

Java 程序员上班那点事儿

前言

不久前，我曾在网上论坛上看到了一个这样的帖子：

大家给我个底吧，我是即将毕业的大学生，现在有软设的证书了。

主要擅长于计算机的维护、应用以及开发；

软件方面：现以通过国家软考中心软件设计师资格考试；

主要熟悉的 IDE 环境：VC++，C++BUILDER，VB；

主要使用的数据库接口：ADO；

主要使用的数据源：ACCESS，SQL SERVER，擅长使用 SQL 语言；

主要使用图形接口：OPENGL，对 DirectX 接口也有一定了解；

其它语言：JAVA（J2EE、J2ME）；

网络方面：熟悉 ASP，PHP，JAVASTRIPT，以及网络构架设计、施工、调试，对安全知识也有相当的了解；

系统维护方面：有三年的计算机维护经验，熟悉系统工作原理；

其它：熟悉 CAD、PHOTOSHOP 等几乎所有常用软件的使用，UNIX 系统的应用；

我投了很多简历，并面试了几家公司，目前只有一家公司出1500，大家说我去吗？

这个"准程序员"朋友的简历真可谓是"高、大、全"，所掌握的技术很是全面，几乎是无所不知无所不晓。然而，为什么只有一家公司愿意聘用他，而且才给他 1500 元的月薪？他很迷茫。

过了几天，有一个在软件公司开发团队做 Team Leader 的朋友想招聘 Java 软件工程师，找我帮忙推荐，招聘要求如下：

软件工程师（1人）

- 1、计算机或相关理工科专业；2年以上工作经验；
- 2、熟练运用 Java 等编程语言，熟悉其他编程语言如，VB\VFP 等者优先；
- 3、熟练使用 MySQL 等主流数据库，熟悉 Oracle 者优先；
- 4、对 Linux 操作系统熟练，熟悉 Solaris 者优先；
- 5、工作踏实认真，具备良好的团队合作精神和良好的沟通能力；
- 6、有独立项目开发经验者优先。

（私下透露月薪范围是4500-6000视个人能力而定）

他跟我说他们贴出这个招聘启示已经一个多月了，也没有招到合适的人，请注意这个招聘要求可比上面那位仁兄简历中写的技术少多了。

是什么原因让我们的用人单位总是招不到人，在其并不复杂的用人需求面前如此的困惑，又是什么原因让我们的"准程序员"在应聘工作岗位时四处碰壁呢？

是我们的"准程序员"们掌握的技术不够全面吗？还是用人单位岗位要求太复杂？显然都不是，现实情况却正好相反，那是什么原因造成的这种局面？

我告诉大家，无非是两点原因： 第一点，应聘者对自身没有一个清晰的定位，不是好高骛远，就是定位过低。 第二点，应聘者所掌握的技术知识和工作岗位的需求脱节，要么是学了用不上，要么是有用的没有学。这两点原因无外乎就是"对即将走向的工作岗位没有清楚的了解"。

如果，我们的"准程序员"能够做到"知己知彼"那么自然"百战不殆"了。

本书就是让我们即将走向程序员岗位的朋友们，能够做到既"知己"又"知彼"，在入职前能够对这个职业有一个清楚的了解，在入职后对职业方向有更清晰的认识，从而，使大家更加充满自信的走向工作岗位。

目录

《我工作 我 Java》 1

原名: 《Java 程序员要上班! 》 1

前言 8

第一卷 生存法则 9

第1章 知己知彼, 百战不殆 10

1.1 问题1: 这个职业赚钱吗? 赚多少钱 10

1.1.1 修炼第一层境界: 剑指四方, 试问天下谁与争锋 11

1.1.2 修炼第二层境界: 世界如此之大, 要学的东西很多 11

1.1.3 修炼第三层境界: 最深即最浅, 最浅即最深, 不过如此 12

1.1.4 点评"修炼三层境界" 13

1.1.5 回答这个问题 13

1.2 问题2: 大学毕业生找不到职业入口 13

1.2.1 工作经验都是什么经验 14

1.2.2 他们为什么面试的时候这么问? 15

1.2.3 程序员的笔试 20

1.2.4 回答这个问题 23

1.3 问题3: 跨行业真的这么难吗? 24

1.3.1 跨行业最难的是什么 24

1.3.2 跨行业的入口--原来从事行业的业务知识 24

1.3.3 农民造出了飞机, 为什么他还是农民 25

1.3.4 回答这个问题 25

1.4 秘诀: 经营自己的优势 25

1.4.1 善于发掘和积累自己的优势 26

1.4.2 善于展示与利用自己的优势 26

1.5 点评"经营自我" 28

第2章 软件开发职业的误区 28

2.1 误区1: 软件开发职业是青春饭 29

2.1.1 不必为30岁以后烦恼 29

2.1.2 30岁以后照样可以编程序 29

2.1.3 50多岁的程序员多的是 30

2.1.4 点评"青春饭" 30

2.2 误区2: 做软件开发必须要加班熬夜的工作 30

2.2.1 程序员们熬夜工作的借口 30

2.2.2 常态加班的危害 31

2.2.3 控制好工作的一日时间表才是关键	32
2.2.4 "偏执与狂热"不等于加班加点	33
2.2.5 程序员的大脑与第二大脑	33
2.2.6 点评"加班熬夜"	34
2.3 误区3: 开发一个软件产品一定是集团作战	34
2.3.1 几百人的公司其他人都在干什么	35
2.3.2 需要较全的人员配置的项目	38
2.3.3 坚信, 只要是"人"做到的我就能做到	38
2.3.4 点评"集团作战"	39
2.4 误区4: 程序员不是一般人	39
2.4.1 程序员的与众不同与平凡	39
2.4.2 程序员们就是一般人	41
2.4.3 点评"一般人"	42
2.5 误区5: 存在"软件蓝领"岗位	42
2.5.1 "软件蓝领"是个"美梦"	42
2.5.2 "让程序员放弃思考?"是个"噩梦"	43
2.5.3 软件开发流程"理想"的"不理想"	43
2.5.4 "软件蓝领概念"忽略的东西	46
2.5.5 现实中的软件开发团队	48
2.5.6 点评"拧螺丝"	50
2.6 关于误区	50
第3章 程序员的"菜鸟心态综合症"	51
3.1 症状1: 指望着公司里有一个老师	51
3.1.1 临床表现	51
3.1.2 治疗1: 弄明白她为什么辞职	51
3.1.3 治疗2: 理解公司的本质是什么	51
3.1.4 治疗3: 要搞清楚你的主管是你的老板, 他绝对不是你的老师	52
3.2 症状2: 不知道怎么让自己前进	52
3.2.1 临床表现	52
3.2.2 治疗1: 你只管低着头上山, 暂时不要向山上	52
3.2.3 治疗2: 明知山有虎, 偏向虎山行	53
3.2.4 治疗3: 没有过不去的火焰山	53
3.2.5 治疗4: 虚心使人进步, 骄傲自满要不得	54
3.3 症状3: 想做圈养的羊, 不想做野生的狼	55
3.3.1 临床表现	55

3.3.2 治疗1: 理解"丛林法则"生存的法则	56
3.3.3 治疗2: 向掠食动物学习如何生存	57
3.4 症状4: 缺乏自信, 总对自己说 No	57
3.4.1 临床表现	57
3.4.2 治疗1: 生活取决于自己	57
3.4.3 治疗2: 找到通往高楼的那扇门	58
3.4.4 治疗3: 逐个排除你恐惧的理由	58
3.5 症状5: 缺少幸福感, 内心总在跳跃	59
3.5.1 临床表现	59
3.5.2 治疗1: 比一比到底谁最幸福	59
3.5.3 治疗2: 要努力进取也要找到幸福的理由	60
3.6 症状6: 困兽心态, 焦躁与不安	60
3.6.1 临床表现	60
3.6.2 治疗1: 像猴子一样生活	60
3.6.3 治疗2: 多和团队成员沟通	61
3.7 症状7: 缺少感激心, 心存感激让你受益匪浅	61
3.7.1 临床表现	61
3.7.2 治疗1: 心存感激不等于低人一等	62
3.7.3 治疗2: 首先应该对你的领导心存感激	62
3.7.4 真心换真心	63
3.8 症状8: 不知道什么是"团队合作"	64
3.8.1 临床表现	64
3.8.2 治疗1: 分析在公司上班的三个目标	64
3.8.3 治疗2: 团队合作--就是团队主管的目标	65
3.9 点评"心态"	65
第4章 换位思考, 项目主管的招聘技巧	66
4.1 招聘就像大海捞针	66
4.2 技巧1: 是否有独立完成项目的经验	66
4.2.1 独立完成一个项目的经验是什么经验	67
4.2.2 没有独立生存的能力不能有最佳团队合作	68
4.2.3 测试手段	69
4.2.4 锻炼攻略: 需要主动寻找独立工作的机会	71
4.3 技巧2: 是否有独立解决问题的能力	72
4.3.1 见招拆招的能力	72
4.3.2 程序员的韧性	72

4.3.3 测试手段	72
4.3.4 锻炼攻略：训练创意思维	74
4.4 技巧3：评价程序员的思考方式	74
4.4.1 一切皆程序	75
4.4.2 找到规律	76
4.4.3 锻炼攻略：抓住中心把复杂的事情变简单	77
4.5 点评"换位思考"	79
第5章 程序员，保持你前进的步伐	80
5.1 程序员前进的四个阶段	80
5.1.1 第一阶段，找到一个编程语言去入门	80
5.1.2 第二阶段，用所这门语言去分析和推理	80
5.1.3 第三个阶段，新知识新技术的积累	81
5.1.4 第四个阶段，大道无形	81
5.2 学习与积累	81
5.2.1 找到好书，相当于找到一个好老师	82
5.2.2 每本书都没有从头看到尾是不是等于不用功？	84
5.2.3 学会建立沉淀目录	86
第二卷 制胜法宝	88
第6章 Java 程序员的七种武器	89
6.1 武器1：编程 IDE 开发工具	89
6.1.1 Team Leader 的嗜好	89
6.1.2 什么功能是程序员最需要的	89
6.1.3 下面这些 IDE 你都得到	93
6.1.4 点评"开发工具"	98
6.2 武器2：数据库系统	98
6.2.1 广告与市场的力量	98
6.2.2 不要盲目选择数据库，根据用途选择合适的数据库	99
6.2.3 以下这些数据库绝不能仅仅是"眼熟"	100
6.2.4 研究一下 JDBC 源程序	102
6.2.5 不熟悉数据库就会"绕远"	113
6.2.6 点评"真相"	117
6.3 武器3：Web 服务器软件	118
6.3.1 Web 服务器是如何工作的	118
6.3.2 支持 JSP 的 Web 服务器的原理	119
6.3.3 常用的 WebServer	120

6.3.4 研究一下 Web Server 的源程序	120
6.3.5 点评"深入研究"	124
6.4 武器4: 操作系统	125
6.4.1 让我们看看这个招聘启事	125
6.4.2 Java 程序员为什么需要研究操作系统	125
6.4.3 我们应该更关心操作系统的哪些方面	125
6.4.4 哪些操作系统我们要重点关注	126
6.4.5 点评"Linux"	131
6.5 武器5: 编程语言	132
6.5.1 Java 程序员只会 Java 语言行吗	132
6.5.2 各个编程语言的特长	134
6.5.3 点评"第二门语言"	136
6.6 武器6: 辅助设计工具	136
6.6.1 UML 图设计工具	136
6.6.2 常用 UML 设计工具	137
6.6.3 UML 要"灵活"的掌握	140
6.7 武器7: 版本控制工具	140
6.7.1 工作原理	140
6.7.2 常用版本控制工具	140
6.7.3 融入团队的开发氛围	141
6.8 点评"武器"	141
第7章 破除 Java 开发中的封建迷信	142
7.1 迷信1: Java 占内存到底大不大	142
7.1.1 测试一: 让程序去裸奔	143
7.1.2 测试二: 针尖对麦芒	146
7.1.3 让人不再"迷信"的测试结果	147
7.1.4 先天与后天	147
7.2 迷信2: Java 和 C 到底谁快	148
7.2.1 测试一: 让程序转起来	148
7.2.2 测试二: 读取个大文件吧	149
7.2.3 测试三: 内存处理的速度	152
7.2.4 测试结果分析	153
7.2.5 也不要过于迷信 C 语言	153
7.2.6 Java 语言与 C 语言之间的应用比较	154
7.3 迷信3: Java 就等于 JSP 吗	154

7.3.1 一个面试的现象	154
7.3.2 JSP 开发时间长了的误解	155
7.3.3 Java 的纯真年代	155
7.3.4 Java 绝对不等于 JSP	156
7.3.5 努力保持一个纯真的心态	156
7.3.6 点评"纯真"	156
7.4 迷信5: C/S 与 B/S 相比一无是处	156
7.4.1 B/S 是一个很好的创意	157
7.4.2 B/S 程序本身也是一个 C/S 程序	157
7.4.3 C/S 程序的优势--速度	158
7.4.4 C/S 程序的应用领域	158
7.5 迷信6: J2EE 的开发必须用 EJB	159
7.5.1 EJB 真人真事	159
7.5.2 我们不禁要问, 什么是"服务集群"? 什么是"企业级开发"?	160
7.5.3 把 EJB 掰开了揉碎了	160
7.5.4 EJB 的最底层究竟是什么	161
7.5.5 EJB 中所谓的"服务群集"	163
7.5.6 这种部署难道是无懈可击	164
7.5.7 EJB 活学活用, J2EE 不是必须使用 EJB	165
7.5.8 "技术"不是神, 不要动不动就"崇拜"	165
7.6 点评"迷信"	165
第8章 揭秘中大型应用系统	166
8.1 何谓"中大型应用系统"?	167
8.2 无法学习与模拟	167
8.3 资深程序员的"经验"	167
8.4 为什么要熟悉系统的运行环境	168
8.5 带你进机房里去看看硬件设备	168
8.5.1 机房的基本情况	168
8.5.2 "U"的概念	170
8.5.3 机房中的设备	171
8.5.4 在机房里我们发现了什么	175
8.6 安全与效率--永恒的主题	176
8.6.1 绝对安全是不存在的	176
8.6.2 RAID	177
8.6.3 负载均衡	178

8.6.4 双机、集群的配置模式	179
8.6.5 网络流量与速率	180
8.6.6 带宽	180
8.7 一个软硬件部署方案实例	180
8.8 点评"经验"	181
第9章 为什么要学习用框架开发	182
9.1 学习框架是因为它"火"	182
9.2 使用框架开发的好处	182
9.2.1 框架的目的是简化编程工作	182
9.2.2 框架是一个应用程序的半成品	183
9.2.3 框架的好处是代码重用	183
9.3 框架不仅仅只有"SSH"	183
9.3.1 WebWork	183
9.3.2 EasyJWeb	184
9.3.3 Click	184
9.3.4 JBlooming	185
9.4 用框架的思想去"自由思考"	185
9.5 自己也可以试着做一个	186
9.5.1 先看看不用框架怎么编写程序	186
9.5.2 从应用程序中找到共性的东西	188
9.5.3 我们试着做一个最简单的框架	188
9.5.4 有了这个框架开发工作被简化	193
9.6 点评"自由思考"	193
第三卷 达人策略	195
第10章 高手有多高菜鸟有多菜	196
10.1 五年工作经验的"菜鸟"	196
10.2 高手是怎样炼成的	196
10.2.1 修炼1: Java 悟道	197
10.2.2 修炼2: 关注程序的品质	197
10.2.3 修炼3: "技术"与"技巧"都很重要	211
10.2.4 修炼4: 走入 Java 的底层程序开发	212
10.2.5 修炼5: 从 Worker 到 Maker	226
10.3 点评"高手有多高, 菜鸟有多菜"	227
第11章 控制内存的功力	228
11.1 别指望 Java 和内存无关	229

11.2 容易被搞晕的--堆和栈	229
11.2.1 堆--用 new 建立, 垃圾自动回收负责回收	229
11.2.2 栈--存放基本数据类型, 速度快	229
11.2.3 何谓栈的"数据共享"	230
11.2.4 实例化对象的两种方法	230
11.3 内存控制心中有数	231
11.3.1 两个读取内存信息函数	231
11.3.2 开发 Java 程序内存看的见	231
11.3.3 必须要介绍的虚拟机的参数"-Xmx"	232
11.4 内存控制效率优化的启示	234
11.4.1 启示1: String 和 StringBuffer 的不同之处	235
11.4.2 启示2: 用"-Xmx"参数来提高内存可控制量	237
11.4.3 启示3: 二维数组比一维数组占用更多内存空间	237
11.4.4 启示4: 用 HashMap 提高内存查询速度	239
11.4.5 启示5: 用"arrayCopy()"提高数组截取速度	241
11.5 内存垃圾回收问题	243
11.5.1 什么是内存垃圾, 哪些内存符合垃圾的标准	244
11.5.2 JVM 垃圾回收的相关知识	246
11.6 点评"功力"	247
第12章 产品和项目是程序员永恒的主题	247
12.1 什么是项目, 什么是产品	247
12.1.1 "产品"的定义	247
12.1.2 "项目"的定义	248
12.1.3 "产品"和"项目"的区别	248
12.2 软件产品开发是"艺术"	249
12.2.1 软件产品开发需要灵感	249
12.2.2 程序作品是你的一个传世的艺术作品	250
12.2.3 软件产品开发需要"前瞻性"	251
12.3 软件项目开发是"军事行动"	253
12.3.1 开发者就是这个程序的"三军统帅"	253
12.3.2 "项目"开发需要"运筹帷幄"	253
12.3.3 项目控制, 一艘船的故事	254
12.3.4 点评"军事行动"	255
第13章 非技术知识对工作的辅助	256
13.1 辅助1: "英语"不需要专业, 因为它只是工具	257

13.1.1 英语与编程无关	257
13.1.2 用英语可以看一些英文文档	257
13.1.3 掌握基本的工作交流时的英语词汇	257
13.2 辅助2: "Google"不是万能的, 但不会用万万不能	261
13.2.1 在网页标题中搜索关键字: intitle	261
13.2.2 在特定站点中搜索关键字: site	261
13.2.3 在 url 链接中搜索关键字: inurl	262
13.2.4 精确匹配搜索: 双引号	262
13.2.5 搜索结果中不希望含某特定查询词: 减号	263
13.3 辅助3: 程序员的常用文档写作	264
13.3.1 程序员在软件开发过程中需要提交的文档	264
13.3.2 程序员在日常工作中需要提交的文档	265
13.4 点评"非技术"	265
第14章 结束语	266
第15章 本书简介	267
第16章 相关人士对本书的赠言	268

第1章 知己知彼，百战不殆

从你决定迈出校门进入社会的那一刻起，你就进入了一个战场，这个战场虽然没有硝烟弥漫，没有炮火纷飞，但却绝不亚于任何一个真正的战场。你要在这个战场上去搏杀，去竞争，利用各种可以利用的手段去赢取战斗。在这个战场上，没有人会因为你的弱小而给予同情，也没有人会因为你是一个新手而手下留情，在这里你会感受到在学校里从来没有感受过的"残酷"，你将深深的体会到一句话，那就是"优胜劣汰"。

我们能在这个战场上获胜的法宝之一就是"知己知彼，百战不殆"。我们如果能够在战斗前对双方的情况了如指掌，那么，取得胜利将会成为必然。

知己：要对自身的情况了解，要找准自己的定位。这个定位是自己对自己充分了解的情况下进行的思考，对于准备迈向程序员职业的战士们来说，进行这个定位是绝对必要的。

知彼：要对未来工作岗位的一切做到尽可能的清楚，这就需要对你的对手有非常清楚的了解。

1.1 问题1：这个职业赚钱吗？赚多少钱

Java 程序员这个职业赚钱吗？能赚多少钱？

我们刚刚进入本书的正题就拿出一个俗不可耐"钱"字来和大家大谈特谈，未免不雅。但是，我还是要在一开始就要说这个问题，因为这是很多朋友关心的问题，为什么不先说？

有很多即将进入这个行业的年轻朋友都很想问这个问题。

认真的面对这个问题，我们的回答是：

先不要急着问能赚多少钱，先要想想你为别人能提供什么服务。

任何一个职业都很赚钱！而且，都可以赚很多钱，想拿高薪不一定非要做程序员。我不是在这里卖关子，这是真理，"三百六十行，行行出状元"。

那么这时，有些朋友听到了我说的这些话也许会很失望，其实也不要失望，这个职业有让你赚到高薪的机会，只是，赚高薪是在什么时候，或者是在程序员的哪个所属层级。

我们来了解一下程序员的修炼三层境界，了解一下这个内容会比较容易抓住本书中的内容要旨。

1.1.1 修炼第一层境界：剑指四方，试问天下谁与争锋

修炼第一层境界的程序员，对 Java 开发技术尽数掌握，开发工具掌握的也较为娴熟。可以将第二层次程序员交给的任务完成的很出色，可以按要求独立完成类，接口和算法的开发。注重技巧，对具体的编程语言非常熟悉。

能力之所及，皆无不用其极，认为所有开发知识，越是看起来深奥的越值得去研究，希望在自己开发的所有项目中，能用上的技术全用上，目的只有一个，就是尽可能多的获得实践机会。总想四处试刀，看看手里的刀到底快不快。满口都在谈，什么框架是最优秀的，C#和Java的优劣，满脑子想着如何将一个程序编写的更复杂。热衷于探讨技术问题，甚至有可能因为一个开发观点而和别人争论的面红耳赤。

在编码中，经常可以看到他们会这样写程序代码：

```
if (a>0)
a++;
else
b++;
```

他们非常想证明自己掌握技术的娴熟程度，没错，他知道这个知识，在这里可以省略大括号。

这类程序员大有"剑指四方，试问天下谁与争锋"的气势，工作具有活力，常常因为一个技术细节加班到深夜，大多属于拼命三郎型。如果项目不能让他们学到他们想要的东西，他们会放弃这些项目，去投靠别的公司，跳槽对于他们来说很平常。

第一层境界特征：

工作时间：三年内

工作任务：按要求编写类和接口的具体实现代码

工作内容：编写具体的代码

开发目标：无所不能

开发特点：注重技巧，对具体的编程语言非常熟悉

工作职位：初级程序员，程序员，软件工程师

参考薪金：¥2000-¥6000（仅供参考）

1.1.2 修炼第二层境界：世界如此之大，要学的东西很多

修炼第二层境界的程序员，他们往往是从事了Java开发好几年了，从第一层境界进阶上来的好手，即，没有被优胜劣汰掉的那批人。之所以说"没有被优胜劣汰"这么"残酷"的用语，并非危言耸听，因为，一般修炼第一层境界是非常艰苦的，没有坚强的意志，没有强健的体魄，完全不可能进阶到第二层境界，也就是说，要头脑始终保持清醒，抱有坚定的信念，同时，你的身体也要非常好，才能过关。

我们经常看到，在这一关掉队的人们，由于志向偏离，或者吃不了苦，或者对困难估计不足，甚至是身体原因放弃软件开发职业。这里之所以强调"身体"，也是因为，第一关是如此的残酷，其实并没有人要求你加班加点，只是你的一腔热血使然。

所以，我们说在第二层境界中是"没有被优胜劣汰"的那批好手。

他们已经经历了若干个开发产品或项目，已经可以利用自己的知识去带领第一层次的程序员开发项目，可以说是一个很有经验的开发者，对在上一个层次阶段没有完全理解的技术知识已经相当的清楚。可以自由的运用开发技术，并分的清楚什么技术用在什么地方。

最让他们头痛的是项目的"工期"和"Bug"，根本无暇顾及什么技术实践的问题。他们往往利用自己最擅长的架构方法去开发和设计整个程序的技术架构。

在编码中，经常可以看到他们这样编写程序代码：

```
if (a>0){  
a++;  
}  
else{  
b++;  
}
```

他们老老实实的加上了"{}"大括号，因为他们知道，这些细节造成的Bug有可能让他们花去数天时间去调试，而这个省去的大括号，根本不会给系统带来任何优化。

他们知道的越多，越觉得世界是那么的广阔，不禁叹息"世界如此之大，要学的东西很多"，对Java 开发技术方面的探求知识，大多是在产品的架构层面，更愿意去研究架构设计方面的知识，比如，他们很清楚什么时候使用EJB，什么时候该设计什么样的一个接口。

他们逐步感到，Java 技术已经不能满足他们工作成功率的需求，他们不得不去花时间去研究项目管理的方法，对总体的技术关注点也从Java 的具体开发技术，逐步的向和Java 无关的其他信息技术方向转移，比如，网络应用层协议，其他平台语言，甚至Linux 内核裁剪等问题逐步纳入他们的视野。

后来他们会发现，他们想进入修炼的第三层境界的阻碍，恰恰是他们较高的技术水平。

第二层境界特征：

工作时间：工作三年以上，或直到退休

工作任务：按开发要求编写并指导第一层次程序员开发

工作内容：带领开发团队，设计架构，并编写关键程序，保证项目工期，对某开发项目的质量负责
开发目标：不求有功但求无过，质量效率胜于一切，合理的技术用在合理的地方
开发特点：注重方法，不关注编程语言细节
工作职位：高级软件工程师、开发经理、系统架构师、项目经理等
参考薪金：¥6000-¥15000（仅供参考）
1.1.3 修炼第三层境界：最深即最浅，最浅即最深，不过如此
修炼第三层境界的程序员，他们通常是在第二层境界"突破自我"之后进阶上来的有智慧的人，所谓"突破自我"就是打破自己而脱掉原有的蝉壳，破壳而出获得新生的过程。
在修炼的第二层境界已经将技术水平练就的如火纯清，甚至个别技术可以用"登封造极"来形容，有自己的一套"绝活"，可以说，靠这些本事在业界应该说是过着"衣食无忧"的生活。
他们逐渐发现，技术永远是技术，原来一直认为最深的技术恰恰是最简单的，而原来最简单的那些技术恰恰是最值得去研究的，其实那些所谓的"登封造极"对于他们来说，仅仅是利用他们所掌握的"原理级"技术，将"应用级"技术进行不同的排列组合而已。
任何"应用级"技术在他们眼里，没有任何区别，他们看着那些被业界炒作的"如火如荼"的技术，像什么，EJB 啊，开发框架啊，或者对于开发语言，什么 Java 啊，Vb 啊，C 啊，或者其他什么语言，基本上是很"淡然"，既不觉得如何好，也不觉得如何不好，只会淡淡的说一句"不过如此"。
所以，他们要突破，他们要进阶，面对他们的是更加广阔的空间，然而，他们会逐渐发现，他们进阶的桎梏恰恰就是原来自己的"优势"，较高的技术水平，使他们更难"抛弃"或"摆脱"。
突破自己的方法就是从技术中跳出来，利用"应用级"技术的不同排列组合去创造，去创新，这些创新要紧密的结合市场，要紧密结合应用业务。
他们不仅仅要具备很好的技术知识水平，还要具备更敏锐的产品洞察力，和更灵敏的市场嗅觉，并能够将这些能力充分的发挥并输出技术与市场都响当当的创意。
最终他们成功了，达到了程序员修炼的第三层境界，他们已经突破了原有程序员的传统概念，达到了在业界"横行无阻，任意驰骋"的能力，这就是程序员修炼的最高境界。
第三层境界特征：
工作时间：工作六年以上（经常跳槽的不算）

工作任务："应用级"技术的不同排列组合，以市场为导向去创新与创造

工作内容：面对市场背靠技术开发团队， 指导开发的市场着眼点， 指导市场的获利方式

工作目标：盈利

工作职位：部门经理、研发副总、CTO、解决方案专家、业务专家、产品经理、高级产品经理等

参考薪金：¥15000 以上（仅供参考）

1.1.4 点评"修炼三层境界"

王国维在《人间词话》中对人生三境有如下阐述，"古今之成大事业、大学问者，必经过三种之境界。'昨夜西风凋碧树，独上高楼，望尽天涯路'，此第一境也。'衣带渐宽终不悔，为伊消得人憔悴'，此第二境也。'众里寻他千百度，蓦然回首，那人却在，灯火阑珊处'，此第三境也。"

又有人说人生境界如陶渊明《桃花源记》中所写的"初极狭，才通人。复行数十步，豁然开朗。"

程序员的发展路径， 就是程序员人生路径， 从最初的"看山是山， 看水是水"， 到后来见的多了悟到"看山非山，看水非水"，没想到最终发现"看山还是山，看水还是水"。

1.1.5 回答这个问题

言归此问，"这个职业赚钱吗？赚多少钱？"， 我想我已经回答了这个问题， 世界上没有天上掉下来的馅饼，也不会掉下一个林妹妹。

任何事业，均需如下才可成功：第一要立志，第二要思考，第三要奋斗。

1.2 问题2：大学毕业生找不到职业入口

大学应届毕业生， 现在找工作是如此之难， 本书不回避这个问题。 目前的情况与笔者当年毕业时的情况有天壤之别。这个问题说起来是一个大话题，关系到"社会"、"经济"、"文化"、"教育"等诸多领域，并非一言以蔽之的事情，对于国内教育体制与社会用人需求脱节的问题，大家也已心知肚明。

本书也没有奢望能够在本节将其讲的很清楚， 只希望， 能够结合本行业的具体情况给出一个理由，以及给处于此阶段的同学们一个解决办法而已。

话说到，"大学应届毕业生， 现在找工作是如此之难"这一问题， 从本行业出发， 不负责任的人，无非会给出一个似是而非的解释"缺少工作经验"。

乍听起来，好像是很有道理，但仔细一想，简直是"废话"。

应届毕业生哪里来的工作经验呢？

如果，按此逻辑，凡是毕业生通通在待业，因为，始终没有工作过，哪里来的工作经验，所以永远找不到工作啦。

认真的面对这个问题，我们的回答是：

缺少应聘该职位所必需的技术或者能力。

为什么这么说，要知道，并不是只有应届毕业生找工作难，有"工作经验"但"缺少应聘该职位所必需的技术和能力"的人找工作同样难！

所以，要想解决这个问题，作为我们广大应届毕业生同学，必须要弄清楚"应聘该职位所必需的技术和能力"都是什么。

即，工作经验都是什么经验。

1.2.1 工作经验都是什么经验

我们刚才已经说了，所谓的工作经验就是"应聘该职位所必需的技术和能力"，那么这个技术和能力又具体指的是什么呢？

我们只从行业出发，来剖析这个工作经验，他是包含两个方面的问题，即"技术"和"能力"。

1.2.1.1 首先说说"必备技术"

