从大禹治水看构件与集成

大禹治水

在远古的尧、舜时代，黄河流域经常发生了大水灾，洪水横流，五谷不收，家破人亡。所以尧派鲧去治水，鲧沿用了过去的传统法子，水来土挡，用土筑堤，堵塞漏洞。但由于洪水凶猛，不断冲击土墙，结果弄得堤毁墙塌，洪水反而闹得更凶了。鲧治水九年，劳民伤财，并没有把洪水制服，是一事无成。

舜接替尧后，就把鲧办罪处死，随后命鲧的儿子禹继续治水。大禹领命之后，寻找到了以前治水失败的教训，最后决定用疏导的办法来治理水患。大禹带领百姓是凿了一座又一座大山，开了一条又一条河渠，把黄河的主流加深加宽，把支流疏通与主流相接。同时，把原来的高处培修得更高，把原来的低地疏濬得更深，便自然形成了陆地和湖泽。把这些大小湖泽与大小支流连结起来，洪水就能畅通无阻地流向大海了。

相传大禹三过家门而不入，把整个身心都用在开山挖河的事业中。大禹用疏导的办法治水终于获得了成功。大禹治水，为民造福，永受华夏子孙所称颂，永为炎黄后裔所怀念。

集成与构件

“集成”是看到了信息化建设中的一个个信息孤岛，数据不能交换，资源不能共享，业务不能协同，如同洪水泛滥一样。传统的应用集成 EAI 是典型的“堵”法，可以说总是在事后解决问题，是头痛医头、脚痛医脚，治标不治本的办法。碰到了集成问题，才去想办法去解决，而解决眼前问题的同时又带来更多和更复杂的其它集成问题。所以，就如同鲧治水一样，鲧没有把洪水制服，EAI 当然也不可能从根本上解决信息资源共享利用的问题。

因此解决信息孤岛，要学大禹治水，“疏”比“堵”更重要。“堵”是一时的、眼前的，“疏”是长远的，一劳永逸的。与集成的事到临头相比，构件就是“疏”的方法，是从源头上去堵。在构件体系下，信息资源将按标准、有层次的通过构件展开，数据是构件、展现是构件、流程是构件、服务是构件，一切皆构件。好比大禹治水，开山凿渠是构件库，主流支流是大小构件，贯通无阻是统一标准。

所以，构件可以实现信息资源的大“治”，用计算机术语来讲，就是“同构”，标准统一，架构统一，建设统一，管理统一，开发、部署、运行与维护实现同构，信息孤岛从设计源头上被消灭。

从测试角度看用户手册在软件质量中的地位

对于软件，开发者往往只注意到其功能和性能，而忽略了用户手册。其实用户手册也是衡量软件好坏的一个重要标准。好的用户手册可以帮助用户快速入门，是用户正确、充分使用软件的前提。对于开发者来说，好的用户手册可以减少培训和售后服务的费用。所以在测试中，不能忽略用户手册的重要性，应从以下多个方面考察用户手册的质量。

用户手册的完整性

重点考察用户手册内容的全面性与完整性，从总体上把握用户手册的质量。这一项看似简单，但在实际测试中我们发现，很多开发商还是无法做到这一基本标准。很多软件由于开发过于仓卒，在付诸使用时，用户手册中缺少关于某些模块的说明，让用户使用起来比较困难。在测试工程师的眼里，优秀的用户手册内容应该是包括软件的所有功能模块。

用户手册的描述与软件实际功能的一致性

考察用户手册与软件实际功能的一致程度。当确认用户手册基本完整后，我们还要注意用户手册与实际功能描述是否一致。这种问题往往是由用户手册跟不上软件版本的更新速度

造成的。对用户来说，容易造成对描述不一致的功能的误解和苦螅?进而影响用户对软件的使用。优秀的用户手册应该根据软件的升级而及时更新，手册描述应该与软件实际功能保持一致。

用户手册的易理解性

考察用户手册对关键、重要的操作有无图文说明，文字、图表，是否易于理解。对于关键、重要的操作仅仅只有文字说明肯定是不够的，应该附以图表使说明更为直观、明了。优秀的用户手册应该是图文并举，易于理解。

用户手册提供学习操作的实例

考察对主要功能和关键操作提供的应用实例是否丰富，提供的实例描述是否详细。当前大量软件的用户手册只有简单的图文说明，而无应用实例。这样的用户手册看起来就像是软件界面的简单拷贝，对于用户来说，实际上没有什么帮助。例如财务软件，用户手册就应该提供具体建帐实例及具体帐务处理的实例，这样才能使用户看完用户手册后，能够独立完成新帐套的建立并逐渐学会使用软件处理帐务信息。优秀的用户手册不仅要对主要功能和关键操作提供应用实例，而且对实例的描述应做到详细、充分，易于用户理解。

用户手册的印刷与包装质量

考察用户手册包装的商品化程度，印刷质量。有些用户手册是简单打印、装订而成，过于粗糙，不易于用户保存。优秀的用户手册应提供商品化包装，并且印刷精美。

软件的质量是由各个方面构成的，用户手册就是其中重要的一环。特别是在当前软件业快速增长的时期，软件开发者过于注重功能与性能而忽略用户手册，使得用户手册的质量问题尤显突出。所以对于测试人员应该充分认识到用户手册的重要性，严把用户手册的质量关，以促使软件整体质量有一个提高。

从菜鸟到大师细看程序员的五种层次

软件界一个无可争议的事实是，不同程序员的效率有差别，而且差别很大。许多专家将优秀程序员和一般程序员区分地很清楚。大多数研究得出结论认为，一般程序员跟优秀程序员之间在工作效率和质量上存在 10：1 的关系：优秀程序员和水平较差的程序员的编码时间比例为 1：20；debugging 时间比为 1：25；代码数量比是 5：1；程序执行速度比例是 10：1。而且发现，程序员的代码质量和效率跟工作经验没有关系。

让我们看看一些软件大腕们是如何看待优秀程序员和一般程序员的：

Randall E. Stross：无论是从软件标准、创造性、开发速度、还是设计思路或者解决问题的能力上来说，优秀程序员比差的程序员都何止好一点。

Bill Gates：一个优秀的机床工值一个一般机床工的好几倍，而一个优秀程序员值一个一般程序员的 10000 倍。

Robert C. Martin：90%的代码是由 10%的程序员写出来的。

程序员因此被分为五大类：

1. 大师级程序员（Visionary/Artist Programmer/）

大师级程序员是软件界绝对的稀有种族，他们可以创造出 99.9%的程序员所创造不出来的东西。他们发明新的应用和软件模式来驱动软件产业的发展。Napster, Netscape 以及 World Wide Web 都是大师级程序员创造的。对他们而言，软件更多的是艺术而非科学。在这个级别，速度和质量不是最重要的，他们创造出的财富才是最重要的。许多开发团队或者公司顶多也就一个大师级程序员，通常是这个公司的技术创始人或者 CTO。

2. 开拓者程序员（Trailblazer Programmer）

开拓者程序员通常带来很好的主意和趋势。他们通常是最终产品的原型创作者，他们一天做出的事情大部分程序员需要几周甚至几个月。开拓者程序员总是在尝试新工具、新技术，不断地学习和搜寻方法来提高工作效率，并通常是其他程序员的导师和老师，而且你经常会发现当其他程序员早已离开的时候他们却依然工作到深夜。尽管这样级别的程序员工资很高，但是每个成功的公司或团队还是应该配备一两个开拓者程序员。

3.骨干程序员（ Workhorse Programmer）

骨干程序员是一个公司或者开发团队的脊柱，这些人尽管不是很有创新性，但往往比较高效且值得信赖。给一位骨干程序员一套模板和合适的工具，他们总能以最短的时间交出错误最少的代码。

4.机械程序员（ Drone Programmer）

许多程序员就是朝九晚五地为了填塞下自己钱包的机械程序员。他们不愿意接触新技术、避免学习新事物。许多公司或者开发团队都有许多这样的机械程序员，因为他们很便宜，但岂不知更贵的程序员才真正地更便宜。

5.白痴程序员（ Idiot Programmer）

林子大了什么鸟都有，软件领域也不例外。编程需要抽象和逻辑思维，然而一些尚不具备此能力者由于向往着不错的薪水而加入了该领域。白痴程序员总是对最简单的算法也搞不清楚，他们总是错过软件截止日期，终日无所获。白痴程序员最好的出路就是换行。

从《三十六计》看软件测试之计[1]

《三十六计》是根据我国古代卓越的军事思想和丰富的斗争经验总结而成的兵书，古人用兵最讲究谋略，在中国古代战争史上，精彩的谋略计策层出不穷，令人眼花缭乱，但万变不离其宗，大抵都逃不过这三十六计的范围。时至今日，“三十六计”在我们日常的工作和生活中，同样可以有很广泛的应用。我是一名软件测试工程师，并热爱软件测试这一职业，目前从事测试已有一段时间，我很愿意将自己在从事软件测试工作中积累的一些经验，以及一些心得体会，借助三十六计中的若干计谋加以说明，与诸位同行分享。

总说

【原文】

六六三十六，数中有术，术中有数。阴阳變理，机在其中。机不可设，设则不中。

【解析】

“兵以诈立”，多谋者胜。用兵要讲究谋略，“运筹帷幄，决胜千里之外”。同样的道理，无论从事什么样的工作，都需要讲究方式、方法。有了正确的方式方法，或者适时的运用一些小技巧，往往可以收到事半功倍的奇效。

第一计　瞒天过海

【原文】

备周则意怠；常见则不疑。阴在阳之内，不在阳之对。太阳，太阴。

【译文】

防备周全时，更容易麻痹大意；习以为常的事，也常会失去警戒。秘密潜藏在公开的事物里，并非存在于公开暴露的事物之外。公开暴露的事物发展到极端，就形成了最隐秘的潜藏状态。

【解析】

long,long ago,there is a 很厉害的程序员，名叫关羽，他是计算机专业科班出身，又拥有二十几年的编程开发经验，是当之无愧的资深软件工程师。虽然关羽的专业水平无庸置疑，但是他有一个缺点，就是自视过高，骄傲不可一世，他常常认为自己写的代码十分完美，几乎已经到了自恋的程度。他看不起测试人员，对他们提出的程序错误不仅不屑修改，甚至于

不肯承认，并经常与测试人员起争执。有一年他在湖北荆州负责一个十分重要的大型系统的开发，而负责这个系统测试工作的正是关羽向来都瞧不起的吕蒙。这个吕蒙原本学历不高，只有中专文化程度，并且还不大注重学习，提高自己的能力。直到有一次被他的上司孙权教育了一顿，从此发奋图强，进步神速，技术能力迅速提高，早已不是当日的吴下阿蒙。起先吕蒙将发现的错误上报给关羽，关羽并不理会，还是如同以往一样，找出许多理由来搪塞，一会儿说这是个技术难点，无法修改，一会儿又说这是当初的需求没有写清楚。这吕蒙早就清楚关羽的为人，从此也不与关羽多加争辩，只是兢兢业业的做着自己的工作，将自己在测试过程中发现的所有小错误一一如实记录了下来。等到测试报告出来的时候，关羽傻了眼。由于他一时疏忽而犯下的一个小错误，并且错误扩散到整个系统的每个角落，已经无法修改。客户大为不满，项目终于失败！而老板一气之下也把关羽炒了鱿鱼。关羽的一世英名就这样毁在自己的大意上面。这就是在 IT 业界流传很广，十分有名的“关羽大意失荆州”的故事。

从这个故事中我们可以得到以下几个教训：

1、越是厉害的人物，越容易阴沟里翻船，水平很高的程序员，也很容易因为不注意细节而犯下一些低级的错误。所以身为一名测试者，不能迷信权威或专家，对就是对，错就是错，要勇于怀疑一切。时刻牢记我们代表的是最终用户，并建立这样一个观点：即使一个错误不是程序本身的原因，而是因为操作不便而使用户造成，严格说来，那仍然是一个错误。

从《三十六计》看软件测试之计[2]

2、测试者与开发者的地位是相对独立，但绝不是势同水火，双方彼此同样是项目组的成员，在保证软件产品质量这个大方向上是一致的，彼此都应该互相尊重对方的劳动成果，虚心对待。关羽的直接领导诸葛亮早就告诫过他这一点，让他一定要尊重测试组的劳动成果，不要双方闹翻。可关羽硬是不听，于是造成项目失败。

3、在测试工作中，测试者与程序员的沟通是十分重要的。在双方互相尊重的基础上，彼此都要本着对事不对人的原则，保持严谨的科学态度，共同完成软件的开发。上面的故事中，吕蒙的做法其实也不是十分的正确，不仅遭到了大量关羽的 fans 的指责，而且长久背负着做人不厚道的骂名，这些倒不重要，更为重要的是，最终整个系统、整个开发团队失败了，他同样是个失败者。更好的做法是在测试的工作中，就要注重沟通问题，当一个错误一直不被修改的时候，与开发人员沟通失败后，应该及时上报给项目的管理者，尽早寻求解决的方法，而不是将错误一直留到测试结束后才暴露出来，此时的错误可能已经造成十分严重的后果，测试报告写得再漂亮也已经没有多大的意义。基于以上陈词，本庭宣判：本案关羽负主要责任，吕蒙负次要责任。关羽斩首，吕蒙打五十大板！^_^

