的工程师们早已把液压传动之类的细节封装好，你只要通过一些简单的接口就可以操纵这个机器人。现在，问题来了：怎么让这位一无所知的司机先生通过十字路口呢？“啊哈，”你说，“那太简单了，我现在就可以写出伪码。”话音未落，你写出了这样一段伪码：

```
IF 红灯亮 THEN 停车
ELSE 前进
```

唔，没错。可是，假如面前亮着的是绿灯，却恰好有位老太太颤巍巍地从车前走过，那岂不酿成大祸？“哦，”你说，“一时考虑不周而已，我稍微修改一下。”然后，你把伪码改成了这样：

```
IF 红灯亮 THEN 停车
ELSE
    IF 车前有人 THEN 停车
    ELSE 前进
```

行了，现在这位司机先生算得上谨小慎微。但它的行为方式也未免显得太“菜鸟”了吧？假如车前是一位低着头专心打电话所以没看见信号灯的先生，难道我们不应该摁摁喇叭提醒他一下吗？你也觉得这是个满有道理的建议，于是又做了一点修改：

```
IF 红灯亮 THEN 停车
ELSE
```

```
IF 车前有人 THEN 停车，鸣笛
ELSE 前进
```

哎呀，你的考虑又有点欠缺：有些路段可是禁止鸣笛的。要是司机先生在每个路口都不分青红皂白地摁喇叭，恐怕到月底就会有一大堆罚单寄到你家了。这可不，所以你马上又调整了它的行为方式：

```
IF 红灯亮 THEN 停车
ELSE
    IF 车前有人
        IF 禁止鸣笛
            THEN 停车，鸣笛
        ELSE 停车
    ELSE 前进
```

我就不再刁难你了。不难看出，你很快就会得到一个复杂的、有着十多层嵌套的 IF...ELSE... 结构。如果再加上 AND、OR 之类逻辑条件，恐怕你将很难说出这里的规则究竟是什么，更不用说要去修改它了。更糟糕的是，在这样一个结构里，你很难复用业务规则。比如说“鸣笛”的那一段吧，你打算怎么把它抽象出来复用呢？或者是每逢需要鸣笛时都做这么一堆判断呢？

这只是一个轻松的小游戏。然而在现实工作中，我们或多或少都遇到过类似的情况：业务规则散布在庞杂的逻辑判断中，时间稍长连编写者本人都无法说清究竟有哪些规则，更惶论规则复用了。至于让客户定制规则，就更是想都别想。一旦业务规则有变，开发者们只好再来亲手维护这堆可爱的 IF...ELSE...。面对这种尴尬的情景，我们该怎么办？

规则与规则引擎

路德维希·维特根斯坦在《逻辑哲学论》中这样谈起他的世界观：

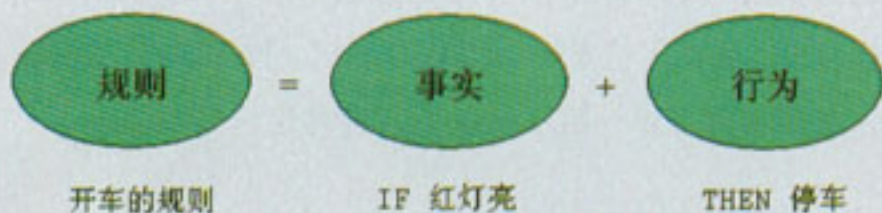
- ◆ 世界是事实的总体
- ◆ 世界为诸事实所规定
- ◆ 事实，就是诸事态 (sachverhalt) 的存在
- ◆ 事态是对象的结合

换句话说，对象 (object) 的属性、以及对象之间的关系构成了诸多的事实 (fact)。下列陈述句都是这样的事实：

- ◆ 用户 A 有 1000 元钱
- ◆ 用户 A 的钱比用户 B 多

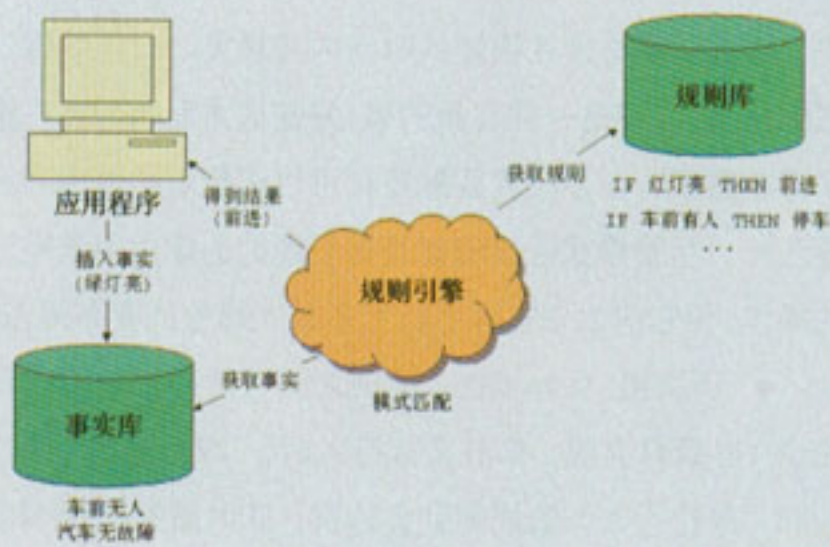
如果把需要关注的所有对象放在一起，我们实际上就拥有了需要关注的所有事实。我们把这些事实称为一个事实库 (fact base)。在一个基于规则的系统 (rule-based system) 中，事实库通常是由面向对象语言在运行时提供的。

除了事实之外，另一个重要的概念是行为 (action)。行为也就是应用程序将要做的一件事、一个动作，例如“开车前进”、“从帐户扣款”等等。负责将事实与行为两者联系起来的便是规则 (rule)。每条规则分为两部分：可能发生的事实，以及此事实发生时应该采取的行为。其中“事实”部分也被称为谓词 (predicate) 或前提 (premise)，“行为”部分也被称为结论 (conclusion)。



可以看到，规则大多数时候可以被描述为一条 IF...ELSE... 语句：如果红灯亮，请停车；如果绿灯亮，请开车前进；如果车前有人，请不要前进……那么，既然都是 IF...ELSE... 的条件判断，引入这个概念又有什么用呢？或者说，是什么让一个基于规则的系统与我们前面那种编程方式有所不同？答案是规则引擎 (rule engine)。

规则引擎是这样的一个程序：你可以在其中维护一个规则库（这些规则通常是以配置的方式放入规则引擎的），并且可以在运行时将一组对象作为事实库交给规则引擎处理；规则引擎将对事实库中的诸事实与规则库中诸规则的“事实”部分进行模式匹配，一旦某条规则指定的事实存在，则执行该规则指定的行为。



这里的关键是：规则引擎处理规则的方式是声明性的 (declarative)，而不是过程性的 (procedural)。换句话说，你不能、也不需要指定规则执行的顺序，规则引擎会执行所有能够与事实库中的事实相匹配的规则（不过你可以指定规则的优先级，譬如“避让行人”的规则优先级高于“绿灯通行”）。这也就意味着你不应该让规则之间有太多的交互（尽管你可以这样做），并且不应该依赖规则执行的先后次序。由于采用了声明性的处理方式，各条规则可以独立地发展完善，添加新的规则也不会对旧规则造成影响。更重要的是，各条规则都能够保持简单，使用户定制规则成为了可能。

用这种声明性的、基于模式匹配的方式来处理业务，规则还带来了一个有趣的特性：规则引擎的处理基本上不会抛出异常（除了陷入无穷递归之后的 OutOfMemoryError 之外）。如果规则库中的规则不够齐备，规则引擎通常也能获得结果（不过很可能是与你预想不符的结果），而不会抛出异常使系统不能服务。不过，这个特性究竟是好是坏，恐怕只有你自己才能确定了。在本文后面部分，我们会看到一个这样的例子。

更正式地说，一个典型的基于规则的应用包括以下几个部分：

- ◆ 推理引擎。也就是前面所说的规则引擎。它负责将事实与规则联系起来，并执行合适的行为。
- ◆ 规则库。保存所有的规则。成熟的规则库通常不会以文本形式保存规则，而是用规则编译器将规则编译成二进制形式，再加上索引保存，以便快速处理。一种常见的规则数据结构就是著名的 Rete 网络 (Rete Network) ——“rete”正是拉丁文中的“net”。
- ◆ 事实库。保存所有的事实。

其中，最重要的部分——推理引擎——又由以下几部分组成：

- ◆ 模式匹配器 (pattern matcher)。事实库中可能包含数千个事实，规则库中的每条规则又可能有两三个前

提，推理引擎必须在很短的时间内将事实与对应的规则匹配起来，这不是一件容易的事。好在这方面的研究已经非常成熟了，很多模式匹配器都可以在极短的时间内完成匹配。尽管模式匹配仍然是基于规则的应用中最耗时的操作，但它的效率实际上比大多数人想象的要高得多。

◆ 调度器 (agenda)。规则之间可能存在冲突。譬如说，当绿灯亮起，车前又有行人时，“绿灯通行”的规则和“避让行人”的规则就会冲突。此时就需要由调度器来决定采用哪条规则。通常规则可以有优先级的设置，调度器将根据优先级来解决冲突。

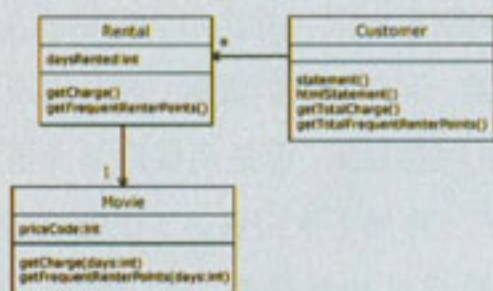
◆ 执行引擎 (execution engine)。最后，规则引擎需要执行规则的行为，执行引擎就负责这项工作。在早期经典的基于规则系统中，规则引擎只能增、删、改事实库中的诸事实，除此之外不能做任何事。但现代的规则引擎通常放宽了限制，例如 Jess 就可以使用位于当前运行时 CLASSPATH 中的任何 Java 类。

看到这里，读者应该对规则、规则引擎、基于规则的应用之类概念有个基本的印象了。下面，我们就来看一个真实的例子，看看如何用 Java 构建一个基于规则的应用。

真实的规则

要找到一个合适的例子来演示规则引擎的运用并不容易，幸亏我的手边恰好就有这么一个例子：来自《重构》(Refactoring, Martin Fowler 著，侯捷/熊节译，电力出版社 2003 年)第一章的“音像店”的例子。因为是这样现成的例子，我不会过多介绍它的背景，读者可以参阅《重构》一书。

在我们的音像店里有三个主要的类：代表“顾客”的 Customer、代表“影片”的 Movie 和代表“租借”的 Rental。每个顾客可能发生多次租借，每次只允许租借一部影片。根据影片类型不同，租借时间长短不同，租金也各有不同。



说得更详细点，影片共分三类：普通影片、儿童片、新片。各种类型没有重叠，也就是说，不会有“新上映的儿童片”这样的分类。各种影片的租金定价如下：

- ◆ 普通影片：两天以内收取 2 元；超过两天，每超出一天加收 1.5 元。
- ◆ 儿童片：三天以内收取 1.5 元；超过三天，每超

出一天加收 1.5 元。

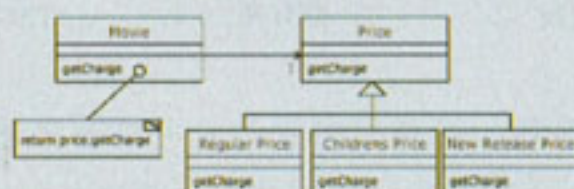
- ◆ 新片：每天收取 3 元。

在最初的实现中，我们看到了一个典型的 switch 语句：根据不同的影片类型（即 priceCode 的值）决定如何定价。各种定价策略混淆在同一个方法中，无法看清各种策略的实质；如果需要修改哪个定价策略，或是添加新的定价策略，都必须修改 Movie 类，给单元测试带来很大的麻烦。这个方法的代码如下所示：

```

double getCharge(int daysRented){
    double result = 0;
    switch (getPriceCode()){
        case Movie.REGULAR :
            result += 2;
            if (daysRented > 2)
                result += (daysRented-2)*1.5;
            break;
        case Movie.NEW_RELEASE :
            result += daysRented * 3;
            break;
        case Movie.CHILDRENS :
            result += 1.5;
            if (daysRented > 3)
                result += (daysRented-3)*1.5;
            break;
    }
    return result;
}
    
```

经过重构之后，我们用一个 State 模式取代了 switch 语句：将“定价策略”封装在一个 Price 对象中，并将“确定租金”的操作委派给 Price 对象的 getCharge() 方法。这样一来，各种定价策略可以独立测试、独立修改，代码的阅读者也可以一目了然地看清各种定价策略。



但这个设计也并非没有缺陷。首先，不同的定价策略被封装在不同的 Price 子类中，这无疑增加了系统的复杂程度。每当希望添加新的定价策略时，我们必须新建一个 Price 子类，还必须修改负责创建 Price 对象的工厂。更重要的是，Price.getCharge() 方法只有一个参数：租借的天数。换句话说，影片的价格必须根据“定价策略”和“租期”两项信息确定。在一个商用软件系统中，这个不起眼的局限很可能成为一个严重的束缚——读者不妨想象一下，如果某位音像店老板希望给顾客积分，并根据积分制订不同的定价策略，我们该怎么办？如果这位老板想做好事，给 60 岁以上的老人特殊优待，我们该

怎么办？照经验来看，“定价策略”是变化非常频繁、非常无规律的一类业务逻辑，我们需要一个更清晰、更灵活、更便于修改的封装。

用 Jess 封装业务规则

Jess 是一个基于 Java 的规则引擎，它采用了 CLIPS 的语法基础——还有多少人记得这种多用于专家系统、人工智能领域的“实验室语言”？在这篇短短的文章里，我无法涵盖 Jess 的方方面面，只能带着读者一边实现前文所述的业务需求，一边蜻蜓点水地介绍 Jess 语言的常用特性。现在，我们开始吧。

首先，我们为“普通影片”制订规则。正如前面说过的，一个规则由两部分构成：当哪些事实存在时触发它（即“IF”部分）；触发之后做什么事（即“THEN”部分）。我们先来看看如何定义一个规则，并为它指定触发条件：

```
(defrule rule-regular
  ?movie <- (sample rule.Movie (OBJECT ?C))
  ?daysRented <- (java.lang.Integer (OBJECT ?I))
```

不难看出，Jess 使用了一种函数式的语法。我们可以用 defrule 函数来定义一个规则，同时为它指定一个名称。随后，我们指定了两个事实 ?movie 和 ?daysRented，它们的含义分别是“事实库中存在一个 sample.rule.Movie 对象（将其命名为 ?C）”和“事实库中存在一个 java.lang.Integer 对象（将其命名为 ?I）”。读者可以看到，这条规则假设事实库中会有一些 Java 对象存在。因为 Jess 是基于 Java 的规则引擎，所以它可以像 Java 程序一样使用 CLASSPATH 中的所有 Java 类。如果 Java 程序把对象放进事实库中，Jess 程序就可以使用这些对象。

如果这里声明的两个事实都成立，我们就应该算出普通影片的租金——或者说，触发这条规则。我们这样声明这条规则的“行为”部分：

```
=>
(bind ?days (call ?I intValue))
(if (> ?days 2) then
  (bind ?charge (new Double (+ 2 (* 1.5 (- (call ?I doubleValue) 2)))))
else
  (bind ?charge (new Double 2.0)))
```

首先，我们需要用一个“粗箭头”（“=>”）来分隔规则“事实”和“行为”两个部分。随后，我们需要取出“租期”的数值——别忘了，事实库中虽然存在“租期”这个对象，但那是一个 Integer 对象，而不是 Jess 能

够处理的数值。用 Jess 提供的 call 函数可以调用一个 Java 对象的方法，bind 函数则可以将一个值赋给一个变量。譬如上面的

```
(bind ?days (call ?I intValue))
```

就等价于下列 Java 代码：

```
Integer i;
int days = i.intValue();
```

明白 call 和 bind 两个函数的用法之后，随后的几行代码就很容易理解了。唯一需要留意的是，算术运算在 Jess 中也是以函数的形式出现的，所以你会看到类似“(+ x y)”这样的表达法。不过，回想大学时曾经熟悉的前序表达式，这样的函数也就不会显得太过突兀了：它无非是表示“用 x 和 y 两个参数执行加法操作（函数）”而已。

最后，我们把计算得到的结果放入事实库。尽管 Jess 程序可以操作 CLASSPATH 中的所有 Java 类，但一般情况下不应该这样做，因为一个构造良好的基于规则的应用唯一能够操作的便是事实库——添加事实、删除事实、修改事实，事实库就是规则的整个运行时环境。调用 assert 函数就可以向事实库中插入新的事实，例如：

```
(assert(java.lang.Double(class(get ?charge class))(OBJECT ?charge)))
```

现在我们已经拥有了第一条规则。但仅仅一条适用于普通影片的规则显然是不够的，音像店的店主马上就提出了新的需求：新片应该收取更高的租金。编写这第二条规则并不困难，唯一不同的是我们需要指定：只有价格代码（priceCode）等于 1 的影片（即“新片”）才适用这条规则。很快地，我们写出了这条规则：

```
(defrule rule-new-release
  ?movie <- (sample rule.Movie (priceCode 1) (OBJECT ?C))
  ?daysRented <- (java.lang.Integer (OBJECT ?I))
  =>
  (bind ?charge (new Double (* 3 (call ?I doubleValue))))
  (assert(java.lang.Double(class(get ?charge class))(OBJECT ?charge)))
```

请留意上面的粗体字部分：对于 ?movie 这个事实，我们不仅要求事实库中存在 Movie 对象，还要求这个对象的 priceCode 属性值等于 1。聪明的读者马上就会想到了，Jess 程序是按照 Java Bean 规则去调用 Movie.getPriceCode() 方法获得这个属性值的。

现在读者要问了：如果事实库中确实有这么一个



“新片”对象，Jess怎么知道应该调用rule-new-release这条规则呢？答案是：Jess不知道。Jess只会忠实地进行模式匹配，凡是匹配合格的规则都会被触发——但是有先后之分。在我们这里，“新片规则”的优先级高于“普通影片规则”，所以我们可以让rule-new-release在rule-regular之后执行，并且始终取出事实库中的最后一个对象作为租金值即可。于是，我们给rule-new-release设置一个较低的salience值（默认为0），让它较晚执行：

```
(defrule rule-new-release (declare (salience -10))
```

用这种方式设置规则的优先级简便明了，而且对已有规则的影响很小——你不需要修改现有的规则，只要给新的规则设置一个较高或较低的salience值即可。但这种做法的缺点也是显而易见的：所有匹配合格的规则都会被触发，如果规则的行为会耗费大量资源，系统的性能就会大打折扣。所以这种做法只适合于“规则的行为耗费资源极少”的情况。

依此类推，读者应该能够写出儿童片的租金计算规则了。现在只剩下最后一个问题：如何把Jess程序与Java程序结合起来。我们将使用JSR-94（Java规则引擎）的参考实现来运行Jess程序，读者可以在<http://www.jcp.org/en/jsr/detail?id=94>下载这个参考实现。按照JSR-94规定的API，我们需要首先获得一个规则服务的提供者对象：

```
Class.forName("org.jcp.jsr94.jess.RuleServiceProviderImpl");
RuleServiceProvider serviceProvider = RuleServiceProviderManager
    .getRuleServiceProvider("org.jcp.jsr94.jess");
```

然后，通过服务提供者获得规则管理器：

```
RuleAdministrator ruleAdministrator = serviceProvider.getRuleAdministrator();
```

将包含了Jess代码的InputStream对象传递给规则管理器，我们就能得到一个规则执行集(RuleExecutionSet)，其中包含了我们编写的所有规则。然后，我们就可以把这些规则注册到规则管理器上：

```
InputStream inStream = getClass()
    .getResourceAsStream("charge.xml");
RuleExecutionSet res1 = ruleAdministrator
    .getLocalRuleExecutionSetProvider(null)
    .createRuleExecutionSet(inStream, null);
ruleAdministrator.registerRuleExecutionSet(res1.getName(),
    res1, null);
```

现在，我们可以得到一个规则运行时对象，并从中创建规则会话。最常用的会话方式是无状态的会话，因此我们创建一个StatelessRuleSession对象：

```
RuleRuntime ruleRuntime = serviceProvider.getRuleRuntime();
StatelessRuleSession statelessRuleSession = (StatelessRuleSession)
    ruleRuntime.createRuleSession(res1.getName(),
        new HashMap(),
        RuleRuntime.STATELESS_SESSION_TYPE);
```

现在，我们把所有事实放进一个List，然后把它交给StatelessRuleSession。这个List就是我们的规则将要使用的事实库，规则需要使用的对象都需要放进这个List传入：

```
List facts = new ArrayList();
// 把需要的事实放入 facts 列表中
List results = statelessRuleSession.executeRules(facts);
```


调用executeRules()方法之后，返回的List对象便是Jess程序运行完毕之后的事实库。按照前面的设计，我们取出其中的最后一个对象，那就是我们需要的租金值：

```
Number charge = (Number) results.get(results.size() - 1);
```

不论定价策略发生怎样的改变，Movie.getCharge()方法也不需要再做修改，我们只需要修改Jess程序中的规则，就能够满足新的需求。而且，如果做了足够的封装，再对客户加以培训，甚至可以让客户自定规则——这正是国外很多银行、电信应用的常见做法。对于计费规则、利率规则等变化频繁、又比较容易表达的业务规则，用这种方式来处理是最合适的。

尾声

本文简单介绍了基于规则的应用系统中的一些主要概念，并带领读者用Java和Jess开发了一个基于规则的示例应用。读者也许会觉得Jess的语法毕竟有些晦涩，有点太难掌握。好在JSR-94只规定了Java规则引擎的API，并没有规定编写规则的语言。另一个出色的Java规则引擎产品Drools (<http://www.drools.org>)就允许使用XML和Groovy来编写规则，大大简化了规则的定制。如果读者希望在真实的项目中应用规则引擎，也许Drools会是一个更好的选择。

这篇文章就此搁笔，希望对熟悉Java的读者了解规则引擎有所帮助。本文的示例代码可以在CSDN网站《程序员》杂志频道下载。我们下次再见。 

■ 责任编辑：欧阳璟 (ouyangjing@csdn.net)

探讨与比较 Java 和

.NET 的事件处理框架

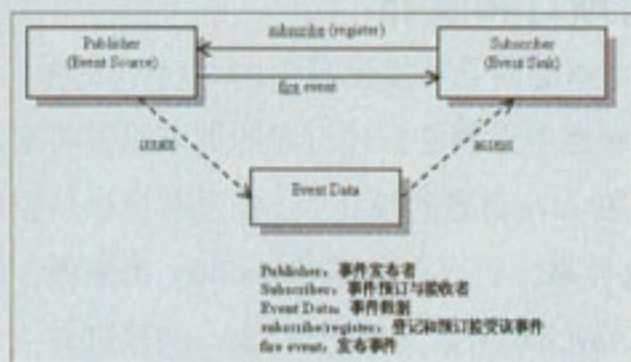
文 / 游智超

本文主要研究和比较 Java 和 .NET 事件处理框架的设计思路。通过比较的方式来思考和学习，可以更清晰的了解“为何”要这样做，而不是只知道“如何”做。软件的发展历史，就是一直试图不断简化处理复杂事务方法的历史，通过和过去的比较，可以更明了发展的轨迹和方向。在 .NET 方面，本文只使用 C# 语言来做说明，VB.NET 其实只是语法上不同罢了，其实大同小异。