这时有些同学可能会说，"我已经会很多程序的开发技术啦，Java、C#、VB，都会呀，这些技术难道还不够吗？"

我可以毫不迟疑的告诉你，"不够！"。

我们再回过头来看一下，我们在"前言"中提到的那个应聘简历：

主要擅长于计算机的维护、应用以及开发：

软件方面：现以通过国家软考中心软件设计师资格考试；

主要熟悉的 IDE 环境：VC++，C++Builder，VB；

主要使用的数据库接口：ADO；

主要使用的数据库原：Access，SQL Server，擅长使用 SQL 语言；

主要使用图形接口：OPENGL，对 DirectX 接口也有一定了解；

其它语言：Java（J2EE、J2ME）；

网络方面：熟悉 ASP，PHP，JavaScript，以及网络构架设计、施工、调试，对安全知识也有相当的了解；

系统维护方面：有三年的计算机维护经验，熟悉系统工作原理；

其它：熟悉 CAD、Photoshop 等几乎所有常用软件的使用，UNIX 系统的应用；

这个简历中，几乎将目前信息系统开发的所有技术都列出来了，生怕用人单位会因为自己的技术不全面而不给他面试机会。

而结果是，仅仅有一个公司让他去面试了。

我们这里所说的技术，并非指的是"广"，而指的是"精"。

倘若这位同学真的搞定了那些技术，我想，每项技术能达到用人单位的要求，都至少需要1年时间，那么，在他的简历中提到的技术大家可以数一数，至少有10种，简单一算，全部掌握需要多长时间呢？

回答：10年！

那么，我请问，这位应届毕业生同学，哪项技术可以单独拿出来工作呢？

有经验的主管们，一眼便知，此君为"应届"。

因此，我们广大同学应该在所掌握技术的深度和精度入手，那才是用人单位最需要的。

1.2.1.2 再说说"必备能力"

说到能力，我们的很多应届毕业的同学们都愿意给自己的评价是"我的学习能力很强！"或者是"具有良好的学习意识"等等。

请看这是某君简历中的自我评价：

本人性格开朗，做事认真，富有开拓精神，不怕挫折，具有良好的团队意识！具有良好的身体和心理素质，有较强的学习意识和自学能力。作为IT行业的一员，我愿意花费更多的时间，不怕困难，努力提高自己的专业水平！

他的这段话看起来似乎还不错，如果看这个自我评价的是他的班主任，一定会感到很欣慰--真是一个勤奋好学的好学生！

然而，仔细看来，他这段话的重点是"自身学习能力和学习意识"，以及"自我提高的强烈愿望"，他很显然很想让用人单位知道他在校的学习成绩是多么的优秀，或者生怕用人单位觉得他在校期间学习并不优秀，总之，他千方百计的要突出自己的学习能力。

说到这里，肯定有些同学会很不服气，会说 "突出学习能力强和自我提高的强烈愿望，这不正是说明他或她是一个好学生，有什么不对吗？"

当然不对啦，如果我作为用人单位，我请问你，"你学习能力强，自我提高的愿望那么强烈，和我们公司有什么关系"，很显然你工作的第一愿望还是提高自己的技术水平，换句话说，你一定是对这份工作心里没底，希望借用"学习能力强"来告诉用人单位"我现在虽然什么都不会，但是我学习能力很强，很快我会学会的"。

我在公司中经常听到的一段对话是：

A 君：.....。

主管："你到这里的目的是学习？还是工作？如果你的目的是学习的话，你应该给我学费，而不是管我要工资。"

说这话确实有点严厉，但是，这恰恰体现出了用人单位对人才能力方面的需求，这个需求就是，"创造价值"的能力，而不是你"自我提高和学习"的能力。

1.2.1.3 结论，工作经验是什么经验？

工作经验是：具备职位所需技术的精度和深度，最好是对那个技术非常精熟，具有为公司服务的意识，有为公司创造价值的能力，至少有为公司节约成本的能力。

通过以上分析，了解了什么是用人单位的招聘真实意图，这样，我们就不难写出符合用人单位需求心理的简历了，同时也就知道了自己的努力方向。

供求关系一旦吻合的时候唯一出现的现象就是"频繁的接到面试通知"。

接下来，我们就要面对两件事"面试"和"笔试"。

1.2.2 他们为什么面试的时候这么问？

面试是一个"简单"而又"复杂"的事情，正因为它具有"简单"和"复杂"的双重性质，才使我们对这个问题不敢掉以轻心。介绍如何面试，有时候甚至可以写一本书，而有时候，你什么都不准备却可以面试成功。

因为，面试的成功与否完全取决于主考官，因此，这门学问的主要科目就是研究主考官的招聘心理。

我们研究好主考官的招聘心理，自然就可以做到"知己知彼"了。

本小节只是说说面试的"心理战"，而想真正的能够对答如流，需要的是"真功夫"，要想具备"真功夫"还是需要真正的技术水平作为前提的，本小节的内容只是给那些已经具备相应技术水平的应聘者提供一定的应聘技巧而已，以避免没有把自己的"真功夫"完全展示出来从

而丧失工作机会。

面试前对自己的心理暗示:

面试并不是考试, 只是和未来的同事聊聊天。

"心理战"对象, 可能出现的主考官如下几类:

人物1, 人力资源部主管(HR)

人物2, 你未来的主管

人物3, 你未来主管的主管

我们分别来分析遇到不同类型的主考官的不同情况。

1.2.2.1 人物1: 人力资源部主管(HR)

"人物1"的出现往往是进行该职位的初审, 给出一个概观定论, 如果合格将会提交给 "人物2"。"人物1"他们所要进行的是对人的心理和基本技能方面的一个判断。

不过, 也有一些公司, 首次面试仍然是由业务主管来进行, 然后再将初审合格的人交给人力部门来复试, 如果, 是这种情况, 你应该就算90%入职成功了, 因为, 这个复试往往是走个形式, 看看此人有没有被主管忽略的大问题, 如果没有, 基本就差不多了。

那么, 我们仅仅以第一种情况为例, 看看"人物1"大多提出的是哪些问题。

1.2.2.1.1 常见提问1: 请你自我介绍一下

这个问题, 是人力部主考官必问的问题, 这个问题的提问并不是真的想了解你的个人情况, 因为, 你的情况基本上在简历上都写着呢。他提出这个问题的主要目的是来考察你的语言表达能力, 和你在表达过程中的一些细节表现。

所以, 我们应该怎么回答呢?

看看如下对话:

HR: "请你自我介绍一下"

A 君: "您看简历吧, 基本上我都写在简历上了。"

HR: 汗...

HR: "请你自我介绍一下"

B 君: "我叫 XXX, 年龄24, 性别男, 籍贯....."

HR: 倒...

HR: "请你自我介绍一下"

C 君: "这话从何说起呢? 话说10年前....."

HR: 晕...

首先，不要认为主考官没有认真的看你的简历，没有看你的简历就让你来面试，是在浪费他自己的时间，所以绝对是首先认为简历比较合适，才约你来的。

第二，自我介绍并不是让你重复一下你简历上的所有内容，那些内容简历上都有，主考官主要是想听听你如何表达和语言的逻辑能力。

第三，这个表达不要滔滔不绝，要有张有弛，有收有放，主要将自己的想说的优势部分分别道来，能够通过你的介绍让人感觉到你的"亲和力"为佳。

"语言表达能力"并非我们日常所说的"能侃"或者"口才"，这是片面的理解，在面试过程中，我们要展现的"语言表达能力"是指，"语言亲和力"，能够让人感受到你的"沟通"能力。

回答范例：

您好！我来自XXX 大学，是应届毕业生，所学专业是计算机应用技术，在校期间参加多项课余工作，参与了多个应用系统的开发与设计，熟练掌握Java 开发工具和应用系统的各种开发方法。在学习与工作期间，总结与实践了各种技术实现手段，有了一些小的积累。

近期已经毕业，看到您公司的招聘信息，感觉自己的技术与能力非常合适，就投递了简历。希望能过了您这关，呵呵.....(注：最后的微笑最好不要太牵强，目的是缓和气氛)

大家看到了，这个回答范例的自我介绍非常简练，但是，什么也没有落下。

既实事求是的说了自己是应届毕业生，又说明了自己虽然是应届毕业生，但是参加了很多的课余工作，并且具有实际的项目开发经验，而且有了一些小的积累。

既说了自己的对这份工作和自己能力之间的考量，又表达了自己对主考官的期望。

意思表达时不卑不亢，言简意赅，让人听起来就感觉很舒服。

这时候你的目的就达到了，展现出自己的"语言表达能力"和"语言亲和力"，同时又能够让人感受到你的"沟通"能力。

1.2.2.1.2 常见提问2：你最大的优点是什么？

这个问题如果是 HR 问，则最好回答了。

回答这个问题的关键是"围绕自己的技术特长"展开话题，为什么这样？

因为，往往 HR 都不懂技术，围绕技术说自己的特长很容易给他说晕（当然，个别懂技术的 HR 除外）。这里要注意的是，在说技术问题的时候，不要让HR 感觉自己什么都不懂，要注意说话的节奏，不要太快，不要太骄傲。

1.2.2.1.3 常见提问3：你最大的缺点是什么？

这个问题是 HR 的杀手锏，可以说这是HR 的狠招，这个问题最难回答，一般应聘者都本着"扬长避短"的心态去面试，冷不防冒出这么一个问题，还真是挺棘手的。

需要清楚 HR 问这个问题的目的，其目的仍然不是要真的需要知道你的缺点是什么，还是看看你的表达能力，尤其是需要考察应聘者面对危机的时候的处理能力。

所以，我们不要用下列方式作答：

说出自己的真实缺点，尤其是在前面谈话中没有暴露出来的缺点

认为说说某些大众化的缺点即可，认为说一两个无妨

说自己没有缺点，强调自己比较完美

总之，HR 心里想的是：需要了解他面前的这个人在面对困难的时候，是如何处理问题的，从处理危机的方法来判断此人的处理事情的灵活性。当然，如果此人自己暴露出自己的缺点当然更好，省得需要去想办法问更多的问题去发现了。

所以，在回答这个问题的时候，要看起来"真诚"、"坦白"，同时，说出来的并非自己的缺点，而是最好在别人看来是优点的那些方面。

这个问题问的概率很大，通常如果求职者说自己小心眼、爱忌妒人、非常懒、脾气大、工作效率低，肯定不会录用你。HR 喜欢求职者从自己的优点说起，中间加一些小缺点，最后再把问题转回到优点上，目的还是突出自己优点的部分。

HR 喜欢聪明的求职者。

这一点比较难掌握，我们也给出范例：

回答范例：

呵呵，这个问题好难回答啊！我想想.....（亲和力表现，也缓解了自己的紧张情绪）

我的缺点是，比较执着，比如在技术方面比较爱钻研，有的时候会为一个技术问题加班到深夜。还有就是，工作比较按部就班，总是按照主管的要求完成任务。另外的缺点是，总在自己的工作范围内有创新意识，并没有扩展给其他同事。这些问题我想我可以进入公司以后以最短的时间来解决，我的学习能力很强，我相信可以很快融入公司的企业文化，进入工作状态。

嗯.....,我想就这些吧。

这个回答范例开头第一句话就让人觉得很自然，因为这个求职者所说的话恰恰表达了一

一般人听到这个问题后的心理状态， 还有你一定会有一个思考的时间， 因为， 谁也不会立刻说出自己的缺点。

后面说出的几个缺点都是一环套一环的，说了自己"比较执着"，但又说自己其实是"比较爱钻研"，说自己总是"按部就班"，但又补充了其实那是"按照主管的要求完成任务"，这时候，如果用人单位觉得，此人是不是没有"创新思维"的时候，马上就补充道"在自己的范围内有创新意识"，至于"没有扩展给其他同事"这件事，其实无关紧要，干脆就卖给HR 吧。

以上回答确实卖弄了些"技巧"，相信 HR 也一定能看的出来，但是，即使看出来了也无妨，HR 也会心领神会，知道你是一个比较善于沟通并且善于表达的人。

1.2.2.2 人物2：你未来的主管

当见到未来主管的时候，往往是应聘者已经过了HR 那一关，或者应聘者已经过了笔试的那一关，因此见到这位人物意味着距离成功已经向前进了一步。

"人物2"的面试也有他的目的，他是和你在日常工作中接触最多的人，作为你的直接上司，他需要在工作中经常给你分配任务，他需要对他的主管负责，因此，他招聘的这个人选必须是可以帮助他完成他整个 Team 的目标的人。

往往那个吸引你来面试的"招聘启示"就是这个人物所撰写的，因此，其实在你和他见面以前，早已经通过"招聘启示"和他有过交往了。因此，从"招聘启示"中就可以初显这位主管的端倪。

注意，主管同志并不是人事领域的高手，不会用各种语言技巧去发掘你身上的缺点或者优点，往往问题都是实打实的，或者比较一针见血的，而且，更偏重于日常工作。

那么，我们下面和他过过招。

1.2.2.2.1 常见提问1：请你自我介绍一下

这个问题，HR 也问过了，到他那里有可能还会问，主管问这个问题和HR 虽然问的问题一致，但是，其目的并不是完全相同的。

他不仅仅想考察一下你的表达能力，同时还想考察一下你思路的清晰程度。

我们在回答他的问题前，一定要想清楚一件事：他是该技术领域的高手，就是我们"程序员修炼三境界"中描述的"第二层境界"的那个人物，如果还想更清楚的了解这个人物，可以去重新看看前面的那个章节。

这个自我介绍最好说的较为简洁，不要过分炫耀自己的技术如何如何强，免得引起这位主管的兴趣，引起他的技术兴趣没有什么好处，只会带来更多的技术问题的发问。

1.2.2.2.2 常见提问2：你最引以为自豪的项目是什么？

他问这个问题的意图是想考察你的成长路径和编程习惯， 因为， 最让你自豪的项目往往是你成长最快的项目， 那个成长最快的项目往往会给你今后的编程习惯留下很多痕迹。

所以， 通过你对那个引以为豪的项目的描述， 有经验的他会很快锁定你技术成长中的缺陷和闪光点， 从而判断是否能够"为我所用"。

你最好拿出一个自己最擅长技术的那个项目进行介绍， 这个项目最好能够比较贴近招聘要求的那些指标。 如果， 没有做过什么有规模的正规项目， 你就拿些自己非常擅长或者有创意的开发作品来说。 这样做的好处是， 他听完你的介绍后， 会接下来进行提问， 他所有发问， 你都成竹在胸了。

切忌拿一个别人的项目， 或者自己参与很少的项目来介绍， 如果这样的话， 一旦他深入的询问这个项目的细节， 很可能你会所答非所问， 反而造成更严重的影响。 你大可以和他大谈特谈你在那个项目中获得的经验， 那会引起此君的共鸣， 有可能的话， 说出一些你自己的小技巧， 他会很高兴。

1.2.2.3 人物3：主管的主管

遇到"主管的主管"的时， 往往已经是复试， 这说明基本上已经是最后一关了。 但要注意， 这个最后一关是非常关键的一"关"。 因为， 往往如果你未来的主管在公司中某个专业够权威的话， 他的主管一般不会管招聘的事情。 因此， 你看到这位"老大"的原因， 多半是你未来的那位主管的专业地位还没有"稳定"。

1.2.2.3.1 最后的"搏杀"

过程大概是这样的， 一般会将两个或三个人提交给"老大"， 让"老大"定夺， 去选择其中的一个人， 因此， 这个阶段是一个最后的"搏杀"关键阶段。

1.2.2.3.2 "老大"关注的问题：成本+人员素质

所以， 我们在和"老大"过招的时候， 一定要注意自己的言行， 切忌不要穿"奇装异服"， 或者男士留有个性的头发或胡子， 总之一定要让人看起来特别的"平常"， 虽然不是"西装革履"但也要"衣冠整洁"。

因为， 老板们最不喜欢"个性"员工， 而最喜欢的是"优秀"的普通员工。

1.2.3 程序员的笔试

前面我们了解了和不同角色的人物见面的面试技巧， 下面再说说程序员们更加频繁遇到的一个考核方式"笔试"。

"笔试"对于初级程序员应聘者来说是一个关键一关，也是刷掉的可能性最大的一关，一次面试中大部分程序员都会由于"笔试"没有过而无缘进入下一个阶段，只有少数程序员得到与主考官见面的机会。

作为即将应聘初级程序员的我们，更应该做好技术的准备工作，这部分工作恐怕要花费比较多的时间去准备。

如何准备？

还是那句话至少应该读完本书，因为除了"技巧"之外，"技能"更重要。

1.2.3.1 笔试目的

(1) 为防止没有实际开发经验和开发技能的人来面试，可以通过"笔试"过滤掉一批人；

(2) 考察掌握知识的扎实程度，及面对问题的思考方式。

1.2.3.2 笔试误区，这些情况在笔试时应避免

(1) 有一道题不会，就放弃了整个笔试

也许你认为这道题很难，其他竞争对手也会觉得很难。

(2) 没有看清题匆忙作答

这不是入学考试，如果没有人给你计时间，你不用那么匆忙，正常速度作答即可。

(3) 不能完整作答的，干脆就空白

这和我们在学校考试不一样，如果你不能回答完整，最好也写上思路，或者写上想和主考官说的话，有的时候他也会给你机会的呦。

(4) 不清楚的一些概念性的问题，用其他同样不太清楚的概念"生搬硬套"

有些概念本来就不太清楚，就别拿另一个你不清楚的问题往上"套"啦，这样做，往往让主考官哭笑不得，一下暴露原来这些概念都不清楚。

(5) 笔试没有60分及格线这么一说

应聘职位的"笔试"和我们在学校的"考试"完全不同，不存在"及格"与"不及格"的问题，也许你某一个道题回答的很精彩，但是总分并不是很理想也会被录用。

1.2.3.3 笔试技巧

我挑了几道笔试题，曾经给入职程序员出过的几个测试题，大家看看应该怎么作答，我们再看看需要注意什么，题目如下：

1.2.3.3.1 例题1：请说出这个程序的目的是什么？返回结果是什么？

```

public Station(URL urla){
    try
    {
        String a="",b="";
        InputStream ins = urla.openStream();
        BufferedReader bReader = new BufferedReader(new
        InputStreamReader(ins));
        String info = bReader.readLine();
        int i=1;
        info=bReader.readLine();
        while(info!=null){
            a=info.substring(0,info.indexOf("@"));
            b=info.substring(info.indexOf("@")+1,info.length());
            if (i==1){
                this.X1=Integer.parseInt(a);
                this.Y1=Integer.parseInt(b);
            }
            if (i==2){
                this.X2=Integer.parseInt(a);
                this.Y2=Integer.parseInt(b);
            }
            if (i==3){
                this.X3=Integer.parseInt(a);
                this.Y3=Integer.parseInt(b);
            }
            if (i==4){
                this.X4=Integer.parseInt(a);
                this.Y4=Integer.parseInt(b);
            }
            i++;
            info=bReader.readLine();
        }
    }
    catch(MalformedURLException e){
        System.out.println(e);
    }
}

```

```
}  
catch(IOException e){  
System.out.println(e);  
}  
}
```

该题解释：

第二问，返回值是什么？

总有人直接回答，说"没有返回值"，这说明没有弄明白构造函数的声明，如果是普通函数没有返回值的话，应该用"void"而不是没有写任何东西。

第一问，这个程序的目的是什么？

这个程序乍一看确实不太明白，其实，这正是在考察应聘者的想象力和理解能力，这个程序一看就应该知道这是一个构造函数，这个构造函数里有四对变量分别是"X"和"Y"作为开头的，让人直接联想到坐标，另外这个构造函数的名字是 **Station**，说明这个类的名字是 **Station**，因此，可以想见，这个确实是这个 **Station** 的四个点的坐标，是用 **http** 协议到一个 **Web** 上去取得一个字符串，再去解出这个字符串中的每一个数字，赋值给相应的属性。

某君的比较精彩回答：

该函数是一个构造函数，他的目的是构造一个 **Station** 类，根据他的名字和属性判断，这个类有可能是描述一个物体的位置信息，这个信息可以解出四个变量分别赋值给不同的属性。疑问，感觉用 **substring** 和 **indexOf** 的方法解析出数据比较麻烦，为什么本程序不用 **String** 的 **split** 方法呢？那会更省事点。另外这个程序是采用 **URL** 类的 **openStream()** 方法得到远端某一个网页上的数据，那网页可能是一个 **JSP**，可以直接从数据库中取得数据，这个方法，省去了一个自建的 **Server** 程序，这种用法我是第一次看到，感觉学习到了一个新的应用方法的知识。

1.2.3.3.2 例题2：请写一段 **html**，完成下列表格的样子。注意：是单线边框

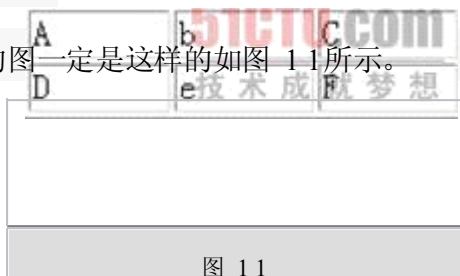
A	B	C
D	E	F

该题解释：

陷阱一：这道题初看起来很简单，似乎就是在考一个简单的"

"标记，于是有很多人都直接画出了一个 table，至于单线边框的问题，他们总是用 "border="1""来描述 table。

这样的话，画出来的图一定是这样的如图 1 1所示。



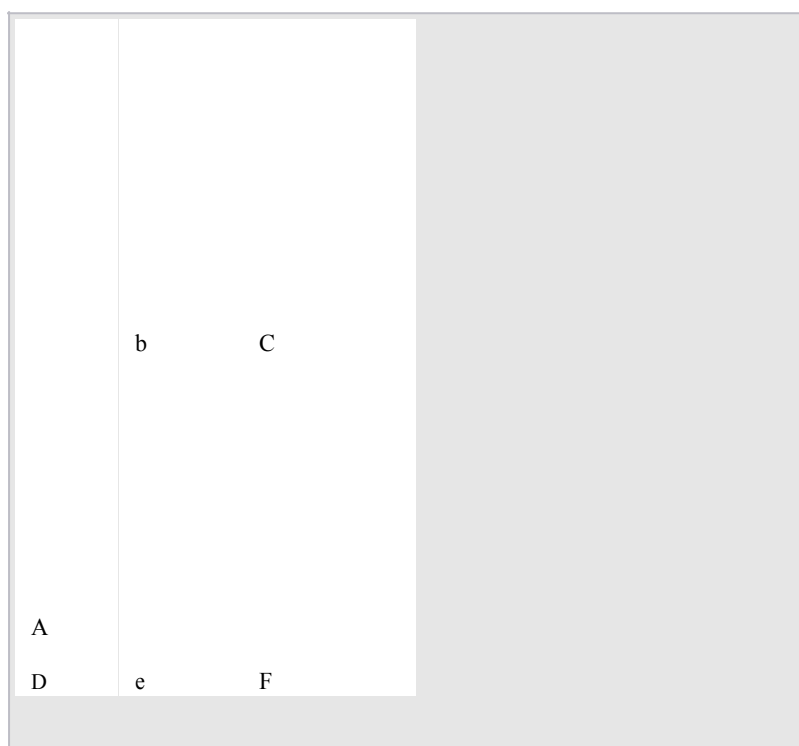
A	b
D	e
F	

图 1 1

还是一个双线表格，并没有像上图一样的单线边框。

陷阱二： 本题表格中的文字是大小写不同的， 有的用的是小写， 有的用的是大写， 一定要按照要求回答问题，这主要考察的是按照要求完成任务的能力。

正确答案：



b	C
A	D
e	F

如果你做过网页中的表格，一定知道应该用这个方法获得单线边框的表格，因为这是一个典型的单线表格设计方法。当然，有的人用CSS 来回答这个问题，也可以算对，只是方法比较复杂了。

1.2.3.3.3 例题 3：一个算法题

再给大家一道据说是难倒很多人的题，这道题曾经有50 人接受过提问，但是，只有 3 个人答对，请大家试试如何回答。

要求：有两个数组一个 N 个元素，另一个 M 个元素，这两个数组中有些元素是相同的，希望通过编写一段程序将两个数组中的相同元素找出来， 请用最少的循环次数完成需求， 请问需要什么方法？

一般程序员马上会想到类似如下的程序：

```
for (int j=1;jfor(int i=1;i.....}}}
```

那么，这个方法的循环次数是多少呢？

答：M*N 个。

但是， 要注意这个题的要求是， 最少的循环次数完成需求， 这时候只要想想一共有几种方法完成这个任务即可， 然后从中选择一个最快的就行了。

正确的答案是，用哈希表的方法，这个方法的循环次数是M+N，一个是将M 装入哈希表的循环，一个是将 N 逐个放倒 M 哈希表中去查询的次数。

这道题有两个陷阱， 第一个，"算法题"， 一般有些程序员只要听到"算法"这个词， 马上晕了，从而影响合理的思考。第二个陷阱是"最少"，用最少的循环次数的方法，而不是普通的，但就是这么两个陷阱使很多应聘程序员落马。

1.2.3.3.4 例题总结

大家看看， 上面的笔试题其实从技术方面来说都不难， 但是陷阱比较多， 而且， 需要你能够有想象力， 与出题者形成互动。 从这几个例题可以看出， 招聘单位最需要的人是实践能力强的人，因此，我们要从这个方面多下功夫，这些功夫获取途径仍然是加强日常的积累，本书的后面章节中也会涉及更多的类似经验，读者可以循序阅读。

1.2.4 回答这个问题

言归此问，"大学毕业生找不到职业入口"，为什么找不到职业入口？

我们的回答是--没有搞清楚什么是职业的入口！

何谓"入口"？

"入口"就是找到自己如何面对用人单位的需求找到自己的突破点，而这个"入口"有代表一种自身完善的方向和方法，当你符合用人单位的需求时，自然就找到了进入职业的途径。

1.3 问题3：跨行业真的这么难吗？

我遇到有很多在别的行业或职业发展的朋友，通过自己的努力实现了程序员的梦。在很多朋友看来，跨行业的发展是非常难的事情，然而，这个跨行业真的那么难吗？

我在本节中仍然要强调的是，"三百六十行，行行出状元"这句"陈芝麻，烂谷子"的话，如果想成功，任何行业都可以发展。如果你现在还不是程序员，甚至是现在只是一个从事其他工作的人，但是你真心的热爱"软件开发"这个职业，并希望"软件开发"将成为你毕生奋斗的一个伟大事业，那么你可以阅读本书内容，如果不是，那么把本书放回书架，打消"跨行业"这个念头吧。

进入软件开发领域需要的是：

真心的热爱，并且有为之奋斗毕生的心愿。

1.3.1 跨行业最难的是什么

跨行业发展的困难之处在于"你是否愿意放弃"。

我曾经看到过原来从事很多别的职业的朋友转到程序开发中来，有学财务的、学商业管理的、学建筑工程的，还有原来从事技术支持的，做网管的，以及做系统集成的，其中我看到跨度最大的是一位"厨师"加入到程序开发中来，这恐怕是跨度最大的了。

虽然我们看到了很多人成功跨越了行业，但是，这个过程确实是很痛苦的，因为，他们必须学会放弃原有的已经从事一段时间的工作，甚至暂时没有收入来源，仅仅凭借着的一颗热忱的心，和一种孜孜不倦的学习精神去支撑着自己。

跨行业最难的就是，在最初先是"不务正业"，然后发展到"在家待业"，跨行业成功了倒还好说，一旦失败会落一个"好高骛远"的名声从而"身败名裂"。

这说的有些夸张，但实际上确实要放弃很多。

想跨行业发展？让我们找到入口！

1.3.2 跨行业的入口--原来从事行业的业务知识

跨行业的入口，就是你原来从事的行业的业务知识！

找到你原来职业和程序员职业相通的点，找到事业的发展路径才是关键。前期从事的职业经历，有很多业务知识是从学校门出来就搞开发工作的所不具备的，因此，跨行业的入口就是你前期从事职业的知识，这些知识可以是你在软件开发行业中迅速成长。

例如这些情况：

学财务的--搞财务软件将得心应手，因为你的财务方面的知识恰恰是帮助你的法宝。

学商业管理的--商业管理知识帮助你理解 ERP、SCM、BOSS 等系统的原理。

从事技术支持的、做网管、以及做系统集成的--更方面的了解用户的某些方面的开发需求。

至于作"厨师"的那位，如果开发餐饮行业方面的软件你一定非常熟悉。

1.3.3 农民造出了飞机，为什么他还是农民

我们在不久前看到这样一则新闻：《张斗三：会造飞机的中国农民》

新闻中这样说的：

普通农民造飞机本身就是一件新鲜事，一件大难事，张斗三平日的职业是广州的一家建筑公司的项目经理，说白了就是一个建筑队的包工头儿。整天与他打交道的不是盖高楼的钢筋与水泥，就是修路、架桥。至于他自己的文化水平，小学三年级，仅限于此。因为儿时家里穷，为了讨生活谋生，13岁的他就背井离乡出门打工，开始了他做劳工砸石头的人生之路。再说，造飞机真不是一件容易的事，它涉及到一大堆的专业理论，物理，数学等等知识都会，这事可真不是一般的人可以做的。但张斗三愣是凭自己的执着做到了。厂地，他选自家天台。飞机设计，他全凭自己脑瓜儿里冒灵感。至于这工具嘛，老张也有高招，这不他把家用的工具，全都派上了阵，叮叮当当的敲打起来了.....