4、想要做好测试工作，学历和技术并不是最重要的，重要的是要有责任心和细心，中专毕业证书是中专学校发的，大学毕业证书是大学发的，而有了责任心和细心的测试员，就不再是普通的测试工程师，而是优秀的测试工程师了。

在开发一个软件产品的过程中，每一个程序员都需要跟成千上万行的代码打交道，他们自己写的代码就像大宝 SOD 蜜一样的天天见，看多了难免叫人头昏眼花胸闷心烦……再加上每天都看见的东西，很容易就会产生思维定式。一个人偶尔犯错并不可怕，可怕的是他对错误熟视无睹，错啊错啊的就习惯了，于是很自然的把错的当成对的。俗话说“老虎也有打盹的时候”，水平再高的程序员也会有犯错误的时候，因此必然会有一些错误和缺陷是很难被自身发现的，其原因可能是他在阅读需求规格的时候惦记着那天和女朋友吵架的事情开了小差，从而没有好好地理解需求；也可能是他那天正好失恋心情不好而犯迷糊，更可气的是他居然说女朋友都跟别人跑了我的程序写得那么好有什么用，我偏要这么写……所以说没有 BUG 的软件是不存在的（否则广大和我一样的软件测试工作者靠什么混饭吃？^_^）。

“阴在阳之内，不在阳之对”，我们那些无比智慧而英明的祖先在这里清楚的告诉我们，一个软件产品中最大、最致命的 BUG，往往不是如你想象的那么隐蔽、那么难找，而经常

是潜伏在你最没有防备、以为最不可能出错的那些地方。这便再一次的证明了这样一个道理：在软件的世界里，不是缺少 BUG，而是缺少发现 BUG 的眼睛！

初学者福音 C 语言的编程风格

缩进格式

Tab 是 8 个字符,于是缩进也是 8 个字符.有很多怪异的风格,他们将缩进格式定义为 4 个字符(设置为 2 个字符!)的深度,这就象试图将 PI 定义为 3 一样让人难以接受.

理由是:缩进的大小是为了清楚的定义一个块的开始和结束.特别是当你已经在计算机前面呆了 20 多个小时了以后,你会发现一个大的缩进格式使得你对程序的理解更容易.

现在,有一些人说,使用 8 个字符的缩进使得代码离右边很近,在 80 个字符宽度的终端屏幕上看程序很难受.回答是,但你的程序有 3 个以上的缩进的时候,你就应该修改你的程序.

总之,8 个字符的缩进使得程序易读,还有一个附加的好处,就是它能在你将程序变得嵌套层数太多的时候给你警告.这个时候,你应该修改你的程序.

大符号的位置

另外一个 C 程序编程风格的问题是对大括号的处理.同缩进大小不同,几乎没有什么理由去选择一种而不选择另外一种风格,但有一种推荐的风格,它是 Kernighan 和 Ritchie 的经典的那本书带来的,它将开始的大括号放在一行的最后,而将结束大括号放在一行的第一位,如下所示:

```
if (x is true) { we do y }
```

然而,还有一种特殊的情况:命名函数:开始的括号是放在下一行的第一位,如下:

```
int function(int x) { body of function }
```

所有非正统的人会非难这种不一致性,但是,所有思维正常的人明白: (第一) K&R 是___对___的,(第二)如果 K&R 不对,请参见第一条. (:-))……另外,函数也是特殊的,不一定非得一致.

需要注意的是结束的括号在它所占的那一行是空的,__除了__它跟随着同一条语句的继续符号.如"while"在 do-while 循环中,或者"else"在 if 语句中.如下:

```
do { body of do-loop } while (condition);
```

以及

```
if (x == y) { .. } else if (x > y) { ... } else { .... }
```

理由: K&R.

另外,注意到这种大括号的放置方法减小了空行的数量,但却没有减少可读性.于是,在屏幕大小受到限制的时候,你就可以有更多的空行来写些注释了.

命名系统

C 是一种简洁的语言,那么,命名也应该是简洁的.同 MODULE-2 以及 ASCAL 语言不同的是,C 程序员不使用诸如 ThisVariableIsATemporaryCounter 之类的命名方式.一个 C 语言的程序员会将之命名为"tmp",这很容易书写,且并不是那么难以去理解.

然而,当混合类型的名字不得不出现的时候,描述性名字对全局变量来说是必要的了.调用一个名为"foo"全局的函数是很让人恼火的.全局变量(只有你必须使用的时候才使用它),就象全局函数一样,需要描述性的命名方式.假如你有一个函数用来计算活动用户的数量,你应该这样命名--"count_active_users()"--或另外的相近的形式,你不应命名为"cntusr()".

有一种称为 Hungarian 命名方式,它将函数的类型编码写入变量名中,这种方式是脑子有毛病的一种表现---编译器知道这个类型而且会去检查它,而这样只会迷惑程序员. --知道为什么 Micro$oft 为什么会生产这么多"臭虫"程序了把!!.

局部变量的命名应该短小精悍.假如你有一个随机的整数循环计数器,它有可能有"i",如果没有任何可能使得它能被误解的话,将其写作"loop_counter"是效率低下的.同样的,""tmp"可以是任何临时数值的函数变量.

如果你害怕混淆你的局部变量的名字,还有另外一个问题,就是称function-growth-hormone-imbalancesyndrome.

函数

函数应该短小而迷人,而且它只作一件事情.它应只覆盖一到两个屏幕(80*24 一屏),并且只作一件事情,而且将它做好.(这不就是 UNIX 的风格吗,译者注).

一个函数的最大长度和函数的复杂程度以及缩进大小成反比.于是,如果你已经写了简单但长度较长的的函数,而且你已经对不同的情况做了很多很小的事情,写一个更长一点的函数也是无所谓的.

然而,假如你要写一个很复杂的函数,而且你已经估计到假如一般人读这个函数,他可能都不知道这个函数在说些什么,这个时候,使用具有描述性名字的有帮助的函数.

另外一个需要考虑的是局部变量的数量.他们不应该超过 5-10 个,否则你有可能会出错.重新考虑这个函数,将他们分割成更小的函数.人的大脑通常可以很容易的记住 7 件不同的事情,超过这个数量会引起混乱.你知道你很聪明,但是你可能仍想去明白 2 周以前的做的事情.

注释

注释是一件很好的事情,但是过多的注释也是危险的,不要试图区解释你的代码是注释如何如何的好:你应该将代码写得更好,而不是花费大量的时间去解释那些糟糕的代码.

通常情况下,你的注释是说明你的代码做些什么,而不是怎么做的.而且,要试图避免将注释插在一个函数体里:假如这个函数确实很复杂,你需要在其中有部分的注释,你应该回到第四章看看.你可以写些简短的注释来注明或警告那些你认为特别聪明(或极其丑陋)的部分,但是你必须要避免过多.取而代之的是,将注释写在函数前,告诉别人它做些什么事情,和可能为什么要这样做.

你已经深陷其中了.

不要着急.你有可能已经被告之"GUN emacs"会自动的帮你处理 C 的源代码格式,而且你已经看到它确实如此,但是,缺省的情况下,它的作用还是不尽如人意(实际上,他们比随便敲出来的东西还要难看- ainfinite number of monkeys typing into GNU emacs would never make a good program)

于是,你可以要么不要使用 GUN emacs,要么让它使用 sanervalules.使用后者,你需要将如下的语句输入到你的.emacs 文件中.(defun linux-c-mode() "C mode with adjusted defaults for use with the Linux kernel."(interactive) (c-mode) (c-set-style"K&R") (setq c-basic-offset8))

这会定义一个 M-x Linux-c-mode 的命令.当你 hacking 一个模块的时候,如何你将-*- linux-c -*-输入在最开始的两行,这个模式会自动起作用.而且,你也许想加入如下

(setq auto-mode-alist (cons '("/usr/src/linux.*/.*\\.〖ch〗$" . linux-c-mode) auto-mode-alist))

到你的.emacs 文件中,这样的话,当你在/usr/src/linux 下编辑文件的时候,它会自动切换到 linux-c-mode .

但是,假如你还不能让 emaces 去自动处理文件的格式,不要紧张,你还有一样东西: "缩进" .

GNU 的缩进格式也很死板,这就是你为什么需要加上几行命令选项.然而,这还不算太坏,因为 GNU 缩进格式的创造者也记得 K&R 的权威, (GNU 没有罪,他们仅仅是在这件事情上错误的引导了人们) ,你要做的就只有输入选项"-kr -i8"(表示"K&R,缩进 8 个字符).

"缩进"有很多功能,特别是当它建议你重新格式你的代码的时候,你应该看看帮助.但要记住:"缩进"不是风格很差的程序的万灵丹.

成长的烦恼：初涉设计模式

相信很多人都喜欢看这部喜剧，我是很喜欢，里面包括了成长中的悲欢离合，你在其中可以寻找你成长的足迹。

编程成长之路何尝不是这样的呢？

故事就是从这里开始的。

小王是刚毕业的学生，进入一家软件公司，薪水不错。年轻人充满干劲，有着远大的目标。前三天参加了公司的培训，三天没写代码了，手痒。第四天，项目经理走过来说：“小王，写一个整型链表的排序算法吧，我们在项目中要用。”

冒泡是小王在脑海中第一个浮现出来的。翻开某某圣经，摘了段冒泡算法，修改了一些代码的书写风格（有些圣经代码风格不咱的），代码大致如此：

```
BOOL    Sort(ListInt)
{
    冒泡排序算法
    {
        比较语句
    }
    return TRUE；
}
```

小王检查了一下，还用测试用例测试了一把，确保万无一失，交给了经理。经理说了句不错，乐坏了小王。

第二天，经理跑过来说：“把你昨天的代码改一下，现在要比较浮点型了，还有能否速度上提高一点？”

小王上网查了一下，选择了快速排序算法，不忘把昨天写的备份了一把，然后在昨天函数的基础上改。代码大致如此：

```
BOOL    Sort(ListInt)
{
    快速排序算法
    {
        比较语句
    }
    return TRUE；
}
```

Easy 吗？测试交差。

一年后……

镜头切换……

小王坐在计算机前熟练的编写着程序，而且旁边还放着本《设计模式》的书。知道了面向对象编程，知道了设计模式，但理解还不够深刻。排序算法也演变成比较文件名了。

一日经理过来说：“小王，现在我们的排序算法要用在嵌入式平台中，你做一些算法的研究工作，给出一份报告。”

这不是策略模式的典型应用吗？定义一系列的算法，把它们一个个封装起来，并且使他们可以相互转换。

这样，小王把一些流行的排序算法都试了一遍，总共有七八种，换一种算法速度也很快，新的算法插入到系统中，老算法从系统中"退休"，实现插件式替换。

```
CSort    *pSort  =  new CBubbleSort;
CClient.ListSort(pSort);
```

如果要改成快速排序，只要如此：

```
CSort    *pSort  =  new CQuickSort;
CClient.ListSort(pSort);
```

测试交差，当然经理自己也有想法，又让小王试了另外的几个算法，小王都能轻松的完成。策略模式的作用在这里淋漓尽致的发挥了，小王心里特别有成就感。

过了些日子，客户提出需要按文件名、日期进行排序，小王觉得这还是比较简单的，改代码的主要工作是 copy-paste，就四个函数，也就很快完成了。

客户的需求是不会停止的，为了加强功能，提出需要按文件大小、文件的类型排序，天知道客户还会提出什么要求。

“再也不能这样活”，小王听着歌，陷入了沉思。

“排序的算法和比较算法分开来会如何呢？把它们脱耦，使得二者可以独立地变化。这句话怎么这么熟悉，我肯定在哪里看到过。”小王忙翻开《设计模式》，开始查阅。

“Got it，这不就是桥梁模式（Bridge）。”一阵欣喜，马上就干。

客户端代码如下：

```
CSort *pSort   = new CQuickSort;
CCompareType    *pType = new CNameCompare;
pSort->SetType(pType);
pSort->Sort(pList);
```

哈哈，客户们，你们尽管提要求吧。

测试之颠，必先利其器

孔子曰：“工欲善其事,必先利其器”，其大体意思是：孔子告诉子贡，一个做手工或工艺的人，要想把工作完成，做得完善，应该先把工具准备好。时至今日想起此话很有道理，在我们的测试工作中又何尝不是呢！只是对其“器”即所谓的工具的范围更广了而也。

在纷繁复杂和反复无常的测试工作中其所用的“器”那是至关重要的，其器可以从两方面来讲：一方面是测试时所具备的工具；另一方面则是测试人员本身，这一点是对器的含义的衍生，相对前者就更加重要了。