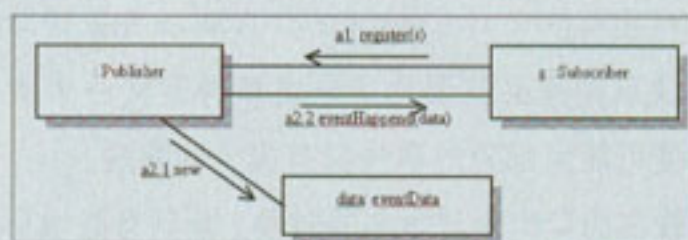
事件驱动模型

事件驱动模型是软件系统平台中的一个重要区域，现代软件系统大量地使用事件驱动的处理方法，尤其在用户界面方面。虽然如此，过去在软件开发语言中一直没有融入事件处理的因子，直到 .NET 的出现，才将事件处理的工作负荷一部分的分派给编译器，从而稍微减轻开发者的负担。

下图显示事件模型的组成份子：



Subscriber 需事先和 publisher 预订要接受其发布的某事件（下图 a1），publisher 在某事件发生以后，必需先生成该事件的相关数据对象（下图 a2.1），然后通过方法调用来通知 subscriber（下图 a2.2），也就是用回调（callback）的方式来通知 subscriber。当然在预订的时候，并不一定要由 subscriber 自身来预订，也可以由另一个对象来帮忙预订。其动态图形示意如下：

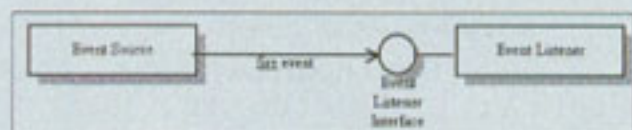


本文并不探讨异步的信息传送，也就是在整个事件的处理过程当中，publisher 和 subscriber 对象皆需要同时存在。如果对于离线（offline）的方式来处理事件有兴趣的话，请参阅 Java 的 JMS (Java Message Service) 和 .NET 的 LCE (Loosely Coupled Events)。

事件是什么？

那么，到底事件是什么？在软件系统中要如何表达一个事件？一个事件应该包括两个东西：识别事件的名称（event identity），和事件的相关的数据（event data）。例如，一个键盘按键被按下的事件可能叫 KeyPressedEvent，事件数据则为该按键的代码。

先前提到发布事件是用调用方法的方式（回调），不过有一个问题，就是 publisher 无法事先知道 subscriber 的类型。在 Java 的编码模式当中，回调可以使用接口模式，也就是 publisher 必需事先定义好一个在发布事件中使用的接口，subscriber 实现该接口中的方法，publisher 则通过调用接口中的方法来完成发布事件的工作。如下图：



这样，在 Java 的编码模式中，一个事件的识别名称就是接口名称和其中的方法名称，而事件数据则自然是接口方法的参数了。Java 对于这个接口的命名风格为 XXXListener，顾名思义就是某事件的倾听者。例如：

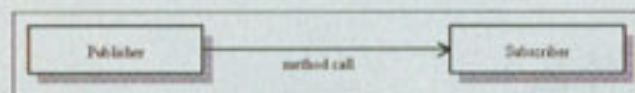
```
public interface KeyListener extends EventListener {
    public void keyTyped(KeyEvent e);
}
```



```
public void keyPressed(KeyEvent e);
public void keyReleased(KeyEvent e);
}
```

由于一个接口中可以包含多个方法，所以Java在设计事件的时候，是将一组相关联的事件放在一起，这样设计的优点是可以很好的将事件做分类，并且在publisher中如果要处理的事件较多的话，可以使用比较少的成员变量来记录subscribers。缺点是如果subscriber只对事件接口中的部分事件有兴趣，也必需要全盘实现该接口（所以在AWT里有java.awt.event.XXXAdapter抽象辅助类）。另一个缺点则是必需要为每一类事件定义一个接口类型，即使可能大部分的事件只有极少的方法。

微软在为C#语言命名的时候，就刻意隐喻C#是从C/C++为基础发展而得的面向对象程序语言，始祖绝不是Java，所以肯定保留一些C/C++的语言机制。在C/C++里面对回调的设计方式就是用函数指针，想当然C#也希望直接使用类似函数调用的方式来作为事件发布的方法。如下图：



所以C#期望使用函数指针类型来作为事件的识别名称，然后用函数的参数来传递事件数据。我们先写一段C++代码来描绘这幅图画：

```
Event type definition;
// 定义KeyPressedCallback 为一个函数指针的类型，
// 该函数接受一个整数型参数，无返回值
typedef void (*KeyPressedCallback)(int keyCode);

Publisher;
class Publisher
{
    public KeyPressedCallback KeyPressedSink = null;
    ...
    void FireEvent(int KeyCode)
    {
        if (KeyPressedSink != null)
            (*KeyPressedSink)(keyCode); //callback
    }
}

Subscriber;
void KeyPressedHandler(int keyCode)
{
    ...
}

...
Publisher publisher = new Publisher();
//register
publisher.KeyPressedSink = &KeyPressedHandler;
```

一个当代的纯面向对象程序语言，是肯定希望要把造成程序复杂和不易维护的指针给去除的。所以在C#语言机制当中，势必要创造新的元素来取代，于是delegate（委托）出现了。如下：

```
Event type definition;
// 定义KeyPressedDelegate 为一个类似函数指针的类型，
// 该函数接受一个整数型参数，无返回值
delegate void KeyPressedDelegate(int keyCode);

Publisher;
class Publisher
{
    public KeyPressedDelegate KeyPressed = null;
    ...
    void FireEvent(int KeyCode)
    {
        if (KeyPressed != null)
            KeyPressed(keyCode);
    }
}

Subscriber;
void KeyPressedHandler(int keyCode)
{
    ...
}

...
Publisher publisher = new Publisher();
//register
publisher.KeyPressed = KeyPressedHandler;
```

一开始，你可以把KeyPressedDelegate当成是与函数指针相类似的东西，通过它你可以引用一个实例方法或静态方法，就好像引用一个对象一样。然后可以通过这个delegate直接调用其引用的方法。但是下面你会看到delegate更扩大了其引用能力。

事件的预订和发布

Publisher必需能够接受多个subscribers的预订，所以在publisher当中必需维护预订者的列表以供将来发布事件使用。在Java语言的模式中，并没有提供特别的东西来帮助这件事，可以自己用collection类来做，例如可以使用ArrayList对象来做记录。Java的预订方法名的风格为addXXXListener()，因为在publisher端必需把subscriber对象“添加”到预订者列表后面，如下图：



对于subscriber来说，预订动作的内部处理是黑箱的，subscriber不用关心publisher是如何做预订记录的。参考以下代码片段：

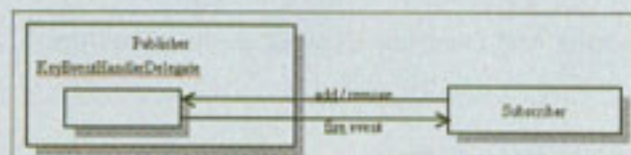

```

class Publisher {
    private ArrayList listenerList = new ArrayList();
    public void addKeyListener(KeyListener l) {
        listenerList.add(l);
    }
    public void fireKeyPressedEvent(int keyCode) {
        Iterator iter = listenerList.iterator();
        while (iter.hasNext()) {
            KeyListener l = (KeyListener)iter.next();
            l.keyPressed(keyCode);
        }
    }
}

```

当然这段代码只是简单的示意, 如果要考虑多线程的安全问题, 可能要在 `addKeyListener()` 前面加上 `synchronized`; 还有, 有预订就必然相应的有“退订” (`removeXXXListener()`), 在这里就不再把它写出来。

如果每个 publisher 都要这样重复撰写这样的代码的确很麻烦。所以在 .NET 中势必希望能够提供一种用来帮助 publisher 记录预订者, 和发布事件的工具。按一般设计者的初步想法, 一定是先提供一个辅助类来协助:



从语法上考虑简化, `add/remove` 动作应该可以用 C++ 的 `operator=()`, `operator+=()` 和 `operator-=()` 来完成。像这样:

```

Publisher publisher = new Publisher();
...
publisher.KeyEventHandlerDelegate += KeyPressedHandler;
// 等同于
// publisher.EventHandlerDelegate.add(KeyPressedHandler);

```

如果可以这样撰写的话, 确实很简单。不过在强制性类型 (strongly-typed) 语言系统中, 必需精确的定义 `add()` 方法参数中的 delegate 类型, 这样似乎无法写出一个可以公用的基础类。所以在 .NET 中, 是结合 delegate 关键字, 通过简单的语法, 借助编译器来帮我们自动生成相关的代码。于是把 delegate 的能力再予以加强了:

```

Event type definition;
public delegate void KeyPressedDelegate(int keyCode);

Publisher;
class Publisher
{
    public KeyPressedDelegate KeyPressed;
}

```

```

...
void FireKeyPressedEvent(int KeyCode)
{
    if (KeyPressed != null)
        // 依次调用记录在 KeyPressed 中的所有方法
        KeyPressed(keyCode);
}

Subscriber;
void OnKeyPressed(int keyCode)
{
    ...
}

void OnKeyPressed2(int keyCode)
{
    ...
}

...
Publisher publisher = new Publisher();
publisher.KeyPressed = OnKeyPressed; // 预订
publisher.KeyPressed += OnKeyPressed2; // 预订另一个

```

这样一个 delegate 不仅可以帮忙记录一个以上的 subscribers, 也可以简单的通过一行的调用语句来发布事件。其实编译器会为每一个 delegate 生成一个相应的类来帮助处理这些工作, 不过是作为一个只是编写应用系统的程序员是不必要去了解这些细节的, 有兴趣的人可以去研究 `System.Delegate` 和 `System.MulticastDelegate` 类。

上面看到的结果应该已经是比较满意的, 但是仍有改善空间。首先, 因为一个 delegate 成员是 `public` 的, 任何人都可以任意的直接接触, 有失面向对象世界中的信息封装和隐藏 (information encapsulation and hiding) 的原则。所以在 C# 中又增加一个关键字 “event”, 用在放在声明一个 delegate 成员变量的前面, 这样表示只有在声明这个 delegate 的类内部才可以直接对它进行 subscriber 调用。

```

public delegate void KeyPressedDelegate(int keyCode);
class Publisher
{
    public event KeyPressedDelegate KeyPressed;
    ...
    void FireKeyPressedEvent(int KeyCode)
    {
        if (KeyPressed != null)
            // 只有在 Publisher 才可以
            KeyPressed(keyCode);
    }
}

// outside of Publisher...
Publisher publisher = new Publisher();
// !!! 不允许 !!! 会编译错误 !!!
publisher.KeyPressed(100);

```


接着, event delegate是以一个成员变量的方式存在, 如果能以属性的方式让外界进行存取, 不是更好吗。于是又增加了 event accessors。在C#语言中, 是使用add和remove来封装实际的 += 和 -= 操作。如下:

```
class Publisher
{
    protected event KeyPressedDelegate m_KeyPressed;

    // event accessor, 定义一个事件属性。
    public event KeyPressedDelegate KeyPressed
    {
        add
        {
            m_KeyPressed += value;
        }
        remove
        {
            m_KeyPressed -= value;
        }
    }

    void FireKeyPressedEvent(int KeyCode)
    {
        if (KeyPressed != null)
            m_KeyPressed(KeyCode);
    }
}
```

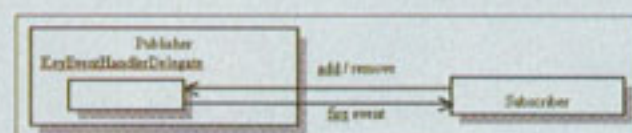
不管是事件变量或者是事件属性, 对声明事件变量和属性的类的外部, 只能对它做 += 和 -= 操作。这样可以加强它的安全性。当然 event accessor 只有 add 和 remove 操作, 所以不管是任何人 (包括声明该事件属性的类内部), 也只能对事件属性做 += 和 -= 操作。

经过这样的改善, 可以理论上更减弱 publisher 和 subscriber 之间的耦合力了。

事件数据

接下来我们谈一谈另一个在事件模型中的重要角色, 就是在事件发布中被传递的“事件数据”。

一个 subscriber 在接受同一种事件的时候, 可能来自不同的 publisher, 所以自然地希望知道发出事件的人是谁, 也就是在传递的参数当中, 必需包含一个 publisher 对象的引用。在Java中, 推荐所有的事件数据类都继承 java.util.EventObject 类。因为在生成一个 EventObject 对象的时候, 必需给一个 event source 对象作为参数。然后可以通过 EventObject 的 getSource() 方法来取得这个对象。在 EventObject 里面, 并没有包含其他任何事件数据, 所以如果在事件的传递过程当中, 有任何事件数据需要传递, 就必需从 EventObject 派生出一个新的子类出来。如下图:



在.NET当中也有一个相似的类叫System.EventArgs, 但是这个类的内容是空的, 如下:

```
public class EventArgs
{
    public static readonly EventArgs Empty;
    static EventArgs()
    {
        Empty = new EventArgs();
    }
    public EventArgs()
    {
    }
}
```

.NET认为不一定所有的subscriber都对event source感兴趣, 所以如果需要的话, 就把event source当成是delegate方法的参数来传递好了。NET定义了一个标准的delegate EventHandler, 以下是它的签名 (signature):

```
public delegate void EventHandler(object sender, EventArgs e);
```

以后, 只要你需要的 delegate 的签名与 EventHandler 相同的话, 就直接用它了。这里所谓的签名相同, 是指参数的类型和返回值的类型皆相同。

Java和.NET都希望用户在定义的事件数据类的时候, 尽可能的使用推荐的基类, 因为这样在publisher对发出的事件数据内容有所变更或扩大的时候, 对subscriber的冲击会比较小, 这是由于多型 (polymorphism) 机制的帮助。

结语

经过这番解析之后, 应该能够比较清楚的了解到Java和.NET事件处理框架的设计思路, 希望有助于读者更进一步理解其框架的形成过程。从语言的角度来看, .NET的确有一些针对性的改善和试图简化对事件的处理, Java则仍保有其一贯简约的风格。读者若有任何意见和指教, 可以通过e-mail与我交流: bruceyou@sina100.com。

关于作者

泓远软件 (上海) 有限公司的技术总监兼副总经理。籍贯台北市, 1991年台湾大学资讯工程硕士班毕业。从事Java/Linux的研究开发多年, 主要精通面向对象相关技术的设计与开发。

■ 责任编辑: 欧阳璟 (ouyangjing@csdn.net)

Java 多线程编程实例

——优化 Cache 并发访问性能

■ 文 / 陈大峰

Cache 是一种提高性能的重要手段。在多线程环境下, 为了避免并发的读写操作可能造成的丢失修改等问题, 往往采用“独占式访问”的方法来确保数据的一致性, 然而这种方法可能会严重损害 Cache 的并发访问性能; 而如果不小心使用了有缺陷的加锁算法, 甚至还会掉入死锁的陷阱。本文最后给出的方法巧妙利用 TreeMap 实现了小粒度加锁, 显著地提高了 Cache 的并发访问性能。

在 产品级软件开发中, 性能向来是一个被高度关注的问题; 而 Cache 可能是在各种提高性能的手段中最为常见的了, 大多数软件产品都针对系统本身的特点定制了自己的 Cache 机制。但是, Cache 本身的性能, 特别是并发访问性能, 往往会受到一些其它因素的牵制, 如果处理得不好不仅会严重影响性能, 甚至还会导致死锁。

背景

我们的网上购物系统对产品数据库的访问非常频繁。产品用 ProductInfo 类来表示, 主要信息包括该产品的唯一编号 (id) 及其当前的存货数量 (stock) 等。另外, 我们用 DBAccessLayer 类用来代表数据库访问层, 它将底层数据库结构屏蔽起来, 为上层提供了一个简洁易用的访问接口。其中 DBAccessLayer.getProductInfoById(long) 方法接受一个 id 作为参数, 通过查询数据库来创建相应的 ProductInfo 对象。由于产品数据库非常庞大, 每次查询访问都非常耗时, 因此我们可以用下面的代码来模拟这种情况:

```
public class DBAccessLayer {
    /* other methods here ... */
    public ProductInfo getProductInfoById(long id) {
        try {
            /*statement.execute("SELECT * FROM ...");*/
            //time consuming operations
            Thread.sleep(1000);
        }
        catch(Exception e){
            return null;
        }
    }
}
```



关于作者: 陈大峰, 分布计算方向研究生, 具有多年面向对象和 Java 中间件开发工作经验。他目前在中创软件公司工作, 从事消息中间件方面的开发和研究。

```
//ProductInfo(long id, byte[] image, int stock)
return new ProductInfo(id, new byte[102400], 10);
}
```

版本 1: 简单, 但很不完善

现在, 为了提高性能, 我们将实现一个 CacheSample 类, 用 LRU 算法对最近访问的 ProductInfo 进行缓冲, 以便减少 getProductInfo 访问数据库的次数:

```
public class CacheSample {
    Hashtable productCache = new Hashtable();
    DBAccessLayer db = new DBAccessLayer();
    /* other methods here */
    public ProductInfo getProductInfoById(long id) {
        Long key = new Long(id);
        if (productCache.containsKey(key))
            return (ProductInfo)productCache.get(key);
        ProductInfo ret = db.getProductInfoById(id);
        //swap-out goes here
        /*if(productCache.size() > MAX_ENTRIES)... */
        productCache.put(key, ret);

        return ret;
    }
}
```

CacheSample.getProductInfo() 方法的实现非常简单直

接，它首先看所需的对象是否存在于内部维护的 productCache 中，如果存在，则绕过数据库查询立即返回；否则仍然从数据库中查询，并将查询结果缓冲到 productCache 中。这看起来没有任何问题，然而不幸的是，就在这短短几行代码中却潜藏着一个危险的错误。

假设编号为 0001 的产品还有 10 件存货，并且尚未缓冲到 productCache 中；如果此时恰好有两个线程（客户）同时访问该产品，会发生什么事情呢？在执行 CacheSample.getProductInfoById(0001) 时，二者都不会从 productCache 中得到结果；于是 DBAccessLayer.getProductInfoById(0001) 将被分别执行，从而使系统中出现了两个对应于 0001 产品的 productInfo。现在，这两个客户都可以购买 10 件产品，并且成功调用 ProductInfo.decreaseStock(10) 来修改存货数量，谁也不知道存货已经透支——这便出现了典型的丢失修改问题。

版本 2：稳定，又笨拙不堪

为了解决丢失修改问题，我们必须进行并发控制。最常用的方法就是对 productCache 进行加锁（我们不能仅仅同步 getProductByld 方法，因为还有其它查询产品信息的方法也可能产生同样的 ProductInfo 对象）：

```
public ProductInfo getProductInfoById(long id) {
    synchronized(productInfo) {
        Long key = new Long(id);
        if (productCache.containsKey(key))
            return (ProductInfo)productCache.get(key);
        ProductInfo ret = db.getProductInfoById(id);
        /*if(productCache.size() > MAX_ENTRIES)...*/
        productCache.put(key, ret);

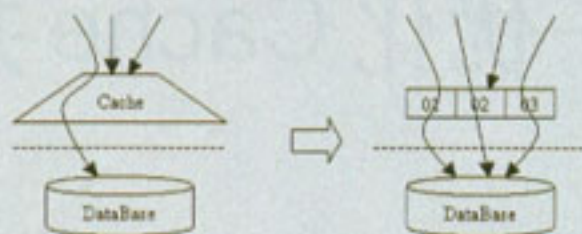
        return ret;
    }
}
```

这是一种典型的“独占式”方法，这种方法实现起来非常简单，但是，只有在 Cache 的命中率接近 100% 的情况下，这样做才是合理的；在命中率较低的情况下，它可能会给 Cache 的并发访问性能造成巨大的伤害——因为它实际上是将所有对产品数据库的访问串行化了，即使是访问不同的产品，两个线程也不能同时执行 DBAccessLayer.getProductInfoById() 方法。在命中率很低而并发度很高的极端情况下，使用这种 Cache 甚至会适得其反，性能可能还不如直接使用“select for update”这样的 SQL 语句。

版本 3：轻快，却惊现陷阱！

为了提高并发访问性能，就必须寻找更优化的加锁

策略。不难看到，独占式访问最主要的问题是，当 Cache 没有命中而不得不执行费时的数据库查询操作时，也会阻塞其它的访问操作。这提醒我们应该减小加锁粒度，最好是只锁住那些有必要锁住的资源，如下图所示。



上图中的锁不是施加于整个 Cache 之上，而是针对每个特定的产品。当 Cache 没有命中而转入数据库查询操作时，它只锁住正在查询的数据，这样访问不同数据的线程就可以并发地查询 productCache 和数据库了，因此显著地提高了并发访问性能。但是，这种方法必须解决一个在实现上颇具挑战性的问题——当要查询的数据尚未存在于 productCache 之中时，应该如何加锁呢？

首先想到的可能是“占位法”，即让首先进入临界区的线程立即向 productCache 中填入一个“哑数据”（dummy）进行占位，然后退出临界区执行数据库查询操作；此后进入临界区的线程将首先判断是否为 dummy 对象，如果是则在该对象上等待（wait），直到数据库查询成功后被唤醒（notify）：

```
public ProductInfo getProductInfoById(long id) {
    Object dummy = "__DUMMY__";
    Object value = null;
    synchronized(table) { //Enter critical section
        value = table.get(key);
        if (value == null) { //I'm the first thread!
            value = dummy;
            table.put(key, dummy); //hold the place
        }
    }
    synchronized(value) {
        if (value == dummy) { //I'm the first thread!
            value = db.getProductInfoById(key);
            table.put(key, value); //replace with real
            dummy.notifyAll();
        }
        else if (value.equals(dummy)) { //I'm NOT the first one
            value.wait(); //wait on the dummy value
            return (ProductInfo)table.get(key);
        }
        else { //the value is already cached
            return (ProductInfo)value;
        }
    }
}
```

不幸的是，上面的代码仍然隐藏着一个错误，只不过更加隐晦！这个错误就在两个 synchronized 代码块之间——注意到了吗？我们能够保证执行 table.put(key,

dummy)的线程一定会抢先执行synchronized(value)吗?不能!也许它在完成占位之后、锁定value之前,时间片刚好发生轮转,被另一个线程抢先执行了synchronized(value)!如果掉进这个陷阱,那么结果将是致命性的一一死锁。这种bug如果编程时不注意,是极难发现的,它甚至能够通过绝大多数的测试,是危险的不定时炸弹。

版本4:巧用TreeMap,鱼与熊掌兼得

经过上面的一系列尝试我们发现,既要保证数据的一致性,又要保证良好的并发度,同时还要保证不会出现死锁,确实不是一件易事。

新方法的核心思路是:我们不是锁住productCache中的value,而是通过锁住它的key来实现加锁。因为即使我们需要查询数据库来获取value,key也可以保持不变。加锁规则如下:假设所求产品的编号为id,它在productCache.keySet()中对应的对象为key,则

(1). 在从productCache中取得结果之前,必须先取得key的锁;

(2). 在执行db.getProductInfoById(id)期间,key将被一直锁住。

不难看出,这两条规则保证了在针对id的数据库查询操作执行完成之前,所有访问id的其它线程都会被阻塞(因为这些线程必然会查询productCache;根据规则(1),它们将被阻塞直到成功取得key的锁)。

规则(1)是利用Comparable接口来实现的,这也正是使用TreeMap的原因所在。它的原理是,TreeMap在查询时,会调用其keySet()中各个入口的compareTo()方法,与参数进行比较(HashMap等不具有这种特性),而我们正好可以利用这个机会来取得key的锁。下面是SyncLong类的实现:

```
class SyncLong implements Comparable {
    long value;
    public SyncLong(long value) {
        this.value = value;
    }
    public void block() {
        synchronized(this){ //try to lock the key itself!!!
        }
    }
    public int compareTo(Object obj) {
        long anotherVal = ((SyncLong)obj).value;
        if (anotherVal == value)((SyncLong)obj).block();
        return (value<anotherVal ? -1 :
                (value==anotherVal ? 0 : 1));
    }
}
```

在SyncLong.compareTo()方法中,如果判断两个key相等,就会执行其中一个key的block()方法,该方法试图取得其自身的锁。如果它已经被锁住的话,当前线程就会被自动阻塞,直到成功取得其自身的锁为止。

规则(2)则是直接在CacheSample.getProductById方法中实现的,我们使用了SyncLong来作为TreeMap中使用的key(为简单起见,这里没有考虑数据库查询失败或结果为空的情况):

```
TreeMap productCache = new TreeMap();
public ProductInfo getProductInfoById(long id) {
    SyncLong key = new SyncLong(id);
    Object obj = null;
    ProductInfo ret = null;
    synchronized(key) {
        synchronized(productCache) {
            //containsKey()implicitly invokes SyncLong.compareTo()
            if (productCache.containsKey(key)){
                return (ProductInfo)productCache
                    .get(key);
            }else{
                productCache.put(key,
                    null); //hold the place
            }
        }
        ret = db.getProductById(id);
        synchronized(productCache) {
            productCache.put(key,ret); //set the real value
        }
    }
    return ret;
}
```

在if(productCache.containsKey(key))这行代码中,SyncLong.compareTo()将会被隐含地调用,从而完成了规则(1)的执行。另一方面,费时的数据库查询操作db.getProductById(id)正好在synchronized(productCache)之外、synchronized(key)之内执行,实现了规则(2)的要求。通过对id而不是value本身加锁,我们既解决了丢失修改问题,又保证了并发度,达到了两全其美的效果!