1998年12月18日，张斗三制造出了自己的第一架飞机"斗强三号"。

.....

乍看起来，这是一个非常令人振奋的消息，"我们国家的农民就是厉害"，但仔细想想我们却发现这则新闻中隐藏这一点点的"悲哀"，那就是"造出飞机也还是农民"的"悲哀"。

这看起来是"张斗三"的个人"悲哀"，但其实这是我们整个社会的一个问题--"出身"。

因为，"张斗三"出生时是一个"农民"，所以，他即使是做建筑工程的"项目经理"他也是"农民"，即使是把飞机制造上天他还是"农民"，即使最后将其称为"科学家"也要冠以"农民科学家"的称谓。

虽然，在这个事情的背后，存在媒体炒作的需要，但这是我们国内社会面临的一个现实问题，科技进步的投入重点始终是在"科班"出身的科学家身上。同时也反映出，我们的"民间科学家"在作跨行业研究工作时面临的窘境。

这一社会现象对于我们跨行业的广大"非科班出身"的求职者有什么启示呢？

这是很值得我们去思考的一个问题。

1.3.4 回答这个问题

言归此问，"跨行业真的那么难吗？"

跨行业发展和"科班"出身的人相比之下，当然会面临一定的困难，这些困难，有的来自于自身，有的来自于家庭，有的来自于社会。你能否看淡这些困难并坚持自己的目标？这是成功的关键，一旦突破自我以后，你会发现：被你认为应该放弃的，你原来的另一个行业的经验，在新的事业中成为你的优势。

这里我们用一句话来结尾：鹰击长空万里阔，壮心上下勇求索。

1.4 秘诀：经营自己的优势

问了那么多个问题，其实秘诀只有一个那就是"经营自我"！

前面说的"大学生入职问题"和"跨行业入职问题"，不管你现在身处于哪种情况，你都要面对同一个问题--"销售自己"。

有几个名词前面一直在提到，如"战场"和"供求"等，这些名词都无疑表露出在这个社会中的现实性，在入职这个问题上我们觉得他更像是一个"市场"。我们每一个人都在经营一个"小店"，这个"小店"中销售的不是别的东西，销售的就是你自己。

1.4.1 善于发掘和积累自己的优势

经营的故事：

世界商业报道：

麦当劳,可谓人所尽知。可又有谁知道，在当时有一家快餐公司和比麦当劳同时起步。

麦当劳很傻，接管餐厅只做快餐，其他的统统外包，钱都让别人赚去了。另一家快餐公

司呢，则很精明，想肥水不流外人田，什么钱都不想让别人赚，做面包要用到面粉，就自己包了块地种植大麦；要用到牛奶，就自己养了几头奶牛，生产线急剧扩大。

若干年后，麦当劳成了世界头号快餐公司，分店遍布全世界。而另一家快餐店则不见踪影。后来，人们在荷兰的一个小农场里找到了他，他早已不开快餐公司了，就养着几头奶牛。

一个企业的成功，首先要弄清楚自己是干什么的，自己的核心是什么。如果不弄清楚这一点，盲目跟风，见什么赚钱就上什么项目，最终必将一事无成。如果，你想把自己成功的销售出去，也要像经营一个公司一样，从自己的优势和特长出发，弄清楚自己为企业提供什么方面的服务，自己的核心竞争力是什么。

俗话说"一招鲜，吃遍天"，依靠一技之长，走遍天下。而这个"一招鲜"不可能是学校培养出来的，如果学校能够培养出来"一招鲜"，那么他就不是"优势"了，因为大家都会。

真正的"一招鲜"是自我发掘和积累的过程，并依靠勤奋得来的，要用敏锐的眼光发掘"市场"，依靠自身积累切入某些市场领域，形成自己的一套"一招鲜"。

在技术积累的过程中，要"有的放矢"，不能眉毛胡子一把抓。程序员要做到"低头做事，抬头看路"，所谓看准"市场"，就是抬头看路的过程，看准路在何方后，并坚实的迈好向这个路前进的每一步，这就是我们程序员们要进行的优势积累过程。

在开发业界，存在很多名词和知识点，这些知识点不需要各个都去研究，有些东西只要知道他们的大概原理即可，对于我们程序员，切忌"什么都会，什么都不精"，要做到"一门灵"，要掌握一个知识就要深入的挖掘它，最终成为自己的特长。

1.4.2 善于展示与利用自己的优势

通过技术积累将自己的优势和特长逐步沉淀，同时，我们还需要学会去展示与利用自己的优势，去争取机会。

曾经有这样一句话"机会只光顾有准备的人"，我们技术优势的发掘和积累就是在时刻"准备"。但是，有时候"机会"不是那么容易自己找到你的，还需要每个人去展示优势去吸引机会。

在简历中要突出自己的优势，展示自己"精深"的特长。

我们看看这个简历，这个简历是截取的一段内容，看看他的简历有什么问题：

职业技能：
熟练掌握Java、JSP、Servlet、C/C++、HTML/JavaScript；
熟悉JBuilder开发工具，能够基于JSP、Servlet进行Web的应用开发；
熟悉J2EE规范、了解MVC架构、XML；
熟悉Weblogic、Tomcat等应用服务器的配置、开发；
熟练掌握关系数据库Oracle、SQL Server2000等；
熟悉软件测试的流程与方法。

1.4.2.1 首先，我们看一下他的"自我评价"：

"本人是一个工作认真负责、为人诚恳、积极主动、适应能力强、善于团队工作的人；思维严谨、乐于学习新的技术知识、适合做技术类的工作。"

这个"自我评价"，你是否觉得似曾相识？

没错，估计在学校每学期写自我评价的时候，都是这样糊弄老师的。

要知道，这个简历是给你未来老板看的，如果你是在经营自己，你这份简历就是你的"产品简介"，要给你的"客户"看，来让"客户"决定是否要花钱购买你。

这个"自我评价"非常的平淡无奇，过于"大众化"，他的"优势"根本没有任何体现，这样的"产品简介"根本不会引起"客户"更多的兴趣，肯定没有人"买"。

如果，我是你的一个比较挑剔的"客户"，针对这个"自我评价"，面试的时候会问你这些问题：

自我评价1"认真负责":

认真负责的是怎么界定的, 什么叫认真负责, 认真负责到什么程度?

自我评价2"善于团队工作":

善于团队工作指的是什么? 团队是一个什么样的团队? 你跟团队是如何配合的? 你怎么那么肯定你善于团队工作呢?

自我评价3"思维严谨":

什么是思维严谨? 如何体现出思维严谨?

很显然, 这个"自我评价"比较不负责任, 用一些比较"高、大、全"的语言来涵盖"自我评价"。需要告诉你, 你未来的老板不像你的老师那么好糊弄的, 要拿出"真本事"人家才会买你的帐。

1.4.2.2 我们再看看他的专业技能:

"熟练掌握 Java、JSP、Servlet、C/C++、HTML/JavaScript; 熟悉 JBuilder 开发工具、能够基于 JSP、Servlet 进行 Web 的应用开发; 熟悉 J2EE 规范、了解 MVC 架构、XML; 熟悉 Weblogic、Tomcat 等应用服务器的配置、开发; 熟练掌握关系数据库 Oracle、SQL Server2000等; 熟悉软件测试的流程与方法。"

这个"专业技能"更是平淡无奇了, 几乎所有Java 程序员都会这些技能, 没有任何特点, 而且我们发现他这个"专业技能"中出现最多的是"熟悉", 要注意"熟悉"和"精通"可是两个概念, "熟悉"只能说明你知道这个技术, "熟悉"这个技术和真正拿它来开发更是不能同日而语啦。

那么这意味着, 他的这个 "专业技能"在明明白白的告诉未来的老板: "我所有概念都'熟悉', 就是没有具体开发过!"。

如果, 咱们换位思考一下, 你是这个公司的老板, 你愿意要这个员工吗?

这样的简历中有没有你的影子呢?

如果有, 那么应该避免这些问题的出现, 重新审视自己的特长, 努力完善吧。

1.5 点评"经营自我"

你是精品店? 还是杂货店? 不管是什么商店, 都需要用心去经营, 你认真对待你的顾客, 你的顾客也会给你相应的回

人生就是这样：

把自己交给市场，用心去经营。

第2章 软件开发职业的误区

我们上一章了解了应聘公司和面试过程，对程序员这个职业有了一个初步的认识，为即将进入应聘公司我们心里打下了底，使我们在应聘前更有信心了。

那么，在本章中要说另一个话题，这是在软件开发职业领域里经常出现的对软件开发职业的误解，这些误解有的时候会成为新程序员的"思想加锁"。所以，有必要在本章中对这些误区予以说明，让我们的程序员们"轻装前进"。

2.1 误区1：软件开发职业是青春饭

我们听得最多的"误解"莫过于"软件开发职业是青春饭"这句话了，这样的说法在人们中间很流行。

这种说法意思是说，只能在"年轻"的时候从事软件开发工作，当然这个"年轻"也给出了一个具体的年龄，就是"30岁"以前，"30岁"以后就不能搞开发工作了，肯定转行做其他工作去了。

这种说法的"理论依据"是，做软件开发非常的"用脑子"，而且都是代码的工作，只有"年轻人"才能有这个精力去投入软件的开发工作中去，年龄大了就干不了啦。

很显然，这个"误解"理论来自于"外行"人，但是，这个"外行理论"却时不常的成为"新程序员"前进的障碍。原因就是，"新程序员"们看不到职业的发展方向，搞不清楚自己的发展路径。

2.1.1 不必为30岁以后烦恼

我们在前面章节和大家说过"程序员修炼三境界"，关于软件开发职业是不是青春饭的问题，我们在那个章节中，从程序员的发展路径中我们基本上可以比较清晰的看到了答案。



这个路径告诉我们，"初级程序员，程序员，软件工程师"并不是软件开发职业的唯一表现形式，这仅仅是职业的初级形式，大约在三年内完成进阶任务。这个"误解"估计主要来自于对"软件开发职业"的理解，认为"软件开发职业"仅仅是指"初级程序员，程序员，软件工程师"，所以，都是年轻人在做这个职位。

我们要充分理解程序员未来的"光明性"和"曲折性"，程序员职业发展的"曲折性"也会表现为一定的"活跃性"，即，部分人会在进阶过程中"掉队"，虽然程序员工作三年可以进阶为"开发经理"，但是，这个过程是"曲折的"并不是一夜之间完成的。

2.1.2 30岁以后照样可以编程序

不管你在30岁以后是否已经进阶为"开发经理"还是继续做"程序员"，一样都可以做编程的工作。认为在30岁以后应该专门做管理工作的观点同样是错误的。30岁以后，虽然做初级程序员的可能性不大了，但是，核心编码工作，尤其是难度较大的那部分编码工作，还是较多涉及的。

2.1.3 50多岁的程序员多的是

据美国调查企业 Evansdata 公司发表的调查报告显示：

从事软件开发的程序员中女性比例逐渐减少，现在仅占9%，大部分开发程序员都是36-50的男性，平均年收入在5.5万美元以上。Evansdata 公司的调查报告搜集了全球1.4万名软件开发程序员回答信息。

这种情况之所以没有在中国出现，其主要原因是中国的信息技术起步较晚，在中国信息技术大面积普及的时间大约是1990年以后，那个时候的年轻人，现在也只不过是30多岁，至多40岁。

在国内找到50多岁的程序员有点难，主要是中国软件开发行业的起步比较晚。但也不是不可能，我就认识这么一位前辈，他目前是一家小公司的老板，他的公司有程序员，但据我所知，个别的程序他也会自己去调试调试，他的这个精神是我等后辈应该学习的。

2.1.4 点评"青春饭"

软件开发职业不仅不是"青春饭"，而且这个职业会让你永葆青春呦！呵呵。

青春饭：

职业本身并没有对年龄的限制，如果你愿意可以一直干下去。

【责任

2.2 误区2：做软件开发必须要加班熬夜的工作

一个外行老板：

有一个朋友在一个国际知名的公司工作，这家公司在北京的中国公司是被一个国内电子商务公司控股的企业。

最近这家公司进行了人事调整，原来的外资管理层被中资管理层取代。

新来的中资管理人员是原来在其他行业的管理者，对IT行业并不是内行，他们上任以来提倡了多个工作作风，其中一项就是，软件技术人员工作必须是早9点上班到晚9点下班。

公司的管理制度所标识的"8小时"工作制度形同虚设，当然，你也可以晚6点离开公司，但是，很快会收到点名批评的邮件。

这是一个外行老板去领导内行的笑话，很明显，这位CEO的想法是，"做软件技术的怎么能不加班？不熬夜呢？"

他的这个想法恰恰是进入了一个职业的误区："做软件开发必须要加班熬夜的工作"。

2.2.1 程序员们熬夜工作的借口

对软件开发职业的这个误解并非"空穴来风"，因为，我们确实看到很多的程序员在没有

硬性规定的公司中加班熬夜的程序员。

正是这一个现象，旁观者们当然会很自然的联想到"程序员们的工作需要这样"的假设。作为程序员，"加班"是有的，但是这个"现象"本身是具有"偶然"性的，而不能成为一种"常态"。

程序员们总能找到熬夜工作的借口，我们来看看都有哪些。

2.2.1.1 代码一气呵成，一定要写完而后快

想一口气编完程序，是大部分程序员们自发加班的主要驱动力。比如，小张在开发一个图形显示组件的过程中，该组件程序已经接近收尾，一定要写完看到想要的效果才肯罢休。

这个加班的动力来自于程序员自身，这种工作的积极性来自于程序员发自内心的对其工作的热爱，是一腔热血使然。这个现象对程序员是有帮助的，对整个项目组也是有帮助的，并应该鼓励，只不过不要成为"常态"。

2.2.1.2 明天要做一个命题演示，一定要调试好程序

明天急着给客户做一个命题的程序演示，还差一点没有搞定，急得抓耳挠腮，这时候，还不加加班？大客户跑掉了，这个责任可承担不起。所以，一定是搞定了才肯回家，明天轻装上阵，顺利完成演示工作。

2.2.1.3 安静的工作环境

我们还听到程序员们抱怨"没有安静的工作环境"，一般两个方面：

其一，程序员这一天工作中，不仅仅需要编写程序，还要花费时间阅读并回复邮件，接电话处理各种问题的询问，协助 HR 面试，甚至审阅产品使用手册，等等，根本没有办法踏实下来一气呵成写完代码。

其二，来自于周围环境，如果程序员运气比较差，正好和商务部门或客服部门做邻居，那还真是存在这个问题，因为那里的电话与谈话声此起彼伏。

面对这两个问题，应该有各自的解决方案：

第一种情况，应该安排好自己的工作时间表，尽量将自己的时间合理的分配。

第二种情况，申请调换座位或心静如水、充耳不闻。

2.2.1.4 项目进度紧张，需要尽快完成任务

"项目进度紧张"经常成为"加班"的借口，一方面是项目经理要求程序员加班的借口，另一方面是部分工作拖沓的程序员加班借口。

之所以这么说，是因为造成"项目进度紧张"的直接责任应该归咎于"项目经理"，而不是

"程序员"。 项目的进度应该由"项目经理"进行合理的安排与调配, "程序员"所需要做的是按照项目进度要求完成自己编码工作, 如果编码工作是按照既定计划完成的, 那么造成项目进度的紧张自然就是项目管理的问题。

不过还有另一个现象, "项目经理"安排的时间得当, 某些程序员工作拖延, 在没有完成既定任务的情况下, 工作时间的做其它事情, 反过头来造成项目进度紧张而"加班"。

所以, 我们说凡是出现这个原因加班的情况, 这一定是一个"警钟", 那就是, "项目出现了问题", 因此, 这个加班现象就一定要避免了。

项目进度紧张而加班:

是项目或者你自身出现问题的信号。

2.2.2 常态加班的危害

偶然的加班是不会造成影响的, 甚至有可能激发程序员的工作热情。 但是当加班成为常态, 则会给工作造成很大危害。

2.2.2.1 危害1: 项目进度不升反降

加班成为常态以后, 尤其是老板要求其工作必须超过12小时的时候, 你会发现程序员将私人事情安排进工作时间。要不程序员该什么时候处理自己的私人事情?

2.2.2.2 危害2: 工作积极性被严重磋商

本来加班是一种工作积极的表现, 尤其是对于由于"代码一气呵成"原因而加班的程序员来说, 更是为项目为公司努力工作的表现。 然而, 当被公司要求加班而不是自发加班的时候, 这部分程序员的积极性可以说是完全被磋商。

2.2.2.3 危害3: 程序员身体造成危害

8小时的工作时间是一个比较科学和合理的, 如果一味的去拼命的使用身体, 必将使身体受到损害, 势必会造成如: 工作没有精神, 记忆力差, 反应不敏捷, 等现象, 这些现象本身给项目造成的损害是更加致命的。

常态的加班危害巨大, 有经验的项目经理或者睿智的管理者, 是不会允许这个现象发生的, 哪里出的问题从哪里找原因, 千万不能通过增加工作时间来弥补事情的真实漏洞。

作为刚刚入行的程序员, 也要清楚的认识到这些危害, 不要陷入"加班熬夜"的工作误区。

2.2.3 控制好工作的一日时间表才是关键

前面提到程序员这一天工作中，不仅仅需要编写程序，还要花费时间阅读并回复邮件，接电话处理各种问题的询问，协助 HR 面试，甚至审阅产品使用手册，等等，根本没有办法踏实下来一气呵成写完代码，无法白天安心编码的问题。

我们程序员应该给自己定一个"一日时间表"，这个一日时间表可以更合理的安排时间，把主要精力放倒最重要的事情上去。

比如，下面这个时间表可供参考：

接满一杯热水，打开电脑，整理思路，阅读并回复昨晚下班后收到的邮件--0.5小时

专心软件开发工作--2.5小时

午餐休息--45分钟

阅读并回复邮件--15分钟

专心软件开发工作--4小时

处理其他事务如接电话等--1小时

阅读并回复邮件--10分钟

在一早来到公司到午餐前这2.5小时是一天中工作效率最高的时间段，应该在这个时间段中完全投入到开发工作，这段时间头脑清醒，思路敏捷，应该尽量避免去做其他事情，你会发现，这个时间的开发进度会事半功倍。

然后午餐，午餐回来后稍事休息，快到1点的时候，可以处理一下邮件。

之后，进入下午的开发时间，这个时间段效率最高的部分也是2.5小时，尽量投入全部精力在2.5小时以内开发编码。

电话等其他事务，会穿插于下午的工作时间中，尽量压缩在1个小时以内。

最后在下班前，阅读并回复邮件。

大家看，这样安排工作时间，是否就不会被繁复的其他工作搞得手忙脚乱了？用效率最高的时间去处理开发任务，用接近休息或者接近下班的时间去处理邮件，这样你还需要加班吗？

反之，如果整天被开发之外的事情搞得团团转，一行代码都没有写，这样的话，不加班都不行了。

时间策略：

好钢用在刀刃上。

2.2.4 "偏执与狂热"不等于加班加点

作为程序员如果想成功是需要"偏执与狂热"力量的，这部分"狂热"的力量促使着程序员永远在关注着最新技术的前沿，关注着最新产品的实现方法。

英特尔公司总裁"安迪·u26684X罗夫"说"只有偏执狂才能成功"，抱定一个信念一直不屑的努力，最终走向成功的彼岸。对于程序员来说，这个"狂热与偏执"是一个持续努力的过程，首先成为软件开发技术的"发烧友"，结合国际与国内的技术形式给自己定位。

这个过程是艰辛的，但是不等于对"工作量"无限堆叠，不等于通过延长工作时间来达到其目标，这个过

狂热与偏执：

强调思考的重要性，而不是延长工作时间。

2.2.5 程序员的大脑与第二大脑

以下是一则新闻：

据科学家最新研究成果表明，在生命体的活动中，除大脑外，还有一个第二大脑。科学家研究确定，在人体中还有一个"组织机构"，即神经细胞综合体。在专门的物质--神经传感器的帮助下，该综合体能独立于大脑工作并进行信号交换，它甚至能像大脑一样参加学习等智力活动。

看到这则新闻，我想在这里武断的假设与猜测一下，那个第二大脑是否就是支配我们梦的编剧呢？

我和一些程序员聊天时，发现大家都有一个非常奇特的现象，就是如果你前一天晚上带着一个难以解决的程序逻辑问题去睡觉，在第二天早晨半睡半醒间，猛地想一想昨晚的那个问题，你会惊人的发现，你很快找到了那个问题的答案。

难道我们的第一大脑是在我们醒着时候活动，而我们的第二大脑是在我们睡着的时候开始活动？

如果，我的这个假设不成立，那么"在人梦里和我说话的那个人是哪个大脑支配的呢？"反正不是我的第一大脑，因为，我在梦里无法预测到那人要怎么说以及怎么做！

好了，打住，快要跑题了，我们不是在写科幻小说，这些人脑问题，还是留给医学和科学专家去分析吧，我们作为软件开发者在这个问题上，只有猜测的权利，没有结论的权利。

如果，姑且认为我的想法是对的，那么.....

提问：为什么半睡半醒之间总能想出困难的软件开发问题的解决办法？

回答：因为，在那个刹那，两个大脑都醒着！

在写作本书的过程中，我做了一个调查，访问了多个程序员，在问及他们是否出现过"记忆力暂时断路"问题时，答案是60%的程序员出现过这个现象。

所谓"记忆力暂时断路"是指，很熟悉的一个事情，突然间忘记，比如，一个很熟悉的演员，看到他以后愣是想不起来他的名字，再比如，一个很常见的字，提起笔来就是忘了怎么写。这些现象，我们称之为"记忆力暂时断路"。

在受访者中经常出现"记忆力暂时断路"的人，80%都有通宵工作的经历。这个现象能够证明什么呢？我想做出一个假设的结论：

如果总是让该睡觉的第一大脑和正在醒来的第二大脑同时清醒，会扰乱人体的生物钟，甚至出现幻觉，即睁着眼做梦。

当然，还是那句话，这个仅仅是作为一个程序开发者以及与第二大脑频繁打交道的人的一个假设的结论，有些科幻性质，读者莫怪。

但无论如何，还是请尽量避免通宵工作，保护自己的大脑和第二大脑。

2.2.6 点评"加班熬夜"

"加班熬夜"对事情的进度于事无补，反而陷入事务急功近利与反复修补拖延的漩涡，并且，还需要付出身体健康的代价，而这些代价有时候是没有办法挽回的。

加班熬夜：

是用你未来的时间工作，看似时间越来越多，其实未来的时间却越来越少。

2.3 误区3：开发一个软件产品一定是集团作战

一提到软件开发，很多人不由自主的会首先想到"很多人一起工作，协同开发"，逐渐形成了一个对软件产品开发的误区"开发一个软件产品一定是好几百人的集团作战"。

我们先看看下面这个软件项目：

MySQL 是以一个客户机/服务器结构的实现，它由一个服务器守护程序mysqld 和很多不同的客户程序和库组成。MySQL 是一个精巧的、多用户、多线程、快速的、强壮的SQL 数据库服务器软件。虽然它不是开放源代码的产品，但在某些情况下你可以自由使用。它提供给使用者比较强大功能以及精巧的系统结构，受到了广大自由软件爱好者甚至是商业软件用户的喜爱。

MySQL 对于 Java 程序员来说可以说是再熟悉不过了，这个软件是一个数据库系统，他的运行速度堪与 Oracle 相媲美，由于其推广方式采用免费下载使用，使其用户数量在全世界非常众多。

而对于这样一个软件来说，他最初的开发版本是由Monty Widenius、David Axmark 等人共同开发与维护的，早期并没有特别多的人参与。

类似的情况还有很多，比如下面的几个软件：

EditPlus 是由韩国人编写的一款共享软件，官方网址是www.editplus.com。最新版本是 EditPlus 2.12。EditPlus 是功能全面的文本、HTML、程序源代码编辑器。主要特点如下：

（1）默认支持 HTML、CSS、PHP、ASP、Perl、C/C++、Java、JavaScript 和 VBScript 的语法高亮显示，通过定制语法文件，可以扩展到其他程序语言。

（2）EditPlus 提供了与 Internet 的无缝连接，可以在 EditPlus 的工作区域中打开 Internet 浏览窗口。

（3）提供了多工作窗口，不用切换到桌面，便可在工作区域中打开多个文档。

（4）正确地配置Java 的编译器"Javac"以及解释器"Java"后，使用EditPlus 的菜单可以直接编译执行 Java 程序。



MRTG (Multi Router Traffic Grapher，多路由器通信图示器)，是一个使用广泛的网络流量统计软件，可以图形方式表示通过SNMP 设备的网络通信的状况。它显示从路由器和其他网络设备处获得的网络通信应用信息及其他统计信息。它产生 HTML 格式的页面和 GIF 格式的图，提供了通过Web 浏览器显示可视的网络性能信息的功能。使用该工具可以方便地查明设备和网络的性能问题。因为MRTG 可以监控任意的路由器或支持 SNMP 的网络设备，所以它可以用于监控边缘路由器与中枢路由器及其他设备。MRTG 收集信息的主要方式是通过 snmpget 命令。但是，用户也可以自定义 MRTG 显示通过其他途径获得的信息。例如，MRTG 可用于查询任何 SNMPMIB 对象，所以可以显示许多附加的性能信息和MIB 对象。

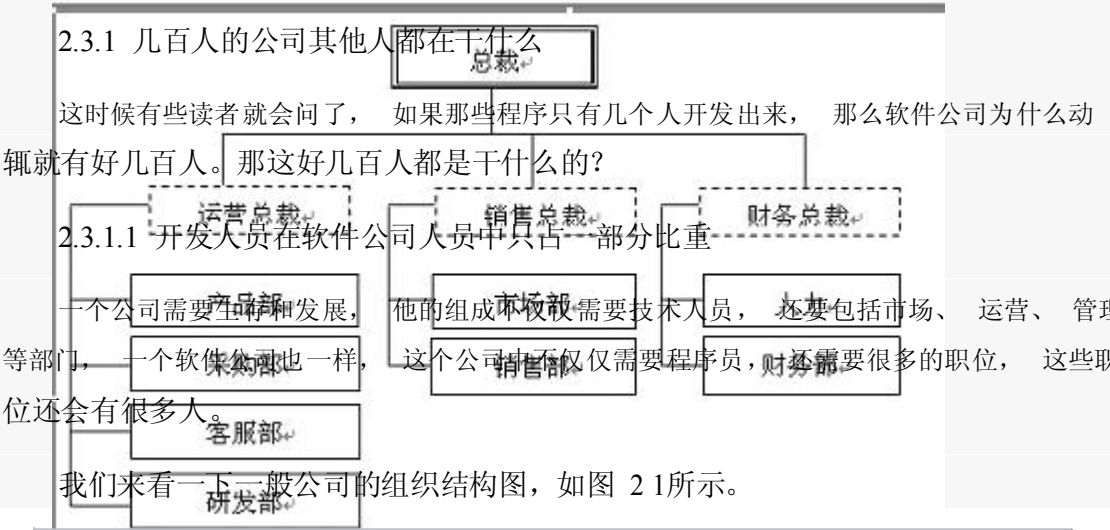
这些软件的用户群也非常庞大，但是，开发这些软件的公司或者团队并不庞大，类似的情况还很多，中国的这样的软件产品也不少，最著名的如：

1988年，中国的IT 界尚处萌芽时期，然而，5月的深圳，中国的办公软件已开始在这个萌动的春天孕育。一个名叫求伯君的技术人员在宾馆的出租房里凭一台386电脑写出了

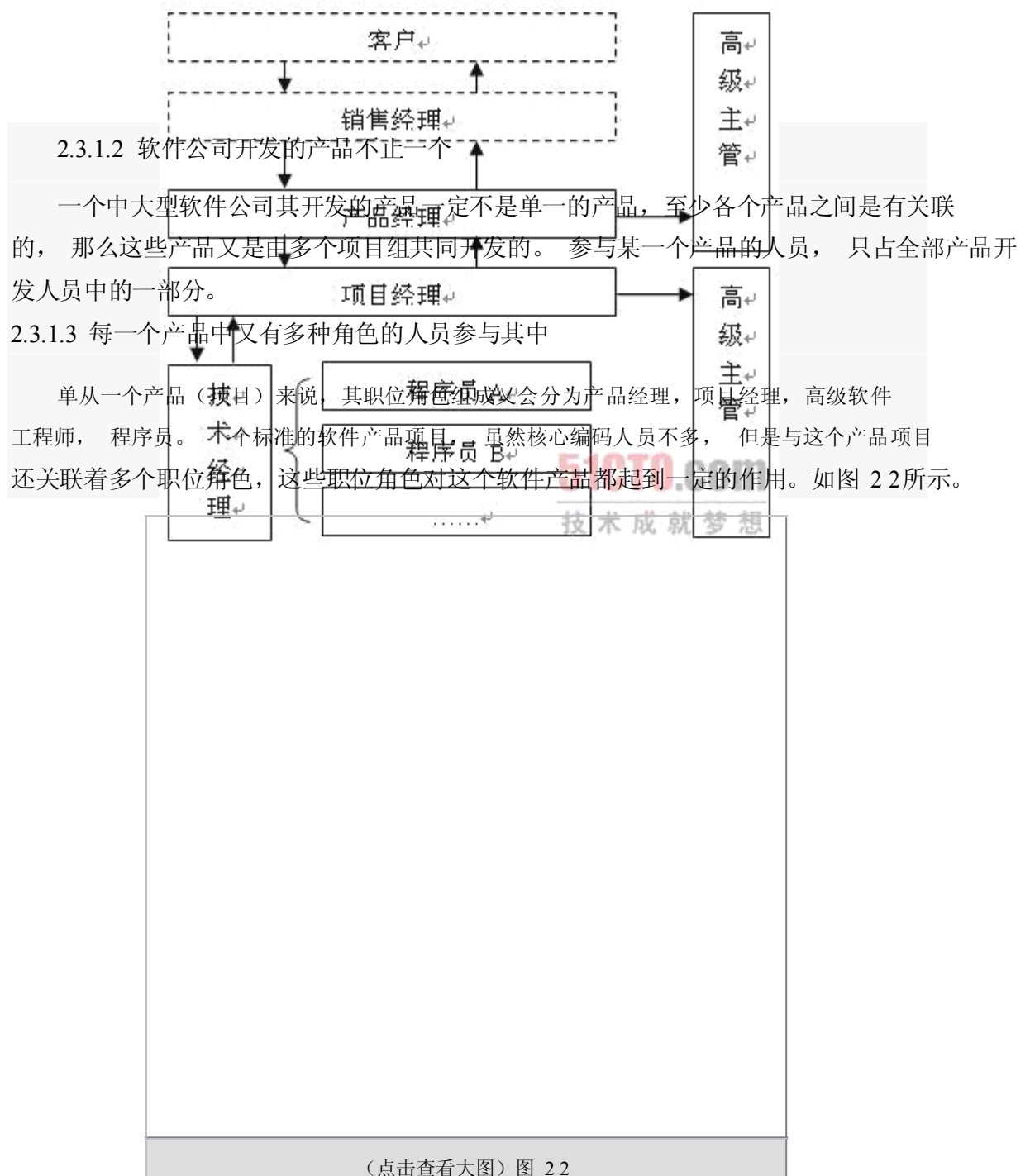
WPS(Word Processing System) 1.0，这几万行代码从此开创了中文字处理时代。