工具是基础，是开展一切事项的根本前提，在人类起源之初因为原始的高级动物具备了工具从而使之开始劳动，进而成为现在我们人类，可想而知其工具的魅力所在。

在测试工作中使用一个好的测试工具是很有必要的，目前市场上所出现的测试工具不难分出如下几类：

为减少重复测试工作的自动化测试工具

高精度专业化的专用测试工具

软件开发过程中各阶段使用的辅助测试工具

面对如此繁多的测试工具我们应该如何对待呢？其实每一个工具均有其自身的特点，这也是每个工具存在的唯一理由，不可能有一种工具什么都能做，或者什么都不能做，要是真有什么都不能做的工具那就不叫工具了。在这里需要强调的是并不是使用的工具越多测试工作就做得越好，其实即使在经济上允许的情况下使用了一些没有必要使用的工具也许是一种累赘，况且据我所知很多公司都不愿意把大把的钱花在买测试工具上，更何况该测试工具是一种累赘。

简单来讲就是在当前环境下我们要使用适合当前项目和符合公司及团队运作的测试工具。所谓的当前环境是指目前在项目进度和项目预算这种前提下是否允许我们使用某个测试工具；所谓适合当前项目是指是否在这个具体的项目中使用这个测试工具能够提高测试效果和效率；所谓的符合公司及团队运作是指公司及团队的发展战略及相关规程是否能够使这个测试工具能够更好的运作起来以此来发挥其最佳效果。

分析好以上所说的“三个所谓”的问题，对于决定是否需要这个测试工具那是一件非常容易的事了，如果没有解决好以上“三个所谓”的问题那么就最好不要选用这个测试工具。关于如何选择一个测试工具请参考 WAYNE 先生的一文《如何选择嵌入式白盒测试工具》，在此文中对于测试工具的选择有非常精辟的阐述。那么拥有一个非常适合的在业界也是比较优秀的

测试工具就够了吗？当然不是了，要不测试人员就要下岗了。

测试人员是灵魂，工具毕竟只是个听从指挥和执行命令的一个实体，不能像人一样发挥主观能动性，不具思维，更不用说是发散思维了，不能执行一些创造性的工作。所以说除了有一个非常适合的在业界比较优秀的测试工具之外，更重要的还需要一个非常优秀的测试人员，需要这个灵魂来*纵测试工作的一切。

一个优秀的测试人员具备如下素质是很有必要的：一、软件开发和设计功底，这个是基础，如果一个不懂得开发的人员来做测试工作其实很容易想象测试工作是多么的糟糕，其道理也很简单，在此也不再提及，但很遗憾的是在这个观点上往往会有人知错犯错，导致总是会有一些测试人员不是很懂得开发的一些东西，这种现状希望在不久后会彻底消失；二、测试理论和测试思想，这点就很容易理解了，这也是作为一个测试人员的基础，要做到这点相对比较容一些；三、不仅要学会使用测试工具，更要在平时的测试工作中加以提炼创造测试工具，以至更好的为测试工作服务，提高测试效率和测试工作的共用性；四、测试人员个人的素养，这里主要是指个人的沟通、交流等方面的能力，还有就是测试人员所具备的比较特殊的发散思维和逆向思维。

在测试工作中除拥有一个好的测试人员外，更加拥有一个适合的比较优秀的测试工具，这两者相结合，那么做好一项测试工作就很容易了，只要拥有了这两项利器相信测试工作会获得更大的成功，需要说明的是并不是每一项测试工作一定需要测试工具来辅助完成，而是需要应该需要的工具。当然还需要组织及团队有效的推动和支撑，这样其测试工作将会发挥到极致，以至登上华山之颠！时至那时，测试行业的发展将会拔开云雾阳光普照。

测试工程师的十二最

测试工程师最开心的事:发现了一个很严重的 bug,特别是那种隐藏很深,逻辑性的错误.偶第一次发现这种问题的时候,听到上司和开发人员的表扬时,高兴的就想扭 pp.不过现在慢慢矜持些了,呵呵.

测试工程师最提心吊胆的事:版本 release 出去后,客户发现了很多或很严重的 bug.经过紧张的系统测试之后,好不容易可以轻松一下了,却又陷入了每天担心正在做验收或使用的客户一封邮件或一个电话说产品有问题.碰到好些的老板还会比较乐观的看这样的问题,最惨的就是有些人一顿臭骂,之前的辛苦,加班全部都给抹杀了.

测试工程师最憎恨听到的话:"为什么这个 bug 没有在测试的发现呢?"这句话经常是客户发现 bug 后,老板对测试人员的质问.当然这里排除那种很明显的错误.其实谁都知道 bug 是不可能全部发现的,这句话其实也是客户对大头,大头对小兵一级一级问下来的.除了希望测试人员警惕之外,还有更多的是一种"踢猫"的行为.对于这句话,偶第一次听到这句话的反映是"我们怎么可能发现所有的 bug 呢",后来变成"制造 bug 的人不是我们,是开发",到现在的"让我查查我的日志,问问开发这个 bug 的原因,为什么我们会没有找到,下次我们会怎样"的回复.

测试工程师最郁闷的事:"刚才那个版本打包打错了,你们要重测".新版本来了,马上投入紧张的测试,希望能够多找些 bug.没想到辛苦了可能大半天,开发人员说打包打错了,你重测吧.这种情况虽然可以通过规范流程之类的办法控制发生的机率,但人总会犯错,多少而已.碰到这样,你除了提醒开发部门下次注意,你除了重测没有太多办法.

测试工程师最不想面对的事:在测试晚期或最新的版本里发现了以前一直存在的问题,特别是当问题很严重时决定到底报不报 bug.报吗,开发人员肯定会问以前有没有这个问题,不报吗,客户发现更惨.毕竟客户或老板的责备比开发部门或主管的责备轻许多,最后还是会报到 bug 库里的.

测试工程师最不想做的事:申请版本推迟发布.由于在版本发现了太多的问题,觉得产品不能达到发布的标准,建议公司推迟发布产品.这时虽然大家都知道产品有问题,尽管你自己也不希望这样,但谁都觉得你是一个制造麻烦的人,毕竟市场的压力很大呀.

测试工程师最丢人的事:辛苦的发现了一个 bug,居然是该配的参数没有配等一些自己的失误造成的.有些该注意的地方居然测试时忘了,找出的问题给开发人员一顿臭扁,无比丢人啦.

测试工程师最怕的事:一天,甚至几天都没有发现一个 bug!经过一段时间的 bug 高峰期后,

有段时间会发现 bug 数量的减少,最可怕的就是一天都没有发现一个 bug.偶有时会难过的吃饭都没心情.搞得偶的开发朋友说了一句最让人吐血的话:"要不要我在代码里放几个 bug 给你呀,hoho"

测试工程师最伤心的事:每年的调薪,发 bonus 或发股票时,测试工程师总比开发工程师少.偶有一同事在调薪的第二天就申请转开发,说测试太没前途了.

测试工程师最有力的保护方法:把你认为是 bug 的问题都提交到一个正式的,可以追踪的地方(一般来说是 bug 库).有时总会碰到一些很小的或是很难判断的问题,犹豫一定是否要报,特别是一些 UI 的问题.有时问开发人员,他们可能会轻描淡写的回复你导致你没有 report 它.但多年的经验一定要报,了解 bug 流程走向的人都知道,后面还有人 verify,还有开发经理判断,如果不是 bug,自然他们会回复,会写明原因.说白了,出了问题也不是你的事情.当然一开始经验不足时会收到一些白眼球,但慢慢经验多了,对系统熟悉了,自然这种情况会少些.人也可以从一些问题中发现自己的弱点.但如果不报,那天客户提出来,你除了懊悔还要面对指责,严重的炒鱿鱼.

测试工程师最任重道远的事:测试驱动开发.碰到这种开发模式的项目,既是测试扬眉吐气的机会,也是可能会陷你于深渊的恶潭.你就必须打起十二分的精神.等于你在引导开发,有什么问题一定要提出来,否则你就等着被盲目的牵着鼻子走了.

测试工程师最期待的事:测试能够越来越受重视,测试工程师的考核越来越合理.

操作系统\UML 与面向对象\数据库考试模块指导

操作系统和数据库是程序员和软件设计师每年的必考内容，从 1987 年到 2005 年春季软考都少不了它们的身影。

近年来，程序员和软件设计师大纲虽做了一些大的改动，但操作系统部分变动并不是很大，上午分值多是 1 到 5 分之间，下午是不确定出题，也就是可能会出到，也可能没有。但不可大意，如 2004 年秋季下午题的第四题就是道操作系统题。

另外，在出题形式上更趋于具体的分析，而不再是纯粹的概念题。如 PV 原语操作就比较多偏向于对生产者/消费者问题的解答。大纲所列知识点虽不能全部都涉及到。不过再通过我们对历年题型的综合分析后(特别是 1995 到 2005 春季)，可以明确的是操作系统方面的题目，一般集中在进程，存储管理和作业管理这几个方面。1998 年到 2000 年这几年的操作系统，有很多是重复出题，而且都集中在上面说的几个方面。希望各位考生在复习时把主要精力放在主要知识点上。

数据库在程序员和软件设计师的出题中比重不小。分值上午一般会有 5 分左右，下午有和软件工程结合出题，或者与 UML 联合出题的情况。这种结合多是考查 ER 模型到关系模式的转换，以及用 SQL 来建立关系模式，2005 年春季考试上下午都有数据库的题，且下午是独立题目。而且我们思达网校的老师一致认为这是考生朋友们应该牢牢抓住分数的部分。具体的重点是很清晰的，ER 模型和关系模式之间的转换，关系代数，关系演算，范式，SQL 语言(查询的比重较大)。复习时应注意掌握以上这些知识点。

面对对象和 UML 是新大纲的新要求，可以参考的并不多。不过对概念的考查火力比较强，考生很不容易在面对对象方面的众多概念中拿到分，这就要求考生朋友们一定要注意平时在复习时就把这些内容有意加强记忆。UML 是在下午题中出现，从 2004 年春季考到 2005 年春季考的下午试题中发现出题 UML 的火力点多在对各种静态图和动态图。

采用简化原型法进行需求分析

顺序阶段的观点[2]，改变了传统的自顶向下的开发模式，降低了软件需求的风险，因此得到了广泛的应用，特别是在致力于某一领域 MIS 开发的软件公司，如致力于电力 MIS 开发的公司。但作者在长期的 MIS 需求分析过程中，发现原型法有以下缺陷：

1）原型的设计和修改工作量大，增加了系统的开发成本；

2）由于用户不关心或不理解原型的概念和实现，而且存在较大期望，使得与实际系统差别较大的原型增加了需求分析人员与用户的交流难度；无论是水平原型，还是垂直原型都不能反映实际系统的全貌；

3）软件需求主要包括：功能需求、界面需求、性能需求、环境需求、可靠性需求、安全保密需求、资源使用需求、软件成本消耗与开发进度需求和目标需求[3]。原型法中的原型难以表达软件的后七项需求；

4）原型法强调用户和开发人员不断对原型进行不断修改和补充，直到用户感到满意为止。在时间紧和任务重的大型 MIS 项目中，这种情况实际难以保证，特别是在用户单位和开发单位距离较远时。

本文结合管理信息系统项目实施的实践，提出一种新的需求分析方法-简化原型法。这种方法根据数据库应用的特点，将需求分析分为两个阶段，并简化了作为需求分析工具的系统原型。

2 简化原型法需求分析的第一个阶段

管理信息系统属于数据库应用。数据库应用需求分析应该围绕数据，而不是功能展开，因此应该首先解决"有什么"，然后再明确"做什么"[4]。第一个阶段就是要解决"有什么"，即由项目经理与用户进行协商，确定系统的技术协议，因此可以称为技术协议阶段。技术协议需要开发方的项目经理与用户单位的技术主管签字并盖章，并以合同附件的形式存在。技术协议的主要内容有：系统的边界、系统处理的业务、与其它系统的接口、工程的进度控制、培训安排和技术服务承诺。

2.1 系统的边界

系统的边界规定系统覆盖的作业范围，主要有地理边界（规定系统运行的部门、分支单位等）、操作员范围（规定操作系统的所有操作员身份、分布和大致权限）和业务范围（规定系统处理的业务，对于不处理的边沿业务特别明确指出）。

2.2 系统处理的业务

系统处理的业务涵盖系统处理的所有业务，包括各种业务的描述、数据来源、实现要求。但是业务规定不要求过细，可以对应实际系统中的一个模块。如：电力 MIS 中输电设施管理子系统中的线路设备管理，不详细描述线路设备管理中的所有功能。

2.3 与其它系统的接口

与其它系统的接口明确规定接口的系统、功能和实施单位。在接口的实施单位中明确是由开发方完成，还是由开发方协助第三方完成。

2.4 工程的进度控制

工程的进度控制规定工程的开始、结束日期和具体工程项目的名称、完成时间、地点、完成标志及责任分工。具体项目一般包括：采购设备到达现场、采购设备安装调试、完成网络布线、开发准备阶段、业务需求调查、系统分析和设计、软件编制、现场调试、数据准备及录入、功能确认、试运行和系统验收。责任分工规定双方对于具体项目的工作内容和配合方式。在配合方式中规定人员组织方式、人员素质要求、提供的设备和场所。完成标志规定具体项目完成提供的文件名称和要求，如：网络布线验收报告和硬件设备验收报告等。