结束语

利用TreeMap的特性,我们能够巧妙地实现小粒度的加锁算法,从而避开独占式访问带来的弊端。我们的持久层Cache在采用这种算法实现之后,在最新的压力测试中,它显示出了将近1倍的性能提升,应该说是相当惊人的。👉

(本文源代码请从csdn杂志频道下载。)

■ 责任编辑:欧阳璟 (ouyangjing@csdn.net)

趣谈 (上)

Functional Programming

■ 文 / 恶魔

本文并非是一个系统性的FP教程，仅仅是揭示了FP背后的一些数学道理，介绍与之相关的一些有趣话题，我相信您一定能从文章中有所收获。本文的例子主要使用Python解说。不过即使您不懂Python，稍微耐心一些也可以看懂。

虽然FP的历史与Fortran一样悠久，但以LISP为代表的Functional Programming一直以来只是被视为学术性的编程风格，而为工业界所拒斥。然而随着主流编程环境动态性的逐步提升，不但具有FP能力的Python等现代语言开始流行，而且Java、C#和C++等传统类型的开发工具，借助泛型等特性，也可以一定程度上实现FP，不管你是否以为然，FP的一些思想和模式已经开始渗入日常编程中。FP的一个核心思想，就是只关心做什么，不关心怎么做，具体的程序逻辑在函数的重重组合中自然实现，其背后有深刻的数学思想，而在现实中亦有其积极意义。用FP方式编写程序，灵活性高、可靠性好，代码短小精悍，便于维护。但是掌握FP思想，确实是一件挺不容易的事。

高阶函数

考虑下面的三个算式，第一个算式计算从a到b的所有自然数之和：

```
2 def sum_integers(a,b):
3     if a>b then:
4         return 0
5     else:
6         a+sum_integers(a+1,b)
```

第二个算式计算从a到b的每个自然数的立方和：

```
2 # 定义立方算式
3 def cube(x):
```

```
4         return x*x*x
5
6 def sum_cubes(a,b):
7     if a>b then:
8         return 0
9     else:
10        cube(a)+sum_cubes(a+1,b)
```

第三个算式，我们来计算一个级数展开式。这是一个计算 $\pi/8$ 的莱布尼兹展开式：

$$\frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \dots$$

```
2 def pi_sum(a,b):
3     if a>b then:
4         return 0
5     else:
6         1.0/((a+2)*a)+pi_sum(a+4,b)
```

很明显，这三个算式内部具有相同的模式。他们大部分是相同的。他们不同之处仅仅在于，每个函数的被加数不同，每次步进的长度不同，当然还有他们的名字不同。是不是已经闻到了badsmell了？让我们来refactoring。根据前面代码可以生成这样一个模板：

```
2 def <name>(a,b):
3     if (a>b)
4         return 0
5     else
6         <term>(a)+<name>(<next>(a),b)
```

这种抽象模式应该很简单，不用多说。不过实际上在数学中这种抽象表达早就存在了，那就是我们非常熟悉的Sigma标记。例如：

$$\sum_{n=a}^b f(n) = f(a) + \dots + f(b)$$

Sigma 标记的威力在于它允许数学家们处理和运算这个概念本身，而不仅仅是具体的加法运算。

同样，对于程序设计者来说，抽象这样的运算也是情理之中的事情。

```

2  def sum(term,a,next,b):
3      if a>b then:
4          return 0
5      else:
6          term(a)+sum(term,next(a),next,b)
7  def inc(n):
8      return n+1
9
10 def sum_cubes(a,b):
11     sum(cube,a,inc,b)
12
13 def sum_integers(a,b):
14     sum(lambda x:x,a,inc,b)
15
16 def psum(a,b):
17     sum(lambda x:1.0/((x+2)*x),a,lambda y:y+4,
18     b)

```

sum 这个函数就称为一个 High order Function。他能够将各种不同的函数连接起来形成一个新算法。我们可以用这个 sum 干很多事情，例如我们可以轻而易举的算出一个函数从区间[a,b]的积分。我们可以用积分逐项展开式来求解：

$$\int_a^b f = \left[f\left(a + \frac{dx}{2}\right) + f\left(a + dx + \frac{dx}{2}\right) + f\left(a + 2dx + \frac{dx}{2}\right) + \dots \right] dx$$

```

2  def integral (f,a,b,dx):
3      dx*(sum(f,a+dx/2.0,lambda x:x+dx,b)

```

我们可以看到，如果换成 C/C++ 语言，我们或许会用函数指针，但是函数指针与生俱来就有很多缺陷，否则 boost 库也没有必要费尽心机做出那么丑陋的 Functional 库。相比之下 Java 的 interface 要比函数指针干净些。但是与函数式语言相比，他的方法也只是二等公民，写出来的代码同样非常累赘不堪。

这是我修改的一个例子：

```

2  class sum
3  {
4  private:
5      lterm term;
6      lnext next;
7      int a;
8      int b;

```

```

9  public:
10     sum(lterm t,int a,lnext n,int b)
11     {
12         term=t;
13         next=n;
14         a=a;
15         b=b;
16     }
17     int do_sum()
18     {
19         if (a>b)
20             return 0;
21         else
22         {
23             int t=a;
24             a=n, donext(a);
25             t, doterm(t)+do_sum();
26         }
27     }
28 }

```

从抽象的效果来说，他们当然是相同的，但语法上实在太累赘。High order Function 最为擅长的就是抽象算法或者业务逻辑。然而在 Java/C++ 中承载逻辑的最基本单元函数却不是第一等的公民，只能负载在类上进行传递。因此对于 OO 来说它的确不善于抽象逻辑和算法。以前 STL 之父说过，不是所有的东西都是对象，算法就不是对象。STL 中的用 functor 抽象算法和 FP 有些像，不过只是东施效颦罢了。FP 并不擅长于运算，而擅长于逻辑推导。这种推导，在数学上称为算子空间。什么叫算子和算子空间？我们从数学中知道 $f(x)$ ， x 有定义域， $f(x)$ 有值域。而我们把 x 的定义域的概念进行扩展，我们把每个函数看成一个集合中的元素。然后将所有算子的集合抽象为空间或者函数空间（也就是说定义域这个集合中的每个元素都是一个函数）。例如上面的各种各样的 term 和 next 就是一系列算子，他们的集合就叫做抽象空间。一个算子可以看作这个空间里面的一个点，算子的组合即可以看作函数空间中的向量组合。在数学里我们空间的两个点表示 $a:[x1,y1,z1]$ ， $b:[x2,y2,z2]$ 。他们的组合就成为一个线段，如果有这个线段产生方向就成为一个 $a \rightarrow b$ 的向量。因为算子和算子之间也是有方向的，最简单就是运算顺序。那么算子和算子之间就是算子向量。由于算子所作用的集合并没有维数的限制，在函数空间基础上所研究出来的算子的各种性质，成为处理用有穷维或者线性空间逼近无穷维集合问题的有力武器。作为函数式语言背后的数学基础，Lambda 演算和范畴论 (Category Theory) 中就大量使用算子，算子向量来解决很多问题。当然这些问题就非常艰深了，我们现在可以看到算子的一个非常直接的功效就是算子空间本身

具有的性质——即N维正交。也就是说每个算子都是独立互不干扰的。他不会有OO那样牵扯不清的各种问题。什么叫正交？

我们来看一个Java例子：

```

2  class A
3  {
4      int t;
5      A()
6      {
7          t=100;
8      }
9      void dec()
10     {
11         System.out.pring(t--);
12     }
13     void inc()
14     {
15         System.out.pring(t++);
16     }
17 }

```

在这个例子中，dec, inc 就不是正交的。Dec 的运算会影响到Inc的运算。而函数式语言中的算子，类似数学中的函数。函数式语言里函数的所有变量不允许进行再绑定。在函数式语言中一个变量x 当它被赋予一个值1 以后它永远就是1。这样的x 就被称为是immutable 的。相反，现在大多数流行的OO 语言里的x 却是mutable 的，那么当你执行OO 语言中的某个函数A(x,y)。其x,y 会受到某个“遥远”变量的影响而使得A的结果出现一种你未曾预料到的情况。这被称为“side-effection”，即边界效应。而函数式语言中A(1, 2)的结果是不会改变的，有什么样的输入一定会有什么样的输出，因此他是“no side-effection”。“no side-effection”会让程序的Bug 更少，使程序更加健壮一些。

介绍完了High Order Function，我们接下去介绍更加有趣的Combinator。

Combinator

“Pascal是为了建造金字塔，Lisp是为了建造有机体，金字塔矗立在那里千年不变，而有机体则必须演化，否则就会消亡。”

本节开头引用的这段话出自非常有名的MIT.6001《计算机程序构造与解释》。这是一本非常令人惊奇的教科书，我想凡是从事计算机工作的朋友都应该读读这本书。这句话说出了函数式语言和普通的OO 语言最大不

同。或许大家认为这些话似乎有些夸张而且不可理解。那么现在请睁大眼睛。

在这节中我将介绍Monad Combinator。好，我们现在先不管那些神神道道的东西。我们先来看个Case，我们可爱的多利羊是个克隆羊，他只有母亲没有父亲。如果我们为一只羊建立一个族谱，也就是说建立一棵二叉树，如下：

```

2          Sheep
3      mother      father
4  mother father  mother father
5  ...

```

这个很简单没有什么可以多说的。

那么我们可以建立这样一个数据结构：

```

2  class Sheep;
3      def __init__(self, name, mother, father);
4          self.name=name
5          self.mother=mother
6          self.father=father
7      if __name__=="__main__":
8          adam=Sheep("adam", None, None)
9          eve =Sheep("eve", None, None)
10         uranus=Sheep("uranus", None, None)
11         gaea=Sheep("gaea", None, None)
12         kronos=Sheep("kronos", gaea, uranus)
13         holly=Sheep("Holly", eve, adam)
14         roger=Sheep("Roger", eve, kronos)
15         molly=Sheep("Molly", holly, roger)
16         dolly=Sheep("Dolly", molly, None)

```

现在我们需要根据一个给定的路径找到这头羊的特定祖先的名字。例如我们给molly->mother->father->father这样一个特定的路径。怎么找出它妈妈的爸爸的爸爸的名字？

当然首先想到的就是我们OO中的‘.’操作，molly.mother.father.father。不过可惜我们有了克隆技术，如果当中某个Ancestor是None的话那么就会出现exception。而且molly.mother.father.father，这种东西不容易定制。那么我们现在就用FP的有机体的思考方式来解决一下这个问题：先不管具体的实现，从最基本的地方开始，我们设计两个函数：

```

2  def mother(sheep);
3      return sheep.mother
4  def father(sheep);
5      return sheep.father

```

这太简单了！我们可以这样想：对于molly.mother.father.father最愚蠢的一种实现是什么？


```

2  if sheep == None:
3      return None
4  else:
5      sheep1=mother (sheep)
6      if sheep1==None:
7          return None
8      else:
9          sheep2=father(sheep1)
10         if sheep2==None:
11             return None
12         else:
13             sheep3=father(sheep2)
14             if(sheep3==None):
15                 return None
16             else:
17                 return Sheep3

```

这个实现够愚蠢吧？但是你可别笑，无论你用什么样的算法转译成机器代码都是这个样子。那么既然这个算法很badsmell，那么再来refactoring一下，我们可以看到：

```

2  if xx==None:
3      return None
4  else
5      function (xx)

```

这是这段实现最基本的材料，有点像细胞一样。我们称为Monad，我们可以抽象出这样的函数：

```
2 a->(a->Maybe b)->Maybe b
```

这里采用了Haskell的描述，因为Haskell的函数类型描述准则是用Haskell Curry(一个数学家)表示，所以表达起来就比较方便。这段是什么意思？也就说一个函数接受两个参数。第一个参数类型为a；第二个参数的类型是一个function。这个function本身接收一个a的参数，并且返回一个值。这个值是一个Maybe类型也就是Maybe b=Nothing| Just b。意思是这个类型要么是b，要么是Nothing。当然在Python中很简单，每个变量都是一个Maybe类型，要么有具体的值要么只有None。Ok接收了a和这个function后，我们返回另外一个Maybe b类型。怎么样？是不是觉得这种表述和上面那段判断代码很像？我们来看看这个Monad如何表示成函数：

```

2  def Combinator(sheep,function):
3      if(sheep==None):
4          return None
5      else:
6          return function(sheep)

```

在这里，sheep就是a，function就是a->Maybe b。这

里的function可以father或者mother，他们接受一个sheep有可能返回一个Sheep，有可能返回一个None。那么整个Combinator的函数的返回值不是None就是一个Sheep也就是一个Maybe b。我们现在再来写一个愚蠢的代码：

```
2 Combinator(Combinator(Combinator(molly,mother),father),father)
```

用逻辑推导一下，它是否就是molly.mother.father.father？当然写这么长的代码的确有些badsmell，那么再来Refactoring这段代码的抽象过程。把molly.mother进行运算然后把结果和father再进行运算，这个模式可以一直继续下去。那么我们可以将这个模式抽象成一个函数。当然这个函数不用我们写了，Python中已经有现成的函数，那就是reduce：reduce(function,[...])。这个函数接受一个函数和一个list作为参数，它首先把list的第一第二个元素用function进行运算，然后把运算结果和第三个元素进行运算，以此类推，例如：

```
2 reduce(lambda x,y:x+y,[1,2,3,4,5])
```

这就是计算从1到5的和。

那么我们就可以将上面那个愚蠢的代码写成这样：

```

2  class Sheep:
3      def __init__(self,name,mother,father):
4          self.name=name
5          self.mother=mother
6          self.father=father
7
8  def mother(sheep):
9      return sheep.mother
10 def father(sheep):
11     return sheep.father
12
13 def Combinator(sheep,function):
14     if(sheep==None):
15         return None
16     else:
17         return function(sheep)
18
19 def gotAncestor(pedigree):
20     return reduce(Combinator,pedigree)
21
22 if __name__=="__main__":
23     adam=Sheep("adam",None,None)
24     eve =Sheep("eve",None,None)
25     uranus=Sheep("uranus",None,None)
26     gaea=Sheep("gaea",None,None)
27     kronos=Sheep("kronos",gaea,uranus)
28     holly=Sheep("Holly",eve,adam)
29     roger=Sheep("Roger",eve,kronos)
30     molly=Sheep("Molly",holly,roger)
31     dolly=Sheep("Dolly",molly,None)
32     test=[molly,mother,father]

```



```

33     print gotAncestor(test).name
34     test=[dolly,father,father]
35     ret=gotAncestor(test)
36     if ret==None:
37         print "None"

```

看到了么? 这就是FP的惊奇所在, 我们之所以把Combinator称为Monad, 因为mother, father只是一些代码的片断, 就像是DNA的ADTG的四个基因一样, 当然这个算式基因只有2个。Combinator就像是规定了基因的排列顺序, 组合成一个单体的细胞。通过reduce又像细胞的无丝分裂那样分裂出其他的细胞, 最后得到一个非常简单的有机体。我也写一个传统的代码:

```

2  def getAncestor2 (child,pedigree,generation):
3      if pedigree[generation]==None:
4          return child
5      else:
6          if(pedigree[generation]=="mother"):
7              return getAncestor2(mother(child),pedigree
8                                  ,generation+1)
9          else:
10             return getAncestor2(father(child),pedigree
11                                 ,generation+1)
12
13 if __name__=="__main__":
14     getAncestor2 (molly,["mother","father",
15                           ,"father"],0)

```

这样的代码就像是金字塔, 从一开始我们就把整个算法规划好了, 这个代码不能做任何的改动。与上面的有机体是截然两种味道。一旦我们要refactoring这个代码, 我们可能就要做更多的工作, 而且有可能是推倒重来。而Functional语言不一样, 他所有的代码都是一小片的片断: 首先这些小片段的改动将比改动整个金字塔来的更加容易, 其次这些小片断天生就是独立正交不相互干扰。如果修改了mother, 不会干扰father, 而OO要做到这样就要动用接口, 模式这种非常笨重的方法。如果将Combinator用Java实现, 你会发现你的代码很“华丽”。所以XP和refactoring对于FP来说是最自然不过的事情, FP与OO/impervative相比XP和refaction的成本要低得多。其实这个Combinator是一个非常有用的Monad, 不仅能算羊的祖先(这只是最简单的Monad, 我们还有其他更加强大的Monad), 还可以用在Hash表或者DataBaseQuery里。举个小例子: 如果我有一个数据库表形态如下:

```

2  ID,name,Age,Gender,Position


```

假设我们现在还没有发明SQL语句。我们要来查询年龄>20, Gender是男性的, Positon为Manager的人员怎么做呢?

```

2  class employee:
3      def __init__(self, ID,name,Age,Gender,Position):
4          self.ID=ID
5          self.Name=name
6          self.Age=Age
7          self.Gender=Gender
8          self.Position=Position
9
10 def checkAge(Emp):
11     if(Emp.Age<20):
12         return None
13     else:
14         return Emp
15
16 def checkGender(Emp):
17     if(Emp.Gender!="male"):
18         return None
19     else:
20         return Emp
21
22 def checkPosition(Emp):
23     if(Emp.Position!="manager"):
24         return None
25     else:
26         return Emp
27
28 def Query(Rowset, QuerySet):
29     map(lambda record:reduce(Combinator,record
30                             ,insert(record,-1)), Rowset)
31
32 rowset=[.....]#构造数据
33 Query (rowset,[ checkAge, checkGender,checkPosition])

```

先说明一下: map和reduce类也是一种combinator, 它接收一个函数, 一个列表, 把列表的每个元素送入函数进行运算, 然后把结果组合成一个新的列表。这个代码可能大家闻上去有很多的badsmell, 不过没有关系, Functional语言的工作方法就是一开始写最臭不可闻的代码, 然后慢慢地归纳算法的规律, 慢慢演化。我们先不管这些badsmell, 关注一下Query(rowset,[checkAge, checkGender, checkPosition])这段代码, 你会发现, [checkAge, checkGender, checkPosition], 这个列表和我们即将要发明的SQL语言是多么的相似! 只要我们能想办法使CheckAge可提供参数(其实这个非常简单, 大家可以慢慢琢磨), 那么这个Query就是一个最基本的SQL雏形了。其实我们经常使用的SQL语言就是一种Functional语言。所以大家没有必要对Functional语言有任何的陌生感, 我们早就在使用Functional语言进行程序的编写了, 只是我们不知道而已。其实我们身边的Functional语言还有很多, 例如使用非常广泛的XSLT就是一种Functional语言。下期杂志我们将介绍一个非常有趣的Combinator: Y Combinator。 

■ 责任编辑: 欧阳璟 (ouyangjing@csdn.net)

任何软件产品都有最擅长解决的问题，如何围绕核心问题安排软件的功能，减少用户在软件使用中遇到的麻烦是软件产品最重要的设计原则。

这里的麻烦，不仅指软件运行中可能出现的异常情况，还包括不必要的功能，过分花哨或者过分单调的界面，拥挤的控件摆放等。本文将通过分析 Skype 这个软件的产品设计的优缺点向大家解释软件产品设计的步骤和原则。

Skype 是最近出现的一款 IP 电话软件。在此以前的 6 年里，我尝试过不下 7 种 IP 电话，只有 Skype 用起来最顺手，没有磕磕绊绊，不需要阅读手册，也没有“机会”到处寻找一个按钮，一切都非常自然。

原则 1：如何围绕核心问题安排软件的功能，减少用户在软件使用中遇到的麻烦，是软件产品最重要的设计原则。

现在的大多数软件都提供可以从互联网上下载的试用版。从用户决定希望试用这一刻起，软件厂商必须经历一场严格的淘汰赛，用户在任何一步感到不满意都不会再进行到下一步。

网站设计



图 1：Skype 网站首页

整个过程的第一步就是设计一个能够让用户非常容易找到软件并且非常方便就可以下载的网站。这个目标虽然看起来容



作者简介：董洵是一家软件公司的 .NET 架构师，负责设计了 508Wizard.NET 版本的所有功能 (<http://www.508Wizard.com/>)，同时还创建了北京 .NET 用户协会 (<http://www.mscommunity.com/UserGroup/>)。