1997年--1998年， 盘古的失利使金山进入了发展的低谷， 一些员工开始陆续离开， WPS 97 一度仅有4名程序员坚持开发。 终于摸索出了WPS 97， 是一套运行在Windows3.X、 Windows95 环境下的中文字处理软件， 在保留了原有的文字编辑方式的同时， 支持"所见即所得"的文字 处理方式。

不要认为只要是软件都是很多人开发的成果， 其实， 有很多的软件产品的核心人员只有 几个人， 甚至只有一个人。 到一个比较大的软件公司去之后， 发现自己身处一个只有几个人的 的项目组中去编程，你千万不要惊讶，因为，有可能就是这样的。



我们来看一下一般公司的组织结构图，如图 2 1所示。



2.3.1.3.1 产品经理

产品经理对该项产品负责，他设定产品的目标，进行产品的定位，他要制定产品的营销计划，与销售部门协调，使得产品的开发、文档、测试、客户支持等事项能配合销售方面的工作，并进行产品的选型，他要进行信息管理、价格管理、广告管理、促销管理。产品经理是对产品未来适用性的主要负责人。

2.3.1.3.2 技术经理

技术经理是团队中对程序最熟悉的程序设计师，负责整个程序的框架合理性，并确保软件内部各个组成模块和协调一致性，确保所有的开发工作符合设计标准。还需要对团队内的程序员进行训练，带领程序员共同完成相关的开发任务。他通常也负责技术文档的建立与更

新，包括软件设计图等。

通常这个角色也是由团队中最资深的程序设计师担任。

2.3.1.3.3 项目经理

项目经理是项目的主要负责人， 项目经理负责拟定软件开发计划， 并监督开发工作按照开发计划有序的进行， 负责协调项目开发过程中的其他相关事项， 负责申请并调配公司提供给该项目的相关资源， 以保证项目最终完成。负责向高级主管报告本项目的进展状况。

通常这个角色是由团队中最资深的程序设计师担任， 偶尔也写点程序， 但那是他的次要工作。

2.3.1.3.4 软件开发人员

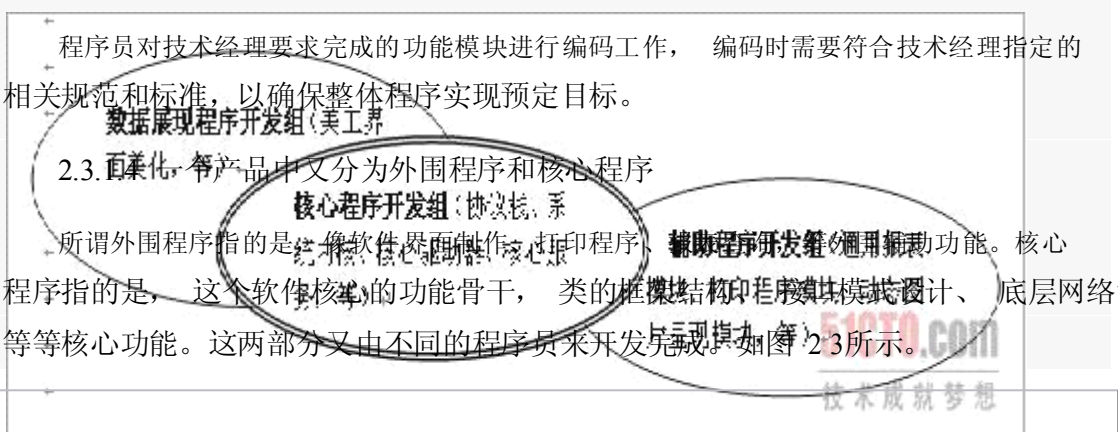
程序员对技术经理要求完成的功能模块进行编码工作， 编码时需要符合技术经理指定的相关规范和标准， 以确保整体程序实现预定目标。

数据展现程序开发组(美工界)

2.3.1.3.4.1 产品化产品又分为外围程序和核心程序

核心程序开发组(协议、系统)

所谓外围程序指的是，像软件界面制作、打印程序、辅助程序开发等通用模块功能。核心程序指的是， 这个软件核心的功能骨干， 类的框架结构、接口模块设计、 底层网络协议组件， 等等核心功能。这两部分又由不同的程序员来开发完成，如图 2-3 所示。



2.3.2 需要较全的人员配置的项目

我们经常能看到这样的招聘启示：

XX 软件公司招聘：

职位名称	工作 地点	发布 日期	截止 日期	招聘 人数
Java 高级工程师	北京	xx-x	xx-xx	1

	市	X-X	-XX	
Java 软件工程师	北京 市	XX- XX-X	XX- XX-XX	5
数据分析工程师	北京 市	XX- XX-X	XX- XX-XX	1
Java 后台开发高级工程师	北京 市	XX- XX-X	XX- XX-XX	3

很显然，这个软件公司在为某个软件项目招聘人才，有可能已经有一个项目在实施中，或者已经签署了开发协议，等待开发人员到场，那么这个公司是要开发一个什么样的项目呢？

2.3.2.1 需要客户化开发定制的项目

这样的系统特征基本上是需要大量的客户化开发定制，为某个集团企业制作某一领域的大型管理系统，由于这样的庞大系统一般涉及很多的分支系统，以及很多的系统模块，而且这些模块又需要符合大客户的需求细节，所以，这样的系统一般需要较全配置的人员投入。

例如：ERP 系统、BI 系统、SCM 系统，等等。

2.3.2.2 大型服务型运营平台系统

大型服务型运营平台指的是一个运营服务提供商，为了给接受服务的个人或者企业客户提供相应的服务产品，而建立的运营服务系统平台。

这个系统平台通常涉及的内容较为广泛，不仅包括通信协议、多媒体传输、或者音频混音，等等底层技术内容，还有可能涉及到网络硬件层面的部署，并且还可能包括，计费、帐务核算、财务分析、客户关系管理等诸多模块。之后，这些看似不同领域的软件系统之间又需要紧密联系，才能使运营平台系统发挥它的最终效能。

因此，这个大型服务运营平台系统也需要较全配置的人员投入。

例如：软交换服务管理平台、视频会议运营服务平台、短信服务管理平台、网络游戏产品，等等。

2.3.2.3 复杂的系统化软件产品

复杂的系统化产品指的是一个功能完善，有极强的系统扩展性，包含复杂的功能模块，完成复杂的系统任务的软件产品。

例如：项目管理软件产品、OA 软件产品、标准化的 ERP 产品，等等。

2.3.3 坚信，只要是"人"做到的我就能做到

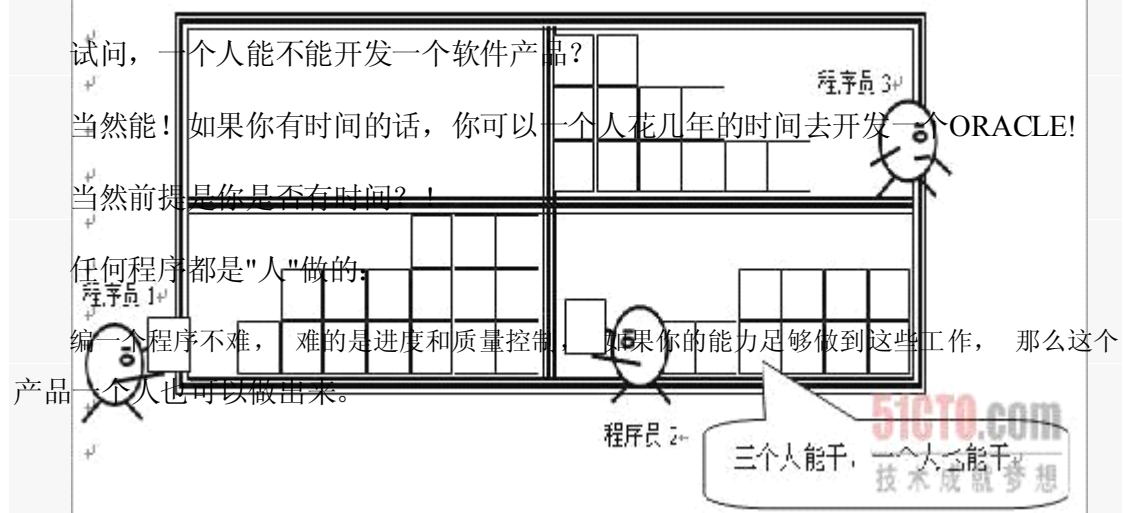
所以， 根据我们上面的分析， 那些看着非常复杂， 看似高不可攀的软件系统的核心， 其实都是几个人甚至是某一个人领导开发的， 由其搭好框架制定好规范后， 再由程序员堆砌工作量便可以完成复杂的软件项目， 是否符合要求那就看项目管理水平和产品定位了， 与程序员工作无关。

如果面对一个庞杂系统的程序开发职位或者任务的时候， 不必惊慌， 要认清自己在整个项目中的所处地位， 你会逐渐发现， 那些程序并不是最开始想象中的那么困难。

2.3.4 点评"集团作战"

虽然开发一个软件产品不是动辄上百人的"集团作战"， 但也不是毫无章法的"散兵游勇"。

作为程序员， 一方面， 要具有"一夫当关， 万夫莫开"之勇， 知道软件产品开发也不过如此， 另一方面也要知道， 软件开发的科学性和复杂性。



2.4 误区4：程序员不是一般人

从一个做会计的女生对程序员的误解说起：

那天和一个做会计的女生聊天，问她对程序员或者编程工作的看法。她搬出了一大堆词汇，立刻让我晕倒。她说："编程很深奥，工作很枯燥，程序员编程累脑子，干的时间长了看着程序员都有点木，这活儿可不是一般人能干的。"

2.4.1 程序员的与众不同与平凡

我自问我自己就是普通人一个，两个肩膀扛着一个脑袋，再普通不过了。再看看我身边的这些程序员，哪个也没长出三头六臂来，正是这么一群普通人却在人们眼中是那么的不一样。

2.4.1.1 程序员的与众不同

程序员的与众不同也许正是让人们产生不是一般人的误解的原因，作为程序员由于工作和思维习惯的不同，已经逐步形成了一种生活方法和定式，这些行事风格让人一眼就可以看出他是一个程序员。

2.4.1.1.1 逻辑思维推理能力强

程序员的工作性质决定了这个特性，开发应用程序过程中的每一个代码组合都是需要较强的逻辑思维推理能力将其堆叠出来。

2.4.1.1.2 有强烈的危机意识

开发应用程序的时候，能够时刻保持警惕，随时准备处理各种危机，面临困难镇定自若，冷静的处理困难，一步一步逼近各种技术难题，并将BUG赶尽杀绝。

2.4.1.1.3 有完美的控制意识

程序员们对解决方案最完美的追求可谓是发挥到了极致，如果有更好的解决方案，则会毫不留情的清除掉数千行耗时数日的代码，用新的解决方案来取代。

程序员最瞧不起利用简单的临时程序处理关键问题的不负责任的编程行为，比如：为了解决守护进程调度的多线程程序中进程无法正常退出的异常问题，采用编写一个脚本在进程启动前 KILL 全部该名称进程的做法，是优秀程序员们嗤之以鼻的。

2.4.1.1.4 人生的战略规划意识

程序员们对自己未来的发展路径看得很清晰，对自己的每一个发展计划都有比较有高度的战略规划，能够在长期的应用软件开发过程中持之以恒，工作一直保持严谨的工作态度，有张有弛，忙而不乱。

2.4.1.1.5 强烈关注开发细节

程序员们非常关注应用程序开发的细枝末节，对于人机界面中某一个按钮的摆放位置，或者对于代码中的拼写错误，排版不一致，甚至更小的看起来不影响程序运行的小的瑕疵都不能容忍。这正是一个优秀程序员所应该具备的品质。

2.4.1.2 程序员的平凡

程序员们是那么的与众不同，这都是职业习惯，有人也把这些习惯称作是"职业病"。然而，程序员们却又是如此的平凡，他们在具有优秀特征的同时也显露出了一些平凡的欠缺之处。

2.4.1.2.1 逻辑思维推理能力强，但容易钻进牛角尖

程序员们的逻辑思维能力可以让他们在程序的世界中翱翔驰骋，多么复杂的程序在他们面前也都是小菜一碟，任何难题对他们来说都是无往而不利。然而正是由于他们具有较强逻辑推理能力，才使他们对无关紧要的事情也容易陷入逻辑推理惯性思维。

像这样在普通的无关紧要的事物上进行逻辑推理的现象，我们把这个称作是"钻牛角尖"，也正是因为此，才被别人误解为爱"钻牛角尖"的一群人。

2.4.1.2.2 有强烈的危机意识，但总是患得患失

危机无处不在，为了应对一个可能发生但还没有发生的潜在危机，做出数种应对策略，划出多道马奇诺防线，这点用在程序开发上让程序员面对任何风浪都能拿出解决方案。然而，在生活中这种危机意识使程序员总是患得患失，总是在那些还没有发生的事情上苦恼，这也是常有的事情。

2.4.1.2.3 有完美的控制意识，眼睛里揉不得沙子

我们看到程序员们对解决方案最完美的追求可谓是发挥到了极致，这一点用在工作中没有问题，但是，在生活中却是一个绝对完美的追求者，不免让人产生有"眼睛里揉不得一点沙子"的感觉。有时候不能容忍一件事情的一点瑕疵，也是程序员们苦恼的来源。

2.4.1.2.4 人生的战略规划意识，把人生当成编程

虽然程序员们对自己未来的发展路径看得很清晰，对自己的每一个发展计划都有比较高度的战略规划，但是，程序员们往往把生活和人生也看成是编写的程序，生活的每一步都是按照预先设定好的程序运行的，人生如果变成了程序未免太缺乏乐趣了吧。

2.4.1.2.5 强烈关注开发细节，忽略的自己的身体

程序员们非常关注应用程序开发的细枝末节，他们经常因为一个界面中的按钮位置而熬夜，因为一个代码的效率而忘记吃饭，为了一个程序运行的小的瑕疵而一天都不喝一口水。

2.4.2 程序员们就是一般人

程序员们得到了那些"不一般", 但失去了那些"一般", 然而, 毕竟程序员还是普通人, 程序员们啊, 还是应该非常认真的去做一个一般人。

2.4.2.1 工作和生活是两个事情

程序员的欠缺其主要原因就是生活和工作没有区分开来, 工作和生活必须分开才能将好的品质用于工作而并没有将他们带入生活。

工作就是工作, 生活就是生活, 在工作中要全力以赴展现出程序员的全部优秀品质, 在生活中尽可能的忘掉全部工作去拥抱生活, 去享受生活, 只有这样才能成为一个健康快乐的一般人。

2.4.2.2 程序员的头发与桌子

将这两个毫无关联的东西放到一起说, 是因为一个有趣的现象, 一个程序员桌子越乱, 他的头发就越乱, 不知道是巧合, 还是必然规律。

2.4.2.2.1 头发问题

请十二分的注意你的头发, 不用"油光可鉴", 也最好让其"各就各位"。很乱的头发, 并不能代表你的个性, 相反会让你接触的人感觉你的工作风格有问题。

2.4.2.2.2 桌子问题

请二十四分的注意你的办公桌面, 不用"层次分明", 也最好让其"一目了然"。不要把技术书籍散落到你的桌面的各个角落, 你桌子上书的多少并不能代表你的技术水平的高低。

2.4.2.3 编程序要多多补充水分

程序员生活中的这个细节也需要注意, 很多程序员在工作中由于精神往往是处于高度集中的状态, 在这种状态下工作, 造成的直接结果是喝水较少。

人的每天水的摄入量必须是一定的, 否则也会导致程序员的身体问题, 最终影响工作质量, 比如直接影响到大脑的灵活程度。

最近香港卫生署做了一次大型调查, 最终结果表明, 喝水不够, 大脑会迟钝。专家提醒, 一个健康的成年人每天应喝6-8杯(每杯约240毫升)水或其他饮料(如果汁、茶等), 否则极易出现脱水, 对身体健康造成伤害。

作为程序员的我们, 为了让我们以更充沛的精力来编写代码, 减少返工次数, 请务必多多补充水分。

2.4.2.4 增加运动保护视力

较多的程序员们容易出现对着电脑工作长达数小时，保持一个姿势不动，或仅仅做一些细微的坐姿调整等现象。这样的习惯会对腰椎、脊椎、下肢、坐骨等部位造成影响，长久的盯着屏幕，眼睛不能得到休息，从而对视力造成影响。

因此，在工作一段时间后，要活动一下身体，并向尽可能远的方向瞭望，以使的身体和眼睛得到休息。

2.4.3 点评"一般人"

没有比做个一般人更令人向往的事情了，永远不要指望自己是一个超人，因为超人不存在。不要让自己离正常的社会越来越远，不要让自己的身体受到伤害。

一般人：

是地球上数量最多的人。

2.5 误区5：存在"软件蓝领"岗位

不知道从什么时候开始在社会流行一种说法，说是在软件行业中存在一个"软件蓝领"的职位概念。这个概念无非就是像蓝领工人一样，只需要按照设计师的要求完成工作，而不需要过多的了解这个程序的原理，也不需要了解整个软件的架构，只要负责组装零配件即可。

2.5.1 "软件蓝领"是个"美梦"

一个事物要存在，就必然要有它存在的价值和有它存在的空间才可能。"软件蓝领"要想存在，同样也必须找到他存在的空间，然而，很遗憾，在现实社会中"软件蓝领"的生存空间实在太有限了。与其说中国软件行业近期不需要"软件蓝领"，还不如说中国软件行业根本就没有"软件蓝领"的生存空间。

"软件蓝领"的理想来自于"软件大工厂"的渴望，人们希望通过人海战术组建成"软件大工厂"，这个工厂中能够容纳上千人甚至上万的软件工人，工厂中工人的大部分工作跟制鞋厂缝制布鞋的女工以及流水线上拧螺丝的工人差不多，高中生经过简单培训就足以胜任。



这个"软件大工厂"把软件开发的流程看成是传统工厂制造商品的生产流水线，讲求规范与标准，把软件的一个个功能模块看成传统商品上的机器零件，各种员工分工明确。

只有"软件大工厂"在当前中国软件行业中出现，才会需要大量的"软件蓝领"。然而在现实中这种"软件大工厂"的想法非常理想化，在中国目前软件行业中并没有出现，因此"软件蓝领"自然没有了生存空间。没有"软件大工厂"这个根基的依托，"软件蓝领"这个理想仍然是一个理想。

2.5.2 "让程序员放弃思考？"是个"噩梦"

软件蓝领就是软件生产线上的工人，是依照软件的详细设计进行编码的程序员。软件蓝领是纯粹的软件工人，他们根本不参与软件的设计，也许根本就不知道自己写的这段代码是用在哪个项目的哪个模块里，他们只知道按照接口和功能规范编写代码。

这就像机械工人按照图纸做螺丝钉，却不知道这颗螺丝用在什么地方。假如真的有"软件蓝领"这样的工作岗位，那一定是程序员的悲哀。

因为，让一个程序员放弃思考本身就是程序员的噩梦。程序员是思维活跃的一群人，你不让他们思考，还不如让他们去死！

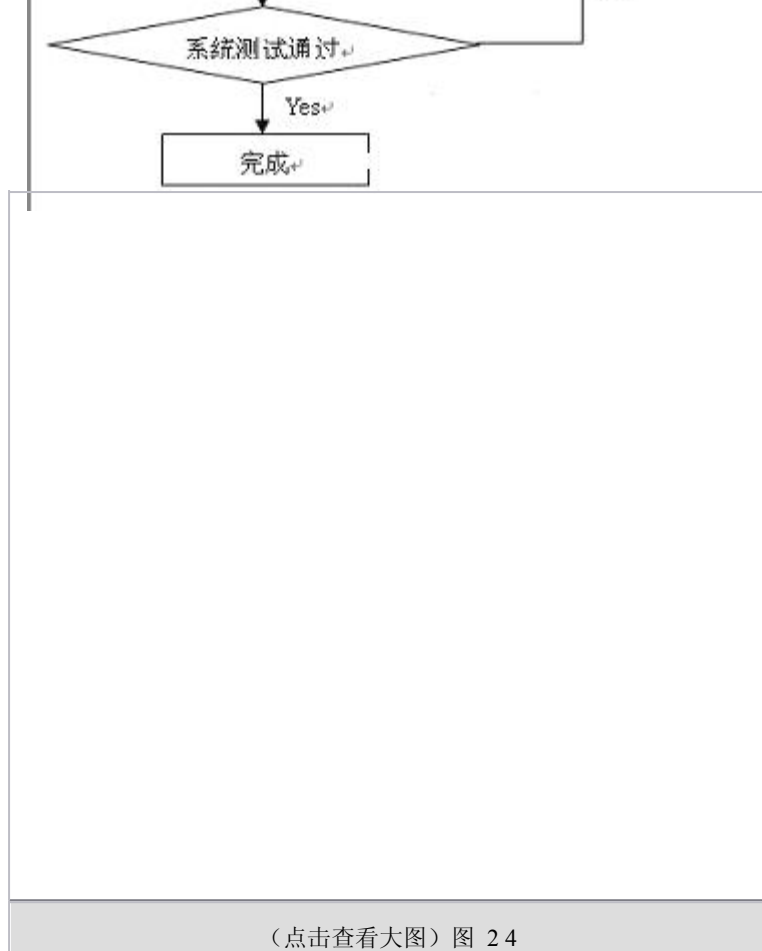
不过万幸的是，这个噩梦不是真的，在当代中国还没有出现这样的岗位。

所以，我们的程序员要有心理准备，你一旦当了程序员，就别想过不动脑子的休闲日子，需要利用你的知识和智慧糊口。

2.5.3 软件开发流程"理想"的"不理想"

在"软件蓝领概念"中存在着一个软件开发流程的"假设"，这个"假设"是大多数软件开发项目的"理想状态"，这个概念的全部理论都来自这个开发流程的假设。

我们来看看这个软件开发总体流程的"理想状态"大概应该包含这么几个阶段，如图2 4所示。



上面所画的流程图是"软件蓝领概念"的开发软件流程的"理想"状态，我们来分别解释一下这个"理想"的项目开发过程的各个阶段。

2.5.3.1 阶段1：项目计划阶段

项目计划阶段是一个项目开发初期的关键阶段，在这个阶段中不需要编写程序，他只是确定项目展开的计划和实施步骤。当公司得到一个商业机会或者按照产品发展计划确定开发某项产品时，根据初步商业计划来完成项目的计划草案，分析项目存在的各项风险并确定优先级，还要制定风险解决方案。本阶段的目的是确立产品开发的经济理由。

在这个阶段还需要制定软件开发计划、人员组织结构定义及配备、过程控制计划。

其中，软件开发计划的目的是收集控制项目时所需的所有信息，项目经理根据项目计划来安排资源需求并根据时间表跟踪项目进度。项目团队成员根据项目计划以了解他们的工作任务、工作时间以及他们所依赖的其他活动等。

可将计划分成总体计划和详细计划，总体计划中每个任务为一个里程碑，详细计划中必须将任务落实到个人。软件开发计划还应包括产品的验收标准及验收任务（包括确定需要制订的测试用例）。

另外，过程控制计划的目的是收集项目计划正常执行所需的所有信息，用来指导项目进度的监控、计划的调整，确保项目按时完成。

2.5.3.2 阶段2：需求分析阶段

需求分析的过程是将用户的需求变成可以实现的软件项目方案的过程，在这个过程中需要不断的和用户进行沟通，需求分析阶段的目的是在系统工作方面与用户达成一致。

在需求分析阶段，需要分析用户的工作流程，数据分布权限，详细说明系统将要实现的所有功能。用户界面原型等方面。

2.5.3.3 阶段3：系统设计阶段

软件开发阶段是将系统分析的结果用软件的形势予以实现，本阶段从物理上实现目标系统。在这个阶段需要软件架构设计、类设计、数据库设计等。

首先要针对软件架构进行设计，说明软件的组织结构、部署结构及运行环境。必要的情况下进行类的设计，定义类之间的关联和类的属性、方法。然后就是数据库设计，定义数据库表之间的关联和各个表的字段。

这个阶段往往是将系统分析文档中的主要内容进行细化，从而形成系统设计文档。系统设计文档直接指导着系统编码的实现。另外，那些关键类的编写也许也可以放到这个阶段来做，当然也可以放到编码阶段来做。如果这个项目不是"白手起家"那些关键类的代码一定都是可重复使用的现成类。

2.5.3.4 阶段4：软件编码阶段

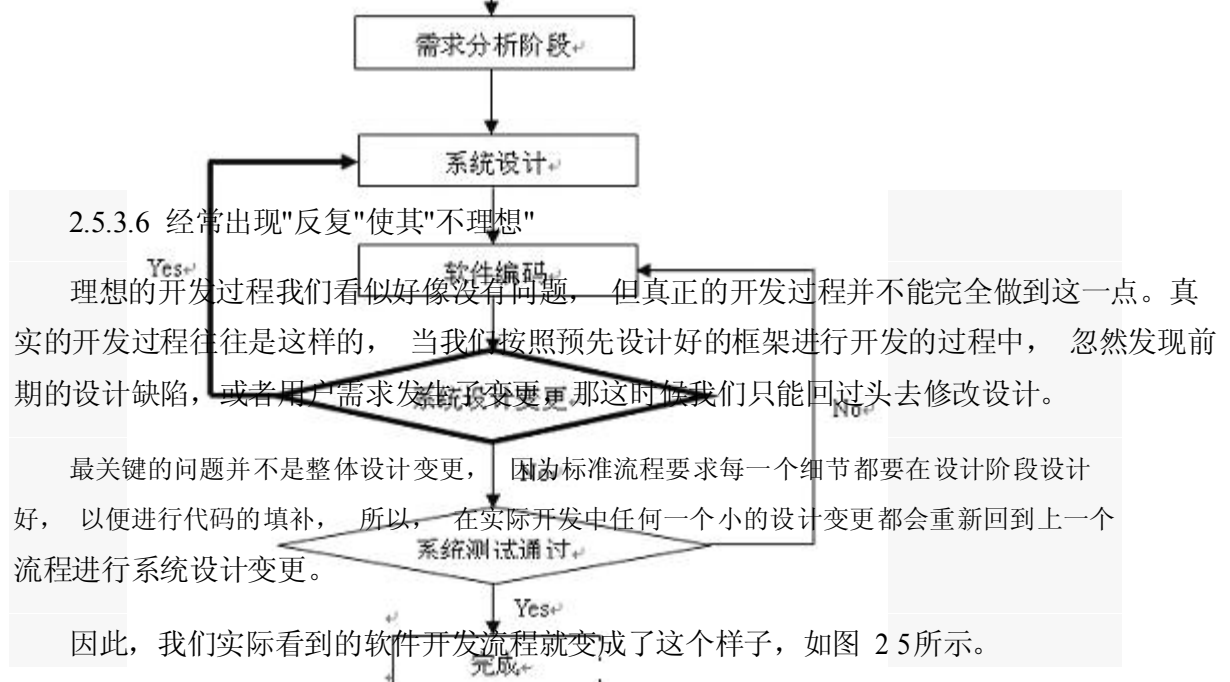
编码阶段是将系统设计阶段的设计内容用编码的方式实现，最终形成可运行的软件代码。在这个阶段需要在系统设计的框架内按照系统设计文档进行编码。

这个阶段往往就是"软件蓝领概念"中所说的那个非常需要人的地方。

这个阶段中不仅仅是需要编码，还需要进行单元测试，每完成一个模块应进行单元测试。最后，进行集成，按软件组织结构的要求将各个子系统组合起来。

2.5.3.5 阶段5：测试阶段

测试的目的是在发布之前找出程序的错误。包括：核实每个模块是否正常运行、核实需求是否被正确实施。首先制定测试计划，收集和组织测试信息，为测试工作提供指导。然后添加测试数据，尽量使用真实数据。最后，测试完成出测试报告，记录测试结果，详细描述问题，提出解决办法