2.5 培训安排

训包括操作员和系统维护人员的培训。培训安排包括每种培训的人员数量、培训内容、培训时间、地点、组织方式和教材，并规定教员和学员的素质要求，及培训后学员达到的水平。

3 简化原型法需求分析的第二个阶段

如果说第一个阶段解决"有什么"的问题，那么第二个阶段解决"做什么"的问题。主要工作有需求调查准备、到用户单位进行需求调查分析和进行需求评审。

3.1 需求调查准备

需求调查准备工作，在系统的技术协议签订后，严格依照技术协议进行，主要有向用户单位发放业务调查表、建立需求分析文档原型和建立系统简化原型。业务调查表在系统的技术协议签订后，立即通过传真发送到用户单位，要求用户单位在需求调查人员到达现场之前完成。业务调查表内容包括：具体业务的名称、上级业务、下级业务、发生条件、处理的数据和详细流程（处理岗位、处理方式和审核细节等）。需求分析文档原型是根据技术协议编写的需求分析说明书原型，它的格式与标准的需求分析说明书相同。其中的状态迁移图和各种表证单书等不明确的内容，采用相似系统的或由系统分析人员根据技术协议和以往经验设计。

系统的简化模型根据技术协议的要求，仿照相似系统设计。简化模型采用可视化的数据库编程语言设计，一般采用数据库应用开发人员熟悉的 PowerBuilder(PB)或 Delphi。简化模型的主要设计要求有：1）充分考虑系统的设计与实现，不得与实际系统脱节；2）尽量仿真实际系统的操作界面，与实际系统的操作过程完全相同；3）可以单机安装运行，不与实际数据库连接；4）演示数据的存储可以通过文本文件、单机的数据库或 PB 外部数据源的数据窗口；5）对于界面中容易误解或难以理解的操作，在功能帮助按钮中给出说明；6）界面中难以实现或工作量很大的功能，以标注方式详细说明；7）运行稳定，并比实际系统对硬件要求低。

3.2 需求调查分析

需求调查分析在确认需求调查准备的三项工作完成后，由开发单位的系统分析人员到用户单位进行。系统分析人员与用户单位安排的业务主管共同讨论业务调查表和系统简化原型，并不断修改完善系统简化原型和文档原型，最终形成共识，并要求业务主管在需求分析说明书上签字。最终系统简化原型和源代码留在用户现场，便于系统的操作员进一步理解分析，直到最终掌握；而且有利于提出进一步的改进意见。改进意见可以随时通过邮件或传真直接发到开发单位，或由用户单位的系统维护人员修改简化原型后，随时发到开发单位，从而便于开发人员及时修改系统的设计和编码。

3.3 进行需求评审

需求评审一般由用户单位组织，评审团成员由同行专家、系统分析、设计和测试人员组成。评审的依据不仅有需求分析说明书，还有系统简化原型；同时在评审过程中，系统简化原型不断进行优化。评审的目标是要求需求分析说明书具有正确性、可行性、必要性、具有优先级属性、可验证性和无二义性[5]。需求评审报告作为对需求分析的补充和修正，由双方负责人签字，以需求分析说明书附件的形式存在，同样指导下一步的系统设计工作。

4 几点说明

1、此方法适合各种 MIS 工程的需求分析，特别适合致力于某一领域 MIS 开发的软件公司。采用此方法，开发同类项目越多，需求分析工作的效率越高。

2、在需求分析过程中，由于需要设计系统简化原型和文档原型，并充分考虑到系统的设计与实现，因此与其它需求分析方法向比，提高了对需求分析人员的要求。在实际工作中，一般由资深的软件分析和设计人员进行。

3、此方法不仅适合 MIS 软件工程，同样适合其它大型软件工程。

4、由于需求分析工作本身的难度和重要性，此方法同样要求用户单位和需求分析人员对需求分析所有工作内容，引起足够重视；科学安排需求分析工作步骤，某些步骤可以同时进行；所有工作步骤不得应负或疏忽。

5 结束语

目前简化原型法已经在多个电力 MIS 工程中应用，大大提高了需求分析的工作效率。实践证明，简化原型法具有以下特点：1）简化的系统原型开发工作量大大降低，修改和补充方便；2）简化原型大大缩短了需求分析人员与业务主管之间的距离，便于交流；并大大加强了需求分析人员与业务主管对系统的认识，有利于发现和解决问题；3）简化原型的设计提前考虑了系统的设计与实现，大大降低了软件工程的风险；4）简化原型增加了系统操作员对实际系统的认识，大大简化了工程实施后系统的操作培训；5）简化原型可以直接指导工程的设计和编码，便于系统开发的组织。这种方法也可以用于其它软件工程，对于其它需求分析方法的改革也具有指导意义。

不要将默契式管理和人性化管理划等号

学而优则仕在中国还是一个挺普遍的现象，给批判得也很多！但是原话是“仕而优则学，学而优则仕”，见于《论语·子张》，其真正含义并不是我们片面理解的“学习优秀的就去做官”。同样，“仕而优则学”难道是告诉管理者：“做好了官，有余力则学习！”如果你认定等我有余力才学习吗？呵呵，那你永远也没有有余力的一天。

在 IT 行业，不少管理者都是技术出身，从论语这句话，我们应该学习到：当走上管理岗位，更加要“边干边学”，从工作中学习提高业务知识和管理体系，持续改进，才能进入“仕而优则学，学而优则仕”的良性循环。（看来和 CMMi 的本质一致的！）

案例一：

某日抽检项目组 O 工作，检查项目经理 L 对任务的执行情况了解和把控，抽查本周内项目组成员 C 的工作任务时，发现项目经理 L 对 C 昨天的工作任务 T1 的工作完成情况并不了解。

为了进一步了解 C 的工作完成情况，和项目经理 L、同事 C 一起检查昨天 C 的全日任务 T1，该任务是测试业主方的 FTP 上传、下载接口，而本日的任务 T2 是开发 FTP 上传模块。先是询问了同事 C 在昨天是否还开展了其他任务，答案是没有；接着询问任务 T1 除了使用 java 的 sun.net.ftp 程序包外，是不是还有特别的业务逻辑以及异常处理等处理，答案是异常需要写日志记录；接着还告诉同事 C，程序为了稳健性，处理正常的业务逻辑外，还有很多是针对异常的控制逻辑，除了使用 log4j 写入程序日志，还有什么特别的地方，C 的回答是没有。

事后和项目经理 L 分析，项目组 O 的工作方式，停留在按模块给成员派任务，成员将任务计划上报，项目经理 L 和技术经理 LL 则合并计划，解决人员间的任务冲突！但是任务的时间是否合理，没有进行评估和监控，小组评估法也好，专家评估法也好，总之没有评估监控环节，项目经理 L 和技术经理 LL 都缺位了。

另外，工作的执行没有进行督促，没有每日的检查或者抽查，那么小组 O 里面，谁做到好，谁做得不好，只是拍脑袋；还有，项目组里面到底应该鼓励谁呢，谁是项目组成员的榜样呢？

案例二：

某日抽查某项目新进入成员 G 的工作，通过他的工作可以看到他融入项目组工作的成效。我询问 G 的工作汇报给谁，G 告诉我汇报给技术经理 L。G 接着谈到他上周的工作情况，提到了他上周开发过程中存在需求变更。我告诉他这个事情有以下几种情况：

在 G 开发的过程中，业主告诉“我们理解的需求”需要变动；

上级同事 L 没有及时将“我们理解的需求”变动告知 G；

上级同事 L 没有将任务的内容清楚告知 G；

同事 G 没有真正清晰地理解同事 L 的要求；

我询问 G，现在回头看上周这个问题，到底是上面那种情况，G 的回答是 1；我接着询问同事 G，他的上级 L 怎么验收工作成果，G 告诉我 L 将内容交给了项目组中的测试人员。

事后和技术经理 L 分析，拿到这个事情的判断，L 的回答是 4，L 告诉我是在任务中检查 G 发现他对业务理解有错误。具体的事实是怎样并不重要，重要的是同事 G 和同事 L 对工作的理解、判断的标准是不同的，而且截至到目前为止一周，他们还是没有达成共识的。为什么会造成这样结局？是因为他们两都没有将自己对待事物的分析见解摆出来。如果 G 有错误理解的地方，L 并没有在 G 刚进入项目组就告诉他这样做是不对的，那么以后情况会怎么样，可想而出；如果 G 这样的思考没有问题，那么 L 就得考虑为什么会这样想 G？是不是得加强日常的工作沟通，将目标和要求明确告知。

案例三：

同事 C 完成项目 E 二期后，进入另一个项目 S 二期，目前负责带领 4 个同事开展开发工作，我刚好有天早上经过他的办公地点，发现 C 正在给同事 T 评点 T 昨天开发内容，告诉 T 那些有问题需要改进，我会觉得 C 的同事每天的工作成果都是在他的控制下的。

事后我了解到同事 C 在工作的每一天，都能够将团队成员的工作进行 Review，并且以邮件 PPT 的方式发送给小组，抄送给领导 Z，并且在 PPT 中，讲述了每个人的工作进展，与计划的偏差以及本日的工作目标。当然有的读者可能觉得没有必要吧，每天都发出 PPT，呵呵，其实是否 PPT 形式并不是重要，重要的是内容的质量，从我阅读他的 PPT 内容后，我觉得他的做法有以下的优点：

C 让小组同事、领导、监督管理者了解到小组整体的工作情况；让管理者的眼光更多关注到他的项目；

C 让小组的成员都知道，我们的工作成果，每个人的工作好坏，上层的管理者都是知道的，每个人也是可以相互评价的；

C 让小组的成员都知道，C 的要求是什么，我今天要检查的是什么？写出来的总要比只是说说要有契约精神。培养团队的契约精神，是团队战斗力的一个基石。

（注：后续我还需要去了解 C 对整个项目计划 Plan 的更新和跟踪习惯。）

案例总结：很多从技术转型到管理的读者，都是从人性化的小公司开始做上来的，公司小的时候因为人员少，边界简单，工作大家都看得见，很容易检查到并且调整。当转型为管理者的时候，默契式管理就会成为一个“过去式、怀念式”的理想，当你怀念为什么团队以前小的时候没有这样的理解、处理难题，那就很有可能已经患病不轻。请不要将默契式管理和人性化管理划等号。

我给初入管理工作的 IT 技术员工的建议：

做好 TO DO LIST 的记录和跟踪，进行 Daily Check；

做好管理者的心态切换，做一个管理者而不是一个处理者；人性化管理更加不是不要求、不监管；

团队建设不是请客吃饭，对做好的同事进行鼓励和宣传，对不对或者没有做好的同事要及时教育；

要特别注意简单分守山头的工作任务分配方式，记住这种简单的分配方式，一个刚毕业的学生就会的，所以可能是错误的；

管理工作涉及的多个方面中，减少库存量可能是最好的工作方式。

标志设计中应该遵守的艺术规律

标志艺术除具有一般设计艺术规律(如装饰美、秩序美等)之外，还有它本身独特的艺术规律。

1.符号美

标志艺术是一种独具符号艺术特征的图形设计艺术。它把来源于自然、社会以及人们观念中认同的事物形态、符号(包括文字)、色彩等，经过艺术提炼和加工，使之结构成具有完整艺术性的图形符号，从而区别于装饰图和其它艺术设计。

标志图形符号在某种程度上带有文字符号式的简约性、聚集性和抽象性，甚至有时直接利用现成的文字符号，但却完全不同于文字符号。它是以图形形式体现的(现成的文字符号

须经图形化改造)，更具鲜明形象性、艺术性和共识性的符号。

符号美是标志设计中最重要的艺术规律。标志艺术就是图形符号艺术。

2.特征美

特征美也是标志艺术独特的艺术特征。

标志图形所体现的不是个别事物的个别特征(个性)，而是同类事物整体的本质特征(共性)，或说是类别特征。通过对这些特征的艺术强化与夸张，获得共识的艺术效果。这与其它造型艺术通过有血有肉的个性刻划获得感人艺术效果是迥然不同的。

但它对事物共性特征的表现又不是千篇一律和概念化的，同一共性特征在不同设计中可以而且必须各具不同的个性形态美，从而各具独特艺术魅力。

3.凝练美

构图紧凑、图形简练，是标志艺术必须遵循的结构美原则。标志不仅单独使用，而且经常用于各种文件、宣传品、广告、映像等视觉传播物之中。具有凝练美的标志，在任何视觉传播物中(不论放得多大或缩得多小)都能显现出自身独立的完整的符号美。

本地化测试错误分析与测试方法[1]

软件本地化测试是项系统性任务，讲究团队协作精神。软件测试工程师负责测试、发现、报告软件错误。软件错误修复工程师的工作是及时正确修复处理这些软件错误。为了便于软件错误修复工程师能够迅速重现报告的错误，寻找错误产生的原因，然后及时修复错误，需要测试工程师正确完整地报告发现的错误。另外，不同类型的软件错误，又分别由来自不同公司的软件错误修复工程师进行修复。例如，软件功能错误和软件国际化错误只能由源语言软件供应商的软件错误修复工程师进行修复，而本地化错误由本地化服务商的软件错误修复工程师进行修复。