他山之石 可以攻玉

——软件产品设计实践

■ 文 / 董洵

软件作为一种商品，开发技术只是导致其成功的诸多因素中的一个，更重要的是产品的设计与规划。对于软件开发人员来说，虽然产品设计远没有软件开发复杂，但是缺乏对产品设计的理解很容易导致自己呕心沥血做出来的产品不被用户和管理人员接受。

在这期专栏中，让我们将注意力转移到软件产品的设计上，从开发人员的角度介绍软件产品设计的原则和一些常见的问题。

易，但是仍然有很多软件产品的下载网站实在令人难以满意。常见的问题是：

- ◆ 域名和软件产品名不一致。典型反例就是 CuteFTP，这个大名鼎鼎的软件属于一个几乎不为人知的公司 GlobalScope，也许出于推销公司的目的，无论在网站还是开始菜单上都找不到 CuteFTP 的名字，很多人因此埋怨 CuteFTP 没有开始菜单中添加启动项。比较推荐的方法是 1) 直接注册产品名为网址，例如 <http://www.skype.com/>。这样的好处是当有人听到产品名字的时候，第一反应必然是下载一份，而首先要去的就是以这个软件名命名的网址；2) 以产品名注册一个 Google AdWord，这样当潜在的客户通过 Google 搜索软件的时候，你的网址可以立刻显示出来。

- ◆ 没有对软件产品明确的定位描述。正如前面提到的，所有产

品都有一个中心定位,因此在互联网上没有任何特点的软件是没有竞争力的。图1中 Skype 就在网站的首页明确标明这个软件的优势为“Skype is Free Internet Telephony that Just Works.”。这句宣传中突出了 Skype 的两大特点:免费和易用性。

- ◆ 没有明显的软件下载标志。经过统计,大约有15%左右的软件下载地址是无效连接,还有35%的软件在下载地址无法很快找到。大部分用户从来都是只阅

原则2:评价软件产品设计的唯一标准是最终用户是否能在无需帮助的情况下使用所需功能。

读动画,图片,大写黑体,很少有人能够有足够的耐心把网站首页上所有的字读完,因此必须非常明确的提供下载链接。从图1中可以看出 Skype 在顶部使用了一个与整个页面的红色有强烈反差的绿色框标明“Download Now(It's Free)”。同时这个绿色框被修饰成按钮型,给人非常直观的感觉。最坏的链接莫过于哪怕用户找到了仍然以为自己“又”找错了了。

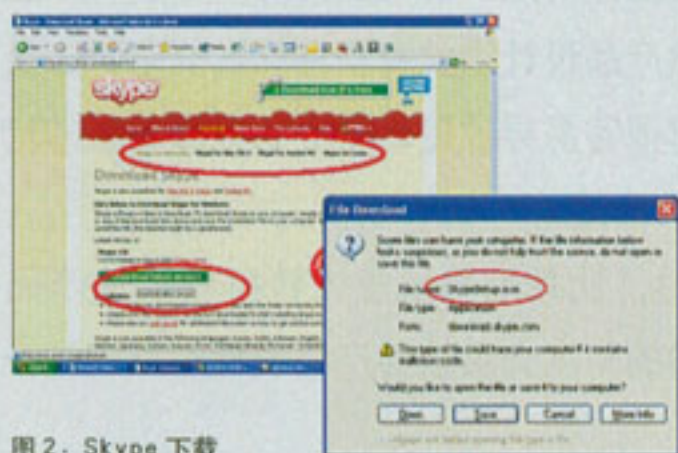


图2: Skype 下载

- ◆ 软件下载版本繁杂。很多软件都要支持多个平台,网站也会提供多个版本供人下载。这对于试用者来说是非常糟糕的事情,因为试用者被强迫停下来仔细阅读你的文字才能确定应该选择哪个版本。在 Skype 下载的网页仍然选择用亮绿色标明“Download latest version”,同时在网页首部给出其他操作系统版本的下载链接,一目了然。

- ◆ 软件包文件名不利于记忆。软件下载的文件名必须是直接明了的。Skype 的文件名为 SkypeSetup.exe,应该没有任何其他的文件名能比这个更直接的了。我个人反对在安装包的文件名上标注版本号,例如 SkypeV10.exe 这样的名字,因为这样对于用户增加了无

谓的记忆负担,还同时让用户迷惑,不知道这到底是一个应用程序还是一个安装程序。

安装

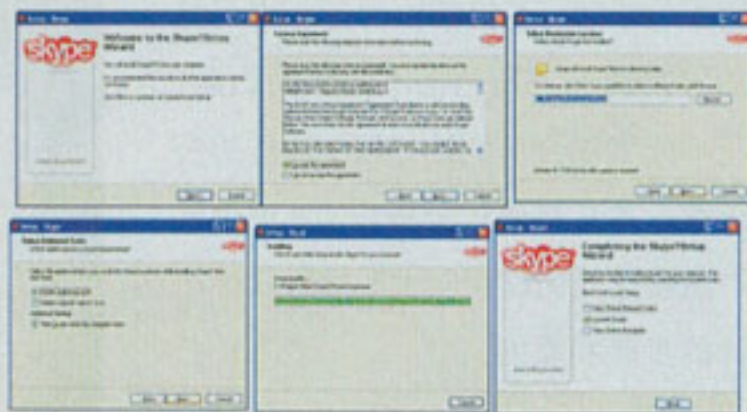


图3: Skype 安装

在用户成功下载完毕后,下面一步要进行的是软件安装。对于软件安装最需要注意的是测试、测试、再测试。软件安装时出现任何问题,无论出于什么原因,都会给用户软件品质不高的第一印象,这往往是致命的。任何一个正式版本的发布都需要给安装包测试留下至少一周的时间,如果软件需要支持的操作系统比较多,就应该给出更充分的时间进行测试。Skype 的安装共有六步,非常简练,没有任何可以去掉而不会影响用户的功能。

原则3:对于软件安装最需要注意的是测试、测试、再测试。

下面是软件安装设计需要注意的一些要点:

- ◆ 在安装时尽量避免用户设置,如果有可能就选择大部分用户可以接受的默认设置,把对软件定制的任务放在软件设置中进行。最理想的状态是80%的用户只需要连续点击“下一步”就可以完成所有的安装操作。
- ◆ 如果软件有任何需要注册到操作系统的功能,一定要在干净的环境中进行测试。常见的注册包括 COM/ActiveX 注册,修改注册表以及将 .NET Assembly 注册成为 COM/Automation 对象。推荐的方法是使用 VirtualPC 或 VMWare 针对每种需要支持的操作系统制作一个干净的虚拟机,在这个环境测试后将虚拟机重新恢复到初始状态。
- ◆ 在安装的最后一步提供“运行该程序”的选择框,默认选中。这样就可以免去用户到开始菜单中寻找的麻烦。

原则4: 确定任何功能是否必须的判断标准是去掉这个功能后, 是否会给用户带来使用上的麻烦。

配置

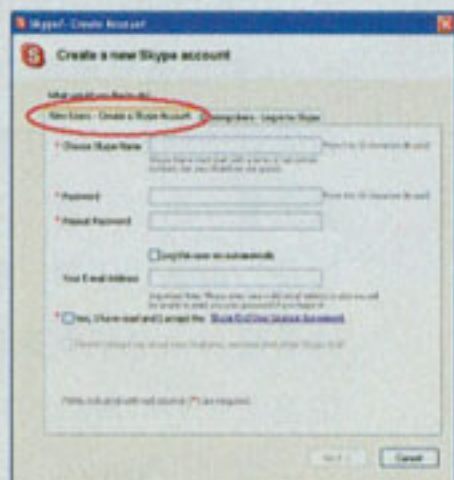


图 4: Skype 添加新账号

除非必要, 否则应该尽量避免在软件安装后还需要用户进行额外的设置才能使用软件。因为对于一般用户来说, 安装、配置都是复杂、容易出错的高级操作。

对于类似于 Skype 这样必须进行设置才能使用的软件, 应该尽量减少用户必须输入的信息量。Skype 只需要输入用户名、密码和电子邮件就可以了, 至于个人信息、联系方式、昵称之类的信息完全可以等到软件运行起来后再输入。

原则 5: 尽量避免软件安装后还需要用户进行额外的设置才能使用。

使用

用户通过前面所有的步骤, 终于进入到应用程序的主界面。

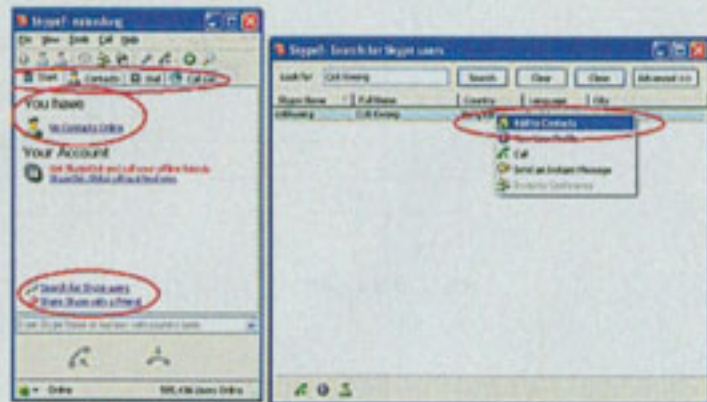


图 5: Skype 主界面

◆ 对于任何启动时间超过 1.5 秒的软件都需要提供一个开始画面; 对于启动时间超过 5 秒的, 推荐在启动画面上显示进度信息。

◆ 对于新安装的软件, 必然需要进行的操作是各种添加操作: 如添加联系人, 添加账户, 添加分类, 添

加记录。如何方便、直接的进行必要的添加操作, 会决定用户对软件本身的第一印象。通常的做法是在工具栏、系统菜单或者右键菜单放置相关操作。推荐的做法是在软件界面最明显的位置放置这些功能。



图 6: Skype 拨叫用户界面

◆ 在软件设置完成后, 最主要的任务就是尽量简化常规操作。越简单的常规操作, 越能够得到用户的认可。最理想的情况是所有常规操作都只需要一步操作即可。对于 Skype 来说, 最常进行的操作就是打电话和接听电话。在 Skype 中拨叫任何用户的操作只有一步: 在用户列表中双击希望呼叫的用户名, 就会出现图 6 的拨叫画面; 挂断电话的操作也只有一步: 点击屏幕下方红色的放下电话标志; 接听电话的操作仍然只有一步, 点击屏幕下方接听电话的绿色标志就可以接听。

原则 6: 越简单的常规操作, 越能够得到用户的认可。最理想的情况是所有常规操作都只需要一步。

问题

Skype 产品最大的问题在于没有在服务器端为客户保留一份通讯列表。当我从笔记本切换到家里电脑的时候, 必须把所有的联络人全部重新添加一次。

原则 7: 软件必须提供方便透明的迁移服务, 保存用户以往的个人设置, 外观喜好。

总结

“成功的软件是类似的, 而失败的软件却各不相同”。本文通过对 Skype 从下载到使用全过程的分析, 希望能够让您更加形象的理解软件产品设计针对的阶段和相关原则, 提高您对产品易用性设计的关注。

■ 责任编辑: 欧阳瑾 (ouyangjing@csdn.net)

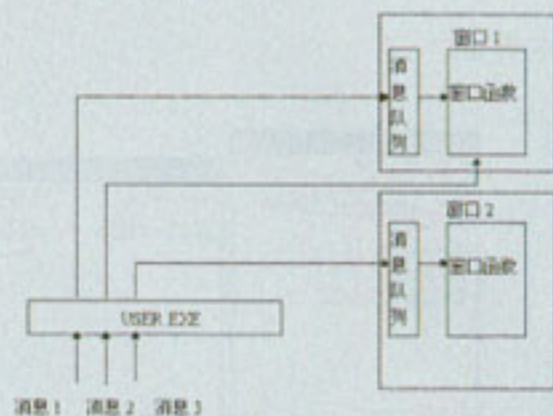
在 .NET Framework 中, `System.Windows.Forms.dll` 承担了 Windows 操作系统平台上的图形用户接口部分的工作(也就是我们常说的“窗口”)。它的主要内容是 .NET 控件和 WinForm (也可以将它看成一个特殊的 .NET 控件)。从操作系统的角度来看, `System.`

也是理解 Windows 窗口管理的基础。对于 .NET 开发者来说, Windows 的窗口消息机制的概念也同样重要, 会经常接触, 尤其是在进行一些“高级”的程序设计时, 不可避免的会碰到此类问题, 例如当你要在 `Control.`

`WndProc()` 或是 `Form.WndProc()` 内写一些代码的时候, 就会涉及窗口消息的概念。另一方面, 从程序设计的角度来看, Windows 窗口消息机制是 Windows 窗口管理的核心。站在窗口编程的角度来看, 语言环境、编程平台都不过是一种表达工具, 最终都是要围绕着窗口消息机制来做文章。所以说搞清楚 Windows 的窗口消息机制对于在 Windows 平台上的 .NET Control 开发人员来说, 是一件完全必要的事情。

Windows 的窗口消息机制

简单回顾: Windows 系统的窗口管理部分是一个典型的消息驱动系统, 也就是说它是依靠消息来协调窗口系统各部分协同工作。例如在屏幕上拖动一个窗口, 会引起其它窗口的重画。整个动作的完成是依靠各个窗口的处理程序通过接收、发送消息协调工作, 一起完成的。如图所示:



上面的图示只是帮助理解, 不够精确。消息的处理过程准确地说是这样的:

消息是由系统或者应用程序生成, 分为 `Queued Message` 和 `Non-Queued Message` 两种类型。对于 `Non-Queued Message`, 系统会直接将它传给窗口的消息处理函数。 `Queued Message` (鼠标、键盘消息多属于此类消息) 的处理过程相对复杂。首先, 它会进入系统的消息

深入.NET 控件开发

——System.Windows.Forms 中的 Windows 窗口消息机制

■ 文 / 唐泉

如果从 Windows 窗口管理的角度来看, 所有控件 (Control) 都不过是一个或多个子窗口, 都是围绕着子窗口进行用户接口 (UI) 处理的构件程序。 .NET 控件也不例外。本文主要介绍 .NET 控件是如何和底层 Windows 窗口系统关联起来的, 以及在 `System.Windows.Forms` 中如何找出一些机会可以应用 Window 窗口消息机制做开发。

本文需要大家熟悉这些背景知识: Windows 的进程线程模型, 窗口模型, 消息队列, 以及它们之间的关系。网络上介绍这些知识的资料很多。也可以参考 MSDN (ms-help://MS.MSDNQTR.2004JAN.1033/winui/winui/windowsuserinterface/windowing/messagesandmessagequeues/aboutmessagesandmessagequeues.htm)。

`Windows.Forms.dll` 并不是操作系统的一部分, 也不是操作系统的扩充, 而是基于现有操作系统平台独立架构的, 完全依赖于前者。因此, `System.Windows.Forms.dll` 依靠调用传统的 Win32 API 来完成它在 .NET 平台中所承担的图形用户接口部分的工作。

对于传统的 Windows 开发者来说, 理解 Windows 的窗口消息机制是开始学习 Windows 窗口编程的第一

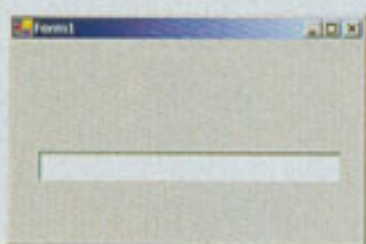


队列, 等候被分发到拥有其归属窗口的线程的消息队列中; 然后, 该线程请求系统分发它消息队列里的消息到相应的窗口处理函数; 最后, 窗口处理函数会处理所收到的消息, 同时对于自己不处理的消息, 窗口处理函数将继续把它传递给系统默认的处理函数去处理。

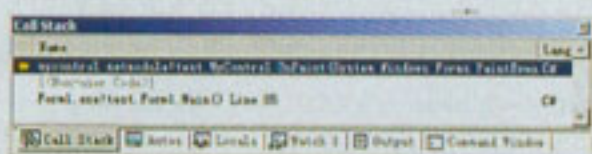
分析 .NET Control 的窗口消息处理过程

下面我们可以通过一个十分简单例子来看 .NET Control 的消息处理过程:

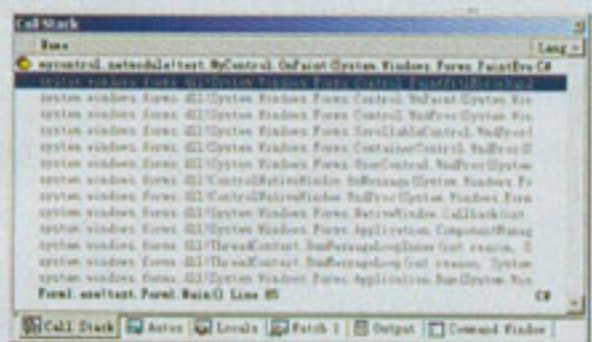
写一个 MyControl, 从 System.Windows.Forms.UserControl 派生, 覆盖 (override) 它的 OnPaint 方法。然后创建一个 Windows Form, 新建一个 MyControl 的实例 (Instance) 放在上面。界面大致如下:



然后用 Debugger 程序调试它 (大家可以选用 Visual Studio .NET 或者 .NET SDK 提供的 DbgCLR 皆可, 两者使用起来都很方便, 都有友好的图形调试界面)。调试过程如此: 在前面提到的 OnPaint 方法内设置断点; 当程序运行并中断到此处时, 查看调用函数窗口 (Call Stack Window); 这时候一般会看到这样的信息:



碰到这样的情况, 大家只需在 Call Stack 窗口内点击鼠标右键, 选择 "Show Non-user Code", 就可以看到如下信息:



从上面的图示里, 大家可以十分清楚的看到 MyControl 是如何获得机会响应 WM_PAINT 消息的。不幸的是, Microsoft 的调试器不提供反汇编的功能, 大家只能用它看到难懂的汇编代码。不过它列出了相关的类名函数名, 我们可以自己用 Reflector 软件去反编译 System.Windows.Forms.dll, 然后找到我们关心的

函数。(Reflector 是一个免费的 .NET 反编译软件; 作者 Lutz Roeder; 下载地址: <http://www.aisto.com/roeder/dotnet/>)

大家顺着 Call Stack 里的函数调用顺序, 用 Reflector 浏览一遍它们的代码。不难发现: 要想完全理解这些代码很困难, 但是看清整个消息的响应处理过程的脉络却是不难。

暂且搁下这个例子, 下面我们开始用工具软件 Reflector 查看 System.Windows.Forms.Control 的反编译代码。可以发现 Control 的窗口部分是由 internal sealed class ControlNativeWindow 完成。ControlNativeWindow 是从 System.Windows.Forms.NativeWindow 派生, 并根据 Control 的实际需要, 整理了一下消息派发, 并把一个 Control 和一个窗口 (NativeWindow) 关联起来。实际上 NativeWindow 是整个 System.Windows.Forms 的窗口基础, 负责管理窗口的整个生命周期, 从创建到销毁, 以及在其生命周期里, 窗口消息的管理和分发。从 NativeWindow 的对外使用接口上看, 这个类很简单, 事实并非如此。大家可以用 Reflector 查看这个类, 它十分复杂。对它展开全面的分析超出了本文的内容范围, 我们只关注它对窗口消息的分发和处理过程。不过有兴趣的读者可以自己进行一些研究, 搞清楚这个类可以更好地帮助了解 Windows 系统和 System.Windows.Forms 的关系。

回过头来, 继续查看 System.Windows.Forms.Control 的代码。我们可以总结出窗口消息是从 NativeWindow.WndProc(ref Message m) 开始, 到 NativeWindow.DefWndProc(ref Message m) 结束。于是大家自然会问: 这些消息从哪里“流入”到 NativeWindow.WndProc(ref Message m), 又最终从 NativeWindow.DefWndProc(ref Message m) “流”向何方?