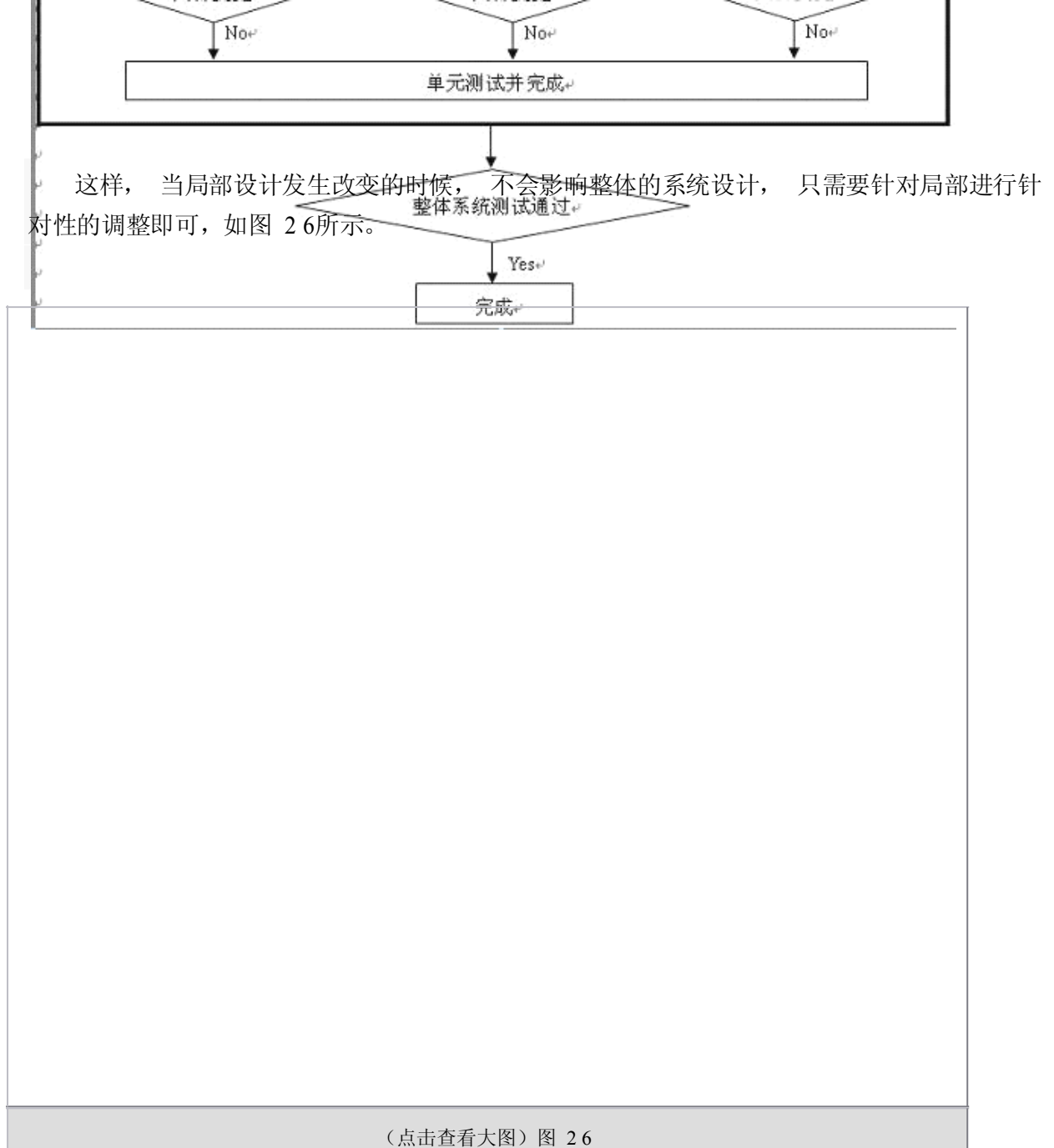
(点击查看大图) 图 2 5

"理想"的开发流程要求系统设计人员能够考虑到开发中的全部细节，只有这样，才能避免在实际填补代码的时候只需填补而不会产生任何疑问。然而，真能做到这一点是非常困难的，做到这一点几乎成为系统设计的"理想"。

正是因为这个过程中的"反复"，才使这个"软件蓝领概念"越来越像一个"理想"。

2.5.3.7 现实中的软件开发流程

为了避免系统设计的频繁变更的问题，软件开发流程被我们聪明的项目经理改了改，将"系统设计"改成了"系统整体设计"，把"软件编码"变成了"局部设计与编码"。



(点击查看大图) 图 2 6

2.5.4 "软件蓝领概念"忽略的东西

"软件蓝领概念"认为，一个软件企业的标准体型应该是上尖下宽的金字塔型，软件企业保持70%到80%的软件蓝领是比较合理的。

"软件蓝领概念"的软件开发人员构成，如图 2 7所示。



(点击查看大图) 图 27

在这个结构中，所谓的高级设计人员不需要太多，只要做设计即可，下面的全部是软件蓝领，去按照系统设计的框架去填补代码。

然而，"软件蓝领概念"忽略了导致这个金字塔形的人员组成结构不能成立两个问题。

2.5.4.1 不是每个人的技术水平都一样

"软件蓝领概念"有一个要求就是处于金字塔下方的基础人员的技术水平相似，以便完成填补代码的工作。

这一点恐怕很难做到，原因是：

软件组成员工作经历不同

工作经验积累程度不同

每个组成人员的职业理想不同

让不同工作经历、不同经验积累和不同职业理想的人组合在一起去填补某个框架的代码，大家想像一下，是多么枯燥的让人窒息的工作啊，这样的开发团队的工作效率与开发质量可想而知。

2.5.4.2 全世界的软件开发不仅仅只是做 MIS 系统

"软件蓝领概念"中所说让蓝领们去填写代码，让系统设计者们去做系统设计，系统设计者把各个模块框架设计好，让蓝领们去填补。大家想一想，这样的开发方法是一个什么样的项目呢？

最好是具备以下特征：

采用技术比较单一

各个项目的技术变化不大

和数据库相关，最好只需要建立针对数据库操作的各个模块

符合这几个特征的项目有哪些呢？

恐怕只有管理信息系统(MIS)的软件项目适合这种开发方法吧， 猜想那些提出"软件蓝领"的人也都是基于这个类型的软件项目而提出的。

但是， 实际软件开发的项目中， 不仅仅包含 MIS 项目， 还有很多种类型的软件项目， 这些软件项目的技术用到的是多种多样的。

我们列举几个非 MIS 系统的例子：

网络游戏相关软件系统

多媒体相关软件系统

网络安全相关系统

网络通信相关系统

厂矿、航空、铁路等大型软硬件平台传感或控制系统

等等，不胜枚举

这些系统虽然都会和数据库有些关系， 但还有很大程度上是和网络或者硬件设备或者相关子系统相关的软件开发项目。 类似这样的软件项目如果仅仅通过填补代码来完成， 恐怕必然是一个理想， 因为， 填补代码的工作所占的比重还没有系统设计的工作量大。

开发部门主管经理

2.5.5 现实中的软件开发团队

开发项目经理(高级软件工程师)

显然"软件蓝领"的这个概念并没有被中国的广大软件Teamleader 所认可， 因为， 大多数软件团队的组成结构并不是一个"金字塔"形， 而更像是一个"阶梯型"。

程序员(或兼任测试)

如图 2-8所示。

实习程序员(实习生或临时员工)

51CTO.com

技术成就梦想

2.5.5.1 角色1：实习程序员(实习生或临时员工)

本来按照常理来说不应该出现这个层次的程序员的，但是我们发现一些公司中的确有这一个层次的程序员。他们往往刚刚毕业或者还没有毕业，以实践学习为自身的最主要目的。

由于这个层次的程序员也许并没有和公司签订正式的劳动合同，仅仅是实习合同，因此，往往称这个类型的程序员为"实习生或者临时员工"。

这部分员工虽然是临时性质但是往往对其工作上的要求并不轻松，有时候和正式员工的工作量是相同的，因为，往往这部分员工会成为正式员工的后备军。

职责：按软件工程师的要求进行编码和单元测试。

要求：良好的编程技能和测试技术。

2.5.5.2 角色2：程序员(或兼任测试)

如果该公司没有实习程序员这个层次，那么程序员这个层次往往是刚刚参加工作的初级程序员。如果，该公司有实习程序员的话，那么往往这部分员工是实习程序员转正的。他们是正式员工和公司签订正式的劳动合同，他们可以独立完成程序开发任务，基本上不需要高阶的编程员的指导即可完成日常开发工作。他们有时候会被称作"程序员"。

这个层次的程序员有时需要兼任单元测试或者整体测试的工作。

职责：按软件工程师的要求进行编码和单元测试。

要求：较强的编码整合能力和良好的编程技能和测试技术。

2.5.5.3 角色3：软件工程师

这个层次是程序员中资格较老技术掌握较全面的那些员工，正因为他们的经验在某些领域比程序员级别的员工丰富，因此，被称作"软件工程师"。

软件工程师这个层次的员工往往独立负责大型软件项目的某个模块，带领几个实习生或者程序员一起开发。他们具有一定的设计能力，可以将系统设计方案变为软件模块，并执行项目经理的项目开发进度要求。

职责：

按项目经理的要求满足项目开发进度和质量要求

指导程序员或实习工程师进行开发

对项目或产品的某个模块负责

要求:

较丰富的软件开发经验

方案的整合能力

指导初级程序员开发的能力

2.5.5.4 角色4: 开发项目经理(高级软件工程师)

开发项目经理对整个产品或者项目的开发质量和进展负责, 往往兼具系统分析和系统设计的工作。开发项目经理有时候就是高级的软件工程师, 具有相对多年的开发经验和管理经验才使其能够胜任该职位。

项目经理负责安排各个模块的具体开发工作, 各个模块的开发负责人向他汇报。项目经理负责统筹各个模块之间的关联关系, 以便最后进行整合。

项目经理对开发部门经理负责, 项目经理有时候直接面向客户, 对客户的要求进行设计实现, 并在开发结束后对客户进行产品交接和培训等工作。

职责:

制定产品或项目的开发目标

制定各个工作的详细任务表, 跟踪这些任务的执行情况, 进行控制

组织会议对程序进行评审

综合具体情况, 对各种不同方案进行取舍并做出决定

协调各项目参与人员之间的关系

要求:

对产品有激情, 具有领导才能

对问题能正确而迅速地做出确定

能充分利用各种渠道和方法来解决问题

能跟踪任务, 有很好地日程观念

能在压力下工作

2.5.5.5 角色5：开发部门主管经理

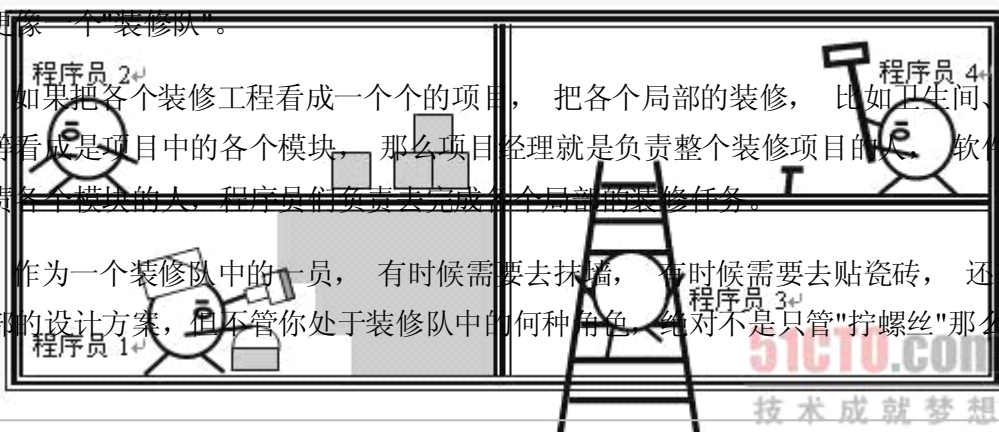
开发部门主管经理对部门所属各个产品的开发质量和进展负责。开发部门主管经理负责各个产品和项目的开发总体目标，协调公司各个方面的资源以满足该部门的业务进展要求，统筹安排人员需求计划，人员晋升计划，以及负责对该部门所述人员的工作绩效进行考评。

2.5.5.6 现阶段的软件团队更像是一个"装修队"

如果非要用一个更形象的词汇来形容当前中国的软件开发团队的话，我想现在的软件团队更像一个“装修队”。

如果把各个装修工程看成一个个的项目，把各个局部的装修，比如卫生间、阳台、木地板等看成是项目中的各个模块，那么项目经理就是负责整个装修项目的人，软件工程师就是负责各个模块的人，程序员们负责去完成各个局部的装修任务。

作为一个装修队中的一员，有时候需要去抹墙，有时候需要去贴瓷砖，还有可能做一些局部的设计方案，但不管你处于装修队中的何种角色，绝对不是只管“拧螺丝”那么简单。



2.5.6 点评"拧螺丝"

如果你想做一个只“拧螺丝”的程序员，恐怕在当前软件行业中找不到你的位置，因为这么简单的工作根本不存在。

拧螺丝：

如果软件开发是“拧螺丝”将不只是程序员的悲哀，而是软件产业的悲哀。

2.6 关于误区

如果把职场看做是"战场", 那么"误区"就是这个战场上的"雷区", 这些"雷区"是阻碍我们程序员前进的障碍。程序员们背起行囊准备好战斗武器, 绕过雷区, 攻入敌方阵地, 去赢取战斗的最后胜利。

第3章 破除 Java 开发中的封建迷信

刚刚看到本章的标题, 大家一定会迷惑不解, 所谓"封建迷信"是什么? 难道在 Java 开发的过程中还有"装神弄鬼"的事情? 我告诉大家, 虽然在Java 开发领域里没有那么多"装神弄鬼"的事情, 但也有很多"故弄玄虚"概念和理论困惑着我们的 Java 程序员。

本章就想打开这些"封建迷信"的枷锁, 使 Java 程序员们更轻松愉快的工作。

3.1 迷信1: Java 占内存到底大不大

"Java 程序的运行会占用非常大的内存", 这一说法恐怕是所有Java 程序员都已经默默承认的说法。然而, 你知道为什么 Java 程序比 C 占用内存吗?

如果你不知道其中的原因, 那就是"迷信", 想不"迷信"还得自己亲自做一个测试, 只有自己亲自做过测试, 弄明白了其中原因才算不"迷信"。

测试环境:

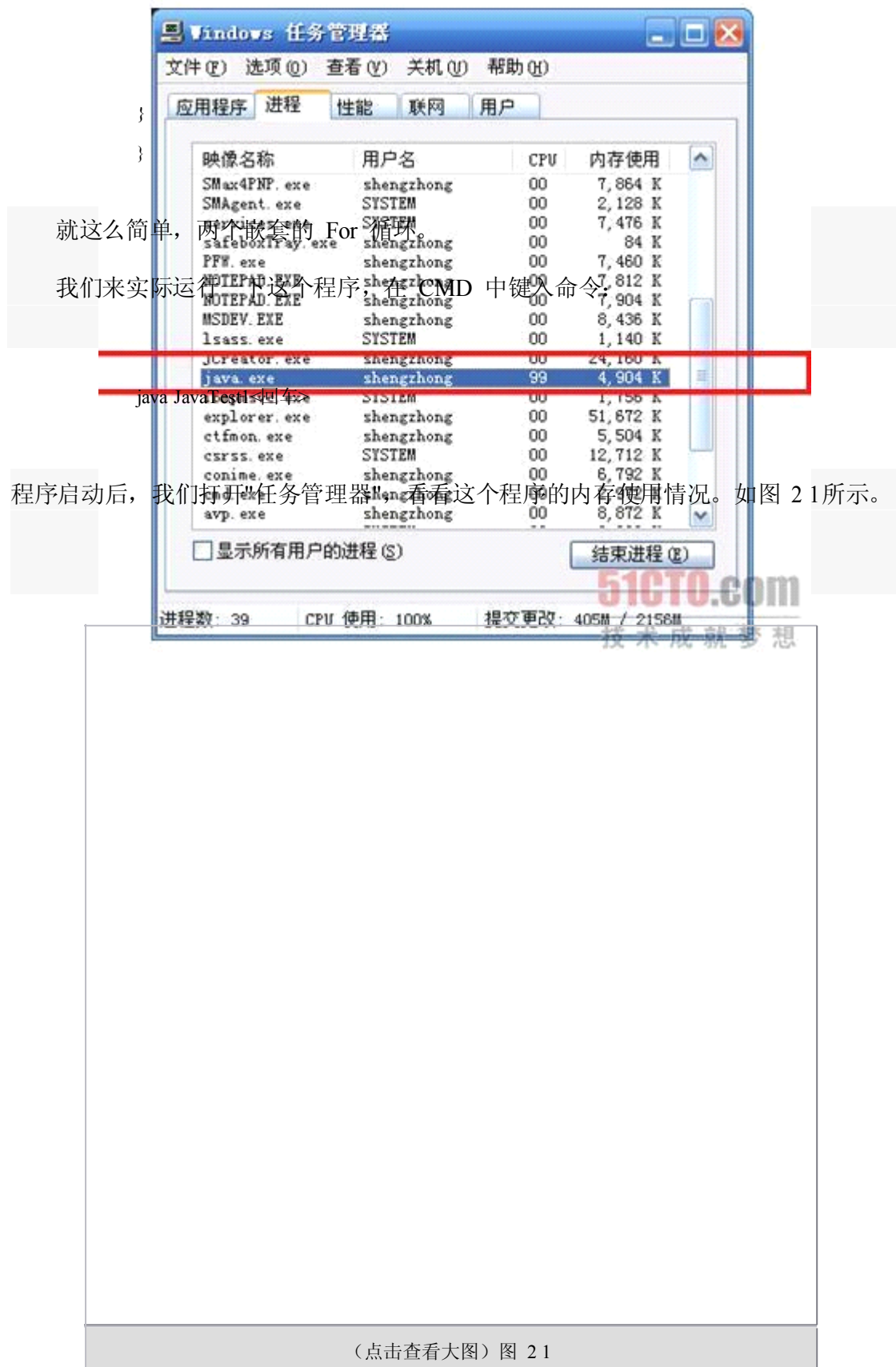
计算机: IBM T42笔记本电脑
CPU: 迅驰 1.7GHz
内存: 1.5GB
JDK: 1.4.2

3.1.1 测试一: 让程序去裸奔

想测试一下到底是不是占用内存, 就将程序去除一切包装, 让那些程序裸奔起来, 再请你告诉我 Java 程序占用多少内存?

裸奔程序代码如下:

```
public class JavaTest1 {  
    public static void main(String[] args) {  
        for(int i=1;i<100000;i++){  
            for(int i2=1;i2<100000;i2++){  
            }  
        }  
    }  
}
```

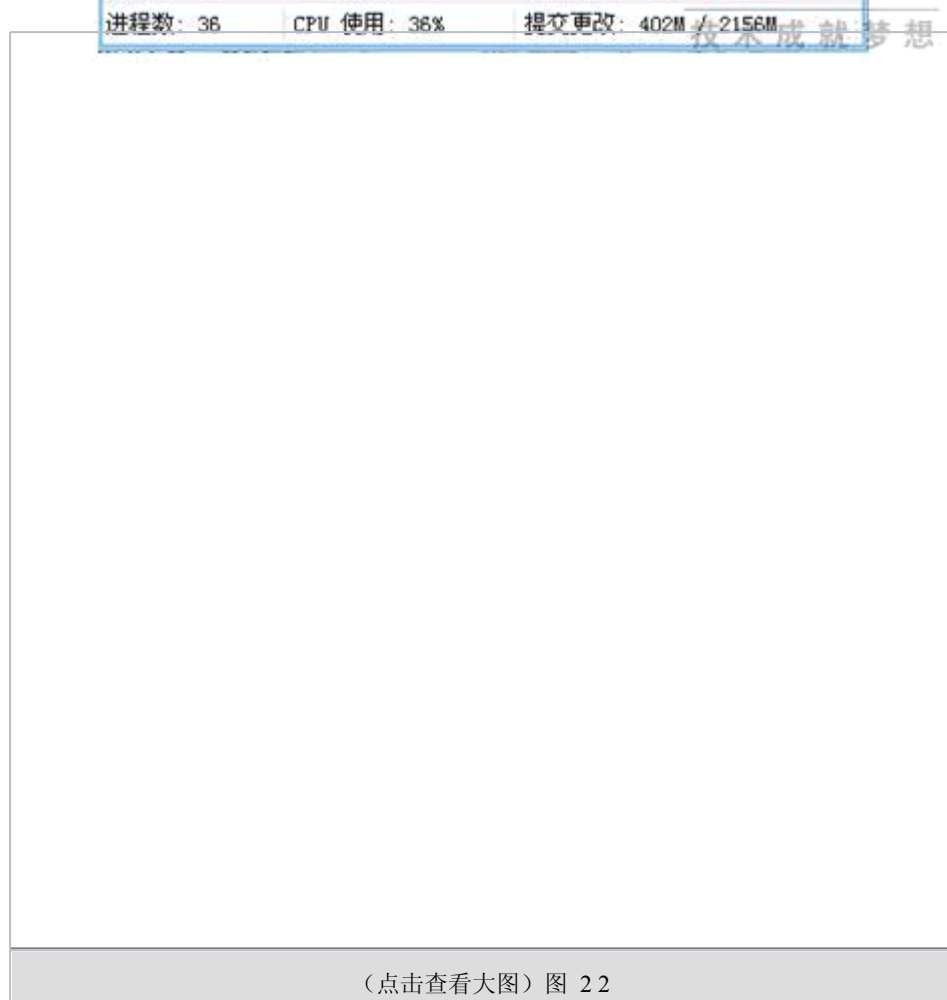
我们看到目前的这个进程的内存占用是 "4904K"，但是，这个进程的数字里还包含了 "java.exe"程序本身的内存占用，我们还需要将"java.exe"自身所占用的内存减去。

在 CMD 中键入命令：

java @###@¥<回车>

java.exe	shengzhong	10	4,368 K
cmd.exe	shengzhong	00	1,136 K
explorer.exe	shengzhong	00	51,796 K
ctfmon.exe	shengzhong	00	5,512 K
csrss.exe	SYSTEM	00	5,308 K
conime.exe	shengzhong	00	7,156 K
cmd.exe	shengzhong	04	3,016 K
avp.exe	shengzhong	00	8,876 K
avp.exe	SYSTEM	06	5,276 K
ati2evxx.exe	shengzhong	00	6,180 K
ati2evxx.exe	SYSTEM	00	2,928 K
AntiArp.exe	shengzhong	00	8,984 K

这是故意键入错误的参数，以便获得"java.exe"自身实际的内存占用量。这时候我们看到任务管理器的内存占用情况，如图 2 2 所示。



(点击查看大图) 图 2 2

任务管理器显示，目前 Java 的内存占用是"4368K"。

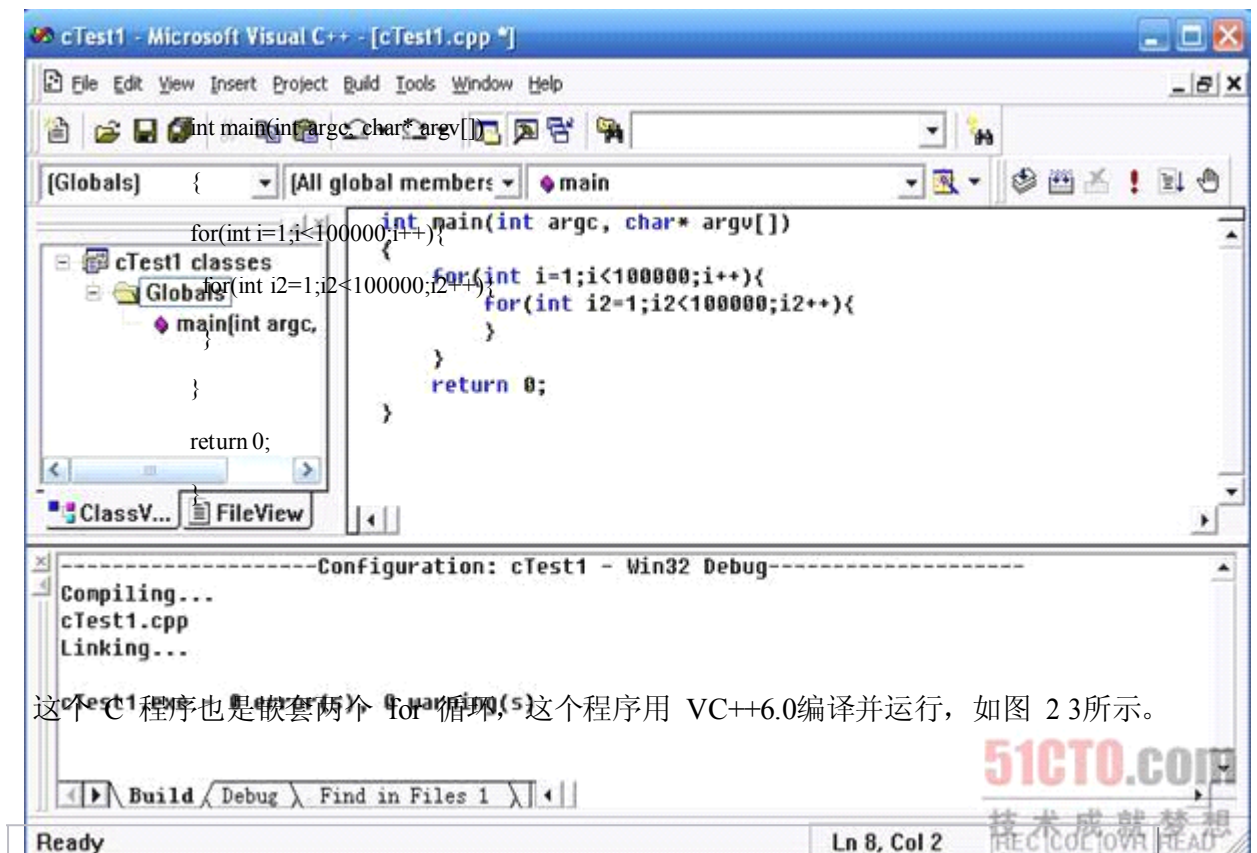
那么，现在我们假设 java.exe 的实际内存占用是"4368K"的话，那么，Helloworld 这个程序的内存占用了多少呢？

$$4904K - 4368K = 536K$$

Java 程序运行结果：Helloworld 的内存实际占用量是"536K"。

好了，我们现在再编辑一个 C 语言的程序。

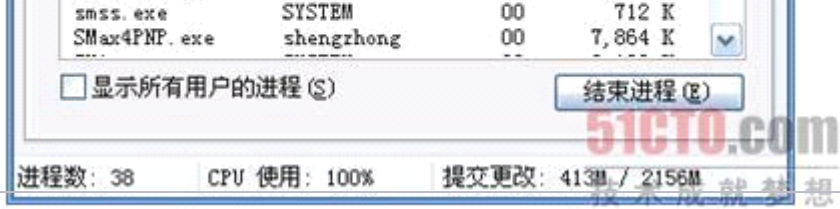
程序如下：



这个 C 程序也是嵌套两个 for 循环，这个程序用 VC++6.0 编译并运行，如图 2 3 所示。

(点击查看大图) 图 2 3

运行后，我们在查看任务管理器的内存占用情况，如图 2 4 所示。



(点击查看大图) 图 2 4

C 程序运行结果：内存实际占用量是"684K"。

测试结论：

总体而言，Java 进程"4904K"比 C 的进程的"684K"还是多了一些。如果去除 Java 本身占用的内存，那么，这个 C 语言程序实际占用的内存为"684K"，和我们刚才的 Java 程序的"536K"非常接近。

3.1.2 测试二：针尖对麦芒

上面的测试程序不过瘾，我们在来一个针尖对麦芒的测试，都去开一个内存空间，看看两个程序的内存增量。

Java 程序如下：

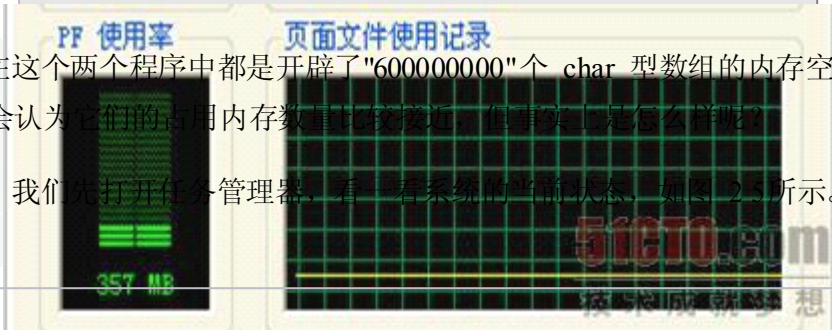
```
public class javaCopyMem {  
    public static void main(String[] args){  
        char[] array=new char[600000000];  
    }  
}
```

C 语言的程序代码如下：

```
void main(int argc, char* argv){  
    char *array=new char[600000000];  
    delete array;  
}
```

我们在这个两个程序中都是开辟了"600000000"个 char 型数组的内存空间，还没有运行前你也许会认为它们的占用内存数量比较接近，但事实上是怎么样呢？

首先，我们先打开任务管理器，看一看系统的当前状态，如图 2.5 所示。



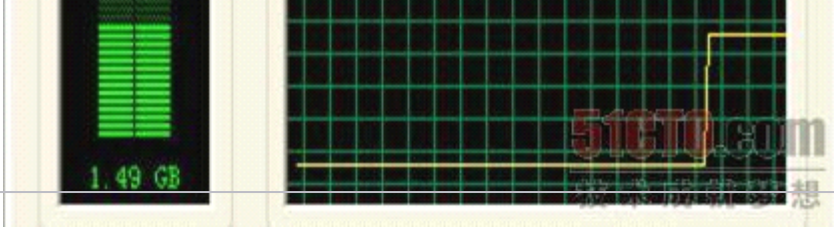
(点击查看大图) 图 2.5

好，现在我们来运行一下 Java 程序，在 CMD 中键入命令：

```
c:\>java -Xmx1240m javaCopyMem
```

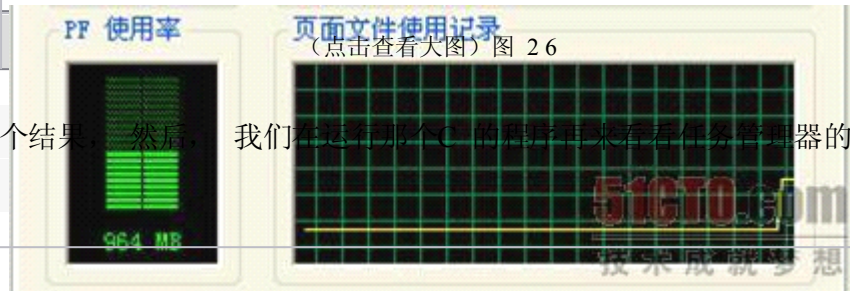
这个命令中的"-Xmx1240m"指的是给 Java 虚拟机开辟的可用堆内存空间是"1240M"，为了做到尽量准确的来开辟可用空间，试了很多个数字，最终确定，只有在给虚拟机开辟至少"1240M"的空间基础上才可以正常运行，否则都会报"java.lang.OutOfMemoryError"的错误。

当我们运行该程序以后，再看看任务管理器的系统状态，如图 2.6 所示。



(点击查看大图) 图 26

看到了这个结果，然后，我们在运行那个C 的程序再来看看任务管理器的系统状态，如图2 7 所示。



(点击查看大图) 图 27

我们发现了一个有趣的现象，在这两个程序中都开了600M 的内存空间，然而，当程序实际运行时 Java 程序所占用的内存一下猛增到了1.49GB，而 C 语言老老实实的增加了600MB。

3.1.3 让人不再"迷信"的测试结果

通过这两个测试程序我们得知，当 Java 仅仅在作循环等操作时并没有比C 占用更多的内存空间，真正使 Java 变得占用内存的东西是开辟内存空间。

这个测试结果才使我们不再"迷信"了，通过亲自的测试活动基本了解了Java 的内存占用的一些规律。

不过这个结果还是挺让 Java 程序员失望的，本来还指望着有一个意外的结果。

但不管怎样，我们至少亲自测试并亲眼目睹了这个结果。

3.1.4 先天与后天

由于 Java 分配内存的原理导致其对内存的占用量相对多一些，这可谓是"先天"的因素，然而，我们在此面前就无可作为了吗？

Windows 程序 WINWORD.EXE 随便打开一个文档也得占个十兆八兆的，MSN 这样的即时通信软件，占用的内存竟然也有60M 左右。

那么从这个角度说，谁占得内存更大呢？

通过我们的测试得知了这个事实之后，对于Java 程序员自身而言养成良好的编程习惯，这就是我们需要"后天"所努力的方向，这样才能更好的扬长避短，发挥好Java 的优势。

3.2 迷信2：Java 和 C 到底谁快

在一本谭浩强主编的 Java 入门教材上是这样描述的：

.....