下面对本地化软件的错误的三种典型类型进行分类讨论，探讨错误的表现特征，产生的原因，测试要求，发现错误的方法。

1、功能错误

表现特征

不能实现设计要求的功能。

产生与设计要求不符合的结果。

绝大多数存在于源语言软件和本地化软件，也有仅出现在本地化软件中。

经常出现在软件的菜单项、工具栏按钮和对话框的功能按钮中。

产生原因

源语言软件编码错误。

错误本地化，如将程序中的变量字串名进行了翻译等。

测试要求

保证输入数据正确，或者打开了正确的测试用例。

明确正确的输出结果和中间数据数值及格式。

测试方法

对于菜单项或工具栏按钮，通过全面测试各个选项，认真观察每一步是否正确执行，输出结果（包括格式和数值）是否正确。

对于对话框，可以逐个执行各按钮，各个列表选项等观察执行结果。

对于命令行形式的多个并列选项，采用路径跟踪法，按分支顺序测试嵌套的全部子项。

说明

特别注意不同选项、不同按钮相互操作的影响。

注意检查快捷键是否遗漏，是否多余，是否不同，是否起作用。

本地化测试错误分析与测试方法[2]

2、国际化错误

表现特征

控件或对话框中显示不可辩识或无意义的明显错误的字符。

不支持双字节字符的输入和输出，包括双字节的文件名和路径名。

与当地不符合的默认打印纸张大小。

与当地不符合的日期和时间格式。

本地化软件的列表项排序错误。

某些没有本地化的字符串。

仅出现在本地化后的版本中。

产生原因

源程序在设计时没有正确进行国际化设计，例如，没有提供双字节字符集的支持。

源程序在设计时没有将可以本地化的字符串与软件代码进行彻底分离。

软件本地化后，单字节字符向双字节字符转化过程中，由于单字节和双字节之间的差别，可能使得某些本地化后的双字节字符的显示乱码。

测试要求

本地化后的软件字符显示正确完整，无乱码或明显错别字。

与本地区域有关的数据类型（日期/时间，货币符号，纸张大小，字体，度量单位等）的显示符合本地的格式要求。

测试方法

执行菜单或按钮，检查对话框中的字符。

打开帮助文档，检查所有需要翻译的字符。

说明

注意检查对话框下拉列表中需要拖动滚动条才能显示的内容。

需要确认源语言软件对应项正确，仅本地化软件存在错误。

3、本地化错误

包括翻译错误和控件大小和位置引起的布局错误。

表现特征

应该翻译而没有翻译的英文字符。

不应该翻译而翻译的中文字词。

错误翻译的字词。

较多隐含在对话框各控件以及帮助文档中。

只在本地化版本中存在该类型错误。

控件相互重叠或排列不均匀。

控件中字符显示不完整。

主要出现在本地化版本的对话框中。

白盒测试用例设计问题演示

问题：

对这样一段代码：

```
if (a>2 && b<3 &line;&line; (c>4 && d<5))
statement;
```

请问，按照各种覆盖方法应该怎么考虑它的测试？

我们这里只给出 Condition/Decision Coverage 和 Modified Condition/Decision Coverage 两种覆盖方法的用例设计。

Condition/Decision Coverage:

条件				结果
a<2	b>3	c<4	d>5	(a<2 && b>3 &line;&line; (c<4 && d>5)
T	T	T	T	T
F	F	F	F	F 这个很容易，就不解释了。

odified Condition/Decision Coverage：

基本思路：

表达式可以理解为(a<2 && b>3) &line;&line; (c<4 && d>5)；

将表达式的理解为两个组合条件 A or B 形成的表达式，其中 A 为(a<2 && b>3)，B 为(c<4 && d>5)；

对这个表达式，当 A 为 F 时，B 是独立变量；当 B 为 F 时，A 是独立变量；

则第一步的分析可以围绕 A、B 进行：

条件		结果
(A)	(B)	A or B
F	T	T
T	F	F
T	F	F
F	F	F

其中最后一组取值重复，最终根据这三种取值进一步分析。

5. 第二步的分析，考虑 A 表达式，A 为(a<2 && b>3)，当 a<2 取值为 T 时，b>3 为独立变量；b>3 取值为 T 时，a<2 为独立变量；因此，A 条件取值为 F 的 MC/DC 用例为：

条件		结果
(a<2)	(b>3)	(A)
T	F	F
F	T	F

A 条件取值为 T 的用例为 T，T；

6. 第三步的考虑，分析 B 表达式，B 为(c<4 && d>5)，同对 A 的分析，B 为 T 的用例为 T，T；B 为 F 的用例为 T，F 和 F，T；

7. 综合 4、5、6 的分析，最终得出结果：

条件				结果
a<2	b>3	c<4	d>5	(a<2 && b>3 &line;&line; (c<4 && d>5)
T	F	T	T	T
F	T	T	T	T
T	F	T	F	F
F	T	T	F	F
T	F	T	F	F
F	T	F	T	F
T	T	T	F	T
T	T	F	T	T

WindowsXP 系统下安装 apache+php+mysql

Apache 和 mysql 的安装较简单，主要是安装前请保证 80 端口未被占用 比如 iis 以前安

装过的 apache mysql 先停止运行 phpmyadmin，主要是配置文件的问题，把 phpMyAdmin 安装目录下 Libraries 目录下面的 Config.default.php 复制到 PHPmyAdmin 根目录下，改 名为 Config.inc.php; 用记事本打开 Config.inc.php， 把 “$cfg['blowfish_secret'] = '';” 改为 “$cfg['blowfish_secret'] = '什么都可以';” ，其中“”是我为了避免"加上的，后面“'”中的“什么都可以”你可以改成其它什么都可以，就是不能为“无”，也可是空格;接下来把“$cfg['Servers'][$i]['auth_type'] = 'config';”改为“'$cfg['Servers'][$i]['auth_type'] = 'cookie';”;最后一步就是点下保存。

更详细的设置方法：

搜索$cfg['PmaAbsoluteUri']，设置你的 phpmyadmin 的 URL，如：$cfg ['PmaAbsoluteUri'] = 'http://localhost/phpmyadmin/'; 注意这里假设 phpmyadmin 在默认站点的根目录下

搜索$cfg['blowfish_secret']，设定好 root 密码后这里也要填写比如 ROOT 密码 123456 则设置为$cfg['blowfish_secret'] = '123456';

搜索$cfg['DefaultLang'] ，将其设置为 zh-gb2312 ;

搜索$cfg['DefaultCharset'] ，将其设置为 gb2312 ;

搜索$cfg['Servers'][$i]['auth_type'],默认为 config,是不安全的,不推荐,推荐使用 cookie,将其设置为$cfg['Servers'][$i]['auth_type'] = 'cookie';

注意这里如果设置为 config 请在下面设置用户名和密码!例如：

$cfg['Servers'][$i]['user'] = 'root'; // MySQL user-----MySQL 连接用户

$cfg['Servers'][$i]['password'] = '123456';

打开浏览器，输入：http://localhost/phpMyAdmin/ ，若 IIS 和 MySQL 均已启动，输入用户 ROOT 密码 123456 即可浏览数据库内容。

phpMyAdmin 出现如下错误：配置文件现在需要绝密的短语密码(blowfish_secret)

因为$cfg['Servers'][$i]['auth_type'] = 'cookie';，所以需要密码。

在 phpMyAdmin 安装目录打开 config.inc.php 文件，找到$cfg['blowfish_secret'] = ''，输入密码即可。例：mysql 密码为“666666”，则设置为：$cfg['blowfish_secret'] = '666666'

至此所有安装完毕。

TITLE:xp 系统下安装 apache+php+mysql

WindowsXP 管理搞定 Vista、XP 双系统

本文另辟蹊径，实现 XP、Vista 双系统并存，实际上是非常合理的。Vista 之家建议大家可以借鉴学习一下，下面直接进入本文主题：

一、双系统的实现介绍

1、特性

本双系统 XP，Vista 都是独立的，互不影响，各安装在一个主分区中

启动过程是，bois--mbr--ntldr--在这里选择启动到 XP 还是 Vista，也就是说用 XP 通过 boot.ini 启动 Vista，Vista 启动 XP 实在太麻烦了，对 bootmgr 说再见吧。

2、磁盘工具 1：备份 mbr 的软件

dos 下的 debug，XP、Vista 下的 winhex 13.7 SR7，linux 下的 dd（保存 mbr 有很多工具）

3、磁盘工具 2：分区工具

能更改活动分区的分区软件（同样有很多），其实在 XP 和 Vista 的磁盘管理里可以改变活动分区，在主分区上点右键--将磁盘分区标为活动的。

4、系统安装顺序

任意，先 XP 后 Vista，先 Vista 后 XP 都可以

二、硬盘分区

划分 2 个主分区，一个安装 XP，一个安装 Vista，最好将 XP 分区放在前面，因为 boot.ini 里的 multi(0)disk(0)rdisk(0)partition(1)里面要记录分区顺序.

下面的叙述采取按该方案

三、安装系统

a. 先装 XP 再装 Vista

1、激活第一个分区，安装 XP

2、将 mbr 保存为 mbr.1

3、激活第二个主分区，然后安装 Vista

4、将 mbr 保存为 mbr.2

5、激活第一个分区，启动进入 XP

6、将保存的 mbr2 复制到 XP 的分区的根目录下，在 boot.ini 里面添加一行

C:\MBR.2="Microsoft Windows Vista Ultimate "

b. 先装 Vista 再装 XP

1、激活第二个分区，安装 Vista

2、将 mbr 保存为 mbr.2

3、激活第一个主分区，然后安装 XP

4、将 mbr 保存为 mbr.1

5、启动进入 XP

6、将保存的 mbr2 复制到 XP 的分区的根目录下，在 boot.ini 里面添加一行

C:\MBR.2="Microsoft Windows Vista Ultimate "

有兴趣的朋友可以比较下 mbr.1 和 mbr.2

四、取消双系统

a.要取消 XP 系统，激活第二个主分区，然后处理第一主分区

b.要取消 Vista 系统，删除启动菜单，然后处理第二分区

注：如果采用 pq 分区的，XP 下 boot.ini 引导 Vista 可能会出现问题，请用 XP、Vista 的自带的分区工具重新分区。推荐 spfdisk。

附：保存 mbr 的方法

运行 winhex 13.7 SR-7

按"F9"，选择"物理磁盘"，点到磁盘第一个分区，然后点击菜单：编辑--复制扇区--快捷键是"Ctrl+Shift+N"那一项，输入一个文件名即可。

WindowsXPSP3 预览版新功能深入探秘

一个月前，Windows XP SP3 的一个预览版被泄漏出来，而在近日，首个官方 Beta 测试版送交到了测试人员手中，也披露了这个备受期待的补丁包的更多详情。

相关推荐：有什么变化 深度剖析 Windows XP SP3

泄漏版和官方测试版 XP SP3 的英文版文件名都是

“windowsxp-kb936929-sp3-x86-enu.exe”，Build 版本号 3205(泄漏版为 3180)，Hash 校验码为：

CRC: 56e08837

MD5: c8c24ec004332198c47b9ac2b3d400f7

值得注意的是，XP SP3 的体积并非此前传言的恐怖的 1GB 之多，而是只有 334.2MB，比泄漏版大了仅仅 2MB。除了英文版，微软还提供了日文版和德文版，体积也都差不多。

XP SP3 兼容各种 x86 版本 XP 系统，包括专业版、家庭版、媒体中心版、Starter 版、Embedded 版、Fundamentals 版。

作为一个补丁升级包，XP SP3 的最大任务自然是汇总此前分散发布的各个更新补丁。据悉，XP SP3 里一共有 1073 个新的 Patch、Hotfix 补丁，其中第一个是 2006 年 4 月 7 日的 KB123456，最后一个是 2007 年 9 月 29 日的 KB942367。

在这 1000 多个补丁中，有 114 个修正了安全方面的漏洞，另外 959 个涉及性能和稳定性提升、bug 修复、核心模式驱动模块改进、蓝屏死机(BSOD)问题更正等等。当然，最终正式版的补丁数量可能还会有所变化。

与 XP SP2 一样，XP SP3 里的补丁不但包含了通过各种途径公开发布的补丁，也有针对特殊问题提供给特定客户的补丁。当然，XP SP3 也整合了 XP SP2 里的所有补丁，因此不需要重复安装，这也是微软 SP 的惯例。

除了安全和常规补丁，XP SP3 也提供了不少全新特性，使之不仅仅是一个简单的补丁集合，比如新的 Windows 产品激活(WPA)模型(安装期间就像 Vista 那样可以选择不输入序列号)、网络访问保护(NAP)模块和策略、新的核心模式加密模块、黑洞路由检测功能功能等。