用 Reflector 反编译 NativeWindow, 分析它的代码, 我们可以发现 NativeWindow 做了两件关键的事情: (1) 将自己的窗口消息响应入口函数注册到窗口系统; (2) 联接系统默认缺省的窗口消息响应函数。从而将自己和底层 Windows 窗口系统关联起来。这样一来, Windows 窗口系统就可以将窗口消息派发到 NativeWindow 的窗口消息响应函数; 当 NativeWindow 的窗口消息响应函数完成相应的处理, NativeWindow 会调用系统默认缺省的窗口消息响应函数, 从而完成了一个完整的消息处理链。(这里涉及到几个十分重要的 Win32 API 窗口函数, 涉及到对整篇文章的理解 RegisterClassEx, CreateWindowEx,

SetWindowLong, DefWindowProc。大家如果对他们不够了解, 请查阅 MSDN)

具体的分析过程如下:

我们用 Reflector 反编译出 NativeWindow.WindowClass.RegisterClass() 函数的 C# 源代码。发现 .NET 的窗口消息响应入口处是 NativeWindow.Callback(...) 函数, 默认缺省的消息响应函数是 User32.dll 的 DefWindowProc()。(请注意下面粗斜体并且带下划线的代码)

```
private void RegisterClass()
{
    string text1;
    WNDCLASS_1 wndclass_i1;
    bool flag1;
    int num1;
    WNDCLASS_D wndclass_d1 = new WNDCLASS_D();
    if (NativeWindow.userDefWindowProc == IntPtr.Zero)
    {
        text1 = ((Marshal.SystemDefaultCharSize == 1) ?
        "DefWindowProcA" : "DefWindowProcW");
        NativeWindow.userDefWindowProc =
        UnsafeNativeMethods.GetProcAddress(new HandleRef(null,
        UnsafeNativeMethods.GetModuleHandle("user32.dll")), text1);
        if (NativeWindow.userDefWindowProc == IntPtr.Zero)
        {
            throw new Win32Exception();
        }
    }
    if (this.className == null)
    {
        wndclass_d1.hbrBackground = UnsafeNativeMethods.
        GetStockObject(5);
        wndclass_d1.style = this.classStyle;
        this.defWindowProc = NativeWindow.
userDefWindowProc;
        this.windowClassName = string.Concat("Window.",
        Convert.ToString(this.classStyle, 16));
        this.hashCode = 0;
    }
    else
    {
        wndclass_i1 = new WNDCLASS_1();
        flag1 = UnsafeNativeMethods.GetClassInfo
        (NativeMethods.NullHandleRef, this.className, wndclass_i1);
        num1 = SafeNativeMethods.GetLastError();
        if (!flag1)
        {
            throw new Win32Exception(num1, SR.
            GetString("InvalidWndClsName"));
        }
        wndclass_d1.style = wndclass_i1.style;
        wndclass_d1.cbClsExtra = wndclass_i1.cbClsExtra;
        wndclass_d1.cbWndExtra = wndclass_i1.cbWndExtra;
        wndclass_d1.hicon = wndclass_i1.hicon;
        wndclass_d1.hCursor = wndclass_i1.hCursor;
        wndclass_d1.hbrBackground = wndclass_i1.
        hbrBackground;
        wndclass_d1.lpszMenuName = Marshal.PtrToStringAuto
        (wndclass_i1.lpszMenuName);
    }
}
```

```
this.defWindowProc = wndclass_i1.lpfnWndProc;
this.windowClassName = this.className;
this.hashCode = this.className.GetHashCode();
}
string[] textArray1 = new string[5];
textArray1[0] = Application.WindowsFormsVersion;
textArray1[1] = ".";
textArray1[2] = this.windowClassName;
textArray1[3] = ".app";
textArray1[4] = Convert.ToString(AppDomain.CurrentDomain.
GetHashCode(), 16);
this.windowClassName = string.Concat(textArray1);
this.windowProc = new WndProc(this.Callback);
wndclass_d1.lpfnWndProc = this.windowProc;
wndclass_d1.hinstance = UnsafeNativeMethods.GetModuleHandle
(null);
wndclass_d1.lpszClassName = this.windowClassName;
if (UnsafeNativeMethods.RegisterClass(wndclass_d1) == IntPtr.
Zero)
{
    this.windowProc = null;
    throw new Win32Exception();
}
this.registered = true;
}
```

NativeWindow.WindowClass.Callback(...) 的反编译代
码如下:

```
internal NativeWindow targetWindow;

public IntPtr Callback(IntPtr hWnd, int msg, IntPtr wParam, IntPtr
lparam)
{
    try
    {
        UnsafeNativeMethods.SetWindowLong(new HandleRef
(null, hWnd), -4, new HandleRef(this, this.defWindowProc));
        this.targetWindow.AssignHandle(hWnd);
        return this.targetWindow.Callback(hWnd, msg,
wparam, lparam);
    }
    catch (Exception)
    {
    }
    return IntPtr.Zero;
}
```

NativeWindow.Callback(...) 的反编译代码如下:

```
private IntPtr Callback(IntPtr hWnd, int msg, IntPtr wParam, IntPtr
lparam)
{
    Message message1 = Message.Create(hWnd, msg, wParam,
lparam);
    try
    {
        this.WndProc(ref message1);
    }
    catch (Exception exception1)
    {
        this.OnThreadException(exception1);
    }
}
```



```

    if (msg == 130)
    {
        this.ReleaseHandle(false);
    }
    return message1.Result;
}

```

这样一来,对于熟悉 Win32 API 的开发者,事情就很清楚了,可以说是一目了然。

现在我们再结合前面的例程,可以归纳出.NET Control 内的窗口消息流程是这样的:

```

NativeWindow.Callback-->ControlNativeWindow.WndProc-->
ControlNativeWindow.OnMessage-->Control.WndProc-->Control.
DefWndProc-->NativeWindow.DefWndProc

```

应用介绍及经验总结

于是,我们可以总结出 System.Windows.Forms 至少提供了这么几个地方可以做文章:

```
Application.AddMessageFilter(IMessageFilter msgFilter)
```

(注:虽然上面没有讨论过这个函数,读者可以自己用 Reflector 去看,它提供了一个机会,让程序可以查看线程的消息队列,也就是说在 Queued 消息被送入窗口的消息处理函数之前)

```

virtual NativeWindow.WndProc(ref Message m)
virtual Control.WndProc(ref Message m)
virtual Control.DefWndProc(ref Message m)
virtual NativeWindow.DefWndProc(ref Message m)

```

也许有人会问,了解学习以上的知识对大家的实际开发工作会有什么帮助?这就好比医学中的人体解剖学,了解学习它虽然不能使你立刻成为医生,开始诊治病人,但至少让你对“人”有了一个比较专业的背景认识。上面谈到的 Windows 消息机制也就是类似性质的知识,可以帮助大家对 Control 的运行有比较深刻地理解。在此基础上,才有可能对 Control 进行庖丁解牛般地开发。

例如,在一些开发中,需要快速重画 Control 的显示内容。为了提高性能,常常需要利用 GDI 来做这项工作而不是 GDI+, 因为其速度较慢。于是我们就很有必要去了解 WM_PAINT 消息;如果在一个标准的 TextBox 上想再添画一点什么东西,就还需要了解 Control 是如何处理 WM_PAINT 消息的,从而能够找出一个适当的时机完成我们想要的效果,例如,覆盖(Override) TextBox 的


OnPaint 方法,或者覆盖(Override) TextBox 的 WndProc 或 DefWndProc 方法以拦截 WM_PAINT 消息。

再举一个比较抽象的例子:在开发输入类的 Control 时候,常常会碰到切换“Focus”的问题。围绕着“Focus”的切换,Control 会做自己的处理,在后台处理 IME,在前台发 Enter, GotFocus, LostFocus, Leave 等一系列的事件。我们常常需要利用这些逻辑,不能影响它的运行而又要加进去自己的逻辑。在这种情形下,不了解 Windows 消息,不了解 Windows 消息机制,是不可能把这件工作做好的。

简而言之, System.Windows.Forms 对 Control 的开发做了很多工作,提供了 System.Windows.Forms.Control, System.Windows.Forms.UserControl, 还有一些常用的 Control, 并对它们提供了 Design-Time 支持,构成了一个比较完整的框架,一个平台。不过在提供这些便利的同时,不可避免地将一些东西做得比较死,从而又限制了 Control 的开发。在这种情况下,对于一些对用户接口有着特殊要求的 Control, System.Windows.Forms 就时常显得无能为力。使得我们不得不研究系统,找出一些这样的时机,绕开系统的限制(当然,同样也有可能绕开了框架平台提供的便利。意味着许多工作你不得不自己做)。充分利用前面提到的几个口子,并且结合 Win32 API 的直接调用,不但可以帮助解决在开发 Control 过程中遇到的一些难题,能够绕开 System.Windows.Forms 的限制,而且,还可以利用它们搭建出自己的 Control 开发框架。简单地说,这样做的优缺点可以归结如下:

优点:充分利用了 Win32 API 平台的功能,摆脱.NET 目前框架的限制,可以开发出功能强大的 Control 产品。

缺点:由于和 Win32 API 平台捆绑过于紧密,不符合 Windows 目前的发展方向,所以会存在一些问题。例如, .NET Control 如果直接调用了 Win32 API,运行时受.NET 的运行权限制约,难以在 Web 页面上应用;所开发产品在下一代 Windows 平台 LongHorn 上生死未卜,有可能需要重新开发。

希望本文的内容对大家的开发工作有所帮助。最后祝编程愉快! 

■ 责任编辑:欧阳璟 (ouyangjing@cson.net)

早选择 早收获!

新东方职业教育前身为新东方IT教育，它创造出极具特色的IT培训模式，即由“教学、教材、开发、就业”四大业务模块组成的综合型IT教育模式。并在IT培训行业内率先推出了“就业班”课程理念和类型。5年来，“就业班”课程也已成为深受新东方学员喜爱的经典课程之一。

面向岗位技能的课程体系：在新东方职业教育标准课程设置之前，新东方职业教育已在全国范围内对IT企业的职位需求、人才需求进行了全方位的调研。在对企业需求的充分把握下，首批推出了软件开发工程师、网络及网站工程师、电脑美术设计师三大类课程及十一个专业方向的培训及考试体系。

具有企业真实项目的案例教学：通过项目案例实践，在模拟企业全真工作条件的环境中学习相应知识和技能，了解一个完整项目的流程，迅速培养一定的项目开发能力。学员通过3-4个月的实战学习，掌握了与目前IT企业应用相适应的技术，数已万计的学员通过新东方职业教育踏上了IT职业发展之路。

团队协作的案例式考试：它不是以简单客观题为主的标准化考试，而是引入了以企业实际项目为基础、企业参与命题和判卷的“案例考试”方式，每一科目的案例考试题目均取自相应技术领域内知名企业的实际项目，从而保证能够以最小的误差评定考生对于特定技术岗位的工作实践能力，50多家著名IT企业优先招聘通过“NIT-PRO”考试学员。

专业、系统、实战的学习恢复您IT就业信心!

全面热招中

现在报名更有多重惊喜!

北京 广州 长沙 哈尔滨 武汉 长春
唐山 福州 石家庄

咨询电话：
010-82622266

新东方职业教育

新东方职业教育与教育部考试中心已结为合作伙伴，面向IT人才需求，推出了《全国计算机应用技术证书（职业技能类）考试》，简称NIT-PRO项目。

北京市海淀区新东方职业教育 网址：<http://it.neworiental.org> <http://www.nit-pro.org>
地址：北京市海淀区北四环西路15-1号新东方学校西侧二层

010-82622266(北京) 020-87530539/87530450(广州) 027-87812157/027-87304229(武汉)
0731-4169216/4169226(长沙) 0451-86651691(哈尔滨) 0591-87549332/87549039(福州)
0431-5112115(长春) 0311-6044119(石家庄)

利用英特尔软件开发工具 释放 IA 架构上程序的最佳性能

■ 文 / 高源奕

本篇是介绍如何运用英特尔软件开发工具提高软件性能的最后一篇。前两篇分别介绍了利用英特尔编译器提高编译后代码执行性能以及通过英特尔 VTune 寻找代码性能瓶颈入手优化。本篇将介绍如何利用高度优化的函数库来提高代码开发效率和性能。由于函数库所提供函数都针对英特尔处理器进行过专门优化,直接调用接口就可方便使用,开发者就好比站在巨人的肩膀上,不再需要以硬编码的方式针对处理器的优化,省去了大量时间精力,从而提高开发效率,使产品更快推向市场。

英特尔基于开发者的不同需要,提供了三套高度优化的函数库,它们分别是针对信号处理与多媒体软件的“英特尔集成性能原件”(英特尔 IPP),针对三维图形“英特尔图形性能原件”(英特尔 GPP)以及针对数学、工程、科学计算的“英特尔数学内核库”(英特尔 MKL)。下面,就让我们开始优化之旅,了解这些函数库的功能与运用。

英特尔集成性能原件, IPP

这是一个跨平台的信号与多媒体软件库,包含一个将功能从底层处理器抽象出来的低级层。这样,应用程序就可以透明地使用英特尔体系结构的最新增强

功能,例如 MMX(tm)技术、数据流单指令多数据扩展指令集(SSE)、第二代数据流单指令多数据扩展指令集(SSE2),以及安腾体系结构和 Intel XScale微体系结构。“英特尔 IPP”提供了广泛的多媒体功能:音频、视频及语音编解码器;图像处理、信号处理、数学支持例程以及计算机视觉。使用英特尔 IPP 可以构建许多标准的编解码器,包括:MP3与AAC音频;H.263、MPEG-1、MPEG-2、MP3及MPEG-4 视频;JPEG与JPEG2000图像;以及G.723.1与G.729语音。

“英特尔 IPP”包含各种各样的函数,用于进行矢量与图像处理、颜色转换、过滤、分屏、设置阈值、变换,以及算术、统计、几何与形态运算。对于每个函数,“英特尔 IPP”均支持多种数据类型和布局,同时保持了数据结构数量的最小化,它提供了丰富的选项供用户在设计优化应用程序时选用,使之不必去编写汇编代码。

IPP 的优化效果更是立竿见影,在基于奔腾 4 处理器的平台上进行的最新测

英特尔 IPP 领域	测试变量的数量	性能增益 ¹ , 英特尔 IPP 对 C 代码
图像	12,330	390%
信号	20,429	310%
JPEG	152	310%

图1 “英特尔集成性能原件”(英特尔 IPP)优化效果

试表明,“英特尔 IPP”库的性能增益超过编译的C代码。图1显示测试结果。

英特尔 IPP 主要针对以下系列的英特尔处理器:(图2),



图2 英特尔 IPP 针对的英特尔处理器

英特尔图形性能原件, GPP

该库提供了一套内容丰富、功能强大的三维图形函数,并专为采用 Intel XScale 技术的“英特尔个人互联网用户端架构”(英特尔 PCA)应用处理器进行了优化。具体用法模型如图3。

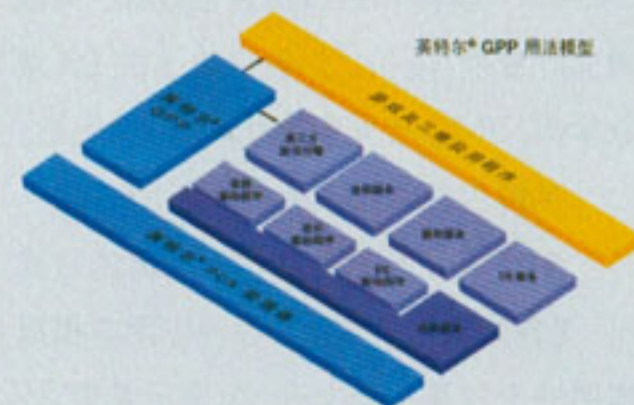


图3 英特尔 GPP 用法模型

英特尔 GPP 包含范围广泛的三维图形函数,其中包括数据类型转换、算术、

三角、向量、矩阵及光栅等。英特尔 GPP 库中的函数针对 Intel XScale 微体系结构进行过优化，使开发人员不必编写底层汇编代码即可最大限度发挥英特尔处理器的性能。不愧是一种便捷而强大的手段。这套“现成的”最佳解决方案可以帮助降低开发成本，加速推向市场。尽管英特尔 GPP 库并非一套全面的图形引擎，但它提供了一套基本的预制代码块，可用于创建专为 Intel XScale 微体系结构度身定制的三维引擎。这些原件包含上至引擎组件级的基本函数，在引擎体系结构与实施方面具有极大的灵活性。

此外，英特尔 GPP 也解决了三维软件渲染系统中的许多常见问题，例如缺乏对整数除法的支持、缺乏专用浮点硬件、系统内存有限、手持与移动设备显示区太小，等等。英特尔 GPP 与许多流行的嵌入式操作系统兼容，如运行于英特尔 PXA250 处理器上的 Microsoft Pocket PC* 2002。不过，这些原件非常低级，在设计时已经注意避免对主机操作系统的依赖性。英特尔 GPP 对应用程序移植提供了很好的支持，同时不会以牺牲性能为代价。英特尔 GPP 还可以优化针对手持与移动设备的游戏应用程序。

英特尔数学内核库，MKL

本库由经过高度优化的函数组成，这些函数涉及英特尔平台上性能要求很高的数学、工程、科学及金融等领域的应用程序。此函数库的功能区包括由 LAPACK 与 BLAS 组成的线性代数、离散傅立叶变换 (DFT)、向量超越函数 (向量数学库/VML) 及向量统计库 (VSL) 函数。

英特尔数学内核库中的新增功能：

◆ 当今在基于英特尔处理器的台式机、工作站及服务器上，各种应用与模拟程序的工作量不断上升，为满足其需求，英特尔 MKL 通过不断提高性能，增加更多的数值功能，继续保持着出众的表现。

◆ 通过提供一维与多维 DFT 例程

(最高可达 7 维，其变换长度不包括 2 的幂次，支持混合基数)，进一步扩展了英特尔 MKL 的 FFT 功能。

◆ 向量统计库提供了可手工汇编、调整及矢量化的高性能伪随机数生成器。这些随机数生成器子例程同时提供基本的连续和离散分布。

英特尔 MKL 的性能优化可以体现在以下几方面：

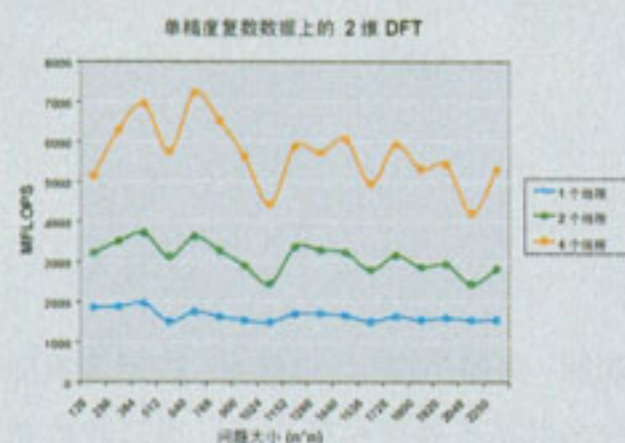


图 4

DFT 函数优化

此图显示在一系列的矩阵大小上，使用单精度复数时二维 DFT 函数的性能，以每秒百万次浮点运算 (MFLOPS) 计。英特尔 MKL 内置的线程技术可以利用多个处理器。此图充分显示了英特尔 MKL 中这些 DFT 函数优化带来的出众性能，以及线程数增加时的性能放大效果。

线性代数与 DGEMM

矩阵乘法是立方运算，作为乘数的矩阵大小翻倍时，运算量会达到原来的 8 倍。双精度普通矩阵 (DGEMM) 在致密线性代数中是司空见惯的问题。在许多非常依赖大型方程求解的应用程序中，正确编写的解算器的性能直接取决于 DGEMM 性能。

图 5 显示单处理器与双处理器 2 的 DGEMM 性能比较，以每秒百万次浮点运算 (MFLOPS) 计，这里使用基于英特尔至强处理器的平台，二级缓存 512-KB，运行 Microsoft Windows。NET* 服务器。图上的矩阵尺度综合了大、小两

种矩阵大小。随着矩阵大小 M 与 K 的变化，会标出产生的 MFLOPS 性能，并用各种颜色的带状区表示性能变化范围。在多处理器系统上，英特尔 MKL 使用其它可用的处理器来加速性能并完成任务，随着处理器数量的增加，性能几乎呈线性放大。

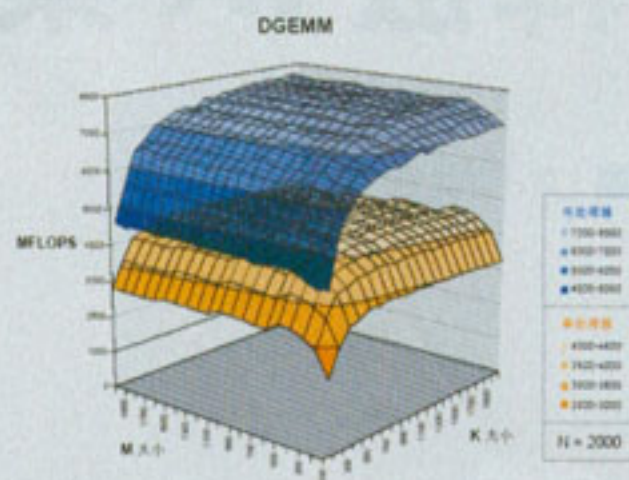


图 5

英特尔 MKL 优化功能甚至可以让台式机发挥大型机的威力，英特尔 MKL 的各种功能区 (LAPACK、BLAS、DFTs、VML 及 VSL) 均能达到极优的性能。英特尔 MKL 中的许多函数都已线程化，在对称多处理 (SMP) 系统上，可以产生极佳的性能。这样可以带来并行运算的好处，不必任何额外工作，便能让应用程序随线程增加产生优异的性能放大效果。

小结

以上三套函数库都从性能与兼容性两大方面入手，方便开发者提高程序性能。利用这些函数库不必针对特定的处理器专门去编写代码，即可利用处理器的高级功能。另外，这些应用程序编程接口 (API) 可以跨越许多平台使用，使得多媒体应用程序开发人员能够轻松实现跨平台兼容，有助于降低开发成本。以上函数库均提供 Windows 与 Linux 版本。

通过以下地址便可获得英特尔软件开发工具的免费试用版：

<http://www.intel.com/cn/products/intel/index.htm>

■ 责任编辑：罗景文 (ljw@csdn.net)

international developer

They get IT, do you?



International Developer
Written by developers for developers

《程序员》全球合作伙伴 www.intldeveloper.co.uk

性能

——企业级应用开发的最后一环 (下)

■ 文 / 吴启新

继上期杂志文章(见10月《程序员》119页)之后,本文进一步对APM的概念以及各厂商的产品作了详细阐述,并对各种厂商产品的优点和不足之处提出了一些参考意见。

技术的融合已经成为了IT世界注定的发展规律,众多厂商提倡的基础架构理念就是顺应这个潮流的产物。现在主流的IT系统构架方式,大多遵循基础架构的原则,以某种开放的技术规范为基础,以中间件和应用服务器为核心,遵循通用和开放的原则,把数据管理、业务逻辑、表示逻辑和流程整合等原本独立的技术实现,都作为插件的方式围绕在作为核心的应用服务器周围,实现最大程度的可扩展性。

既然宿主发生了变化,处于附着物地位的管理工具也要亦步亦趋的跟上变化。

近一两年以来,一个流行趋势主导了性能管理和调优市场的发展变化,那就是“端到端”的性能管理解决方案,APM(Application Performance Management,即应用性能管理)是这个技术发展趋势的一个具体实现。它的着眼点是整个应用系统。

一个最大的变化出现在对于性能

管理效果的评价,原有性能调用工具的服务对象是IT部门中的开发人员和维护人员,而现在APM将应用的使用者,也就是最终用户的满意程度和提升他们的操作体验作为终极目标。IT部门不再是单纯的成本中心,他们提供的基于服务水平协议(SLA)的IT服务,能够直接改善企业的运营效率,成倍的提高企业盈利水平,而性能的保证成为这一切的基础。

APM 厂商介绍

Quest Software

Quest 产品战略是为客户提供应用系统的服务质量保证。在APM领域,Quest 提出了包括发现、诊断、解决和报告四个步骤的方法论(图1),实现对于应用系统的全程监控、故障分析和问题排除,并且通过报表的形式提供服务水平报告。

Quest 使用 Foglight/Spotlight 对应用系统进行监控和性能调整,作为一个7×24的监测工具,Spotlight 以插件(Cartridge)形式对多种技术组件提供支持,Spotlight 则包含了针对数据库、Web 服务器、Exchange 服务器、应用服务器和打包应用系统的性

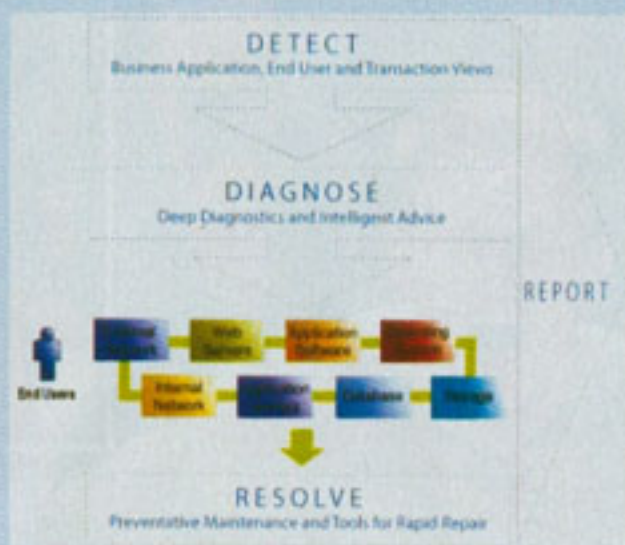


图1

能分析组件,在对于性能问题的调优工具方面,Quest 提供 Quest Central for Oracle, Quest SQL Optimizer, 服务于主流的关系型数据库系统。Quest 还收购了 Sitraka 公司的 PerformaSure 和