Java 的语言特点：

.....

9、高性能

设计字节码时已经把机器码的翻译问题考虑进去了，所以实际翻译过程非常简单，编译器在对程序进行优化后生成高性能的字节码。

尽管字节码翻译执行的速度已经足够快，但有时也会要求有更高的性能。程序运行时，字节码将被快速的翻译成当前 CPU 指令，在某种程度上相当于将最终机器指令的产生放在动态加载器中进行。在SUN Microsystems SPARCStation 10计算机上进行的一项30万个方法调用的实验，证明解释型字节码翻译成机器码的速度和C/C++几乎没有区别。

以上的这段文字说了这么一大套，无非就是想告诉Java 程序员，安心用Java 吧，Java 和 C 的速度没有什么区别。

然而，我们却看到很多文章指出，C 语言程序比 Java 程序快多少多少云云，于是，我们每一个程序员也就基本也认同这个观点。但是，过了些日子又有好多人出来"拨乱反正"，说 Java 实际上比 C 快很多。

于是，我们善良的程序员们晕了，到底谁更快，我们应该相信谁？

回答，相信自己吧！

测试环境：

计算机：IBM T42笔记本电脑
CPU：迅驰 1.7GHz
内存：1.5GB
JDK：1.4.2

3.2.1 测试一：让程序转起来

我们来建立两个最简单的程序，一个Java 的程序，一个 C 语言的，这两个程序几乎完全一样， 循环次数也完全一样， 我们分别在程序中来计算循环时间， 让我们实际运行一下看看有什么差距。

Java 程序如下：

```
public class JavaTest {  
    public static void main(String[] args) {  
  
        long startTime =System.currentTimeMillis();  
  
        for(int i=0;i<100000;i++){  
            for(int i2=0;i2<5000;i2++){  
            }  
        }  
  
        long totalTime = System.currentTimeMillis()- startTime;  
  
        System.out.println(totalTime);  
    }  
}
```

C 语言代码如下：

```
#include "StdAfx.h"  
  
#include "windows.h"  
  
void main(void)  
{
```



```

long startTime = GetTickCount();           //取得当前时间作为开始时间

for(int i=0;i<100000;i++){
    for(int i2=0;i2<5000;i2++){
        }
    }

long totalTime = GetTickCount() - startTime; //取得当前时间与开始时间的差值

printf("%d \n",totalTime);                //将差值结果打印到屏幕
}

```

这个程序是一个非常简单的 C 语言程序，读者可以看出来，除了取得当前时间的方法语句不同外，其他语句几乎完全相同，为了让两个程序更像，我们把两个程序所写的行数都力求一致。

测试结果：

循环算子	循环次数	Java 运行时间	C 运行时间
i=100000,i2=100 0	1亿次	260ms	261ms
		271ms	331ms
		261ms	240ms
i=100000,i2=200 0	2亿次	541ms	490ms
		541ms	551ms
		540ms	541ms
i=100000,i2=500 0	5亿次	1562ms	1212ms
		1582ms	1252ms
		1573ms	1232ms
i=100000,i2=100 00	10亿次	3284ms	2423ms
		3265ms	2444ms
		3275ms	2433ms

我们通过这个测试得到了非常直接的结果，我们发现，当循环次数逐渐加大以后，二者

之间的速度差距越来越大，尤其是在5亿次以上时最为明显。然而，在循环次数较少2亿次以下的时候二者的差距并不大。

3.2.2 测试二：读取个大文件吧

刚才我们只是测试一下 for 循环，下面我们再来测试一下针对硬盘上的文件读写速度，看看哪个更快一些。我们还是来建立两个最简单的程序，这两个程序几乎完全一样，都是读取文件后再复制到另一个文件中，让我们实际运行一下看看有什么差距。

Java 程序如下：

```
import java.io.*;

public class CopyBytes {

    public static void main(String[] args) throws
    IOException {
        long startTime =System.currentTimeMillis();

        String sFile;
        String oFile;

        if(args.length<2){
            System.out.println("USE:java CopyBytes source file |
            object file");
            return;
        }
        else{
            sFile = args[0];
            oFile = args[1];
        }
        try {
            File inputFile = new File(sFile);//定义读取源文件
            File outputFile = new File(oFile);//定义拷贝目标文件
            FileInputStream in = new FileInputStream(inputFile);
            BufferedInputStream bin = new BufferedInputStream(in);
            FileOutputStream out = new
            FileOutputStream(outputFile);
            BufferedOutputStream bout = new
            BufferedOutputStream(out);
```

```

int c;
while ((c = bin.read()) != -1){
    bout.write(c);
}
bin.close();
bout.close();
}
catch(IOException e){
    System.err.println(e);
}
long totalTime = System.currentTimeMillis() - startTime;

System.out.println(totalTime);
}
}

```

C 语言程序如下：

```

#include "stdafx.h"
#include "windows.h"
#include <stdio.h>
#include <stdlib.h>
#define MAXLEN 1024

int main(int argc, char *argv[]){

    long startTime = GetTickCount();

    if( argc < 3 ){
        printf("usage: %s %s\n", argv[0], "infile outfile");
        exit(1);
    }

    FILE * outfile, *infile;
    outfile = fopen(argv[2], "wb" );
    infile = fopen(argv[1], "rb");
    unsigned char buf[MAXLEN];

```

```
if( outfile == NULL || infile == NULL )
{
printf("%s, %s",argv[1],"not exit\n");
exit(1);
}

int rc;

while(      (rc      =      fread(buf,sizeof(unsigned      char),
MAXLEN,infile)) != 0 )
{
fwrite( buf, sizeof( unsigned char ), rc, outfile );
}

fclose(infile);

fclose(outfile);

long totalTime = GefTickCount() - startTime;
printf("%d\n",totalTime);
return 0;
}
```

我们在硬盘上建立两个文件，一个是1M，另一个是10M，然后，分别予以测试。

C 和 Java 敲的命令分别是：

```
C:\> Java CopyBytes c:\filea c:\fileb
C:\> Copyfile c:\filea c:\fileb
```

测试结果：

文件大小	Java 运行时间	C 运行时间
1M	90ms	40ms
	80ms	51ms
	70ms	60ms
10M	802ms	210ms
	771ms	250ms

	792ms	320ms
--	-------	-------

我们通过测试得知，二者之间的读写文件的速度存在着明显差距，显然C 在这方面的运行速度要远远快于 Java 的运行速度。

3.2.3 测试三：内存处理的速度

刚才我们测试了针对硬盘文件的读写速度，现在再建立两个最简单的程序，这两个程序仍然几乎完全一样，我们这回要处理的是开一段内存空间，让我们看看他们有什么差距。

Java 程序如下：

```
public class javaCopyMem {
public static void main(String[] args){
    long startTime =System.currentTimeMillis();
    char[] array=new char[10000000];
    long totalTime = System.currentTimeMillis()- startTime;

    System.out.println(totalTime);
}
}
```

C 语言代码如下：

```
#include "stdafx.h"
#include "malloc.h"
#include "windows.h"
#include "stdio.h"

void main(int argc, char* argv[])
{
    long startTime = GetTickCount();
    char *array=new char[10000000];
    long totalTime = GetTickCount() - startTime;
    printf("%d\n",totalTime);
    delete array;
}
```

用以下命令方式执行 Java 程序， 加"-Xmx1260"的目的是为了给 Java 虚拟机开辟更多的内存空间：

```
c:\>java -Xmx1260M javaCopyMem
```

测试结果：

空间大小	Java 运行时间	C 运行时间
10M	30ms	20ms
	40ms	10ms
	40ms	20ms
100M	301ms	160ms
	311ms	190ms
	300ms	170ms
600M	1703ms	851ms
	1812ms	861ms
	1932ms	900ms

我们通过测试得知，二者在处理内存方面的对比更加明显的表现出C 语言的实力，C 语言在内存的处理速度上远远的将 Java 甩到了后面。

3.2.5 也不要过于迷信 C 语言

从以上那三个程序证实，基本上打击了之前谭浩强主编的那个教材中所说的理论， 我猜想，也许那个理论是 SUN 公司给 Java 做的一个广告吧。

当然，本小节中 Java 和 C 语言的这样的测试是一种很极端的情况，如果，C 语言的程序写得非常烂，一样非常慢。比如再看一下下面的这两个程序是谁快？

比如，用 Java 写：

```
int getXXX(int x, int y)
{
    return x+y;
}
```

用 C++写：

```
int getXXX(int x, int y)
{
    for(int i = 0; i < 10000000; i++){
        string ss = "I am so slow";
    }
    return x+y;
}
```

就上面的两个程序而言，当然还是 Java 比 C 快。因此，我们在处理程序开发的具体问题时也不要过分的迷信 C 语言快。

曾经的一个这样的团队：

某个 Java 开发团队项目中遇到了一个"数据导出速度慢"的问题，这个"数据导出速度慢"的原因是由于其导出方式导致大数据量在网络上传递，导致网络传递速度慢进而影响整体导出速度慢是主要原因。

然而，这个团队的主管错误的认为，导致速度慢的原因是由于Java 本身的速度原因导致的。于是乎花大力气招聘C 语言软件工程师，让C 语言工程师来重新完成这个程序的编码。

最终，C 语言工程师开发了近两个月，由于完全仿照原Java 导出程序的程序处理方式，结果是仍然速度很慢，没有任何的改进。

这个团队的开发主管没有从影响导出速度的"大数据量"方面入手解决问题，而是想当然的认为速度的影响主要来自 Java 语言本身，这当然最终导致程序的失败。

3.2.6 Java 语言与 C 语言之间的应用比较

Java 的可以迅速的组建应用程序，它对于我们的开发者来说，建立应用程序的速度要远远的高于 C 语言，如果考虑到网络集群计算环境，Java 的优势就更加明显了。

世界上又有多少人能用 C 语言写出又快又正确的大型程序？

在这些人中间，又有多少人有能力用C 语言写出一个在大型的、异构的网络环境下能够充分发挥各节点计算能力的大规模并行程序？

也就是说，你也许有能力把程序效能提高一倍，从而充分发挥一台价值6000元人民币的PC 的计算潜力，为客户节省1000元钱。但如果是在一个由100台机器组成的大型异构网络并行计算的环境下，你写的 C 程序恐怕性能还会远远低于对应的 Java 程序，更不要说巨大的后期维护成本，而由此带来的损失可能是1000万或者更多。

C 语言能干的 Java 也能干的如下：

网络应用层协议服务程序开发：如WebServer、FTPServer、MailServer、DNSServer等都可以用纯 Java 语言来开发；

嵌入式开发： 基于Linux 的嵌入式程序开发用 Java 都可以做， 因为Java 虚拟机可以很方便的移植，包括专属设备的图形化接口也可以开发独立的GUI；

不愿意用类似"java 类名"这样方法启动 Java 程序， 想跟C 语言一样， 编译出来的程序直接运行，可以用 gcj 去编译 Java 程序；

多媒体开发方面：图像、语音、3D 图像，Java 都可以开发，包括网络游戏和视频会议系统等等，都可以用纯 Java 语言开发完成。

C 语言能干的 Java 做不到的如下：

操作系统驱动程序的开发：如网卡驱动。用Java 直接开发是做不到的，因为操作系统就是 C 写的；

原始套接字的建立：可以利用网卡的混杂模式，获取封包详细信息。目前Java 也没有提供原始套接字，只提供 UDP 和 TCP 套接字的开发；

系统级程序开发：如基于 Windows 系统的消息钩子程序等。

3.3 迷信3：Java 就等于 JSP 吗

近期我们越来越发现，当前的行业内部充斥着一种风气，那就是用 Java 程序制作 Web 应用的项目要比其他项目多的多。 让我们有些Java 程序员逐渐淡忘了 Java 其实可以做的不仅仅是 Web 应用。

3.3.1 一个面试的现象

前几天，我参与了一个面试的考核活动，发现了一个有趣现象，请看如下对话：

考官："你知道网络的7层架构吗？"

A 君： （笑）

考官："为何发笑"

A 君："没听说过还有7层架构，只听过3层架构"

考官： （惊）"哪三层？"

A 君："MVC"

考官： （晕）.....

考官："你知道 Java 最擅长的地方是什么？"

B 君："网络"

考官：（喜）"网络的哪些方面？"

B 君："Web 应用"

考官：（汗）.....

面对这个面试现象，有点哭笑不得，这几位我们很难说他们对Java不熟悉，但是，确实搞出了这样的笑话。

由于现在开发基于 Web 应用的程序的比重越来越大，以至于很多程序员从一上班开始就一直用 Java 开发 Web 应用程序，从而导致很多的Java程序员已经对基础越来越陌生了，并且，已经对 Java 的用途逐步的产生了误解。

3.3.2 JSP 开发时间长了的误解

很多是去搞所谓的企业级开发、Web 开发了，没有多少人真的理解 Java 的内存模型，很多使用 Java 多年的人没有写过 Socket 程序，不了解 Java 多线程的开销，更不清楚如何进行性能诊断和调优，而这些是写高水平程序的必备的技能。

我只能说对于这些程序员而言，现在的开发时代不再纯真。

3.3.3 Java 的纯真年代

用 Java 去写跨平台的基础软件，利用Java优秀的网络处理能力，去探寻异构系统跨平台 Java 多线程服务程序。Java 的 Socket 程序也许是你用得最多的一个应用方向。每天都在为 Java 多线程的开销而烦恼，不断的进行性能诊断和系统的调优。

对真实计算机体系结构非常清楚，对于 Java 虚拟出来的那个计算环境更是不在话下。甚至还在研究如何将虚拟机更好的进行移植。

有的时候为了解决 Java 的内存消耗太大的问题彻夜未眠。为了降低内存的消耗，减少与磁盘交换数据的可能性而烦恼。为了让 Java 程序跑得快，不断的去尝试简化应用程序，为了更简化程序甚至尝试的去打开 JDK 的源代码一探究竟。

那个年代 Java 开发崇尚的是自由、直接、透明、简单、高效，要像匕首一样锋利，像战士一样勇猛，像农夫一样朴实，反对繁琐华丽的设计，反对架床迭屋的层层抽象，反对复杂的结构和不必要的灵活性。吃饭就是吃饭，捧起碗来喝酒，甩开膀子抓肉。那个年代没有那么多讲究这个讲究那个，没必要吃饭个饭还要斋戒沐浴，漱口净手，战战兢兢，毕恭毕敬。

现在的情况可复杂多了，明明顶多就是一个Socket 报文吧，还要嵌套个十个八个类才能使用，不仅这样，还要美其名曰未XXX 设计模式。甭说现在一个应用层协议了，就连一个简简单单的 Web 开发也要弄出个这个框架那个框架的，什么都要拉开架子摆足了谱儿，生怕

别人说自己开发的程序不专业不正宗。

其实这些真的不是 Java 语言的问题，Java 本身并不是如此复杂的。从语言本身来看，Java 也可以是轻快直接的，也可是酣畅淋漓的。只是不知道为什么经过如此多年的演变，被冠以如此复杂的开发文化，这种文化也许是IBM 倡导的，也许是某位大师倡导的，甚至也许是 SUN 的主意。

但不管怎样，似乎 Java 的纯真年代已经离我们越来越远了，我们面对如此彷徨与繁复的开发文化，我们又能做些什么呢？

3.3.4 Java 绝对不等于 JSP

Java 不是只能开发 Web 应用程序的，用它可以构造出许许多多更加丰富多彩的应用程序来。

你如果问："如果不做 Web 应用，还能做什么呢？"

那么，我问你："Tomcat 是用什么语言开发的？"

显然，这样的例子还远远不止于 Tomcat，我们下面列出来一些例子：

Jbuilder：一个可视的 Java 编程工具。

Eclipse：一个强大的 Java 编程工具。

HSQLDB：是纯 Java 开发的关系型数据库,并提供 JDBC 驱动存取数据。

Mckoi DataBase：是由纯 Java 开的数据库。

ArgoUML：使用 Java 编写的开源 UML 产品。

Columba：是基于 Java 的 EMail 客户端。

FreeCol：一个Java 开发的游戏，是殖民帝国的一个开源版本。它是一个类似于策略游戏《文明》需要玩家征服新的版图。

.....

每一个伟大的应用程序，看起来都是那么的简单。

3.3.5 努力保持一个纯真的心态

如果，你目前的工作目标就是开发一个很棒的JSP 应用程序，那么，让你保持纯真的办法就是好好的了解一下 WebServer 的工作原理，理解隐藏在 JSP 背后的故事。

只有这样，才能够让你编写的应用程序一尘不染，让你的程序开发的更加直接，更加高效，更加优化。

3.3.6 点评"纯真"

让我们变得不再纯真的并不是 JSP 本身，也不是Web 开发工作本身，让我们不再纯真的是包裹在一个单纯的 Java 开发技术外的复杂的、八股的、晦涩的概念，让我们变得越来越虚伪，越来越务虚。

当你变得不再纯真，甚至有点桎梏的时候：

你才知道纯真是上帝赐予你的最重要的东西。

3.4 迷信5: C/S 与 B/S 相比一无是处

所谓 C/S 结构是一个程序的运行方式，它是Client/Server 的简称，即客户机和服务器结构，而 B/S 结构指的是 Browser/Server 的简称，即浏览器和服务器结构。随着互联网在政治、经济、生活等各个领域的不断发展，使基于浏览器的B/S 的应用程序逐步的发展起来，也有人把这种应用形式称作"瘦客户机"程序。所谓瘦客户机指的就是，在客户端无需安装过多的软件即可以实现与服务器的交互。

当 B/S 程序发展到了今天，C/S 的程序越来越少了，现在C/S 似乎已经成为过时的一种开发方法，B/S 的程序如日中天，一发不可收拾。甚至有时候，我们的好多程序员对C/S 程序嗤之以鼻，一提到 C/S 程序简直就是不屑一顾。

然而，C/S 的程序真的一无是处吗？

3.4.1 B/S 是一个很好的创意

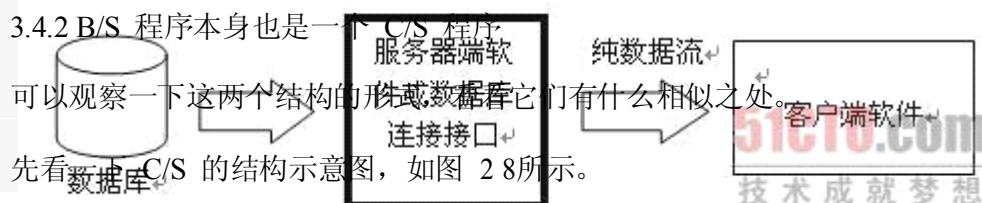
在互联网应用程序应用初期，大多数应用都是建立在C/S 结构基础上的，浏览器的应用促使动态页面的实现，于是就有人提出来，能不能用浏览器作为应用程序的客户端呢？

于是，B/S 程序结构便应运而生，浏览器和服务器结构是对C/S 结构的一种变化或者改进的结构。在这种结构下，用户工作界面是通过WWW 浏览器来实现，极少部分事务逻辑在前端（Browser）实现，但是主要事务逻辑在服务器端（Server）实现，形成所谓三层结构。这样就大大简化了客户端电脑载荷，减轻了系统维护与升级的成本和工作量，降低了用户的总体成本。

以目前的技术看，局域网建立B/S 结构的网络应用，并通过Internet/Intranet 模式下数据库应用，相对易于把握、成本也是较低的。它是一次性到位的开发，能实现不同的人员，从不同的地点，以不同的接入方式（比如LAN, WAN, Internet/Intranet 等）访问和操作共

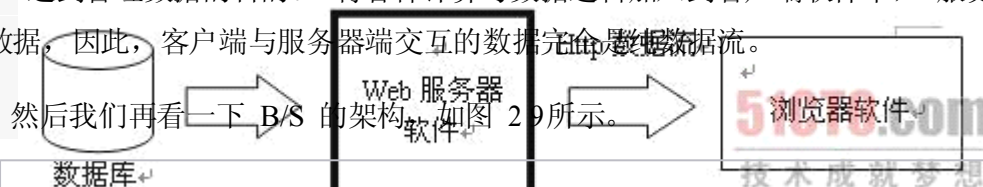
同的数据库；它能有效地保护数据平台和管理访问权限，服务器数据库也很安全。特别是在 Java 这样的跨平台语言出现之后，B/S 架构管理软件更是方便、快捷、高效。

传统的 C/S 体系结构虽然采用的是开放模式，但这只是系统开发一级的开放性，在特定的应用中无论是 Client 端还是 Server 端都还需要特定的软件支持。



(点击查看大图) 图 28

客户端软件是为了完成更多的管理功能而开发的，通过它将与数据库系统进行数据交换，达到管理数据的目的。将各种计算与数据逻辑加入到客户端软件中，服务器端只负责存取数据，因此，客户端与服务器端交互的数据完全是纯数据流。



(点击查看大图) 图 29

B/S 结构的客户端是浏览器，数据逻辑与计算基本上都是在服务器端进行，客户端仅仅是将计算与数据取得的结果予以呈现，而这时候所呈现的内容也依赖服务器端所返回的 Http 数据流中包含的标记。

大家看看这两个结构图的差异，不难发现他们的基本结构极其相似，都是有 "客户端" 的，只不过 B/S 的客户端是浏览器而已。这说明，从本质上来说，B/S 结构的系统也是一种 C/S 的软件。

3.4.3 C/S 程序的优势--速度

他的速度优势就是来自于网络传输的内容。在C/S程序中所传递的是纯数据流，而在B/S中所传递的不仅仅是数据还要包含格式数据，而往往格式信息的数据量会超过纯数据的信息量。另一个影响速度的地方是，逻辑运算部分，B/S的逻辑运算是在服务器端进行，而C/S结构的逻辑运算都分别在不同的客户端进行，因此分担了服务器端程序的CPU资源。如图2 10所示。



C/S 程序的应用服务器运行数据负荷较轻。最简单的C/S 体系结构的数据库应用由两部分组成，即客户应用程序和数据库服务器程序。二者可分别称为前台程序与后台程序。运行数据库服务器程序的机器，也称为应用服务器。一旦服务器程序被启动，就随时等待响应客户程序发来的请求；客户应用程序运行在用户自己的电脑上，对应于数据库服务器，可称为客户电脑，当需要对数据库中的数据进行任何操作时，客户程序就自动地寻找服务器程序，并向其发出请求，服务器程序根据预定的规则作出应答，送回结果，应用服务器运行数据负荷较轻。

数据的储存管理功能较为透明。在数据库应用中，数据的储存管理功能，是由服务器程序和客户应用程序分别独立进行的，在服务器程序中不集中实现。所有这些，对于工作在前台程序上的最终用户，是"透明"的，他们无须过问背后的过程，就可以完成自己的一切工作。

3.4.4 C/S 程序的应用领域

我们了解到 C/S 结构的程序的优势，那么一般 C/S 程序应用到什么领域呢？

速度要求高

实时性要求比较高

用户一旦用上改动不大

比如，超市的销售收款程序，当顾客拿着物品准备交款的时候，却在等着那个缓慢呈现

的 Web 页出现的时候，他会怎么想？因此，C/S 结构并不是一无是处，没必要一提到C/S 结构就嗤之以鼻。对于上述领域 B/S 程序是根本没有办法和 C/S 程序相提并论的。

3.5 迷信6：J2EE 的开发必须用 EJB

最近看到网上有这么一个帖子：

看了很多 EJB 的书和程序，感觉都可以用 JSP 或者 ASP 来写，简单很多。什么时候用 EJB ？？什么时候不用 EJB ？EJB 的资料上说了服务器集群，但是没有一本书能够说清楚 EJB 怎么搞服务器集群！ ！！如果只用一个 Server 要什么 EJB ??，到底什么时候用 EJB 啊，开发一个简单的 JSP 用不用 EJB 啊，谁能告诉我？

看到这个帖子心里都替这个程序员着急啊！

想想这个程序员能不真着急吗？整天看着网上满天飞着"EJB"的文章，这些文章基本都是出自所谓的"高手达人"之手。并且，文章还在不断的说"服务器集群"啊，"企业级开发"啊，等等让程序员们摸不着头脑的虚无概念。但是，就是搞不明白怎么回事，你说他能不急吗？

并不是他一个人着急，估计每一个没弄明白的程序员都挺着急的。这不是什么好事，弄得程序员们产生了一个错觉"J2EE 开发必须要用 EJB"。

3.5.1 EJB 真人真事

一个朋友的故事：

我前些年(大约是02年)在广州认识一个 Java 程序员，当时的 JSP 开发的水平所有别人都不怎么样，没有那么多理论，也没有什么框架之类的，完全是用纯MVC 开发 B/S 的应用程序。这个程序员很聪明，他用自己的方法开发了一套应用系统框架，特别像现在的所谓的"SSH"框架。当时，我觉得他的开发思想真的很先进，而且有自己的想法，充分利用了Java 面向对象的能力，开发出 JSP 的 B/S 应用程序。

这在当时显得并不是很起眼，他自己也说，只是自己琢磨出来的一套编程方法，这样开发应用程序会比较快。他当时根本没有意识到所谓的"框架"这个概念。原因是，"框架"这个概念还没有人炒作。

当时，他应该是27岁左右，我觉得他想法挺好，一定是很有前途的程序员。

然而，两年以后我再和他见面的时候，得知他已经从事技术支持工作了，他放弃了程序员的工作。我很惊讶，问他，你为什么不继续进行编程了？

回答：做技术开发工作我似乎已经作不下去了，新的技术实在是太多了，我觉得我已经

有点跟不上了。

我追问，哪些技术？

答： EJB 啊什么的， 不用EJB 程序搞不大， 做不了企业级开发， 现在没有精力搞他们了。

这时候， 他是29岁， 他毅然决然的放弃了他奋斗多年的技术开发工作， 而放弃的原因竟然是"EJB 搞不明白"！

我不得不给这位兄弟喊一声"冤"！

一个很早就用自己的理念开发出"框架"的高手， 竟然倒在了"EJB"脚下。

我们也许要为了这个兄弟的"气节"鼓掌， 但是， 我们也为他的草率感到"惋惜"。现在又有多少程序员像我们这位兄弟一样， 徘徊在技术开发的十字路口呢？

想必这些程序员都被如下内容搞晕的：

问题1： J2EE 和 JSP 有什么不同？

问题2： 如果 JSP 是 J2EE 的一部分， JSP 是否可以做"企业级开发"？

问题3： EJB 是 J2EE 中专门做"企业级开发"的组件， 那 JSP 怎么和 EJB 结合使用？

结论： 既然EJB 是"群集与企业级开发"的组件， 那么， 不用EJB 一定是效率非常低的了！

带着这些疑问去玩命的查阅各种关于EJB 的书籍， 去找各种理论的网站， 最终也找不到自己想要的答案， 郁闷至极。 最郁闷的是， 将这个所谓的"企业级开发"部署到自己的电脑上都非常困难。

于是， 我们看到了那么多的"强烈疑问"的帖子， 而这些帖子往往也无果而终。 因为， 大家都没有意识到， 这个问题的出发点就错了！

我在这里不得不说一句"EJB 害人不浅啊！"。

3.5.2 我们不禁要问， 什么是"服务集群"？ 什么是"企业级开发"？

既然说了 EJB 是为了"服务集群"和"企业级开发"， 那么， 总得说说什么是所谓的"服务集群"和"企业级开发"吧！

这个问题其实挺关键的， 因为 J2EE 中并没有说明白， 也没有具体的指标或者事例告诉广大程序员什么时候用 EJB 什么时候不用。于是大家都产生一些联想， 认为 EJB"分布式运算"指得是"负载均衡"提高系统的运行效率。然而， 估计很多人都搞错了， 这个"服务群集"和"分布式运算"并没有根本解决运行负载的问题， 尤其是针对数据库的应用系统。

为什么？

我们先把 EJB 打回原形给大家来慢慢分析。

3.5.3 把 EJB 掰开了揉碎了

我们把 EJB 的概念好好的分析一下，看看能发现些什么蛛丝马迹。

3.5.3.1 EJB 概念的剖析

我们先看一下，EJB 的官方解释：

商务软件的核心部分是它的业务逻辑。业务逻辑抽象了整个商务过程的流程，并使用计算机语言将他们实现。

.....