VPN 建设安全环节概要

识别和 IPSec 接入控制

设备验证采用了预共享密钥或数字证书，以提供设备身份，预共享密钥有三种：通配符、分组和独立。独立预共享密钥与某一 IP 地址有关，分组预共享密钥与一组名称有关，仅适用于当今的远程接入。通配符共享密钥不可应用于站点到站点设备验证。数字证书与独立预共享密钥相比，可更理想地扩展，因为它允许一台设备验证其他设备，但不具备通配符密钥的安全特性。数字证书与 IP 地址无关，而是与企业 CA 认证的设备上独特的标志信息有关。

IPSec

IPSec 提供了多种安全特性，对于管理员如何确定其工作方式提供了可配置选择：数据加密、设备验证，以及保密、数据完整性、地址隐藏和安全机构(SA)密钥老化等功能。IPSec 标准要求使用数据完整性或数据加密两种功能之一，具体使用哪个可任选。思科公司强烈建议加密和完整性二者都使用。而改变以上那些数值会提高安全水平，但与此同时， 也增加了处理器开支。

IP 编址

正确的 IP 编址对于用作大型 IP 网络的 VPN 的成功有着重要意义。为保持可扩展性、性能和可管理性，强烈建议远程站点使用主网的子网，以便进行归纳。增加 ACL 输入会降低性能，使故障查寻复杂化并影响可扩展性，适当的子网化还可支持简化的路由器头端配置，以实现分支到分支相互交流，要对所有设备的信息流进行归类，所需的隧道也较少。IP 编址还可影响 VPN 的多个方面，包括重叠网络的远程管理连接。

多协议隧道

IPSec 作为一种标准，只支持单点广播流量。对于多协议或 IP 多点广播隧道，必须使用另一种隧道协议。

网络地址转换

NAT 可发生于 IPSec 之前或之后。了解 NAT 何时发生是十分重要的，因为在某些情况下，由于隧道构建受阻或信息流穿过隧道，NAT 都可能对 IPSec 构成影响。除非提供接入是必须

的，否则将 NAT 应用于 VPN 流量将不失为上策。

单一目的和多目的设备

在网络设计过程中，您需要选择是在联网或安全设备中采用集成功能，还是采用 VPN 设备的特殊功能。集成功能通常是很吸引人的，因为您可以在现行设备上实施，且该设备经济有效，其特性可与其他设备互操作，从而提供功能更理想的解决方案。指定的 VPN 设备通常在对功能的要求很高或性能要求使用特殊硬件时，才会使用。当决定了采取何种选项，可根据设备的容量和功能对决策进行权衡，并与集成设备的功能优势相对照。在整个体系结构中，两类系统都有所使用。由于 IPSec 是一种要求严格的功能，随着设计规模的提高，选择 VPN 设备取代集成型路由器或防火墙的可行性也日趋增大。注意，对 VPN 设备这一概念的了解不是件容易的事情。当今的许多 VPN 设备可提供理想的性能和 VPN 管理选项，与此同时，也提供有限的路由选择、防火墙或 CoS 功能，而它们可能与集成设备有关。如果所有这些高级功能都得以实现，从性能和部署选项的角度来看，这种设备也开始越来越像集成型设备。同样，除了路由选择和安全特性的全面实施以外，可支持全部 VPN 功能的 VPN 路由器，可在 VPN 单独环境中进行配置，其特征更象一种应用。

入侵检测、网络访问控制、信任和 VPN

IPSec VPN 在部署时，周围通常环绕着多层访问控制和入侵检测。网络入侵检测系统(NIDS)是一项用于减少与扩展安全范围有关风险的技术。在 VPN 设计中，NIDS 完成了两项基本功能。首先，NIDS 可用于分析源自或送至 VPN 设备的信息流。其次，NIDS 可用于加密后，确认仅仅是加密信息流被发送并由 VPN 设备接收。

除 NIDS 以外，通常使用防火墙的访问控制应该在 VPN 设备前后设置。当今的部署都将防火墙安置在 VPN 设备的前面。当安置在前面时，对用户信息流的种类不具备可视性，因为信息流依然是加密的。防火墙在 VPN 设备前所能提供的大多数优势此时已丧失殆尽。

VLAN 如何在内网上进行划分?

在介绍具体应用之前，我们先来了解一下 VLAN 的定义、标准及如何划分等内容。

VLAN 的定义

VLAN（Virtual Local Area Network）即虚拟局域网，是一种通过将局域网内的设备逻辑地而不是物理地划分成一个个网段从而实现虚拟工作组的新兴技术。

VLAN 技术允许网络管理者将一个物理的 LAN 逻辑地划分成不同的广播域（或称虚拟 LAN，即 VLAN），每一个 VLAN 都包含一组有着相同需求的计算机工作站，与物理上形成的 LAN 有着相同的属性。但由于它是逻辑地而不是物理地划分，所以同一个 VLAN 内的各个工作站无须被放置在同一个物理空间里，即这些工作站不一定属于同一个物理 LAN 网段。

VLAN 是为解决以太网的广播问题和安全性而提出的，它在以太网帧的基础上增加了 VLAN 头，用 VLAN ID 把用户划分为更小的工作组，限制不同工作组间的用户二层互访，每个工作组就是一个虚拟局域网。虚拟局域网的好处是可以限制广播范围，并能够形成虚拟工作组，动态管理网络。

一个 VLAN 内部的广播和单播流量都不会转发到其他 VLAN 中，即使是两台计算机有着同样的网段，但是它们却没有相同的 VLAN 号，它们各自的广播流也不会相互转发,从而有助于控制流量、减少设备投资、简化网络管理、提高网络的安全性。既然 VLAN 隔离了广播风暴，同时也隔离了各个不同的 VLAN 之间的通讯，所以不同的 VLAN 之间的通讯是需要有路由来完成的。

VLAN 的划分

有四种方式：

1.根据端口来划分 VLAN

许多 VLAN 厂商都利用交换机的端口来划分 VLAN 成员。被设定的端口都在同一个广播

域中。例如，一个交换机的 1，2，3，4，5 端口被定义为虚拟网 AAA，同一交换机的 6，7，8 端口组成虚拟网 BBB。这样做允许各端口之间的通讯，并允许共享型网络的升级。但是，这种划分模式将虚拟网限制在了一台交换机上。

第二代端口 VLAN 技术允许跨越多个交换机的多个不同端口划分 VLAN，不同交换机上的若干个端口可以组成同一个虚拟网。

以交换机端口来划分网络成员，其配置过程简单明了。因此，从目前来看，这种根据端口来划分 VLAN 的方式仍然是最常用的一种方式。

2.根据 MAC 地址划分 VLAN

这种划分 VLAN 的方法是根据每个主机的 MAC 地址来划分，即对每个 MAC 地址的主机都配置它属于哪个组。这种划分 VLAN 方法的最大优点就是当用户物理位置移动时，即从一个交换机换到其他的交换机时，VLAN 不用重新配置，所以，可以认为这种根据 MAC 地址的划分方法是基于用户的 VLAN，这种方法的缺点是初始化时，所有的用户都必须进行配置，如果有几百个甚至上千个用户的话，配置是非常累的。而且这种划分的方法也导致了交换机执行效率的降低，因为在每一个交换机的端口都可能存在很多个 VLAN 组的成员，这样就无法限制广播包了。另外，对于使用笔记本电脑的用户来说，他们的网卡可能经常更换，这样，VLAN 就必须不停地配置。

3.根据网络层划分 VLAN

这种划分 VLAN 的方法是根据每个主机的网络层地址或协议类型(如果支持多协议)划分的，虽然这种划分方法是根据网络地址，比如 IP 地址，但它不是路由，与网络层的路由毫无关系。

其优点是用户的物理位置改变了，不需要重新配置所属的 VLAN，而且可以根据协议类型来划分 VLAN，这对网络管理者来说很重要，还有，这种方法不需要附加的帧标签来识别 VLAN，这样可以减少网络的通信量。其缺点是效率低，因为检查每一个数据包的网络层地址是需要消耗处理时间的(相对于前面两种方法)，一般的交换机芯片都可以自动检查网络上数据包的以太网帧头，但要让芯片能检查 IP 帧头，需要更高的技术，同时也更费时。当然，这与各个厂商的实现方法有关。

4.根据 IP 组播划分 VLAN

IP 组播实际上也是一种 VLAN 的定义，即认为一个组播组就是一个 VLAN，这种划分的方法将 VLAN 扩大到了广域网，因此这种方法具有更大的灵活性，而且也很容易通过路由器进行扩展，当然这种方法不适合局域网，主要是效率不高。

VLAN 的标准

对 VLAN 的标准，我们只是介绍两种比较通用的标准，当然也有一些公司具有自己的标准，比如 Cisco 公司的 ISL 标准，虽然不是一种大众化的标准，但是由于 Cisco Catalyst 交换机的大量使用，ISL 也成为一种不是标准的标准了。

· 802.10 VLAN 标准

在 1995 年，Cisco 公司提倡使用 IEEE802.10 协议。在此之前，IEEE802.10 曾经在全球范围内作为 VLAN 安全性的同一规范。Cisco 公司试图采用优化后的 802.10 帧格式在网络上传输 FramTagging 模式中所必须的 VLAN 标签。然而，大多数 802 委员会的成员都反对推广 802.10。因为，该协议是基于 FrameTagging 方式的。

· 802.1Q

在 1996 年 3 月，IEEE802.1Internetworking 委员会结束了对 VLAN 初期标准的修订工作。新出台的标准进一步完善了 VLAN 的体系结构，统一了 Fram-eTagging 方式中不同厂商的标签格式，并制定了 VLAN 标准在未来一段时间内的发展方向，形成的 802.1Q 的标准在业界获得了广泛的推广。它成为 VLAN 史上的一块里程碑。802.1Q 的出现打破了虚拟网依赖于单一厂商的僵局，从一个侧面推动了 VLAN 的迅速发展。另外，来自市场的压力使各大网络厂商立刻将新标准融合到他们各自的产品中。

下面给出一个 VLAN 的实例，进行说明:

某 IT 公司现有行政部、技术部、市场部。VLAN 的划分：行政部 VLAN10，技术部 VLAN20，市场部 VLAN30，各部门之间还可以相互通讯。

现有设备为:Cisco 3640 路由器，Cisco Catalyst 2924 交换机一台，二级交换机若干台。

交换机配置文件中的部分代码如下：

```
.....
!
interface vlan10
ip address 192.168.0.1
!
interface vlan20
ip address 192.168.1.1
!
interface vlan30
ip address 192.168.2.1
!
.....
```

路由器配置文件中的部分代码如下：

```
.....
interface FastEthernet 1/0.1
encapsulation isl 10
ip address 192.168.0.2
!
interface FastEthernet 1/0.2
encapsulation isl 20
ip address 192.168.1.2
!
interface FastEthernet 1/0.3
encapsulation isl 30
ip address 192.168.2.2
!
.....
!
router rip
network 192.168.0.0
!
```

VisualC++设计超强仿 QQ 自动伸缩窗口(1)

一、观察

模仿前最重要的一步就是观察，经过半天对 QQ 的摆弄和摸索，总结出了以下一些特点：

1、窗口开始粘附时，检测的是鼠标坐标与桌面边界的距离，特别地，粘附在下面的时候，检测的是与任务栏的距离；

2、在向上移动窗口时，窗口边界永远不会超出桌面上面边界；

3、窗口是个 TopMost 风格；

4、当窗口粘附在上面、左边或右边并显示时，你把鼠标移动到最顶端，光标变成改变窗口大小的图标，而单单是把窗口的 top 坐标设置为 0 是不行的；

5、粘附在下面的时候，当处于移动状态，那么窗口的底边是与任务栏顶边对齐的，但从隐藏到显示的时候，窗口的底端是与屏幕底边对齐的；

6、隐藏后显露出来的那条线可能是一个 Border，但肯定的是绝不包含 Client 区域；

7、关于响应鼠标的进入与移出窗口，绝对不是 WM_MOUSEMOVE、WM_MOUSELEAVE。证明：你以及其慢的速度接触隐藏状态的 QQ 边界，你会发现几乎是“一触即发”，你又以及其慢的速度移出显示状态的 QQ，你会发现它的收缩反而不是“一触即发”的，而是离边缘 10 象素左右。而 WM_MOUSEMOVE，WM_MOUSELEAVE，只有在进入、移出 Client 区域才响应，明显和 QQ 不同，其实从第 6 点也可以知道；

8、粘附在两边的时候，高度会调整为桌面上边界到任务栏下边界的距离；

9、在“拖动时显示窗口内容”模式下（桌面属性－外观－效果），粘附在两边的拖动出来时；如果收缩之前高度比收缩后小则回复原来高度，在非“拖动时显示窗口内容”模式下，光栅会回复原来高度，但释放左键时，高度却是收缩时调整后的高度，一开始我以为这是个 BUG，但我编写时同样出现这个问题，发现这两种模式会影响 WM_MOVING 参数的意义；

10、粘附在两边的时候当你设置任务栏自动隐藏，QQ 窗口会自动调整高度充满屏幕高度；

11、窗口显示或隐藏不是一瞬间的，这点在第 9 点提到的两种模式下，会有所不同；

12、任务栏并不显示 QQ 窗口；

二、编写代码

观察完毕，就开始编写了。

首先新建一个基于对话框的 MFC 程序，命名为 QQHideWnd，在对话框属性的 styles 页把 border 改为 Resizing，你也可同时把 Entended styles 的 tool window 钩上，对于这点我在程序了动态修改了。