JProbe 产品,来增强自己在 J2EE 领域性能管理的技术实力。

作为历史最悠久的性能管理和调优工具厂商,Quest 在这个市场的知名度是其制胜的法宝之一,此外,产品对开发环节中的代码优化、资源规划等方面提供支持,而且简单易用的图形界面,也是其工具的特色。

由于 Sitraka 的产品还没有足够的时间溶入 Quest 产品家族,造成在一致性方面的不足,而且,Quest 产品在监控和采样 J2EE 和 Tuxedo 的过程中,对于系统的开销也是比较高的。

Mercury Interactive

Mercury 提出了业务技术优化 (Business Technology Optimization) 的概念,是最早将IT技术和业务需求相关联的厂商之一。其产品主要覆盖应用的测试和性能管理领域(图2)。

Mercury 的测试和性能管理解决方案,侧重于在J2EE和ERP/CRM打包软件领域,主要产品线分为四个体系,称为质量中心、性能中心、业务可用性中心和解决中心,覆盖应用开发、部署、测试和运行的若干过程。其中的LoadRunner产品被广泛采用于压力测试和可用性规划方面,Topaz产品(现在称为Mercury Diagnostics & Monitoring)针对在J2EE应用和打包的ERP/CRM应用中的问题隔离和定位,可以从最终用户的某个交易,通过层层拨茧的方式,依照方法调用路径一直可以分析到SQL



图 2

语句层,同时对于系统资源的消耗情况,如CPU、I/O和死锁情况能够进行分析。

好像Quest通过购并的方式扩展自己的产品覆盖一样,Mercury也在不断的丰富自己的产品系列,从原有主要针对应用的开发环节提供测试软件,向应用的

生产环节进行扩展,并且从最终用户和业务人员的角度,从交易、地点和客户等方面监控和管理应用,提出了以业务为中心的性能管理解决方案。

然而,Mercury的解决方案缺乏对于各个技术层次的全面覆盖,难以形成统一和相关联的性能检测和分析;并且,由于其产品在对应用系统进行采样的过程中,导致了较大的系统负载,从而大多数的客户还只是在开发和测试阶段使用该产品。对于评价APM产品的一个关键指标是对未来性能的纪录和对未来发展趋势的预测,这都是以大量存储性能采样数据为基础而实现的,Mercury缺乏专门的性能指标存储,导致其对于长期性能的分析有所欠缺,这也是性能测试产品固有的问题所在。

Wily Technology

Wily的核心产品是Interscope(图3),提供了最广泛的J2EE应用服务器的支持,而且通过SNMP同主流的管理软件进行集成。对于特定的J2EE应用服务器和中间件,Wily通过Interscope Powerpack提供扩展的性能管理和监控,能够针对关键的参数设定阈值,当出现性能下降的趋势时为管理人员发出警告信息,并且在管理员进行参数配置时提供建议。

在J2EE领域,一个明显的技术发展趋势是借助这个标准,不仅完成应

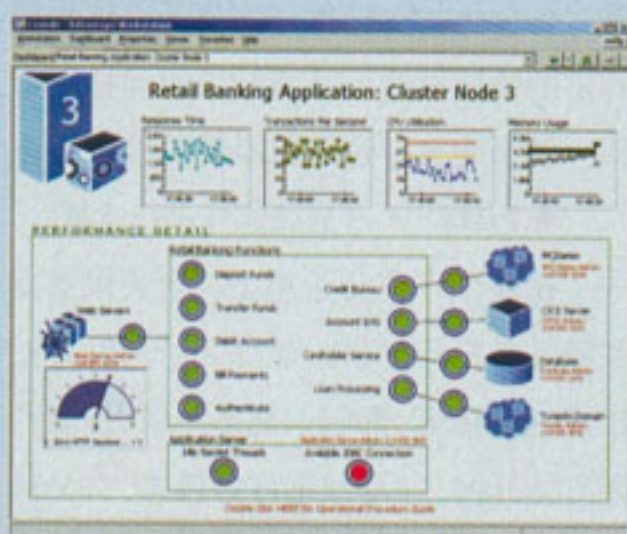


图 3

用的开发,而且完成应用的整合,不仅完成业务逻辑和业务流程,而且统一用户

的操作界面,涌现出了基于J2EE的门户(EIP)和应用整合(EAI)技术。Wily也紧跟这个趋势,推出了针对EIP和EAI的性能管理和优化工具。看来Wily打算将J2EE应用的调优进行到底了。

也是由于Wily的专注,所以它实际不能算作一个APM厂商,因为它违反了APM视野广泛和关联性能分析的首要特性。在故障分析这个技术领域,Wily能够进行层层深入式的分析过程,但是缺乏自动化和智能化的分析工具,对使用人员的技能要求较高。

Veritas的 APM 解决方案

2003年7月,著名的存储软件厂商Veritas收购了Precise,作为其进入APM领域,进而完善其效用计算远景的一个重要步骤。Precise最早是研发针对基于Oracle应用系统的调优工具的,进而通过

扩展其产品覆盖,提出了i³解决方案。

i³是一种典型的APM解决方案,它基于一种旨在便捷地对应用性能问题提供快速隔离、分析、纠正和校验的方法论,由以下5个阶段构成:

- 1、检测:识别并显示性能问题症状
- 2、查找:识别问题来源,将其隔离

在某个技术层次

- 3、聚焦:发现问题的根源所在
 - 4、改进:提供基于最佳实践的建议,以改进性能
 - 5、验证:确保实施措施达到预期目标
- 以上的五个步骤依照以下顺序执行:
- 1、使用VERITAS i³的告警功能,进

行主动和/或被动检测。这种告警可能发送到相应的基础架构组件、组织或个人,此外,负责性能的组织中的相关人员可以长时间检测服务,同时查看各种进程(如EJB、URL)的性能趋势;

2、诊断专家可以端到端地查看应用,根据每个技术层次在响应时间中所占比例,查找是哪个层次导致性能降低;

3、在将问题聚焦到某个技术层次之后,通过“钻取”功能,彻底定位根源问题;

4、VERITAS i³提供经过验证的最佳实践,协助使用者改正应用问题。它甚至可以同特定的数据库服务器、应用服务器等基础部件紧密关联,取得和改写某些配置参数;

5、在将改进真正提交到生产环境之前,VERITAS i³提供测试功能,以验证预期的效果是否达成,并且确定没有对相关应用的性能造成负面影响;

APM 是在实际生产运行环境中,从应用的角度对系统性能进行监控,分析和优化的管理方案。

VERITAS i³是一个集成的应用性能管理解决方案,专注于确定用户体验和服务水平。它能够识别端到端的资源占用情况,从浏览器开始,跨越中间件和应用服务器层,通过数据库一直到物理存储器的监控和检测应用性能情况,以用户响应时间为统一的标准来衡量各个技术层面的性能。这种方式,避免了负责各个技术层面的厂商和技术人员相互推卸责任,大大提高了隔离问题的速度。

在确定问题之后,深度钻取的能力又向使用者提供了专家级的修改建议,协助快速修改问题。相比之下VERITAS i³覆盖面较广且更加深入,可以覆盖企业级应用的广泛领域,并能从代码,甚至物理存储得角度来进行性能的深入管理。

VERITAS i³的产品组成为 Insight、Indepth 和 Inform 三个部分。这些产品无缝集成,共同关注关键应用组件——这样就提供了一个完整的端到端性能管理解决方案。此外,VERITAS i³的每一个组件都能作为一种独立产品单独运行,以满足特殊环境和应用的需求。图4说明了 VERITAS i³的体系结构组件。



图 4

◆ Insight ——“什么原因导致运行缓慢?”

当前的应用利用了多层体系结构,以提供多功能性,但这些体系结构却使我们很难确定性能降低的原因。Insight带有TotalCorrelation,能够测量所有体系结构层的响应时间,并将响应时间按层分段,从而确定导致运行缓慢的原因。

◆ Indepth——“应该怎样进行修复?”

在整个IT基础设施上,全天候捕捉关键性能指标(KPI),确保无论何时开始出现性能降低,都能识别出问题。然后利用SmarTune,钻取到部件级技术,如Web服务器、应用服务器、中间件、数据库以及存储器等,以自动确定最终问题根源,生成专家调节建议。Indepth可识别“为什么”产生问题,以及“如何”修复问题。

◆ Inform——“现在是否存在性能问题?过去有没有?未来会不会有?”

了解应该“何时”专注于性能问题,与“如何”修复问题同样重要。如果没有掌握趋势、基线、期望值和告警,则很难有效管理应用性能。即使发生最轻微性能降低,Inform也能发出及时通知,便于用户使用SmarTune提

供的调节建议。无论通知的原因是无法达到服务水平,或性能标准开始降低,用户都能及时收到信息。

VERITAS i³关注应用的整个生命周期,VERITAS i³不仅着眼于生产系统,并且能够服务于开发、质量保证部门:

在开发环境中,该解决方案能为我们测试系统和了解性能提供极大的帮助,在设计和体系结构问题在生产系统中影响到终端用户之前,提早捕捉这些问题。SmarTune提供的全面数据捕捉和自动分析功能,有助于应用工程师有效地确定开发工作的优先级。在质量保证、测试过程中,VERITAS i³提供了统一的测量标准,并且凭借独特的自动记录功能,质量保证和开发部门能够将大量记录保存在文档中。

总结

通过采用基础架构和多层次模型,我们已经能够满足大型企业计算在高可用性、可扩展性和可扩展性方面的要求,然而,伴随而来的性能问题却导致了最终用户体验的下降。不同于以往应用系统的技术实现,复杂性和关联性成为解决应用性能问题新的障碍,APM同性能单一性能调优工具的一个显著区别就是在复杂系统中,对于问题的定位和隔离的能力。此外,对于性能的管理应该从应用的开发环境扩展到生产环境,APM为系统管理和运行维护人员提供了前所未有的支持。对于应用在开发阶段的调优工作,在其正式投入使用之后就基本结束了,而应用的运行阶段的性能管理则更是一个长期的过程。依靠对于应用性能指标长期的采样和积累,APM能够知晓应用系统在过去和现在的性能状况,并且可以对于将来的发展趋势得出一个科学的预测。依靠APM进行应用性能管理的终极目标是确保系统性能的稳定和可控,一个性能稳定的系统比在发现问题之后进行补救更加重要。

■ 责任编辑:罗景文(ljw@csdn.net)



一、邮局订阅
全国各地邮局均可办理。

二、向互联网周刊直接订阅

◆ 邮局汇款

北京市东城区建国门南大街乙1号金龙大厦4号楼5层
《互联网周刊》发行部收(100005)

为确保您及时准确的收到《互联网周刊》，
请在汇款人栏清晰注明您的单位、地址、邮编、电话。

◆ 银行汇款

户名：北京互联网周刊服务有限公司

开户行：北京市商业银行天桥支行

帐号：01090359500120105037186

请将您的回邮地址、电话、收件人姓名、汇款单
复印件传真至 010-65282497。

三、网上付费：www.gotoread.com；www.dangdang.com

四、订阅热线：010-65284818 / 65249988-5501、5502

每周一期；定价：6.5元；全年订阅价格：292.5元；
上半年订阅价格：136.5元；下半年订阅价格：156元；

更多了解请您登陆 www.ciweekly.com

网络时代的商业周刊 CHINA INTERNET WEEKLY
互联网周刊

网络时代的商业周刊

软件配置管理的最佳实践经验

基于任务的软件配置管理解决方案

文 / Telelogic 亚太地区产品经理 谷炼

前言

90年代中期,当绝大多数配置管理工具厂商还在热衷于推广基于文件的软件配置管理(File-based Configuration Management)解决方案时,TELELOGIC公司提出了一个全新的理念,重新设计产品体系构架,采用以工作任务为核心的对软件开发中的变更进行管理和控制的新思想,首先创造出基于任务的软件配置管理(Task-based Configuration Management)解决方案:TELELOGIC SYNERGY。

基于任务的软件配置管理解决方案的理念从诞生到被用户接受,从被用户接受到被用户高度认可的过程,完全体现了软件开发从简单到复杂、从单一化到多样化、从集中开发到分散并行开发、软件规模从小到大的演变过程,从而使用户真正认识到,为适应市场各种复杂的需求,软件开发一定要做配置管理,而只有做基于任务的配置管理,才是完整的配置管理解决方案,才可使软件开发过程中的各种变更因素得到有效的控制,发布的产品才能得到有效的保证。

本文将在分析基于文件的配置管理工具带给用户软件开发变更管理不足的基础上,阐明基于任务的软件配置管理解决方案的完善之处,意在引起用户思考如何以

基于任务的配置管理解决方案驱动项目开发流程的改进和项目实施状况的控制管理,与用户共同分享软件配置管理的最佳实践经验。

软件变更管理

软件开发过程可以简单地描述为:开发团队按照一定的软件开发需求,完成各种软件开发任务,最后提交软件产品。因此软件开发中的变更可以抽象成两个要素:即软件开发过程中任务状态的变更以及在完成任务的过程中产生的生成物(各种文件、目录等)的变更。尽管软件产品最终是由不同文件的不同版本配置而成,但这些文件的版本是基于不同的工作任务制作而成的,因此,在软件配置管理的过程中,工作任务和在其基础上生成的文件版本是不可分割的两个变更的要素。而变更才会带动软件开发项目不断进展。如图1所示,变更管理流程与开发流程是密切相关的。在时时

刻都具有变更存在的软件开发项目中,大家不难看出以任务为中心的配置管理的重要性:

- ◆ 项目经理在对项目进行管理时,掌握项目的进展状况是对分配给项目组成员的各个任务状况进行收集统计;

- ◆ 开发人员对具体的文件进行修改时,其目的是为了完成项目经理分配给他的任务;

- ◆ 集成人员有效地集成开发人员的开发成果,是以开发人员所完成的任务为目标进行收集,再进行系统集成和测试;

- ◆ 软件发布经理所关注的是在发布版本中都包括了什么新功能、增强功能和缺陷修复,这些信息是通过团队所完成的开发任务汇总而成的。

基于文件的配置管理

在一个基于文件的软件配置管理系统中,开发人员只关注其修改的文件本身的变化,检出文件、修改文件、最后检入文件放回数据库。这种开发人员对文件的修改是独立的个别变化,如图2所示,在基于文件的配置管理系统中,提交和集成的单元是文件;最终的发布中包含的内容是文件和目录的集合体。

当软件开发进入到构建和集成阶段,负责集成的人员必须被告知(一般是以电子邮件、电话、甚至用

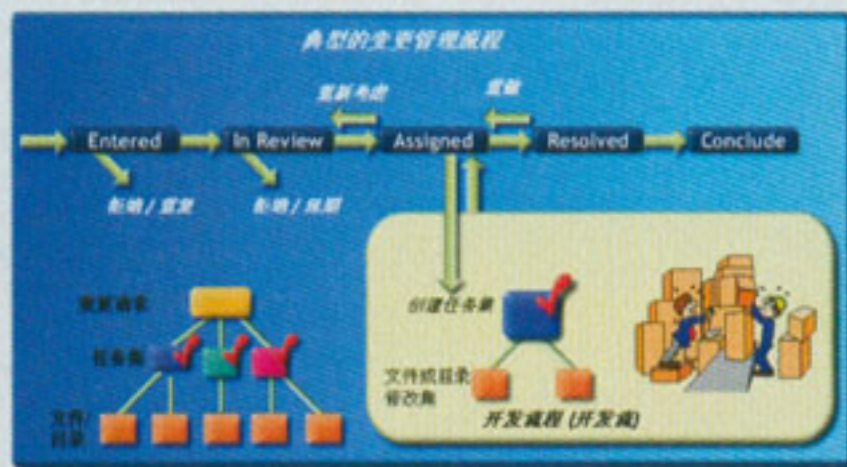


图1 变更管理流程与开发流程的关联性

口头交流的方式) 哪些文件已处于可集成状态或系统测试状态。更简单的做法是集成测试负责人直接到数据库取最新的版本。但这种盲目地去取最新版本的做法, 往往会使集成测试人员收集上来的东西不是其所需的, 或是没有完全完成的修改, 或是不匹配的配置项, 从而造成集成测试阶段出现配置项集成失败、集成效率低下、问题难查的众多问题。甚至更糟糕的是, 由于包含文件的错误版本而造成运行错误或缺陷, 有时在测试阶段很难发现, 这种错误就会埋没在产品中直到产品交付给用户。

那么, 既然基于文件的配置管理系统有这些缺陷, 为什么仍然有开发团队在使用呢? 概括起来有以下几点:

- ◆ 大多数软件以及相关的工件存储在文件中, 开发人员使用的绝大多数软件工具是基于文件处理的。尽管一个逻辑变更会涉及到多个文件或多个文件类型, 用户必须在一个又一个的文件基础上实施该变更。

- ◆ 大多数基础版本管理工具 (例如 CVS, RCS, PVCS Pro, Visual SourceSafe) 全部是基于文件进行操作的, 所以团队已习惯于该方式的操作。加之绝大多数集成开发环境 (IDE) 提供了版本控制系统的用户界面, 使用户可以在不离开 IDE 的环境下, 熟练地对文件版本进行处理, 更使得使用这种方式的用户具有相当的数量。

尽管很多开发工具是基于文件的, 而且也许仍然会继续一段时间, 但典型的软件开发中的变更很少是局限于单一文件的。一般情况下, 软件开发中的一个变更请求会涉及到对多个文件的修改。比如, 为完成一个任务或变更请求, 你也许会修改两个源文件、一个头文件、一个表单文件和一份帮助文档。作为一个开发人员, 你知道如果这五个文件中任何一个没有包含在一个配置项中, 都会导

致从编译出错到不可预计的运行错误行为。所以, 你必须竭尽全力地记住你检入的这五个文件, 同时告知需要此五个文件的其他相关人员 (如: 集成负责人员或发布人员等)。想一想, 你能保证时刻准确地记住吗?

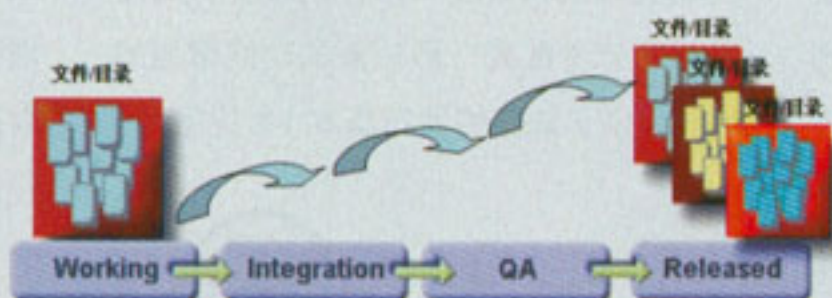


图2 基于文件的开发配置过程

基于任务的配置管理

基于任务的配置管理解决方案的出发点非常简单:

如果开发人员可以报告系统文件变更的原因, 那么系统就能及早地在开发过程中利用这些信息 (并结合已知信息) 去构建有效的配置, 自动识别潜在的问题。这样一个系统可以保证:

- ◆ 一个变更请求和所修改的文件之间的关系是清晰的、相匹配的。

- ◆ 一次软件配置总是包括应得到的全部配置文件, 无所需文件的漏失。

- ◆ 对工作任务之间的直接和间接关系所造成的配置项之间的依赖关系可以简单容易地进行处理、控制和管理。

此外, 开发人员以及集成测试负责人可以根据“逻辑变更” (工作任务), 组织和操作集成过程, 而不是零散地收集个别文件的某个版本。尽管开发人员仍然需要检出文件, 但是那仅仅是配置管理停留在文件级的一个操作。开发人员不必去硬记为完成某个任务都修改或创建了哪些文件, 开发人员甚至可以采用更简便的方式: 即检入一个任务, 这样可同时完成任务时所修改的全部文件检入。如果开发人员想利用其他团队人员的工作成果更新自己的工作空间时, 他可以直接提取其他

人员所完成的工作任务。

同样地, 集成测试人员不必根据文件本身去进行软件配置, 他们可以用基线 (包括已发布的版本) 加上在基线的基础上所完成的任务构建软件配置。如图3所示, 以任务为中心的配置管理, 在整个开发过程中, 提交和集成的单元都是任务; 在最终发布阶段, 发布经理所拿到的是一个层次分明的发布内容, 即: 完成发布项目的全部变更请求单, 所完成的开发任务集以及完成项目所修改的全部变更集 (文件和目录集)。

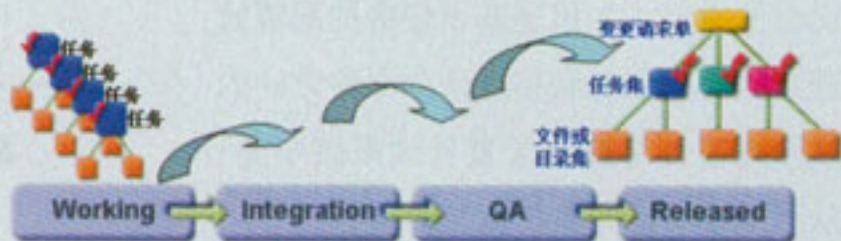


图3 基于任务的开发配置过程

现在, 我们分析一下, 这种基于任务的配置管理可以给整个开发团队带来什么好处:

有效地提高团队生产率

也许基于任务的配置管理系统的最大好处是用户相当容易介入系统。尽管软件开发团队习惯于在文件上工作, 但是他们实质上是为了完成一系列的任务而工作在这些文件上的, 所以所有参加软件开发的人员来说, 基于任务的配置管理系统使开发过程变得更自然、更直接, 从而带动了开发团队整体生成效率的提高。

◆ 开发人员生产率的提高

- ❖ 在基于任务的配置管理系统中, 开发人员可以得到一个清晰表明任务优先级和最终期限的“我的工作列表”。必要时, 开发人员可以更深入地获得此任务是来自哪个变更请求的、该变更请求的原始记录内容是什么?