J2EE 对于这个问题的处理方法是将业务逻辑从客户端软件中抽取出来，封装在一个组件中。这个组件运行在一个独立的服务器上，客户端软件通过网络调用组件提供的服务以实现业务逻辑，而客户端软件的功能单纯到只负责发送调用请求和显示处理结果。在J2EE 中，这个运行在一个独立的服务器上，并封装了业务逻辑的组件就是 EJB（Enterprise Java Bean）组件。

这其中我们主要关注这么几点，我们来逐条剖析：

剖析1：所谓："业务逻辑"

我们注意到在 EJB 的概念中主要提到的就是"业务逻辑"的封装，而这个业务逻辑到底是什么？说的那么悬乎，其实这个所谓的"业务逻辑"我们完全可以理解成执行特定任务的"类"。

剖析2：所谓："将业务逻辑从客户端软件中抽取出来，封装在组件中.....运行在一个服务器上"

既然我们知道了"业务逻辑"的概念就是执行特定任务的"类"，那么，什么叫"从客户端软件中抽取出来"？其实，这个就是把原来放到客户端的"类"，拿出来不放到客户端了，放到一个组件中，并将这个组件放到一个服务器上去运行。

3.5.3.2 把 EJB 这个概念变成大白话

变成大白话就是，"把你编写的软件中那些需要执行制定的任务的类，不放到客户端软件上了，而是给他打成包放到一个服务器上了"。

3.5.3.3 发现问题了

不管是用"八股文"说，还是用大白话说这个 EJB 概念都提到了一个词--"客户端软件"。

"客户端软件"? 难道 EJB 的概念中说的是 C/S 软件?

是的, 没错!

EJB 就是将那些"类"放到一个服务器上, 用C/S 形式的软件客户端对服务器上的"类"进行调用。

快崩溃了吧!

EJB 和 JSP 有什么关系? EJB 和 JSP 有关系, 但是关系还真不怎么大, 至多是在 JSP 的服务器端调用远端服务上的 EJB 类, 仅此而已。

3.5.4 EJB 的最底层究竟是什么

我们揭开了 EJB"八股"概念的真谛, 那么, 再来分析 EJB 的底层实现技术, 通过底层实现技术来分析 EJB 的工作方式。

3.5.4.1 EJB 的实现技术

EJB 是运行在独立服务器上的组件, 客户端是通过网络对EJB 对象进行调用的。在Java 中, 能够实现远程对象调用的技术是RMI, 而 EJB 技术基础正是 RMI。通过 RMI 技术, J2EE 将 EJB 组件创建为远程对象, 客户端就可以通过网络调用EJB 对象了。

3.5.4.2 看看 RMI 是什么东东

在说 RMI 之前, 需要理解两个名词:

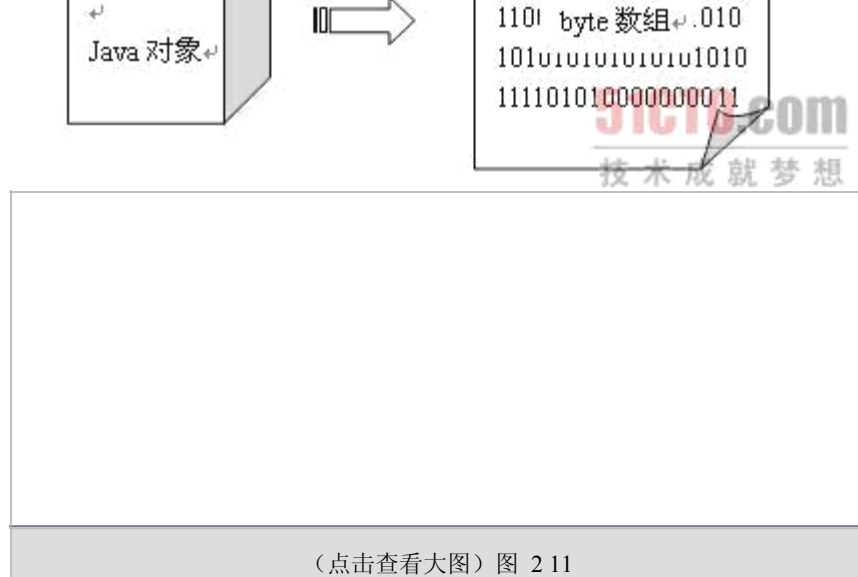
对象的序列化

分布式计算与 RPC

3.5.4.2.1.1 名词1: 对象的序列化

对象的序列化概念: 对象的序列化过程就是将对象状态转换成字节流和从字节流恢复对象。将对象状态转换成字节流之后, 可以用java.io 包中的各种字节流类将其保存到文件中, 或者通过网络连接将对象数据发送到另一个主机。

上面的说法有点"八股", 我们不妨再用白话解释一下: 对象的序列化就是将你程序中实例化的某个类的对象, 比如, 你自定一个类MyClass, 或者任何一个类的对象, 将它转换成字节数组, 也就是说可以放到一个byte 数组中, 这时候, 你既然已经把一个对象放到了byte 数组中, 那么你当然就可以随便处置了它了, 用得最多的就是把他发送到网络上远程的计算机上了。如图 2 11所示。



3.5.4.2.2 名词2：分布式计算与 RPC

RPC 并不是一个纯粹的Java 概念，因为在Java 诞生之前就已经有了RPC 的这个概念，RPC 是"Remote Procedure Call"的缩写，也就是"远程过程调用"。在Java之前的大多数编程语言，如，Fortran、C、COBOL 等等，都是过程性的语言，而不是面向对象的。所以，这些编程语言很自然地用过程表示工作，如，函数或子程序，让其在网络上另一台机器上执行。说白了，就是本地计算机调用远程计算机上的一个函数。

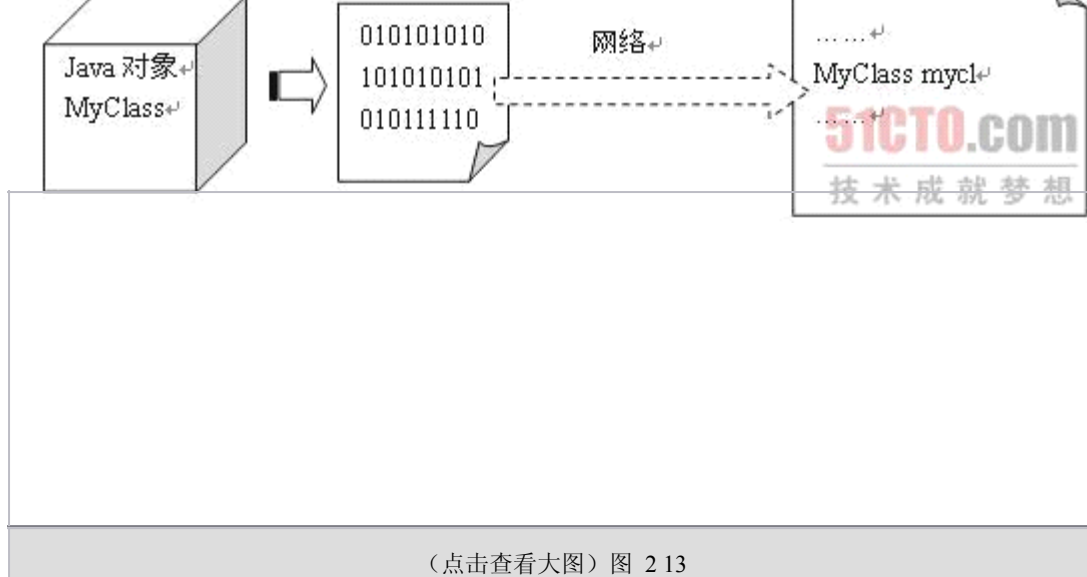
如图 2 12所示。

3.5.4.2.3 名词3：二者结合就是 RMI

RMI 英文全称是"Remote Method Invocation"，它的中文名称是"远程方法调用"，它就是利用 Java 对象序列化的机制实现分布式计算，实现远程类对象的实例化以及调用的方法。说的更清楚些，就是利用对象序列化来实现远程调用，也就是上面两个概念的结合体，利用这个方法调用远程的类的时候，就不需要编写Socket 程序了，也不需要把对象进行序列化操作，直接调用就行了非常方便。

远程方法调用是一种计算机之间对象互相调用对方函数，启动对方进程的一种机制，使用这种机制，某一台计算机上的对象在调用另外一台计算机上的方法时，使用的程序语法规则和在本地机上对象间的方法调用的语法规则一样。

如图 2 13所示。



3.5.4.2.3.1 优点

这种机制给分布计算的系统设计、编程都带来了极大的方便。只要按照RMI 规则设计程序，可以不必再过问在 RMI 之下的网络细节了，如：TCP 和 Socket 等等。任意两台计算机之间的通讯完全由 RMI 负责。调用远程计算机上的对象就像本地对象一样方便。

RMI 可将完整的对象作为参数和返回值进行传递，而不仅仅是预定义的数据类型。也就是说，可以将类似 Java 哈希表这样的复杂类型作为一个参数进行传递。

3.5.4.2.3.2 缺点

如果是较为简单的方法调用，其执行效率也许会比本地执行慢很多，即使和远程Socket 机制的简单数据返回的应用相比，也会慢一些，原因是，其在网络间需要传递的信息不仅仅包含该函数的返回值信息，还会包含该对象序列化后的字节内容。

3.5.4.3 EJB 是以 RMI 为基础的

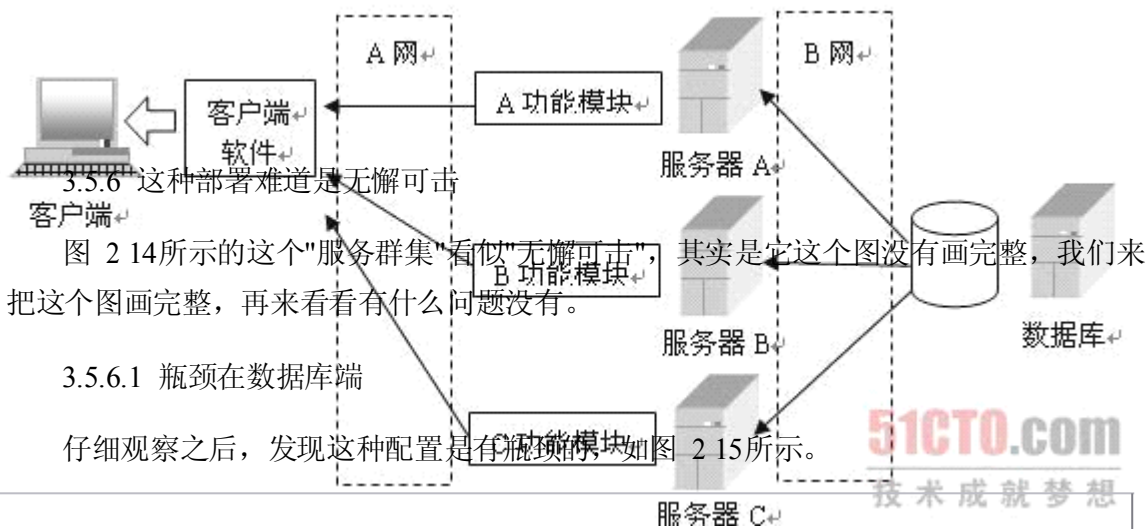
通过 RMI 技术，J2EE 将 EJB 组件创建为远程对象，EJB 虽然用了 RMI 技术，但是却只需要定义远程接口而无需生成他们的实现类，这样就将RMI 技术中的一些细节问题屏蔽了。

但不管怎么说，EJB 的基础仍然是 RMI，所以，如果你想了解 EJB 的原理，只要把 RMI 的原理搞清楚就行了。你也就弄清楚了什么时候用EJB 什么时候不需要用 EJB 了。

3.5.5 EJB 中所谓的"服务群集"

既然已经知道了，RMI 是将各种任务与功能的类放到不同的服务器上，然后通过各个服务器间建立的调用规则实现分布式的运算，也就明白EJB 所谓的"服务群集"的概念。

就是将原来在一个计算机上运算的几个类，分别放到其他计算机上去运行，以便分担运行这几个类所需要占用的 CPU 和内存资源。同时，也可以将不同的软件功能模块放到不同的服务器上，当需要修改某些功能的时候直接修改这些服务器上的类就行了，修改以后所有客户端的软件都被修改了。如图 2 14所示。



3.5.6 这种部署难道是无懈可击

图 2 14所示的这个"服务群集"看似"无懈可击",其实是它这个图没有画完整,我们来把这个图画完整,再看看有什么问题没有。

3.5.6.1 瓶颈在数据库端

仔细观察之后,发现这种配置是有瓶颈的,如图 2 15所示。

(点击查看大图) 图 2 15

我们看看图 2 15的结构图,现在如果想实现各个服务器针对同一个数据库的查询,那

么，不管你部署多少个功能服务器，都需要针对一个数据库服务器进行查询操作。也就是说，不管你的"计算"有多么"分布"也同样需要从一台服务器中取得数据。虽然，看起来将各个功能模块分布在不同的服务器上从而分担了各个主计算机的CPU资源，然而，真正的瓶颈并不在这里，而是数据库服务器那里。数据库服务器都会非常忙的应付各个服务器的查询及操作请求。

因此，通过这个结构图使我们了解数据库根本不能完全解决负载的问题，因为，瓶颈并不在功能模块的所在位置，而是在数据库服务器这里。

3.5.6.2 假如分开数据库，数据共享怎么办

有的读者一定会想到下面的这个应用结构，如图 2 16所示。



(点击查看大图) 图 2 16

就是把每一个功能服务器后面都部署一个数据库，这样不就解决了上节所说的的问题了吗？是的解决了数据库查询负载的问题，然而又出现了新的问题，就是"数据共享"的问题就又不容易解决了。

3.5.6.3 网络面临较大压力，让你的应用慢如老牛

我们再向前翻看看如图 2 15所示的这种架构中存在两个网络，一个是"A 网"一个是"B 网"，这两个网络是不同的。"B 网"往往是局域网，一般带宽是10M/100M，速度较快，因此到还好说，然而，"A 网"往往是互联网或者是利用电信网络互联VPN 网或称广域网。"A 网"的特点是带宽一般较窄，如 ADSL 的网络仅仅有512K-2M 的带宽，由于广域网互联的成本较高，所以一般不会有较高的带宽。

而在这个网络上恰恰跑的是功能模块和客户端软件之间交换的数据，而这部分数据恰恰优势非常占用带宽的。

因此，这个应用架构其运行速度可以想见是多么的慢了。说句不夸张的话，有点想老牛拉破车一样的慢。

一个如老牛的系统：

目前在中国互联网做运营商网络管理系统的一个大公司，它的一个早期的网管软件就是采用了这种架构来做的 C/S 结构的应用系统。

有一次，我作为评估者来对其应用系统进行评估，将其部署到一个非运营商大型的网络中的时候，便出现了我们上述描述的情况，速度已经到了难以忍受的地步，打开一个流量图，有时候需要用15分钟的时间才能呈现完整。然而，该系统在开发阶段并没有发现这个问题，为什么呢？因为，他们没有考虑到应用的实际用户连接网络的复杂性，从而给该公司造成较大损失，以至于，这个开发架构被最终遗弃。

3.5.7 EJB 活学活用，J2EE 不是必须使用 EJB

通过上面小节的讲解似乎好像 EJB 和开发 Web 应用的 B/S 结构的系统关系并不大，其实倒也不然。我们如果把"客户端程序"理解成某一台服务器，这样也是可以被应用的，而且，如果是服务器互相之间做 EJB 的调用的话，也就不存在广域网带宽限制的问题了。

但是，如下情况尽量就不要使用 EJB 了：

1、较为简单的纯 Web 应用开发，不需要用 EJB。

2、需要与其他服务程序配合使用的应用，但调用或返回的自定义的网络协议可以解决的应用程序,不需要使用 EJB。

3、较多人并发访问的 C/S 结构的应用程序，尽量不要使用EJB。

3.5.8 "技术"不是神，不要动不动就"崇拜"

技术的新概念层出不穷，每一个新的技术概念的推出都得到众多程序员的追捧。然而，技术新并不一定是最好的，适合你的才是最好的。

"技术"不是神，不要随随便便的崇拜。

3.6 点评"迷信"

越是不了解真相，就越容易相信这个事物的可怕。所谓"善男信女"指得就是那些很容易被蛊惑那种善良的男孩和轻信他人的女孩。

我们聪明的 Java 程序员们，可不要做软件开发的"善男信女"，要去敢于挑战权威，敢于质疑一切。

迷信：

当你"迷"了的时候，就什么都"信"以为真了.....

第4章 控制内存的功力

那本谭浩强主编的 Java 入门教材说：

.....

Java 的语言特点：

.....

4、健壮性

.....

Java 用真正的数组代替了 C++的指针运算，可以进行数组元素的越界检查。Java 程序在没有授权的情况下是不能访问内存的。所有这些措施，使Java 程序员不用再担心内存的崩溃，因为根本就不存在这样的条件。

这里要澄清一下，我不是有意要和谭教授过不去，主要是我们目前看到的很多Java 程序员都是看着谭教授主编的各种教材作为研究基础入门的，尤其是关于Java 特性方面的描述，被很多后来的 Java 教材所引用，因此还是有一定代表性的。

以上针对 Java 健壮性的描述，不去斟酌每一个字是否合乎实际情况，刚接触Java 的程序员们只是从这段文字的大面上感觉到一个暗示："Java 和内存无关了，可以放心大胆的编程了！"

然而，事实上真的是这样的吗？Java 的程序永远没有因为内存问题而导致的程序崩溃了吗？

天！谁告诉你的？！

4.1 别指望 Java 和内存无关

如果认为"Java 和内存无关了"那你一定是百分之百的错了！要知道，变量、数组、对象等等的操作其实都是针对内存的操作。

那么 Java 的"健壮性"中提到的对内存操作的安全与方便性是什么？其实只是因为Java 中没有指针的概念，不需要像 C++那样去 Delete 堆上开辟的空间，从而使程序员们可以把主要精力放到程序的开发逻辑上去，从而避免了更多的开发隐患问题。

虽然 Java 具有这个优势，但无论如何也不要指望Java 和内存无关。

4.2 容易被搞晕的--堆和栈

由于"堆"和"栈"这两个概念是看不见摸不着的东西， 让很多程序员都整不明白是怎么回事， 其实这两个概念也没有什么好研究的， 因为堆和栈程序员根本没有办法控制其具体内容。

我们只需要了解一点， 栈与堆都是 **Java** 用来在内存中存放数据的地方就行了。然后再弄清楚这两个概念分别对应这程序开发的什么操作， 以及堆和栈的区别即可。

4.2.1 堆--用 **new** 建立， 垃圾自动回收负责回收

1、堆是一个"运行时"数据区， 类实例化的对象就是从堆上去分配空间的；

2、在堆上分配空间是通过"**new**"等指令建立的；

3、**Java** 针对堆的操作和 **C++** 的区别就是， **Java** 不需要在空间不用的时候来显式的释放；

4、**Java** 的堆是由 **Java** 的垃圾回收机制来负责处理的， 堆是动态分配内存大小， 垃圾收集器可以自动回收不再使用的内存空间。

5、但缺点是， 因为在运行时动态分配内存， 所以内存的存取速度较慢。

例如：

```
String str = new String("abc");
```

就是在堆上开辟的空间来存放 **String** 的对象。

4.2.2 栈--存放基本数据类型， 速度快

1、栈中主要存放一些基本类型的变量（ **int**, **short**, **long**, **byte**, **float**, **double**, **boolean**, **char**） 和对象句柄；

2、栈的存取速度比堆要快；

3、栈数据可以共享；

4、栈的数据大小与生存期必须是确定的， 缺乏灵活性。

例如：

```
int a = 3;
```

就是在堆上开辟的空间来存放 **String** 的对象。

4.2.3 何谓栈的"数据共享"

栈其中一个特性就是"数据共享", 那么什么是"数据共享"呢?

我们这里面所说的数据共享, 并不是由程序员来控制的, 而是JVM 来控制的, 指的是系统自动处理的方式。

比如定义两个变量:

```
int a = 5;  
int b = 5;
```

这两个变量所指向的栈上的空间地址是同一个, 这就是所谓的"数据共享"。

它的工作方式是这样的:

JVM 处理 `int a = 5`, 首先在栈上创建一个变量为a 的引用, 然后去查找栈上是否还有5这个值, 如果没有找到, 那么就将5存放进来, 然后将 a 指向5。

接着处理 `int b = 5`, 在创建完 b 的引用后, 因为在栈中已经有5这个值, 便将 b 直接指向5。

于是, 就出现了 a 与 b 同时指向5的内存地址的情况。

4.2.4 实例化对象的两种方法

对于 `String` 这个类来说它可以用两种方法进行建立:

```
String s = new String("asdf");
```

和

```
String s = "asdf";
```

用这两个形式创建的对象是不同的, 第一种是用`new()`来创建对象的, 它是在堆上开辟空间, 每调用一次都会在堆上创建一个新的对象。

而第二种的创建方法则是先在栈上创建一个 `String` 类的对象引用, 然后再去查找栈中有没有存放"asdf", 如果没有, 则将 "asdf"存放进栈, 并让 str 指向"asdf", 如果已经有"asdf" 则直接把 str 指向"abc"。

我们在比较两个 `String` 是否相等时, 一定是用 `"equals()"`方法, 而当测试两个包装类的引用是否指向同一个对象时, 我们应该用`"= "`。

因此，我们可以通过"=="判断是否相等来验证栈上面的数据共享的问题。

例1:

```
String s1 = "asdf";  
String s2 = "asdf";  
System.out.println(s1==s2);
```

该程序的运行结果是，"true"，那么这说明"s1"和"s2"都是指向同一个对象的。

例2:

```
String s1 =new String ("asdf");  
String s2 =new String ("asdf");  
System.out.println(s1==s2);
```

该程序的运行结果是，"false"，这说明用 new 的方式是生成的对象，每个对象都指向不同的地方。

4.2 容易被搞晕的--堆和栈

由于"堆"和"栈"这两个概念是看不见摸不着的东西， 让很多程序员都整不明白是怎么回事， 其实这两个概念也没有什么好研究的， 因为堆和栈程序员根本没有办法控制其具体内容。

我们只需要了解一点， 栈与堆都是 Java 用来在内存中存放数据的地方就行了。然后再弄清楚这两个概念分别对应这程序开发的什么操作， 以及堆和栈的区别即可。

4.2.1 堆--用 new 建立，垃圾自动回收负责回收

1、堆是一个"运行时"数据区，类实例化的对象就是从堆上去分配空间的；

2、在堆上分配空间是通过"new"等指令建立的；

3、Java 针对堆的操作和 C++的区别就是， Java 不需要在空间不用的时候来显式的释放；

4、Java 的堆是由 Java 的垃圾回收机制来负责处理的，堆是动态分配内存大小，垃圾收集器可以自动回收不再使用的内存空间。

5、但缺点是，因为在运行时动态分配内存，所以内存的存取速度较慢。

例如:

```
String str = new String("abc");
```

就是在堆上开辟的空间来存放 `String` 的对象。

4.2.2 栈--存放基本数据类型，速度快

1、栈中主要存放一些基本类型的变量（`int`, `short`, `long`, `byte`, `float`, `double`, `boolean`, `char`）和对象句柄；

2、栈的存取速度比堆要快；

3、栈数据可以共享；

4、栈的数据大小与生存期必须是确定的，缺乏灵活性。

例如：

```
int a = 3;
```

就是在堆上开辟的空间来存放 `String` 的对象。

4.2.3 何谓栈的"数据共享"

栈其中一个特性就是"数据共享"，那么什么是"数据共享"呢？

我们这里面所说的数据共享，并不是由程序员来控制的，而是JVM来控制的，指的是系统自动处理的方式。

比如定义两个变量：

```
int a = 5;  
int b = 5;
```

这两个变量所指向的栈上的空间地址是同一个，这就是所谓的"数据共享"。

它的工作方式是这样的：

JVM 处理 `int a = 5`，首先在栈上创建一个变量为 `a` 的引用，然后去查找栈上是否还有5这个值，如果没有找到，那么就将5存放进来，然后将 `a` 指向5。

接着处理 `int b = 5`，在创建完 `b` 的引用后，因为在栈中已经有5这个值，便将 `b` 直接指向5。

于是，就出现了 a 与 b 同时指向5的内存地址的情况。

4.2.4 实例化对象的两种方法

对于 String 这个类来说它可以用两种方法进行建立：

```
String s = new String("asdf");
```

和

```
String s = "asdf";
```

用这两个形式创建的对象是不同的，第一种是用new()来创建对象的，它是在堆上开辟空间，每调用一次都会在堆上创建一个新的对象。

而第二种的创建方法则是先在栈上创建一个String 类的对象引用，然后再去查找栈中有没有存放"asdf"，如果没有，则将 "asdf"存放进栈，并让 str 指向"asdf"，如果已经有 "asdf" 则直接把 str 指向"abc"。

我们在比较两个 String 是否相等时，一定是用 "equals()"方法，而当测试两个包装类的引用是否指向同一个对象时，我们应该用"=="。

因此，我们可以通过"=="判断是否相等来验证栈上面的数据共享的问题。

例1：

```
String s1 = "asdf";  
String s2 = "asdf";  
System.out.println(s1==s2);
```

该程序的运行结果是，"true"，那么这说明"s1"和"s2"都是指向同一个对象的。

例2：

```
String s1 =new String ("asdf");  
String s2 =new String ("asdf");  
System.out.println(s1==s2);
```

该程序的运行结果是，"false"，这说明用 new 的方式是生成的对象，每个对象都指向不同的地方。

4.3 内存控制心中有数

如果想对内存控制做到十拿九稳，就必须要做到"明明白白"以及"心中有数"。

4.3.1 两个读取内存信息函数

其实，Java 给我们提供了读取内存信息的函数，这两个函数分别是：

1、Runtime.getRuntime().maxMemory()

得到虚拟机可以控制的最大内存数量。

2、Runtime.getRuntime().totalMemory()

得到虚拟机当前已经使用的内存数量。

4.3.2 开发 Java 程序内存看的见

为了看看这两个函数的运行效果，我们可以编一个小程序来看看这两个程序的实际运行效果，让大家也对内存的占用情况做到"看得见摸得着"。

```
public class MemoryTest{
    public static void main(String args[]){
        String s="abcdefghijklmnop";
        System.out.print("maxMemory:");
        System.out.println(Runtime.getRuntime().maxMemory()/1024/1024+"M");
        System.out.print("totalMemory:");
        System.out.println(Runtime.getRuntime().totalMemory()/1024/1024+"M");
        for(int i=0;i<19;i++){
            s=s+s;
        }
        System.out.print("totalMemory:");
        System.out.println(Runtime.getRuntime().totalMemory()/1024/1024+"M");
    }
}
```

这个程序运行效果如下图 3-1 所示。



从这个程序的输出结果可以发现两个值得注意的地方：

1、 竟然虚拟机可以控制的最大内存数量只有63M， 也就是说， 稍微不留神就会发生内存溢出。

2、 在不断的向 String 中追加字符的情况下，所占用的内存存在不断的增长。

不是开玩笑， Java 虚拟机在默认的情况下只能控制"66650112 byte"即"63.56M"。 那么，我们如何才能让 Java 虚拟机能够控制更多的内存呢？

4.3.3 必须要介绍的虚拟机的参数"-Xmx"

Java 程序员往往似乎已经习惯了使用：

```
java -classpath program [回车]
```

来启动 Java 应用程序，这是用于一般程序的启动方式。其实JVM 其中的相关参数还是有很多的，而这些参数对于应用程序的运行是非常有用的。

我们可以进入 Windows 的 cmd 控制台窗口，键入如下指令：

```
java [回车]
```

在屏幕上可以看到如下内容：

Usage: java [-options] class [args...]

(to execute a class)

or java [-options] -jar jarfile [args...]

(to execute a jar file)

where options include:

-client to select the "client" VM

-server to select the "server" VM

-hotspot is a synonym for the "client" VM
[deprecated]

The default VM is client.

-cp <class search path of directories and zip/jar
files>

-classpath <class search path of directories and
zip/jar files>

A ; separated list of directories, JAR archives,
and ZIP archives to search for class files.