在 QQHideWndDlg.h 头文件添加以下成员函数：

protected：

```
//修正移动时窗口的大小
void FixMoving(UINT fwSide, LPRECT pRect);
//从收缩状态显示窗口
void DoShow();
//从显示状态收缩窗口
void DoHide();
//重载函数,只是为了方便调用，实际调用 CWnd 的 SetWindowPos(…)
BOOL SetWindowPos(const CWnd* pWndInsertAfter,LPCRECT pCRect,
   UINT nFlags = SWP_SHOWWINDOW);
```

继续添加成员变量：

```
private::BOOL m_isSizeChanged;//窗口大小是否改变了
BOOL m_isSetTimer;//是否设置了检测鼠标的 Timer
INTm_oldWndHeight;//旧的窗口宽度 INTm_taskBarHeight;
  //任务栏高度 INTm_edgeHeight;//边缘高度
```

```
INTm_edgeWidth;//边缘宽度
INTm_hideMode;//隐藏模式
BOOL m_hsFinished;//隐藏或显示过程是否完成
BOOL m_hiding;//该参数只有在!m_hsFinished 才有效
//真:正在隐藏,假:正在显示
```

增加消息响应，需要注意的是有些消息你只有把右下角的 Filter for message 设置为 window 才能看到。

```
WM_ NCHITTEST
WM_MOVING
WM_CREATE
WM_TIMER
```

然后来到对应的 cpp 文件，在头部定义一些宏：

```
//收缩模式#define HM_NONE0//不收缩
#define HM_TOP1//向上收缩
#define HM_BOTTOM2//向下收缩
#define HM_LEFT3//向左收缩
#define HM_RIGHT4//向右收缩
#define CM_ELAPSE200 //检测鼠标是否离开窗口的时间间隔
#define HS_ELAPSE5//伸缩过程每步的时间间隔
#define HS_STEPS10//伸缩过程分成多少步完成
#define INTERVAL20//触发粘附时鼠标与屏幕边界的最小间隔,单位为象素
#define INFALTE10//触发收缩时鼠标与窗口边界的最小间隔,单位为象素
```

VisualC++设计超强仿 QQ 自动伸缩窗口(2)

然后在构造函数初始化成员变量：

```
m_isSizeChanged = FALSE;
m_isSetTimer = FALSE;m_hsFinished = TRUE;
m_hiding = FALSE;m_oldWndHeight = MINCY;
m_taskBarHeight = 30;
m_edgeHeight = 0;
m_edgeWidth=0;
m_hideMode = HM_NONE;
```

完成了一些初始的工作，那么就开始进入关键的函数实现了。首先是在 OnCreate 做些窗口的初始化和获得一些系统信息。

代码一

```
int CQQHideWndDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
if (CDialog::OnCreate(lpCreateStruct) == -1)
return -1;
// TODO: Add your specialized creation code here
//获得任务栏高度
CWnd* p;
p = this->FindWindow("Shell_TrayWnd",NULL);
```

```
if(p != NULL)
{
CRect tRect;
p->GetWindowRect(tRect);
m_taskBarHeight = tRect.Height();
}
//修改风格使得他不在任务栏显示
ModifyStyleEx(WS_EX_APPWINDOW, WS_EX_TOOLWINDOW);
//去掉关闭按键(如果想画 3 个按键的话)
//ModifyStyle(WS_SYSMENU,NULL);
//获得边缘高度和宽度
m_edgeHeight = GetSystemMetrics(SM_CYEDGE);
m_edgeWidth = GetSystemMetrics(SM_CXFRAME);
return 0;
}
```

接着如何知道鼠标进入或移出窗口呢？在前面我已经证明了 WM_MOUSEMOVE 和 WM_MOUSELEAVE 不符合我们的要求，于是我用了 WM_ NCHITTEST 这个消息，你可以看到我在这个消息响应函数中用了两个 SetTimer，一个用于检测鼠标是否离开，一个用于伸缩过程，不管你喜欢不喜欢，要达到第 7 点和第 11 点，这个是必须的，考虑的效率问题，在不需要的时候关闭这些 Timer 就好了。

代码二

```
UINT CQQHideWndDlg::OnNcHitTest(CPoint point)
{
// TODO: Add your message handler code here and/or call default
CString str;
str.Format("Mouse (%d,%d)",point.x,point.y);
GetDlgItem(IDC_CURSOR)->SetWindowText(str);
if(m_hideMode != HM_NONE && !m_isSetTimer &&
//防止鼠标超出屏幕右边时向右边收缩造成闪烁
point.x < GetSystemMetrics(SM_CXSCREEN) + INFALTE)
{ //鼠标进入时,如果是从收缩状态到显示状态则开启 Timer
SetTimer(1,CM_ELAPSE,NULL);
m_isSetTimer = TRUE;
m_hsFinished = FALSE;
m_hiding = FALSE;
SetTimer(2,HS_ELAPSE,NULL); //开启显示过程
}
return CDialog::OnNcHitTest(point);
}
```

然后在 OnTimer 中，

代码三

```
void CQQHideWndDlg::OnTimer(UINT nIDEvent)
{
// TODO: Add your message handler code here and/or call default
if(nIDEvent == 1 )
{
POINT curPos;
GetCursorPos(&curPos);
```

```
CString str;
str.Format("Timer On(%d,%d)",curPos.x,curPos.y);
GetDlgItem(IDC_TIMER)->SetWindowText(str);
CRect tRect;
//获取此时窗口大小
GetWindowRect(tRect);
//膨胀 tRect,以达到鼠标离开窗口边沿一定距离才触发事件
tRect.InflateRect(INFALTE,INFALTE);
if(!tRect.PtInRect(curPos)) //如果鼠标离开了这个区域
{
KillTimer(1); //关闭检测鼠标 Timer
m_isSetTimer = FALSE;
GetDlgItem(IDC_TIMER)->SetWindowText("Timer Off");
m_hsFinished = FALSE;
m_hiding = TRUE;
SetTimer(2,HS_ELAPSE,NULL); //开启收缩过程
}
}
if(nIDEvent == 2)
{
if(m_hsFinished) //如果收缩或显示过程完毕则关闭 Timer
KillTimer(2);
else
m_hiding ? DoHide() : DoShow();
}
CDialog::OnTimer(nIDEvent);
}
```

暂时不管 OnTimer 中的 DoHide(); DoShow();

先来看看核心的函数之一的 FixMoving，该函数在 OnMoving 中被调用，FixMoving 通过检测鼠标位置和窗口位置来决定窗口的收缩模式，并修正粘附边界时窗口的位置，从而达到像移动 QQ 时出现的效果。

VisualC++设计超强仿 QQ 自动伸缩窗口(3)

代码四

```
void CQQHideWndDlg::FixMoving(UINT fwSide, LPRECT pRect)
{
POINT curPos;
GetCursorPos(&curPos);
INT screenHeight = GetSystemMetrics(SM_CYSCREEN);
INT screenWidth = GetSystemMetrics(SM_CXSCREEN);
INT height = pRect->bottom - pRect->top;
INT width = pRect->right - pRect->left;
if (curPos.y <= INTERVAL)
{ //粘附在上边
pRect->bottom = height - m_edgeHeight;
pRect->top = -m_edgeHeight;
m_hideMode = HM_TOP;
}
else if(curPos.y >= (screenHeight - INTERVAL - m_taskBarHeight))
```

```
{ //粘附在下边
pRect->top = screenHeight - m_taskBarHeight - height;
pRect->bottom = screenHeight - m_taskBarHeight;
m_hideMode = HM_BOTTOM;
}
else if (curPos.x < INTERVAL)
{ //粘附在左边
if(!m_isSizeChanged)
{
CRect tRect;
GetWindowRect(tRect);
m_oldWndHeight = tRect.Height();
}
pRect->right = width;
pRect->left = 0;
pRect->top = -m_edgeHeight;
pRect->bottom = screenHeight - m_taskBarHeight;
m_isSizeChanged = TRUE;
m_hideMode = HM_LEFT;
}
else if(curPos.x >= (screenWidth - INTERVAL))
{ //粘附在右边
if(!m_isSizeChanged)
{
CRect tRect;
GetWindowRect(tRect);
m_oldWndHeight = tRect.Height();
}
pRect->left = screenWidth - width;
pRect->right = screenWidth;
pRect->top = -m_edgeHeight;
pRect->bottom = screenHeight - m_taskBarHeight;
m_isSizeChanged = TRUE;
m_hideMode = HM_RIGHT;
}
else
{ //不粘附
if(m_isSizeChanged)
{ //如果收缩到两边,则拖出来后会变回原来大小
//在"拖动不显示窗口内容下"只有光栅变回原来大小
pRect->bottom = pRect->top + m_oldWndHeight;
m_isSizeChanged = FALSE;
}
if(m_isSetTimer)
{ //如果 Timer 开启了,则关闭之
if(KillTimer(1) == 1)
m_isSetTimer = FALSE;
}
m_hideMode = HM_NONE;
GetDlgItem(IDC_TIMER)->SetWindowText("Timer off");
}
}
```

收缩模式和位置决定后，剩下的工作就由最后两个核心函数完成了：实现收缩的 DoHide()，实现伸展的 DoShow()。在这两个过程中 m_hsFinished，m_hiding 这两个变量起到很重要的控制作用。由于伸缩过程没完成时，hsFinished 始终为 FALSE，所以 Timer 2 不会关闭，于是在 OnTimer 中会重复调用这两个函数之一，在这两个函数体内，窗口位置有规律地递减或递增就可以达到 QQ 的“抽屉”效果了，有趣的是即使伸缩过程还没完成，你也可以在这个过程中改变 m_hiding 这个值来决定他是伸还是缩，正如 QQ 一样。你可以把 Timer 2 的事件间隔调大一点，然后在窗口伸缩时，鼠标来回地进出窗口就会很容易看到这样有趣的效果（还没缩进去又被拉了出来，或者还没拉出来又缩进去了）。

代码五

```
void CQQHideWndDlg::DoHide()
{
if(m_hideMode == HM_NONE)
return;
CRect tRect;
GetWindowRect(tRect);
INT height = tRect.Height();
INT width = tRect.Width();
INT steps = 0;
switch(m_hideMode)
{
case HM_TOP:
steps = height/HS_STEPS;
tRect.bottom -= steps;
if(tRect.bottom <= m_edgeWidth)
{ //你可以把下面一句替换上面的 ...+=|-=steps 达到取消抽屉效果
//更好的办法是添加个 BOOL 值来控制,其他 case 同样.
tRect.bottom = m_edgeWidth;
m_hsFinished = TRUE; //完成隐藏过程
}
tRect.top = tRect.bottom - height;
break;
case HM_BOTTOM:
steps = height/HS_STEPS;
tRect.top += steps;
if(tRect.top >= (GetSystemMetrics(SM_CYSCREEN) - m_edgeWidth))
{
tRect.top = GetSystemMetrics(SM_CYSCREEN) - m_edgeWidth;
m_hsFinished = TRUE;
}
tRect.bottom = tRect.top + height;
break;
case HM_LEFT:
steps = width/HS_STEPS;
tRect.right -= steps;
if(tRect.right <= m_edgeWidth)
{
tRect.right = m_edgeWidth;
m_hsFinished = TRUE;
}
```

```
tRect.left = tRect.right - width;
tRect.top = -m_edgeHeight;
tRect.bottom = GetSystemMetrics(SM_CYSCREEN) -
m_taskBarHeight;
break;
case HM_RIGHT:
steps = width/HS_STEPS;
tRect.left += steps;
if(tRect.left >= (GetSystemMetrics(SM_CXSCREEN) - m_edgeWidth))
{
tRect.left = GetSystemMetrics(SM_CXSCREEN) - m_edgeWidth;
m_hsFinished = TRUE;
}
tRect.right = tRect.left + width;
tRect.top = -m_edgeHeight;
tRect.bottom = GetSystemMetrics(SM_CYSCREEN) -
m_taskBarHeight;
break;
default:
break;
}
SetWindowPos(&wndTopMost,tRect);
}
```

VisualC++设计超强仿 QQ 自动伸缩窗口(4)