- ❖ 自动将修改文件和变更理由相连接, 以变更集为单位提交正确的修改

内容给集成人员。在软件开发过程中,开发人员一般是根据逻辑的变更进行自然地思维的,也就是在内心里他们是把所做的工作任务与修改的文件连在一起进行思维。因此,当他们具备完成了某一个缺陷修正或功能增强的任务条件时,为了让其他团队成员及集成人员获得这些修改,他们需要记住他们所修改的全部文件。如果是在基于文件的配置管理系统中,开发人员一般是用手动记录的方式。这种手动记录的方式存在的问题是容易漏计或记错。而一旦这种现象发生,会造成集成测试人员不知因何集成失败,很难找出问题所在,甚至造成开发项目交付的推迟。如果是基于任务的配置管理系统,则任务和相应修改文件之间的联系是直接存在于配置管理系统中的,从而免去了这方面易出错的行为。

- ❖ 简化代码核实过程。当集成人员发现问题,把开发人员完成的任务退还给开发人员时,开发人员可以以任务的变更集为基础迅速定位应对哪个文件哪个版本进行修改。如果不是基于任务的配置管理,开发人员往往在这以环节浪费大量时间。

- ❖ 项目经理生产率提高和软件发布管理质量的改善

- ◆ 项目经理不必手工收集分配给各个开发人员工作任务进展状况。使用基于任务的配置管理系统,项目经理可以实时获得项目进展状况的报告,甚至可以获得自动报告功能。据统计,并已在一些开发组织中得到证实,仅此功能就可使一个项目经理一周节省4~8小时。

- ❖ 真正的按需集成、按需管理项目的方法。根据各次集成和测试状况,项目经理或集成人员可以决定从一次集成测试中临时移动或排除一个工作任务,进行集成测试,从而帮助项目尽快定位问题所在,迅速找到稳定的解决方案,减少开发时间的浪费。而被排除或移动的任务与其变更集一起,仍然存储在数据

库中。如果项目经理有朝一日还想使用这些任务时,可以随时调出来重新使用,这样不但方便了项目经理对团队的管理而且允许他进行开发任务的增值操作。此外,根据项目开发的状况,项目经理和QA负责人可以以前一发布版为基础,加上已完成且具备发布条件的任务,灵活地进行软件配置,形成新的功能增强的或/和修正缺陷的发布版本(参见图4)。

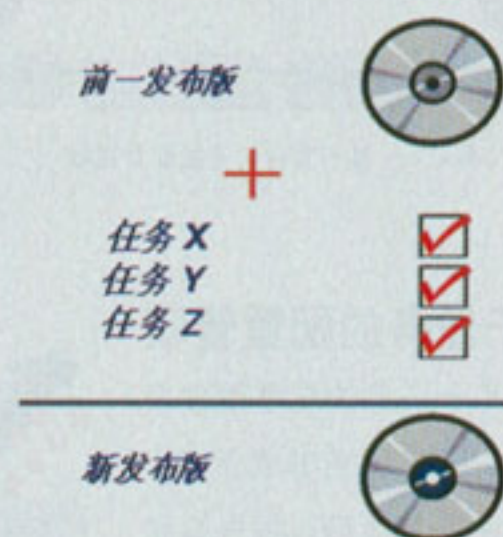


图4 基于任务的配置管理系统可以帮助团队灵活选择发布任务

- ❖ 提供项目经理控制和管理并行开发项目和发布版本的能力。基于任务的配置管理系统,在开发过程的各个阶段其配置项都是以任务单位进行的,整个开发过程也是基于任务进行监控的。集成人员在其集成空间可以自动监控达到集成测试条件的各个开发任务,这样保证了测试尽快尽早地进行。

- ❖ 安全而敏捷的项目管理解决方案。项目经理会经常向集成人员问起一个极为普通的问题:此次集成为何失败?经常会听到“因为某个开发人员忘记了提交一个文件”的回答。基于任务的配置管理系统会自动跟踪完成某个任务所修改的全部文件,因此,开发人员在向集成人员提交工作成果或集成人员在收集开发人员的开发成果时,是以任务为单位进行的,开发人员的“漏交”问题可以绝对消除。此外,基于任务的配置管理解决方案由于自动建立了开发工作和开发成果之间的因果关系,保证了软件开发的配置过程的可预测性和一致性。由于项目经理是通过计算

在某个时间点有多少个任务包含在基线里程碑中来估算项目的进展状况的,所以可预测性和一致性是保证软件开发在预定的计划下按期提供高质量的软件开发产品非常关键的两个因素。

◆ 团队协作和沟通的增强

- ❖ 团队之间的相互理解和透明度增强了。团队中的每一个人都可以看到自己和其他人员的工作进度状况:项目经理在分配工作时,可以基于团队人员的工作列表所显示的工作负荷合理地分配工作;质量保证工程师可以很容易地了解到最新的集成版本中哪些缺陷被修正了、哪些功能增强了;后方技术支持中心也可以清晰地知道新发布版本中,用户提交的缺陷是否被修正了或预计在何时可以修正;管理团队可以实时获取有效的、真实的数据,对项目 and 团队整体状况做出合理的分析和处理。

- ❖ 企业级大型团队沟通的桥梁。在一个企业范围内,一个变更并不一定局限于某一个软件,它可以同时分配任务给许多软件的开发人员,如:Windows开发人员,UNIX开发人员,LINUX开发人员及主机开发人员等等。通过链接这些多功能的任务到这个原始的、更高层次的变更请求,企业可以实现多团队、多层次的协作管理和信息共享。

结束语

综上所述,基于任务的配置管理解决方案,在充分考虑了对软件开发过程中的各种重要变化关系:如版本变化历史关系、并行开发项目之间的关系、不同平台的版本发布管理等,实施跟踪管理的基础上,着重强调以任务为中心的配置解决方案是软件开发团队对项目管理和项目开发流程规范化的基石;是软件开发变更管理的真正意义所在;是软件开发完整的配置管理思想;也是软件开发企业提升企业级管理水平的关键。

■ 责任编辑:罗景文 (ljw@csdn.net)

Dearbook 新会员制度

“全新的感觉，全新的你”

亲爱的会员朋友，感谢您对 Dearbook 的一贯支持。在大家的关怀和支持下，Dearbook 历经了一年的成长历程。通过 8 月份为期一个月的新会员制度的试行后，我们更加完善了会员服务体制，在新的会员制度中，您将会发现更加实惠和真诚的服务。

钻石 VIP 会员：

- * 全场 75 折购书优惠
- * 购书全免配送费用
- * 钻石 VIP 专区经典畅销书 70 折随心选购
- * 赠书、幸运会员……无限惊喜送给你

VIP 会员：

- * 更加实惠的 75 折购书优惠
- * VIP 专区更多畅销书籍 73 折畅享无极限
- * 购书满 100 就可以享受 3 元优惠配送
- * 抽奖、生日祝福……更多活动等你参加！

普通会员：

- * 随时注册，随时加入！
- * 76 折的心动价格
- * 购书满 150 则只收 3 元配送费
- * 会员积分还可以兑换精美礼品哦！

更多详细会员服务请登陆 www.dearbook.com

驻足 VIP 专区，享受更多尊贵权益，体验无限购书乐趣！

更多的优惠，更大的惊喜，尽在 www.dearbook.com/vip

2004 年 9 月，第二书店 **VIP 特区** 正式推出！特区内千余种精品图书，全部以 73 折 / 70 折奉献给 VIP 会员 / 钻石 VIP 会员书！如果您所喜爱的图书不在 VIP 特区内，您可自己或发动朋友，一起推荐到 VIP 特区内，享受更低价格。

VIP 特区部分图书精选：



.NET 本质论 — 第 1 卷：
公共语言运行库 (中文版)
定价：48.00 元
VIP 价：35.00 元
钻石 VIP 价：33.60 元



ASP.NET 基础教程
— C# 案例版
定价：39.00 元
VIP 价：28.50 元
钻石 VIP 价：27.30 元



C# 设计模式
定价：33.00 元
VIP 价：24.10 元
钻石 VIP 价：23.10 元



Microsoft .NET 框架
程序设计 (修订版)
定价：68.00 元
VIP 价：49.60 元
钻石 VIP 价：47.60 元



Microsoft Visual C++.NET
技术内幕 (第 6 版)
定价：106.00 元
VIP 价：77.40 元
钻石 VIP 价：74.20 元



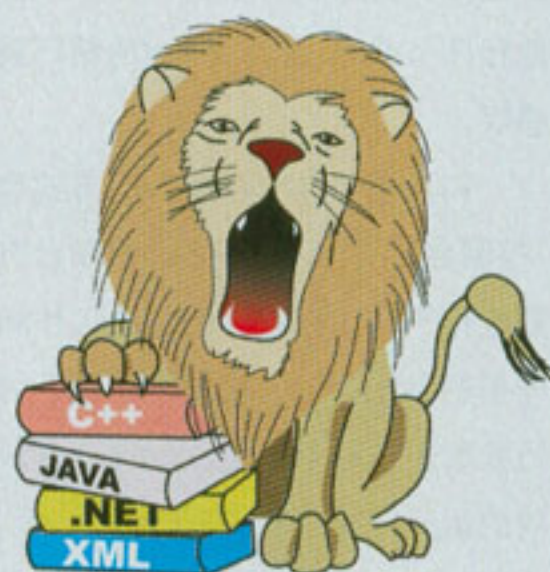
Microsoft Windows 程序设计
— Visual Basic .NET 语言描述
定价：118.00 元
VIP 价：86.10 元
钻石 VIP 价：82.60 元



Visual C#.NET 技术内幕
定价：69.00 元
售价：50.40 元
钻石 VIP 价：48.30 元



面向对象应用程序开发
— 使用 Visual Basic .NET
定价：64.00 元
VIP 价：46.70 元
钻石 VIP 价：44.80 元



程序员给程序员开的网上书店

专业 实惠 快捷 真诚

NUnit .NET 项目测试点评

■ 文 / 陆复名

如果您要问最近几年软件开发界最引人注目的亮点是什么,那么恐怕得要算是推崇“测试先行”的XP开发方法。XP开发的过人之处很大程度上在于对测试工作的极端重视。测试在以往的开发流程中往往处于一种可有可无的地位,许多项目在时间吃紧的时候牺牲了所谓的“可选流程”。这种对产品质量不负责任的做法非但没有加快项目进度,反而让无数心浮气躁的“巨兽”淹没在Bug的“焦油坑”里。

随着Java语言的流行和广泛运用,以JUnit为代表的XUnit系列测试软件越来越得到人们青睐。而C#作为微软公司挑战Java的利器,在自动测试领域一直没有引起人们的关注。

NUnit作为JUnit的C#移植版在使用功能上并没有削弱,而在充分利用C#特性的基础上使得软件的易用性有了很大的提高,因此如果您是一位使用过JUnit的程序员就一定会为NUnit的精巧易用而感叹。

和JUnit一样NUnit的运行方式也分为“控制台”和“GUI”两种。控制台方式主要应用于和其它软件的集成工作和无人守护的自动测试,在此种方式下的命令行参数较多难于记忆。GUI方式是较为直观的运行方式,特点在于操作简单、结果明了。两种方式比较之下,推荐大家平时工作时使用GUI方式。

使用NUnit实现.NET项目测试

先运行一个小例子来断定安装的NUnit是否能正常工作。如果不想用NUnit自带的示例可以自己写一个。运行Visual Studio并新建一个项目,然后在项目的引用中添加nunit.framework一项,否则的话测试类不能通过编译。

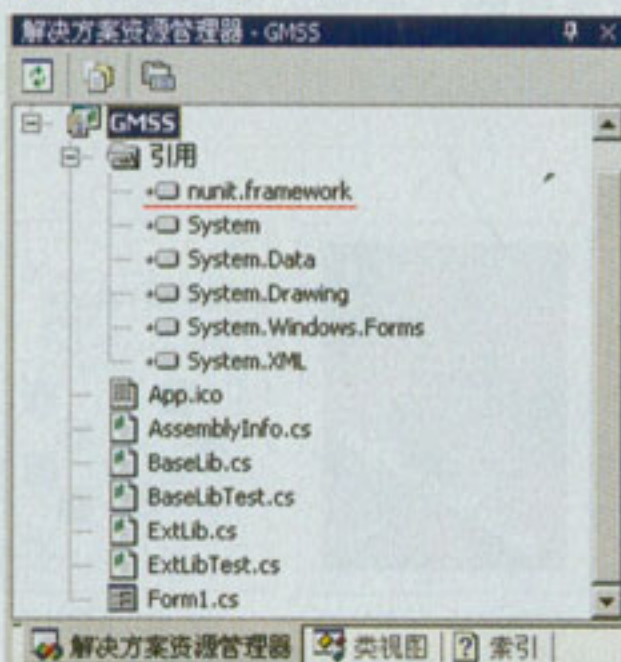


图 1

按照XP的标准,写代码前要先写测试类。有了测试类就可以开始编写函数了。写好以后把项目编译一遍,用NUnit的GUI窗口打开编译生成的.exe文件。

每次重新编译可执行文件,NUnit会自动把以前的测试结果取消(见图2)。

点击Run按钮运行。(见图3)变绿的部分表示测试通过了,假如项目中有一项没通过,那么项目

明左边的圆圈显示为红色。图4红色醒目标明的部分是测试出错的方法。怎么样,是不是一目了然呢?随着代码的数量不断增长,如果没有自动测试工具的帮助的话,一个很小的Bug就很有可能让你欲哭无泪。

NUnit 特点

标签

我个人觉得NUnit对JUnit的一个很大的改进就在于标签,标签就是“[...]”比如[Setup]等等。标签的使用让代码看上去更有条理,逻辑也更清晰,NUnit依靠标签来识别测试类中的各个部分代码的作用。

NUnit的标签共分8种:

1. [TestFixture]标签表示的是某一个类是一个NUnit的测试类。使用方法是:

```
[TestFixture]
public class TestClass {
    .....
}
```

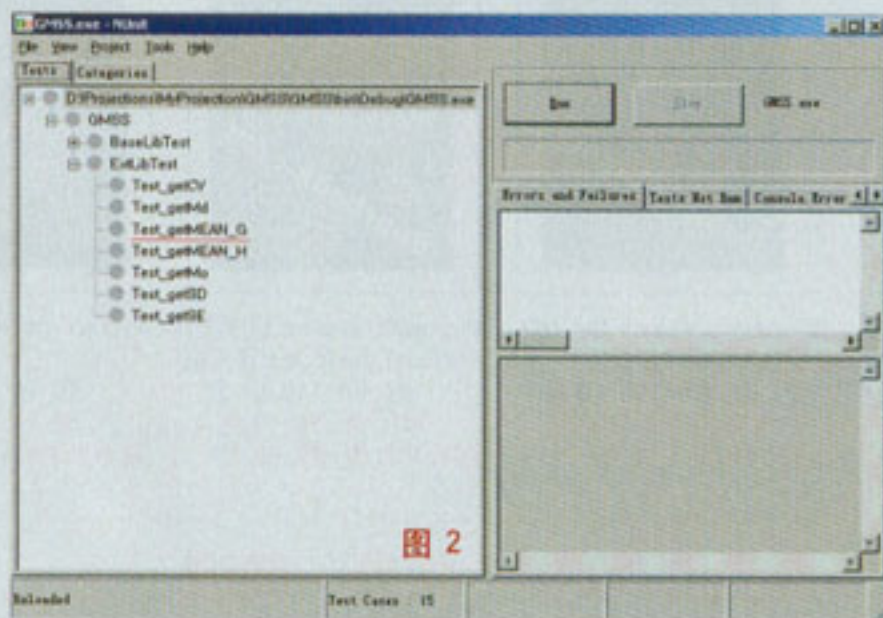


图 2

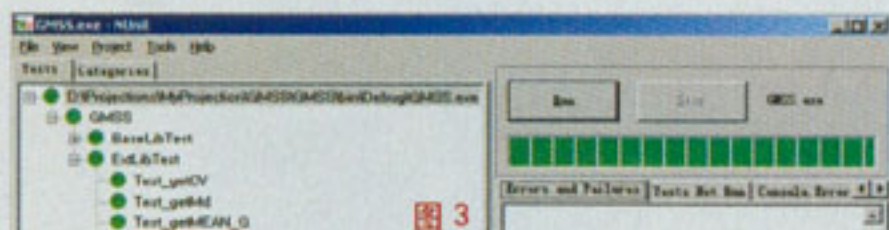


图 3

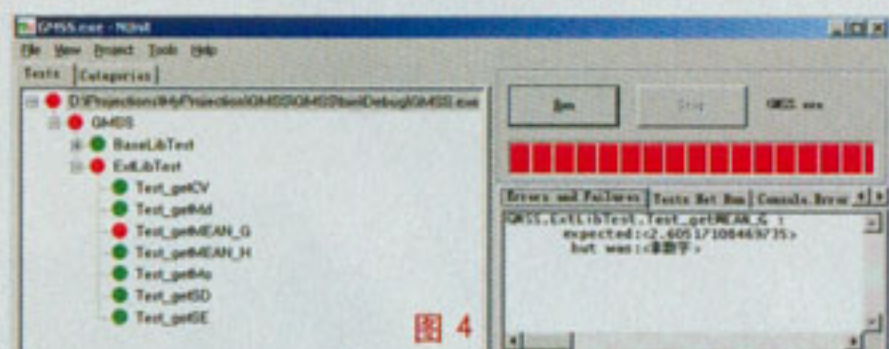


图 4

被声明为测试类的 Class 必须满足 3 个条件

- (1) 测试类必须是 public 类型
- (2) 测试类必须有缺省的构造函数
- (3) 测试类的构造函数必须能重复调用而不会影响其它组件

以上 3 点中前 (1) (2) 两点很容易满足, 而第 3 点需要特别小心注意。因为在网络会话的过程中需要重复构造测试类, 因此如果构造函数引入了改变其它组件的语句就会引发问题。

2. [Test] 标签表示的是某一个方法是 NUnit 的测试方法。使用方法是:

```
[Test]
public void TestMethod() {
    .....
}
```

被声明为测试方法的 Method 必须满足 3 个条件

- (1) 测试方法必须是 public 类型
- (2) 测试方法不可以返回值
- (3) 测试方法不能带参数

违反了上述规则的测试方法可以编译通过, 但是在测试执行的时候会被标识为未执行的黄色。

3. [TestFixtureSetUp]/[SetUp] 和 [TestFixtureTearDown]/[TearDown] 标签表示运行测试方法以前和结束以后要执行的方法, 可以类比为构造函数和析构函数的关系。除了满足 [Test] 标签的要求以外, 一个测试类里只允许有一个 [SetUp] 方法和一个 [TearDown] 方法。多个标签存

在时可以编译但无法运行测试。

4. [ExpectedException] 标签配合 [Test] 标签来测试一个预计会抛出异常的方法。使用方法是:

```
[Test]
[ExpectedException(typeof(
    DivideByZeroException))]
public void DivideByZero() {
    .....
}
```

这个测试如果抛出一个除零异常的话就会通过。

5. [Category] 标签与 [TestFixture], [Test] 标签配合使用来标识一个测试类或方法是属于哪一个测试组的。使用方法是:

```
[TestFixture]
[Category("GroupA")]
public class TestClass {
    .....
}

[Test]
[Category("GroupB")]
public void TestMethod() {
    .....
}
```

运行测试的时候可以选择某个特定的测试组运行而不是全部测试一遍。

6. [Explicit] 标签与 [TestFixture], [Test] 标签配合使用。使用方法是:

```
[TestFixture, Explicit]
[Test, Explicit]
public void TestMethod() {
    .....
}
```

使用 Explicit 声明以后除非特别指定运行这个测试, 否则将被忽略。

7. [Suite] 标签便于用户根据自己的喜好生成 Suites 的子集。同时为以后的版本提供兼容性。

8. [Ignore] 标签标识了一个 [TestFixture] 和 [Test] 暂时不运行。使用方法是:

```
[TestFixture]
[Ignore("Ignore a fixture")]
.....
[Test]
[Ignore("Ignore a test")]
public void TestMethod() {
    .....
}
```

这个标签的主要用途是当目前测试条件不具备时暂时不运行测试, 等将来测试条件具备时再去掉 Ignore 标签。以免忘记测试这个部分。

断言

与 JUnit 一样 NUnit 的主要功能也是通过“断言 Assert”来完成的。NUnit 给我们的断言方法有 8 种:

1. Assert.AreEqual // 用于比较两个数值是否相同。在运行例子中就是使用了这个断言。
2. Assert.AreSame // 用于比较两个引用是否指向同一个对象。
3. Assert.IsTrue // 与 Assert.IsFalse 一起用于判定真假值。
4. Assert.IsFalse
5. Assert.IsNull // 与 Assert.IsNotNull 一起用于判定对象是否为空值 null。
6. Assert.IsNotNull
7. Assert.Fail // 用于生成一个失败的测试, 一般用来构建自己的 Assert
8. Assert.Ignore // 用于运行时忽略某一测试。

以上 8 种断言方法每一个都有几种不同的重载形式, 可以根据实际情况的需要选用。

报告输出

NUnit 的 Tools 菜单有一个 Save Result as XML 的功能, 有了这个功能我们就可以把测试的结果保存为 XML 文件, 这个功能对于经常要提交测试报告的测试员来说可是一个福音。只可惜输出的测试报告是纯 XML 格式的, 看上去没有那么美观, 如果能够配上一个漂亮的 XSL 模板的话就再好不过了。

结语

通过对 NUnit 的接触, 我不得不为设计者的良苦用心所叹服, 一个移植版的软件能有这样的创新真的很难得。虽然现在 NUnit 远没有达到 XUnit 测试家族的水平, 但不能否认这是一款相当优秀的作品。希望大家能够好好地使用这个工具, 让代码的质量不断步步高升。

关于作者

陆复名, 目前在佳能控制系统(上海)有限公司工作, 从事工业控制信息系统的方面的开发和研究。

征订

登录 [HTTP://WWW.DEARBOOK.COM](http://WWW.DEARBOOK.COM)

订阅《程序员》、《MSDN开发精选》享受超值优惠!!!