-D<name>=<value>

set a system property

-verbose[:class|gc|jni]

enable verbose output

-version print product version and exit

-version:<value>

require the specified version to run

-showversion print product version and continue

-jre-restrict-search | -jre-no-restrict-search

include/exclude user private JREs in the version
search

-? -help print this help message

-X print help on non-standard options

-ea[:<packagename>...[:<classname>]]

-enableassertions[:<packagename>...[:<classname>]]

enable assertions

```
-da[:<packagename>...[:<classname>]]
-disableassertions[:<packagename>...[:<classname>]]
disable assertions

-esa | -enablesystemassertions
enable system assertions

-dsa | -disablesystemassertions
disable system assertions

-agentlib:<libname>[=<options>]
load native agent library <libname>, e.g. -
agentlib:hprof
see also, -agentlib:jdwp=help and -agentlib:hprof=help

-agentpath:<pathname>[=<options>]
load native agent library by full pathname

-javaagent:<jarpath>[=<options>]
load Java programming language agent, see
java.lang.instrument
```

其他参数基本上都有相关说明，我们不在这里赘述，现在只关注其中的"-X" 这参数。"-X" 的说明中说："print help on non-standard options"，意思是"打印非标准配置参数的帮助信息"，也就是说，如果直接键入"java -X"的话，将会看到非标准参数的说明。

好，继续在 CMD 中键入：

```
java -X [回车]
```

运行后得到了如下结果：

-Xmixed	mixed mode execution (default)
-Xint	interpreted mode execution only

-Xbootclasspath:<directories and zip/jar files separated by ;>
set search path for bootstrap classes and resources

-Xbootclasspath/a:<directories and zip/jar files separated by ;>
append to end of bootstrap class path

-Xbootclasspath/p:<directories and zip/jar files separated by ;>
prepend in front of bootstrap class path

-Xnoclassgc disable class garbage collection

-Xincgc enable incremental garbage collection

-Xloggc:<file> log GC status to a file with time stamps

-Xbatch disable background compilation

-Xms<size> set initial Java heap size

-Xmx<size> set maximum Java heap size

-Xss<size> set java thread stack size

-Xprof output cpu profiling data

-Xfuture enable strictest checks, anticipating future default

-Xrs reduce use of OS signals by Java/VM
(see documentation)

-Xcheck:jni perform additional checks for JNI functions

-Xshare:off do not attempt to use shared class data

-Xshare:auto use shared class data if possible
(default)

-Xshare:on require using shared class data, otherwise fail.

The -X options are non-standard and subject to change without notice.

现在仍然不管其他参数只看有用的几个参数：

```
-Xms<size> set initial Java heap size
```

设置 JVM 初始化堆内存大小

```
-Xmx<size> set maximum Java heap size
```

设置 JVM 最大的堆内存大小

```
-Xss<size> set java thread stack size
```

设置 JVM 栈内存大小

从这些参数的说明中得知， 我们可以通过这些非标准参数来制定JVM 可以控制的堆内存的大小。只需要在 Java 命令后面加入一个参数"-Xmx"， 这个参数可以指定虚拟机可以控制的内存数量， 这个函数我们在前面章节也曾经见过， 在这里对其详细说明一下。

例如如下命令：

```
java -Xmx1024m -classpath .....
```

这个命令就代表 Java 虚拟机控制的内存为1G， 需要注意的是"-Xmx"和"1024m"之间没有空格， 另外， "1024m"也可以不用"m"而直接写字节数， 但是必须写字节数， 如还是1G 应该写成"1024000000"和"1024m"代表同样的意思。还是上一个小节的那个程序如果我们加上"-Xmx1024m"参数以后， 运行结果如图 3 2所示。

(点击查看大图) 图 3 2

通过我们对虚拟机相关参数及测试方法的了解，使我们做到了Java 内存控制明明白白、心中有数。心中有数了，那就想办法让程序的内存使用效率更高一些吧。

4.4 内存控制效率优化的启示

内存控制效率优化说起来简单做起来难，真正能做到优化，必须从点滴做起，并利用有效的手段加以应用，现在就看看对于控制内存方面都有哪些启示。

4.4.1 启示1: String 和 StringBuffer 的不同之处

相信大家都知道 String 和 StringBuffer 之间是有区别的，但究竟它们之间到底区别在哪里？我们就再本小节中一探究竟，看看能给我们些什么启示。还是刚才那个程序，我们把它改一改，将本程序中的 String 进行无限次的累加，看看什么时候抛出内存超限的异常，程序如下所示：

```
public class MemoryTest{
    public static void main(String args[]){
        String s="abcdefghijklmnop";
        System.out.print("当前虚拟机最大可用内存为:");
        System.out.println(Runtime.getRuntime().maxMemory()/1024/1
024+"M");
        System.out.print("循环前，虚拟机已占用内存:");
        System.out.println(Runtime.getRuntime().totalMemory()/1024
/1024+"M");
        int count = 0;
```

```

while(true){
try {
s+=s;
count++;
}
catch(Error o){
System.out.println("循环次数:"+count);

System.out.println("String 节 字 实
数:"+s.length()/1024/1024+"M");

System.out.print("循环后，已占用内存:");

```

The screenshot shows a Windows command prompt window titled "C:\PROGRA~1\XINOS-1\JCREAT-1\GE2001.exe". The output text is as follows:

```

System.out.println(Runtime.getRuntime().totalMemory()/1024
当前虚拟机最大可用内存为:63M
循环前，虚拟机已占用内存:1M
循环次数:19
String实际字节数:8M
循环后，已占用内存:63.5625M
Catch到的错误:java.lang.OutOfMemoryError
Press any key to continue...

```

A watermark "51CTO.com" is visible in the bottom right corner of the window.

程序运行后，果然不一会功夫就报出了异常，如图 3-3所示。

(点击查看大图) 图 3-3

我们注意到，在String 的实际字节数只有8M 的情况下，循环后已占内存数竟然已经达到了63.56M。这说明，String 这个对象的实际占用内存数量与其自身的字节数不相符。于是，在循环19次的时候就已经报"OutOfMemoryError"的错误了。

因此，应该少用String 这东西，特别是 String 的"+"操作，不仅原来的String 对象不能继续使用，而且又要产生多个新对象，因此会较高的占用内存。

所以必须要改用 StringBuffer 来实现相应目的， 下面是改用 StringBuffer 来做一下测试：

```
public class MemoryTest{
    public static void main(String args[]){
        StringBuffer s=new StringBuffer("abcdefghijklmnop");
        System.out.print("当前虚拟机最大可用内存为:");
        System.out.println(Runtime.getRuntime().maxMemory()/1024/1024+"M");
        System.out.print("循环前，虚拟机已占用内存:");
        System.out.println(Runtime.getRuntime().totalMemory()/1024/1024+"M");
        int count = 0;
        while(true){
            try {
                s.append(s);
                count++;
            }
            catch(Error o){
                System.out.println("循环次数:"+count);
                System.out.println("String 节 字 实
                数:"+s.length()/1024/1024+"M");
                System.out.println("循环后，已占用内存:");
                System.out.println(Runtime.getRuntime().totalMemory()/1024/1024+"M");
                System.out.println("Catch 到的错误:"+o);
                break;
            }
        }
    }
}
```

我们将 String 改为 StringBuffer 以后，在运行时得到了如下结果，如图 3 4所示。



这次我们发现，当StringBuffer 所占用的实际字节数为"16M"的时候才产生溢出，整整比上一个程序的 String 实际字节数"8M"多了一倍。

4.4.2 启示2：用"-Xmx"参数来提高内存可控制量

前面我们介绍过 "-Xmx" 这个参数的用法，如果我们还是处理刚才的那个用 StringBuffer 的 Java 程序，我们用"-Xmx1024m"来启动它，看看它的循环次数有什么变化。



那么通过使用"-Xmx"参数将其可控内存量扩大至1024M 后， 那么这个程序到了1G 的时候才内存超限，从而使内存的可控性提高了。

但扩大内存使用量永远不是最终的解决方案， 如果你的程序没有去更加的优化， 早晚还是会超限的。

4.4.3 启示3：二维数组比一维数组占用更多内存空间

对于内存占用的问题还有一个地方值得我们注意，就是二维数组的内存占用问题。

有时候我们一厢情愿的认为：

二维数组的占用内存空间多无非就是二维数组的实际数组元素数比一维数组多而已，那么二维数组的所占空间，一定是实际申请的元素数而已。

但是， 事实上并不是这样的， 对于一个二维数组而言， 它所占用的内存空间要远远大于它开辟的数组元素数。下面我们来看一个一维数组程序的例子：

```
public class MemFor{
    public static void main (String[] args) {
        try {
            int len=1024*1024*2;           //设定循环次数
            byte[] abc=new byte[len];
            for (int i=0;i<len;i++){
                abc[i]=(byte)i;
            }
            System.out.print("已占用内存:");
            System.out.println(
                Runtime.getRuntime().totalMemory()/1024/1024+"M");
        }
        catch(Error e){
        }
    }
}
```

这个程序是开辟了"1024*1024*2"即2M 的数组元素的一维数组， 运行这个程序得到的结果如图 3 6所示，程序运行结果提示"已占用内存： 3M"。



(点击查看大图) 图 3 6

我们再将这个程序进行修改， 改为一个二维数组， 这个二维数组的元素数量我们也尽量和上一个一维数组的元素数量保持一致。

```
public class MemFor{
    public static void main (String[] args) {
        try {
            int len=1024*1024;          //设定循环次数
            byte[][] abc=new byte[len][2];
            for (int i=0;i<len;i++){
                abc[i][0]=(byte)i;
                abc[i][1]=(byte)i;
            }
            System.out.print("已占用内存:");
            System.out.println(
                Runtime.getRuntime().totalMemory()/1024/1024+"M");
        }
        catch(Error e){
        }
    }
}
```

当我们把申请的元素数量未变， 只是将二维数组的行数定为"1024*1024"列数定为"2"， 和刚才的那个一维数组 "1024*1024*2"的数量完全一致，但我们得到的运算结果如图 3 7所示，竟然占用达到了29M 的空间。



我们姑且不管造成这种情况的原因，我们只要知道一点就够了，那就是"二维数组占内存"。所以，在编写程序的时候要注意，能不用二维数组的地方尽量用一维数组，要求内存占用小的地方尽量用一维数组。

4.4.4 启示4：用 HashMap 提高内存查询速度

田富鹏主编的《大学计算机应用基础》中是这样描述内存的：

.....

DRAM：即内存条。常说的内存并不是内部存储器，而是**DRAM**。

.....CPU 的运行速度很快，而外部存储器的读取速度相对来说就很慢，如果CPU 需要用到的数据总是从外部存储器中读取，由于外部设备很慢，.....，CPU 可能用到的数据预先读到 **DRAM** 中，CPU 产生的临时数据也暂时存放在 **DRAM** 中，这样的结果是大大的提高了CPU 的利用率和计算机运行速度。

.....

这是一个典型计算机基础教材针对内存的描述，也许作为计算机专业的程序员对这段描述并不陌生。但也因为这段描述，而对内存的处理速度有神话的理解，认为内存中的处理速度是非常快的。

以使持有这种观点的程序员遇到一个巨型的内存查询循环的较长时间时，而束手无策了。

请看一下如下程序：

```
public class MemFor{  
    public static void main (String[] args) {  
        long start=System.currentTimeMillis(); //取得当前时间
```

```

int len=1024*1024*3;           //设定循环次数

int [][] abc=new int[len][2];

for (int i=0;i<len;i++){

abc[i][0]=i;

abc[i][1]=(i+1);

}

long get=System.currentTimeMillis();           //取得当前时间

//循环将想要的数值取出来，本程序取数组的最后一个值

for (int i=0;i<len;i++){

if ((int)abc[i][0]==(1024*1024*3-1)){

System.out.println("取值结果:"+abc[i][1]);

}

}

long end=System.currentTimeMillis();           //取得当前时间

//输出测试结果

System.out.println("赋值循环时间: "+(get-start)+"ms");

System.out.println("获取循环时间: "+(end-get)+"ms");

System.out.print("Java 可控内存:");

System.out.println(Runtime.getRuntime().maxMemory()/1024/1024+"M");

System.out.print("已占用内存:");

System.out.println(Runtime.getRuntime().totalMemory()/1024/1024+"M");

}

}

```

运行这个程序：

```
java -Xmx1024m MemFor
```

程序的运行结果如下：

取值结果:3145728

赋值循环时间:2464ms

获取循环时间:70ms

Java 可控内存:1016M

已占用内存:128M

我们发现，这个程序循环了3145728次获得想要的结果，循环获取数值的时间用了70毫秒。

你觉得快吗？

是啊，70毫秒虽然小于1秒钟，但是如果你不得不在这个循环外面再套一个循环，即使外层嵌套的循环只有100次，那么，想想看是多少毫秒呢？

回答：70毫秒*100=7000毫秒=7秒

如果，循环1000次呢？

70秒！

70秒的运行时间对于这个程序来说就是灾难了。

面对这个程序的运行时间很多程序员已经束手无策了，其实，Java 给程序员们提供了一个较快的查询方法--哈希表查询。

我们将这个程序用"HashMap"来改造一下，再看看运行结果：

```
import java.util.*;

public class HashMapTest{

    public static void main (String[] args) {

        HashMap has=new HashMap();

        int len=1024*1024*3;

        long start=System.currentTimeMillis();

        for (int i=0;i<len;i++){

            has.put(""+i,""+i);

        }

        long end=System.currentTimeMillis();

        System.out.println("取值结果:"+has.get(""+(1024*1024*3-1)));

        long end2=System.currentTimeMillis();

        System.out.println("赋值循环时间: "+(end-start)+"ms");

        System.out.println("获取循环时间: "+(end2-end)+"ms");

        System.out.print("Java 可控内存:");
```

```
System.out.println(Runtime.getRuntime().maxMemory()/1024/1024+"M");
System.out.print("已占用内存:");
System.out.println(Runtime.getRuntime().totalMemory()/1024/1024+"M");
}
}
```

运行这个程序：

```
java -Xmx1024m HashMapTest
```

程序的运行结果如下：

取之结果:3145727

赋值循环时间:16454ms

获取循环时间:0ms

Java 可控内存:1016M

已占用内存:566M

那么现在用 **HashMap** 来取值的时间竟然不到 1ms，这时我们的程序的效率明显提高了，看来用哈希表进行内存中的数据搜索速度确实很快。

在提高数据搜索速度的同时也要注意，赋值时间的差异和内存占用的差异。

赋值循环时间：

HashMap: 16454ms

普通数组: 2464ms

占用内存：

HashMap: 566M

普通数组: 128M

因此，可以看出**HashMap** 在初始化以及内存占用方面都要高于普通数组，如果仅仅是为了数据存储，用普通数组是比较适合的，但是，如果为了频繁查询的目的，**HashMap** 是必然

的选择。

4.4.5 启示5：用"arrayCopy()"提高数组截取速度

当我们需要处理一个大的数组应用时往往需要对数组进行大面积截取与复制操作，比如针对图形显示的应用时单纯的通过对数组元素的处理操作有时捉襟见肘。

提高数组处理速度的一个很好的方法是"System.arrayCopy()"，这个方法可以提高数组的截取速度，我们可以做一个对比试验。

例如我们用普通的数组赋值方法来处理程序如下：

```
public class arraycopyTest1 {
public static void main( String[] args ){
String
temp="abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuv
wxyz'
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy z
"
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy z
"
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy z
"
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy z
"
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy z
"
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy z
"
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy z
"
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy z
"
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy z
"
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy z
"
"，
char[] oldArray=temp.toCharArray();
char[] newArray=null;
```

```

long start=0L;
newArray=new char[length];
//开始时间记录

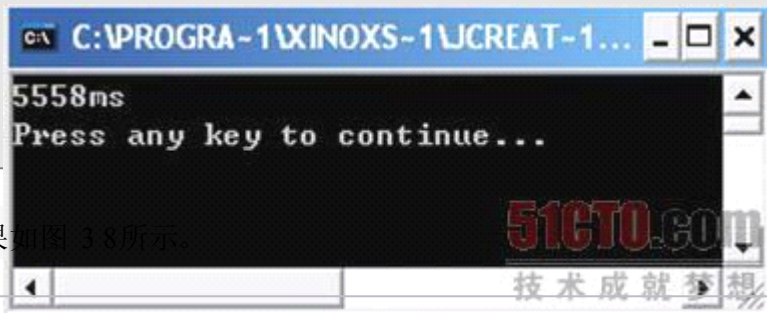
start=System.currentTimeMillis();

for(int i=0;i<10000000;i++){
    for(int j=0;j<length;j++){
        newArray[j]=oldArray[begin+j];
    }
}

//打印总用时间

System.out.println(System.currentTimeMillis()-
start+"ms");
}
}

```



程序运行结果如图 3 8 所示。

(点击查看大图) 图 3 8

那么下面我们再用 `arrayCopy()` 的方法试验一下：

```

public final class arraycopyTest2{
    public static void main( String[] args ){
        String
        temp="abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuv
        wxyz'
        +"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstuvwxy

```

```

"
+"abcdefghijklmnopqrstuvwxyabcdefghijklmnopqrstvwxyz
"
+"abcdefghijklmnopqrstvwxyzabcdefghijklmnopqrstvwxyz
"
+"abcdefghijklmnopqrstvwxyzabcdefghijklmnopqrstvwxyz
"
+"abcdefghijklmnopqrstvwxyzabcdefghijklmnopqrstvwxyz
"
+"abcdefghijklmnopqrstvwxyzabcdefghijklmnopqrstvwxyz
"
+"abcdefghijklmnopqrstvwxyzabcdefghijklmnopqrstvwxyz
"
+"abcdefghijklmnopqrstvwxyzabcdefghijklmnopqrstvwxyz
"
+"abcdefghijklmnopqrstvwxyzabcdefghijklmnopqrstvwxyz
";
char[] oldArray=temp.toCharArray();
char[] newArray=null;
long start=0L;
newArray=new char[length];
//记录开始时间
start=System.currentTimeMillis();
for(int i=0;i<10000000;i++){
System.arraycopy(oldArray,100,newArray,0,120);
}
//打印总用时间
System.out.println((System.currentTimeMillis()-
start)+"ms");
}
}

```

程序运行结果如图 3 9所示。



两个程序的差距再3秒多，如果处理更大批量的数组他们的差距还会更大，因此，可以在适当的情况下用这个方法来处理数组的问题。

有时候我们为了使用方便，可以在自己的tools 包中重载一个 arrayCopy 方法，如下：

```
public      static      Object[]      arrayCopy(int      pos,Object[]
srcObj){
return arrayCopy(pos,srcObj.length,srcObj);
}
public static Object[] arrayCopy(int pos,int dest,Object[]
srcObject){
Object[] rv=null;
rv=new Object[dest-pos];
System.arraycopy(srcObject,pos,rv,0,rv.length);
return rv;
}
```

4.5 内存垃圾回收问题

那本谭浩强主编的 Java 入门教材说：

.....

1、简单性

设计 Java 语言的出发点就是容易编程，不需要深奥的知识。Java 语言的风格十分接近

C++语言， 但要比C++简单得多。Java 舍弃了一些不常用的、 难以理解的、 容易混淆的成分，如运算符重载、 多继承等。 增加了自动垃圾搜集功能， 用于回收不再使用的内存区域。 这不但使程序易于编写， 而且大大减少了由于内存分配而引发的问题。

.....

这样类似的描述出现在众多的 Java 入门级教材中， 非常容易让人们忽略了内存垃圾回收的问题， 其实 Java 的垃圾回收还是需要关注一下的。 这个问题在招聘单位的笔试题中出现的频率也比较高， 我们需要好好的研究一下Java 的垃圾回收机制。

4.5.1 什么是内存垃圾， 哪些内存符合垃圾的标准

我们在前面讲过了， 堆是一个"运行时"数据区， 是通过"new"等指令建立的， Java 的堆是由 Java 的垃圾回收机制来负责处理的， 堆是动态分配内存大小， 垃圾收集器可以自动回收不再使用的内存空间。

也就是说， 所谓的"内存垃圾"是指在堆上开辟的内存空间在不用的时候就变成了"垃圾"。

C++或其他程序设计语言中， 必须由程序员自行声明产生和回收， 否则其中的资源将消耗， 造成资源的浪费甚至死机。 但手工回收内存往往是一项复杂而艰巨的工作。 因为要预先确定占用的内存空间是否应该被回收是非常困难的！ 如果一段程序不能回收内存空间， 而且在程序运行时系统中又没有了可以分配的内存空间时， 这段程序就只能崩溃。

Java 和 C++相比的优势在于， 这部分"垃圾"可以被 Java 虚拟机（JVM）中的一个程序发现并自动清除掉， 而不用程序员自己想着"delete"了。

Java 语言提供了一个系统级的线程， 即垃圾收集器线程（ Garbage Collection Thread）， 来跟踪每一块分配出去的内存空间， 当JVM 处于空闲循环时， 自动回收每一块可以回收的内存。

4.5.1.1 垃圾回收工作机制

垃圾收集器线程它是一种低优先级的线程， 它必须在一个Java 程序的运行过程中出现内存空闲的时候才去进行回收处理。

垃圾收集器系统有其判断内存块是否需要回收的判断标准的。 垃圾收集器完全是自动被执行的， 它不能被强制执行， 即使程序员能明确地判断出某一块内存应该被回收了， 也不能强制执行垃圾回收程序进行垃圾回收。

程序员可以做的只有调用"System.gc()"来"建议"执行垃圾收集器程序， 但是这个垃圾收集程序什么时候被执行以及是否被执行了， 都是不能控制的。 但是虽然垃圾收集器是低优先级的线程， 却在系统内存可用量过低时， 它仍然可能会突发地执行来挽救系统。

4.5.1.2 哪些符合"垃圾"标准

如果了解 JVM 的垃圾回收，就必须要知道 JVM 垃圾回收的标准。

垃圾收集器的"垃圾"标准： 对象已经不能被程序中的其他程序所引用的时候， 那么这个对象的内存空间已经没有用了。

比如当一个方法执行完毕时， 在这个方法中声明的对象就超出其声明周期， 这时候就可以被当作垃圾收集了， 只有当这个方法被再次被调用时才会被重新创建。

例如：

```
.....  
public void function(){  
    OBJ obj=new OBJ();  
    .....  
}  
.....
```

另外还可以将对象的引用变量初始化为null 值， 也可以来暗示垃圾收集器来收集该对象。

例如：

```
.....  
OBJ obj=new OBJ();  
Obj=null;  
.....
```

4.5.1.3 finalize()在该对象垃圾回收前调用

垃圾收集器跟踪每一个对象， 把那些不可到达的对象占有的内存空间收集起来， 并且在每次进行垃圾收集之前， 垃圾收集器都会调用一下finalize()方法。Java 语言允许程序员给任何对象添加 finalize()方法， 但也不能过分依赖该方法对系统资源的回收和再利用， 因为这个方法调用后的执行结果是不可预知的。对于任何给定对象，Java 虚拟机最多只调用一次 finalize 方法。

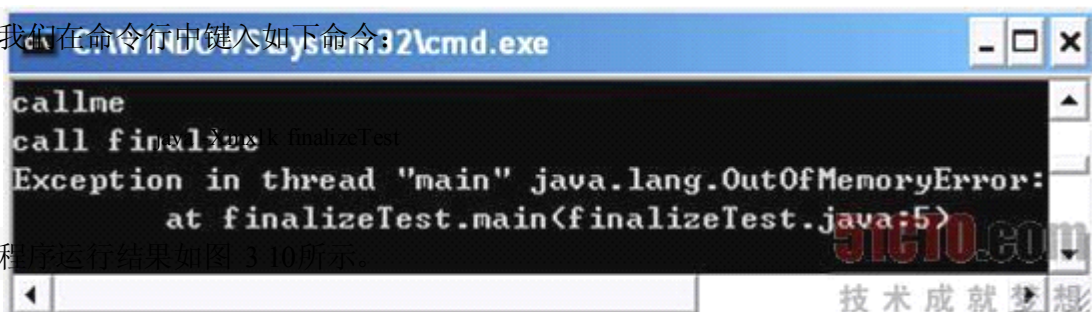
我们用这个程序来演示一下：

```

public class finalizeTest {
public static void main( String[] args ){
    finalizeTest ft=new finalizeTest();
    ft.loading();
    byte bs[]=new byte[1450000];
}
public void loading(){
    test t=new test();
    t.callme();
}
}
class test {
protected void finalize(){
    System.out.println("call finalize");
}
public void callme(){
    System.out.println("callme");
}
}

```

我们在命令行中键入如下命令：



```

callme
call finalize
Exception in thread "main" java.lang.OutOfMemoryError:
    at finalizeTest.main<finalizeTest.java:5>

```

程序运行结果如图 3-10 所示。

(点击查看大图) 图 3-10

这时候，我们将JVM 所许可使用的最大内存设置成"1k"，当内存被占满前JVM 会首先去进行

内存回收，于是失去活动的"test"对象被回收，在回收前调用了"finalize()"。

4.5.2 JVM 垃圾回收的相关知识

JVM 使用的是分代垃圾回收的方式，主要是因为程序运行的时候会有如下特点：

大多数对象在创建后很快就没有对象使用它了。

大多数在一直被使用的对象很少再去引用新创建的对象。

因此就将 Java 对象分为"年轻"对象和"年老"对象，JVM 将内存堆（Heap）分为两个区域，一个是"年轻"区，另一个是"老"区，Java 将这两个区域分别称作是"新生代"和"老生代"。

"新生代"区域中，绝大多数新创建的对象都存放在这个区域里，此区域一般来说较小而且垃圾回收频率较高，同时因为"新生代"采用的算法和其存放的对象的特点，使该区域垃圾回收的效率也非常高。

而"老生代"区域中存放的是在"新生代"中生存了较长时间的对象，这些对象将被转移到"老生代"区。这个区域一般要大一些而且增长的速度相对于"新生代"要慢一些，"老生代"垃圾回收的执行频率也会低很多。

由于 JVM 在垃圾回收处理时会消耗一定的系统资源，因此有时候通过 JVM 启动的时候添加相关参数来控制"新生代"区域的大小，来调整垃圾回收处理的频率非常有用。以便于我们更合理的利用系统资源。

"新生代"区域设置参数是"-Xmn"，用这个参数可以制定"新生代"区域的大小。

我们来举一个例子说明：

我们就用系统自带的程序作为例子，在命令行上键入如下指令：

```
CD C:\java\demo\jfc\SwingSet2[回车]
C:\java\demo\jfc\SwingSet2>java -jar -verbose:gc
-Xmn4m XX:+PrintGCDetails SwingSet2.jar[回车]
```

上面加入了一个新的参数"XX:+PrintGCDetails"，这个参数能够打印出 GC 的详细信息。屏幕输出如下（节选）：

```
[GC [DefNew: 3469K->84K(3712K), 0.0007778 secs]
23035K->19679K(28728K), 0.0009191 secs]
[GC [DefNew: 3284K->171K(3712K), 0.0007283 secs]
22878K->19766K(28728K), 0.0008669 secs]
[GC [DefNew: 3476K->260K(3712K), 0.0008504 secs]
23071K->19855K(28728K), 0.0009862 secs]
[GC [DefNew: 3502K->87K(3712K), 0.0009267 secs]
23096K->19682K(28728K), 0.0010610 secs]
```

我们需要解释一下输出的详细内容的意思，拿第一行输出来说：

"DefNew: 3469K->84K(3712K), 0.0007778 secs"是指"新生代"的垃圾回收情况，这里的意思是从占用3469K 内存空间变为84K 内存空间，用时0.0007778秒。

"23035K->19679K(28728K), 0.0009191 secs"是指总体 GC 的回收情况，整体堆空间占用从23035K 降低到19679K 的水平，用时0.0009191 秒。

那么，这时候我们在将"新生代"的内存设为8M，并把堆的最大可控值设定为32M，再去执行，键入如下指令：

```
java -jar -verbose:gc -Xmn8m -Xmx32m
XX:+PrintGCDetails SwingSet2.jar[回车]
```

得到的结果如下（节选）：

```
[GC [DefNew: 6633K->6633K(7424K), 0.0000684 secs]
[Tenured: 18740K->18820K(24576K), 0.0636505 secs]
25374K->18820K(32000K), 0.0639274 secs]
[GC [DefNew: 6646K->6646K(7424K), 0.0002581 secs]
[Tenured: 18820K->18884K(24576K), 0.0651957 secs]
25467K->18884K(32000K), 0.0658804 secs]
[GC [DefNew: 6611K->6611K(7424K), 0.0000668 secs]
[Tenured: 18884K->18505K(24576K), 0.0931406 secs]
25496K->18505K(32000K), 0.0934295 secs]
```

这个结果说明：

"[DefNew: 6633K->6633K(7424K), 0.0000684 secs]"是指"新生代"的垃圾回收情况，

这里的意思是从占用6633K 内存空间变为6633K 内存空间，用时0.0000684秒。

"25374K->18820K(32000K), 0.0639274 secs"是指总体 GC 的回收情况，整体堆空间占用从

25374K 降低到18820K 的水平，用时0.0639274秒。

"[Tenured: 18740K->18820K(24576K), 0.0636505 secs]"是指"老年代"GC 的回收情况，整

体堆空间占用从18740K 降低到18820K 的水平，用时0.0009012秒。

通过这些参数的调整我们可以看到在处理垃圾收集问题时，从垃圾回收的频率是时间方

面的变化，我们可以根据不同程序的不同情况予以调整。

最后有必要提一下 GC 的相关参数：

-XX:+PrintGCDetails 显示 GC 的详细信息

-XX:+PrintGCApplicationConcurrentTime 打印应用执行的时间

-XX:+PrintGCApplicationStoppedTime 打印应用被暂停的时间

注：":"后的"+"号表示开启此选项,如果是 "-"号那么表示关闭此选项。

4.6 点评"功力"

所谓的"功力"其实就是弄明白一些事情，对事情的深入探究就会比别人明白的东西多，

自然"功力"就越来越深厚了。

功力？

是 N 年磨一剑