个人订阅:

7.6折

2005年《程序员》全年12期.....91元

2005年《MSDN开发精选》全年6期+6CD.....82元

2005年《程序员》12期+《MSDN开发精选》6期+6CD.....173元

另外所有的续订户, 均可免费获赠最新杂志1期

企业订阅:

超值套餐

套餐一.....292元

2005年《程序员》全年12期

国外权威期刊中文精华合集5本

Dearbook VIP帐号

CSDN免费刊登人才信息(一个月)

套餐二.....280元

2005年《MSDN开发精选》全年6期(6CD)

国外权威期刊中文精华合集5本

Dearbook VIP帐号

CSDN免费刊登人才信息(一个月)

套餐三.....400元

2005年《程序员》12期

2005年《MSDN开发精选》全年6期(6CD)

国外权威期刊中文精华合集5本

Dearbook VIP帐号

CSDN免费刊登人才信息(一个月)

注:1.中文合集:

《Java Pro 2002-2003中文精华合集》

《Visual Studio Magazine 2002-2003中文精华合集》

《.NET & XML 2002-2003中文精华合集》

《asp.net Pro 2002-2003中文精华合集》

《Delphi Informant 2002-2003中文精华合集》

2.持有VIP帐号可享受以下优惠:

A)在Dearbook上购书可享受7.5折的优惠。

B)购买VIP专区中的图书可享受7.3折的优惠。

C)购书满100元可享受3元配送(普通用户需要满150元)。

有关VIP帐号的更多优惠信息请登录www.dearbook.com查询

Programmer
程序员

msdn
开发精选

联系人: 王小姐
咨询电话: 010-84540262

Email: wangyu@csdn.net
传真: 010-84540263

更多详情请登录 [HTTP://WWW.DEARBOOK.COM](http://WWW.DEARBOOK.COM)

TLFeBOOK

书评

再读《ASP.NET 揭秘》

■ 文 / 常可

微软的 .NET 技术家族是现今软件开发业界的舆论焦点，其主攻 Web 开发领域的战将——ASP.NET 也开始越来越广泛的个人秀……

So，越来越多的程序员——传统的 ASP 人，VB 人，Java 人，PHP 人都或多或少地开始接触 ASP.NET。虽然也是微软的产品，但平心而论，ASP.NET 的学习曲线并不平坦。“Web 控件”、“ADO.NET”、“数据绑定”、“HTTP 处理器和模块”等等太多的新概念和思想，数千个类名和无数的方法、参数，新的 Web 事件驱动模型……乱花渐欲迷人眼。

如果有一本教程般的书，简要但全面的覆盖到各个知识点，并且搭配大量完整的、实际可用的、短小精干的示例，可以放在机器边，一页页边读边实际演练，之后还可随时做参考，新技术的大门槛也就顺利迈入，再进一步登堂入室也不难了。

现今的出版业界不是一般的浮躁，一项技术一旦流行，大批相关书籍就开始“雨后春笋”。最早面世的当属 Wrox 的 ASP+预览系列，现在自然已没什么实用意义；随后有微软出版社的几本，覆盖高中低层次，读完后仅能进行简单应用的开发，文字上也和 MSDN 一样死板和公式化；还有 Wrox 的 Professional 系列，厚度很可观，可惜一是翻译糟糕，二是例子不好，要么不完整，要么太长和复杂；当然，把这几本的缺点都改正了的，符合前面提到的那些特点的书就是 Stephen Walther 的《ASP.NET Unleashed》。

这本书的第一版名叫《ASP.NET 技术内幕》——俗不可耐的名字，很容易淹没在书海之中。我是很偶然的从公司里的一位前辈那里看到。当时我对于 ASP.NET 已经基本入门，但很多具体的编码技术还不得知，譬如各种高级控件的使用，数据绑定控件如何使用其内置的编辑特性等等。这些东西不难，但是必须充分“过手”才能使开发工作熟练。

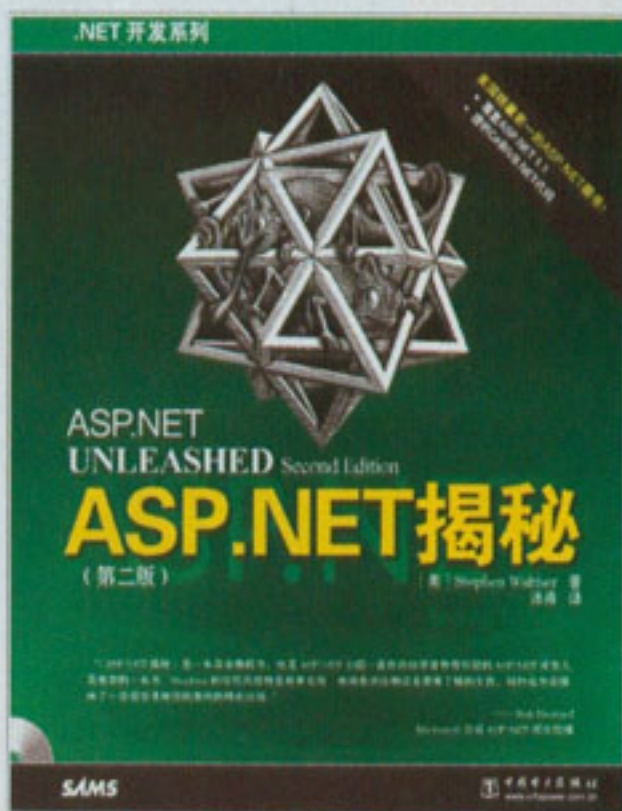
“Unleashed”这本书的最大特点就是内容全面，例子精到，实战性强。书的内容分为九大部分，ASP.NET 中大部分知识点难度上适合初学者，没有很多晦涩的原理解释，许多地方甚至是 step by step 的叙述。例如使用 ADO.NET 访问数据库的基本步骤；书中的例子连作者本人也用了“海量”来形容，几乎每一页都属于“最简”版本，没有可视化 IDE 附加的冗长代码，除了专门章节以外没有用 Codebehind 方式，并且每个例子都是完整

可用的，直接用文本编辑器敲好就能 run，不必再去掌握 Visual Studio .NET 的 Web 工程里那一堆附加框框。

电力出版社出的这本第二版，得到了微软 ASP.NET 项目组长的推荐，也标示了其“美国销量第一的 ASP.NET 图书”的尊贵身份，翻译方面我感觉中规中矩，本来原书的文字就很流畅清晰，译本也如此，读起来没有很多“洋味”，也没有令人费解的段子。这版在内容上变化不大，我感觉 90% 以上的内容可能只是第一版的勘误版本，新增了对 Starter Kit 的介绍，它们是很好的学习范例，书的作者 Stephen Walther 本人也是其中 Community Starter Kit 的作者。再就是附赠了光盘，包括书中全部代码（VB.NET 和 C# 版本，书中印刷出的代码是 VB.NET 的，也许很多 C#er 会不满，不过这两个语言用起来真的很像，我做毕设时没费多大力气就把 VB 代码翻成 C# 的了，似乎也就是类型转换和数据绑定那里有形式上的大区别），以及开发工具，Starter Kit 范例以及一篇讲 ASP.NET 1.1 和 1.0 区别的电子文档。

“Stephen 的写作风格就是简单实用。他将告诉你哪些是需要了解的东西，同时也为你提供了一些很容易使用的案例的精彩回放。”——Rob Howard，微软 ASP.NET 开发组的项目经理如是说。对于想快速系统全面地学习和掌握 ASP.NET 第一线开发技术的程序员，我认为这本书非常值得拥有。具备了基础和熟练的技巧，更上一层楼也就容易了。

■ 责任编辑：罗景文 (ljw@csdn.net)



软件工程师开发技术大全

12559 88 元



12514 68 元



12592 68 元

- 1.国内引进的第一本 Symbian 技术著作
- 2.数十位 Symbian 工程师的经验集成
- 3.手机应用开发的必备资料

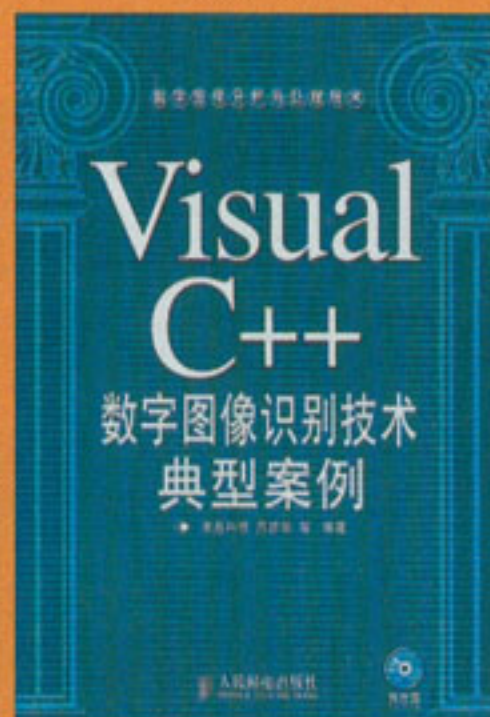
Cg 教程——
可编程实时图形权威指南

12430 38 元



算法的详细介绍和基于C/Visual
C++6.0 的实现代码

11770 72 元



12489 65 元

体会 Forms 编程
分享专家经验

新书上架



CSDN 每月精选上市的新书，由编辑和业界专家亲自点评。诚邀读者推荐新书！！

Windows Forms 程序设计

作者：[美]Chris Sells

译者：荣耀 蒋贤哲

出版社：人民邮电出版社

CSDN编辑评论：不要以为这是一本仅仅介绍Windows Form编程的书，Chris Sell 作为在 Windows 开发社区中备受尊敬的人物，鬓发斑白的时候依旧抵制不了.NET托管开发环境的诱惑，而本书正是在这样的背景下应运而生的。如果你的目标用户是Windows用户，那么此书将是你最好的选择，如果你是Web开发人员，那么其中的很多内容依旧值得参考，特别是设计期集成、资源、数据集与设计器支持等等。通过本书，Chris Sell 将向您展示托管环境下Windows Form开发的威力。

Delphi源代码分析

作者：周爱民

2004.9.16—2004.10.15 dearbook销售排行榜

- ↑ 计算机网络（第4版）（中文版）
- ↓ 精通 Struts：基于MVC的Java Web设计与开发
- ↑ Delphi 源代码分析（70折1周热卖，限北京总站）
- ↓ Microsoft .NET 框架程序设计（修订版）
- ↓ 设计模式——可复用面向对象软件的基础
- ↑ Java2 核心技术（第6版）卷1：基础知识
- ↑ OOD 启思录
- ↑ Delphi 模式编程
- ↑ 数据库系统概念（原书第4版）
- ↑ 重构——改善既有代码的设计（中文版）



出版社：电子工业出版社

CSDN编辑评论：刨根问底的精神永远都是值得学习的，本书作者周爱民就具备这样一种精神。从Delphi 1.0就开始对这项技术核心内容作如此深入研究的人，恐怕不在多数。而真正能够在这个领域内顽强坚持长达5年时间，更是让人值得钦佩。作者并未尝试将自己的作品和李维先生的《Inside VCL》进行比较，而是从纯学习的角度挖掘了Delphi核心，让您深刻体会到Delphi是“如何做到”而非“如何做”的。阅读本书，相信能让Delphi程序员明白许多从前在开发过程中遇到的疑惑。

Java 编程艺术

作者：[美]Herbert Schildt,

James Holmes

译者：邓劲生

出版社：清华大学出版社

CSDN编辑评论：本书作者之一 Herbert Schildt 对于许多90年代的C/C++学习者而言并不陌生，他的多本精彩著作曾引导许多人登堂入室。Schildt 是位多产作家，其作品全球销量超过300

万册。另一合作者 James Holmes 也是位Java顶尖高手，曾获得Oracle年度开发者称号。两位作者在书中为读者深入介绍了Java开发的多个应用场景，给出的代码范例非常精彩，而且实用。其中包括：用Java创建语言解释器、Email客户端、下载工具、Internet搜索机器人、各种统计图表、金融行业应用、AI算法实现等等，设计思想独特，非常值得学习。

深入 Java2 平台安全 ——体系架构、API设计和实现

作者：Li Gong, Gary Ellison, Mary Dageforde

译者：朱岱

出版社：电子工业出版社

CSDN编辑评论：本书是关于Java安全平台的权威且全面的指南，针对当前诸多领先科技企业所采用的Java安全技术进行了更新，以体现其关键的内容追加和版本修订。本书对Java安全体系核心机制的深入解析，描述即使在最苛刻计算环境中仍然可以成功实现的工具和技术。Java总是提供比其它平台更强的安全模型，而本书则回顾了用来增强安全性而不会牺牲功能特性的方法和实践。通过采用一些定制、扩展和优化Java安全体系结构的技巧，用户将能够使用所需的一切方法来保护其信息资产免受内外攻击。

技术凝聚实力 专业创新出版

博文视点 (www.broadview.com.cn) 资讯有限公司是电子工业出版社、CSDN.NET、《程序员》杂志联合打造的专业出版平台, 博文视点致力于——IT 专业图书出版, 为 IT 专业人士提供真正专业、经典的好书。

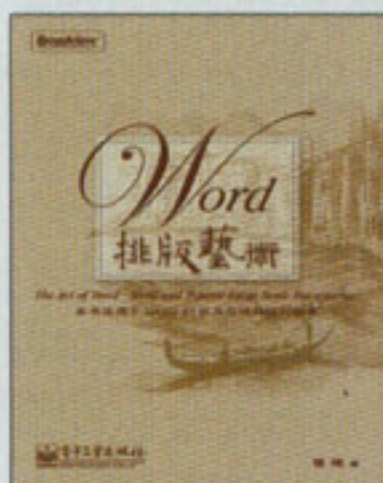
请访问 www.dearbook.com.cn

(第二书店) 购买优惠价格的博文视点经典图书。

请访问 www.broadview.com.cn (博文视点的服务平台)

了解更多更全面的出版信息; 您的投稿信息在这里将会得到迅速的反馈。

博文视点强力推荐



Word 排版艺术

告诉您 Word 的能与不能
作者: 侯捷
出版时间: 2004 年 10 月
ISBN: 7-121-00421-6
定价: 48.00 元
页码: 510
开本: 短 16 开



面向企业的软件研发管理 解决方案: 方法与工具

提供切合中小企业需求的软件研发管理解决方案, 帮助企业快速平稳地提升软件研发与管理能力。
作者: 林锐、范同祥、唐勇
出版时间: 2004 年 10 月
ISBN: 7-121-00434-8
估价: 40.00 元 (含光盘 1 张)
页码: 256 开本: 短 16 开

Java 技术大系



精通 JBoss——EJB 与 Web Services 开发精解

JBoss 和 EJB 开发技术的实践指南
刘洋、魏飞 等编著
出版时间: 2004 年 9 月
ISBN: 7-121-00182-9
定价: 49.00 元 (含光盘 1 张)
页码: 422
开本: 16 开

计算机专业人员书库



ERwin 数据建模

杨国强、路萍、张志军 等编著 CA 公司监制
出版时间: 2004 年 9 月
ISBN: 7-121-00304-X
定价: 55.00 元
页码: 423
开本: 短 16 开



精通 Visual C++ 图像处理编程 (第二版)

周长发 编著
出版时间: 2004 年 9 月
ISBN: 7-121-00309-0
定价: 49.00 元 (含光盘 1 张)
页码: 438
开本: 16 开



ANSYS 高级工程有限元分析范例精选

祝效华、余志祥 等编著
出版时间: 2004 年 9 月
ISBN: 7-121-00346-5
定价: 49.00 元
页码: 468
开本: 16 开

.NET 技术大系



C# 编程语言详解

The C# Programming Language
[美] Anders Hejlsberg, Scott Wiltamuth, Peter Golde 著
张晓坤、谭立平、车树良 译
出版时间: 2004 年 9 月
ISBN: 7-121-00228-0
定价: 55.00 元 页码: 482 开本: 16 开



Visual Basic.NET 应用程序和组件开发

Building Applications and Components with Visual Basic.NET
[美] Ted Pattison, Dr. Joe Hummel 著
韩江 译
出版时间: 2004 年 9 月
ISBN: 7-121-00259-0
定价: 45.00 元 页码: 354 开本: 16 开

我们拥有专业的编辑队伍
我们拥有强大的市场推广平台——www.csdn.net
我们依托于实力雄厚的电子工业出版社的销售平台
我们致力于推动中国 IT 专家原创作品

投稿邮箱: Editor@broadview.com.cn

Broadview 致力原创, 敬请投稿

读书的快乐



文 / 霍泰稳

读书对我来讲一直是快乐的事情，喜欢在书的海洋里漫游，任由思绪在书的引导下飘荡。我一直认为这样做没有什么不好，可最近读到的一本书让我改变了这个观点，意识到这样的快乐不是真正的快乐。这本书的名字是《如何阅读一本书》，由美国著名作家、学者艾德勒与范多伦合著而成。

行文至此，感觉作为本期Magalog的主持人有必要介绍一下自己。一个月前我还在杭州从事开发工作，偶然间看到《程序员》杂志社的招聘广告，以很快的速度将工作交接完毕，来到北京，很幸运地如愿以偿，开始了自己技术编辑的生涯。与从前从事开发工作时甚至一天都不说话的环境相比，编辑部的“谈笑有鸿儒”让我开心不已。编辑们在一起闲聊时的话题多与书相关，《如何阅读一本书》就是同事孟岩向我推荐的。

简洁的包装不说，行文的流畅不谈，其内容的实用真正让我从阅读时表层的快乐深入到内心深处。“这本书会在很大程度上改变我的阅读习惯”，这是阅读这本书时经常显现在大脑中的一句话。究竟会如何改变我的阅读习惯呢？我想从过去自己开发工作过程中阅读技术书籍时的得失谈起效果会更好一些。

很多朋友也许会疑问，《如何阅读一本书》究竟讲了些什么东西让你如此不惜

赞赏之词呢？接下我简单介绍一下此书的梗概，喜欢的朋友可以买来详读。

人分三六九等，物有好坏之分，读书也有不同的层次，艾德勒先生把阅读根据阅读目的的不同分为基础阅读、检视阅读、分析阅读与主题阅读等四个层次。基础阅读最简单，识字而已；检视阅读用略读形容更恰当一些，通过快速翻阅理解作者的写作意图；分析阅读是层次比较高的一种阅读方式，在阅读的过程中要能达到与作者心灵对话的程度；而主题阅读则是最难的，这种方式比较适合写论文时采用，通过对多个作者多本书籍的比较阅读从而获得心智的提高。在阅读过程中要积极主动，具有怀疑精神也是作者强调的一个部分。以一种积极的态度去阅读，在阅读的过程中多问一些为什么，然后根据书中的讲解或自己的阅历尝试着去回答，这样才会使阅读得到升华。

回头检阅一下自己过去阅读的书籍，发现多以书的前1/3部分结束，其余部分则崭新如初，甚至有些书籍是买来之后签上大名然后束之高阁再无问津。为什么会出现这种情况呢？因为在校期间学习成绩比较优异，工作之后也想在很短的时间内做出些成绩来，所以买来的书多为深度探索之类，这种行为的结果是显而易见的，地基不牢何以建

筑大厦。艰涩的术语，难解的程序，让自己一次次以失败告终。这种情况不是主动性不够，怀疑精神不足，结合《如何阅读一本书》的内容，可以理解为没有将自己放在一个合适的位置，没有将书的类别进行细分。做为一个初级的开发人员，如果立志学习Java语言，先去理解一下什么是面向对象，什么是线程等技术是一个不错的选择，否则翻开一本书，满眼的OO、Thread，很容易就会失去阅读的兴趣与积极性。印象中很清楚的是在自己学习JSP的时候，怎么都不能理解什么是脚本语言，现在说来真是惭愧，可当时的那种挫败感让自己很长一段时间都认为JSP是一门艰涩的东西。后来一个偶然的的机会做了一段时间网页设计，还学习了一段时间Servlet，返回来再看JSP书籍的时候，那种顿悟的感觉便忍俊不禁。

阅读是一门学问，阅读技术书籍更是一门艺术，能够通过一些恰当的技巧去学习，从中获得更深层的快乐，也从而使自己的心智得到提高，不是一件令人振奋的事情吗？读书是这样，工作、生活又何尝不是如此呢？以平静的心态审视自己，认清自己所处的位置，应该去做什么样的事情，积极主动地去探索，怀疑批判地去吸纳，那么我们的工作与生活岂不是也充满了快乐！



歪批 IT·漫画管理故事

ISBN 7-302-09741-0

著名管理学家、畅销书《水煮三国》作者成君忆推荐
《IT 经理世界》超人气专栏连载作者力作
被多家 IT 企业和 IT 用户指定为咨询顾问、销售人员、信息技术
人员必备手册

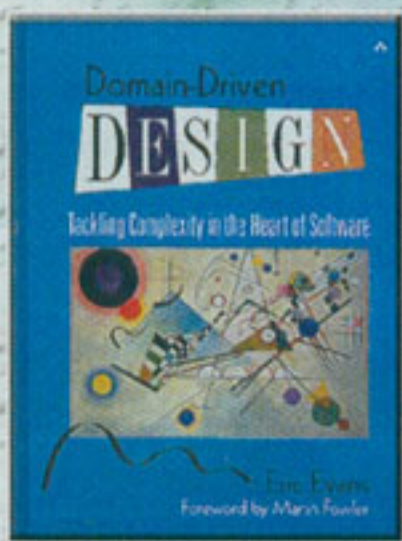
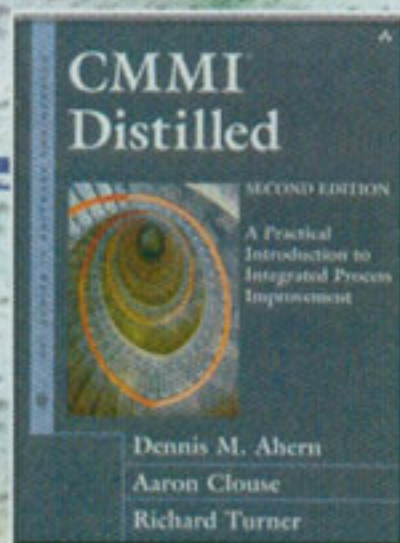
要了解详细信息, 请登录 <http://www.tupwk.com.cn/it/>。

CMMI 精粹 ——集成化过程改进实用导论(第2版)

预计中文版 12 月出版

CMMI 方面的权威著作

该书的第 1 版已经被广大程序员视为 CMMI 领域的宝典, 其中文版
也使很多中国程序员知道何谓 CMMI、如何应用 CMMI。第 2 版根据
CMMI 1.1 版、新的应用和开发环境, 修订了原版存在的不足之处, 添
加了很多实用性资料和面向新技术和需求的应用方法和经验。



领域驱动设计 ——软件核心复杂性应对之道

预计中文版 12 月出版

“每个有思想的软件开发者的书架上都应该有这样一本书”

—— Kent Beck

本书系统地介绍了领域驱动的设计思想和方法。在该书列选、翻
译和审校过程中, 得到很多专家的支持和高度评价。可以肯定的说, 该
书一定能够成为广大开发人员喜爱的经典著作。

软件剖析 ——代码攻防之道

预计中文版 12 月出版

“在有关软件脆弱点方面, 《软件剖析》是我所见过的最好的一本书。”

—— Aviel D. Rubin, 计算机科学教授

“如果不知道自己的处境, 就很难保护自己。这本书提供了攻击者
是如何找到软件漏洞并攻击软件的细节——这些细节将会有助于提升
您的系统安全性。”

—— Ed Felten 博士, 计算机科学教授



清华大学出版社

www.tup.com.cn

liwh@tup.tsinghua.edu.cn

来IBM 2004开发者大会，

IBM

与UML大师

James Rumbaugh亲密互动

时间:2004年11月11-12日

地点:北京香格里拉饭店



IBM developerWorks **Live!** 2004 开发者大会

The Technical Conference with More

无论你熟悉J2EE还是.NET;无论你使用Java还是C++;无论你擅长Windows、Linux还是UNIX平台的开发;也不论你是否是Open Source的支持者,所有的软件开发技术都将遵守同一个主旋律——整合。2004年11月11-12日,北京香格里拉饭店,“IBM 2004开发者大会”邀你快速就位,共谱整合之曲!

■ 整合之道尽在掌握

50多场精彩技术讲座,涵盖软件开发、运行、管理以及应用的全方位内容,让你洞悉软件开发的精髓,掌握IBM最新的整合架构理念。

■ 与UML之父面对面

倍受敬仰的UML创始人之一James Rumbaugh亲临现场,就整合时代的开发方法与理念,为你面授机宜。

■ 软件开发方法论全接触

业界权威,完整的IBM“软件开发生命周期”方法论,将最先进的业务逻辑、建模和设计思想向你完全呈现。胸怀全局,自然运筹帷幄、游刃有余。

■ 现场操作,高人指点

讲座之外还有上机操作,不但现场体验全程方法论的威力,更能亲得高人指点,与UML之父以及来自国内外的众高手论道,更是不可多得的宝贵历练。

所有参会者都有机会当场进行IBM软件全球专业认证免费热考,每项考试价值千元,良机岂容错过?一展实力,赢得全球喝彩!大会更备有光盘资料等超值大礼,让你满载而归!

注册与会或了解详情,请即刻登录www.dwlive.com.cn网站,“IBM 2004开发者大会”欢迎你!

ON 随需应变的业务