代码六

```
void CQQHideWndDlg::DoShow()
{
if(m_hideMode == HM_NONE)
return;
CRect tRect;
GetWindowRect(tRect);
INT height = tRect.Height();
INT width = tRect.Width();
INT steps = 0;
switch(m_hideMode)
{
case HM_TOP:
steps = height/HS_STEPS;
tRect.top += steps;
if(tRect.top >= -m_edgeHeight)
{ //你可以把下面一句替换上面的 ...+=|-=steps 达到取消抽屉效
果
//更好的办法是添加个 BOOL 值来控制,其他 case 同样.
tRect.top = -m_edgeHeight;
m_hsFinished = TRUE; //完成显示过程
}
tRect.bottom = tRect.top + height;
break;
case HM_BOTTOM:
```

```
steps = height/HS_STEPS;
tRect.top -= steps;
if(tRect.top <= (GetSystemMetrics(SM_CYSCREEN) - height))
{
tRect.top = GetSystemMetrics(SM_CYSCREEN) - height;
m_hsFinished = TRUE;
}
tRect.bottom = tRect.top + height;
break;
case HM_LEFT:
steps = width/HS_STEPS;
tRect.right += steps;
if(tRect.right >= width)
{
tRect.right = width;
m_hsFinished = TRUE;
}
tRect.left = tRect.right - width;
tRect.top = -m_edgeHeight;
tRect.bottom = GetSystemMetrics(SM_CYSCREEN) -
m_taskBarHeight;
break;
case HM_RIGHT:
steps = width/HS_STEPS;
tRect.left -= steps;
if(tRect.left <= (GetSystemMetrics(SM_CXSCREEN) - width))
{
tRect.left = GetSystemMetrics(SM_CXSCREEN) - width;
m_hsFinished = TRUE;
}
tRect.right = tRect.left + width;
tRect.top = -m_edgeHeight;
tRect.bottom = GetSystemMetrics(SM_CYSCREEN) -
m_taskBarHeight;
break;
default:
break;
}
SetWindowPos(&wndTopMost,tRect);
}
BOOL CQQHideWndDlg::SetWindowPos(const CWnd*
pWndInsertAfter,
LPCRECT pCRect, UINT nFlags)
{
return CDialog::SetWindowPos(pWndInsertAfter,pCRect->left,
pCRect->top,
pCRect->right - pCRect->left, pCRect->bottom - pCRect->top, nFlags);
}
```

到此，程序终于完成了。在我的源代码中还有对 WM_SIZING 的处理和定义了与之相关的宏，这些主要是控制窗口在调整大小时不能超过最小的宽度和高度，与 QQ 的自动伸缩无关，所以不在这里提及了。

三、结束语

虽然还不能算是完美的模仿，但效果已经非常非常的接近了。也许有人会奇怪为什么要用 Tool Window 风格，这是因为，这样在任务栏中不会显示窗口。从 QQ 的标题栏高度也可以判断出他也是这种风格，但这样一来就不能拥有最小化、最大化按键了。实际上 QQ 的最大化、最小化和关闭按键都是用 DC 画上去的。如何在 Caption 上增加按键，外国一些开源网站有源代码，我下载并看了一下，发现里面有个知识点很有趣，那就是更改消息路由，有兴趣的可以去下载来学习一下。

QQ 的成功很大部分在于他的界面比较人性化（用了 MSN 后深有感受），而这些界面实现起来原理也许很简单，难的是观察东西心要细、设计东西心要密、开发东西心要异。

VisualBasic9.0 隐式类型的局部变量

在隐式类型的局部变量声明中，局部变量的类型是通过局部声明语句右侧的初始值设定项表达式推断的。

例如，编译器推断以下所有变量声明的类型：

以下是引用片段：

```
Dim population = 31719
Dim name = "Belize"
Dim area = 1.9
Dim country = New Country With { .Name = "Palau", ...}
```

因此，它们完全等效于以下显式类型声明：

以下是引用片段：

```
Dim population As Integer = 31719
Dim name As String = "Belize"
Dim area As Float = 1.9
Dim country As Country = New Country With { .Name = "Palau", ...}
```

由于局部变量声明的类型是通过新增的 Option Infer On(新项目的默认值)推断的，因此不管 Option Strict 的设置如何，对此类变量的访问始终是早期绑定的。程序员必须在 Visual Basic 9.0 中显式指定后期绑定，方法是将变量显式声明为 Object 类型，如下所示：

以下是引用片段：

```
Dim country As Object = New Country With { .Name = "Palau", ... }
```

推断类型可防止意外使用后期绑定，更重要的是，它允许为新数据类型(如 XML)绑定强大扩展，如下所示。

For...Next 或 For Each...Next 语句中的循环控制变量也可以是隐式类型的变量。指定循环控制变量时(如 For I = 0 To SmallCountries.Count 或 For Each country In smallCountries 中所示)，标识符定义一个新的隐式类型局部变量，其类型通过初始值设定项或集合表达式推断且作用于整个循环。通过应用此类型推断，可以重新编写打印所有小国家/地区的循环，如下所示：

以下是引用片段：

```
For Each country In smallCountries
Console.WriteLine(country.Name)
Next
```

country 的类型被推断为 Country，即 SmallCountries 的元素类型。

UML 业务建模实例分析[1]

对于大中型信息系统，很难直接进行需求分析设计，需要借助模型来分析设计系统，根据系统调研数据，建立起目标系统的逻辑模型。

在软件工程的历史中，很长时间里人们一直认为需求分析是整个软件工程中最简单的一个步骤，但在过去十年中越来越多的人认识到它是整个过程中最为关键的一个过程。假如在需求分析时分析者们未能正确地认识到客户的需求的话，那么最后的软件实际上不可能达到客户的要求，或者导致需求的频繁变更，而软件无法在规定的时间里完工。

在需求分析阶段，要对经过可行性分析所确定的系统目标和功能作进一步的详细论述，确定系统“做什么？”的问题，最终建立起目标系统的逻辑模型。

首先是获得当前系统的物理模型。物理模型是对当前系统的真实写照，可能是一个由人工操作的过程，也可能是一个已有的但需要改进的计算机系统。首先是要对现行系统进行分析、理解，了解它的组织情况、数据流向、输入输出，资源利用情况等，在分析的基础上画出它的物理模型。然后抽象出当前系统的逻辑模型。

逻辑模型是在物理模型基础上，去掉一些次要的因素，建立起反映系统本质的逻辑模型。接下来建立目标系统的逻辑模型。通过分析目标系统与当前系统在逻辑上的区别，建立符合用户需求的目标系统的逻辑模型。最后补充目标系统的逻辑模型。对目标系统进行补充完善 ,将一些次要的因素补充进去，例如出错处理等。

UML（The Unified Modeling Language，即统一建模语言）是一种编制系统蓝图的标准化语言，可以对复杂的系统建立可视化的系统模型，目前已经被工业标准化组织 OMG（Object Management Group）接受，一经推出便得到许多著名的计算机厂商如 Microsoft、HP、IBM、Oracle 等的支持，也在逐步开始应用到需求分析过程中。

在使用 UML 建立当前系统逻辑模型过程中，初学者通常会遇到一些问题：

1.什么时候真正需要业务模型？什么时候用例模型独立存在？

2.在进行精确的业务建模时能用哪些 UML 图形？如何知道是否用顺序图或者交互图？

3.业务模型如何涉及到其他模型（如领域模型，用例模型等等）呢？如何有机地组织这些模型？

本文将通过图书馆管理系统这个简单而典型的实例来进行一次 UML 需求分析实践之旅。

许多读者对图书馆图书管理工作比较熟悉，主要是围绕读者、图书和工作人员的借还书展开工作。我们先看看图书馆工作人员和部分读者的需求。

读者来图书馆借书，可能先查询书库的图书记录。查询可以按书名、作者、图书编号、关键字查询。查询有两种结果，如果查到则记下书号，交给工作人员，然后等候办理借书手续。如果该书已经被全部借出，则可做借书登记，等待有书时被通知。如果图书馆没有该书的记录，则做缺书登记。

UML 业务建模实例分析[2]

办理借书手续时先要出示图书证，没有图书证则去申请图书证。如果借书数量超出规定，则提示“借书数量超限，不能继续借阅”。工作人员登记借阅人信息、借阅的图书信息、借出时间和应还书时间。系统自动修改书库的图书记录、读者库信息。

当一位读者还书时，工作人员根据图书证编号，找到读者的借书信息，查看是否超期，如果已经超期，则进行超期处罚。

如果图书有破损、丢失，则进行破损处罚。清除借阅记录，同时系统自动查看是否有等待借阅登记，如果有则发出通知，修改书库记录，该书设置为已预订状态，否则设置为可借状态。

图书采购人员进行图书采购时，要参考各类图书的库存数和借阅率，注意合理采购。如果有缺书登记则随时进行采购。正在采购的图书组成一个采购中书库。

采购到货后，进行验收，编号，同时加入图书库，修改采购中书库，并且查看订阅库，发出到书通知，并且已经修改书库的图书记录为已预订状态。

借书登记是当欲借的书被借空后，读者自愿选择的一种操作，它应该记录读者名和联系方式，一旦有这本书后可通知读者。

到书通知，当读者预订的书来到之后，按照读者给出的联系方式发出通知。

缺书登记是当读者需要的书库内查询没有记录时，将此信息转入缺货库，通知采购员采购。

图书注销，如果图书丢失或旧书淘汰，则将该书从书库中清除。

根据需求描述整理一张需求表：

需求分析时首先要识别出系统的参与者，在简单的图书馆管理系统中，可以划分出两种参与者：读者和管理员。当然，根据业务的复杂程度，参与者也可以进行细分，比如读者可以再分为学生读者、教师读者、校外读者，管理员根据业务和权限的不同可以再细分为库房管理员、借还书操作员、系统维护人员、图书馆管理人员等不同角色。在这里，为了简化处理，我们只列出了读者和管理员。对参与者描述如下：

（1）读者

描述：读者可以借阅、预定、归还物理书刊，可以对书籍和个人信息进行查询，可以取消预定，可以提出办卡申请。

示例：持有借阅卡的任何人和组织。

UML 业务建模实例分析[3]

（2）管理员

描述：图书管理员对系统进行维护，包括读者信息的创建、修改、删除，书刊信息的维护，条目信息的维护，还有系统信息的维护。

示例：图书管理员。

通过识别的参与者，对需求进一步分析，将业务需求进行分解，获得每个参与者的使用用例。在本例中，我们可以得到以下用例：

1.书籍借出：提供借阅物理书刊的功能。

2.书籍归还：提供归还物理书刊的功能。

3.读者办卡：提供为读者办理借阅卡的功能。

4.预定书刊：提供对某一个种类的书刊的预约功能。

5.取消预定：提供对预定进行取消的功能。

6.书籍查询：为读者提供网上的书籍查询功能。

7.信息查询：为读者提供信息查询的功能。

8.读者信息维护：提供读者信息的录入、修改、查询、删除的功能。

9.书刊信息维护：提供物理书刊的录入、修改、查询、删除的功能。

10.条目信息维护：提供书刊条目的录入、修改、查询、删除的功能。

11.系统信息维护：提供对系统的参数的设置。

12.登录：管理员需要先登录才能进入系统。

并且，可以画出如下系统用例图：

通过用例图，可以对系统功能有一个大概的了解，对于复杂系统，我们可以结合 IDEF 方法，通过分层分解，逐步细化的方法来描述系统的功能。对于用例图，建议不要画的过于复杂，特别是用例之间的关系，因为复杂的用例图不仅不能让需求分析人员与客户之间更好的沟通，反而是制造了一种沟通障碍。

UML 业务建模实例分析[4]

下一步就是编制每一个用例的详细说明，对用例说明的主要信息包括有：用例名称、编号、

用例的简短描述、用例的参与者、与其他用例的管理、用例启动的前提条件、用例结束后的事后条件、用例的输入、输出、用例的执行事件流等。在实际项目中，我们并不一定要面面俱到，而是根据实际情况对用例描述进行裁减。其中有几点重要信息是不能裁减的：用例名称、描述、输入、输出、执行事件流、参与者。另外，如果实际情况需要，还可以使用 MS Visio 等工具画出界面的示意图来。

如上例所述，我们对每一个用例都进行详细的描述，建立当前系统的功能用例模型。需求沟通与分析是一个迭代的过程，通过与用户的不断沟通，最终达成对目标系统的一致理解。如果用户确认了需求分析的成果，一般是需求规格说明书之后，项目开始进入系统分析设计阶段，也就是开始构造目标系统的逻辑模型。

为了让系统设计能够以结构、组织方式和代码重用的形式表现出来，要对系统进行设计规划，设计阶段应该与分析阶段交迭。需求是不断地发展，而设计本身也会推动需求的发展(反之亦然) 。在图书馆管理系统的建模设计中，以下 3 个方面的问题是要关注的：业务对象的表示、业务服务的实现、用户界面的组织。

业务对象的表示

在图书馆管理系统系统中，业务对象主要是数据库和数据实体类的表示方式。建模时，可以构造出系统的静态模型，也就是系统类图来表示。如下图则描述了借书这一用例的静态结构图。为了体现类之间的关系，在下图中没有显示出每一个类的属性和基本操作。

业务服务的实现

业务服务的实现需要完成的功能是各种业务规则和逻辑的实现,如借书处理的业务逻辑。每个模块的信息录入、修改、删除、查询等。业务规则和逻辑的实现基本相似，没有太多的规律可循。采用 UML 来进行业务服务的建模，可以使用 UML 的序列图、状态图、活动图。这个部分的工作,通常通过一系列的类之间的交互来完成。为了在更动态的层面上描述系统，UML 提供了许多其他类型的图。

对于 B/S 系统设计而言，情节图(Scenario Diagram) 特别有用。情节图分成两种：协作图(Collaboration Diagram) ，序列图(Sequence Diagram) 。UML 建模工具 Rational Rose 能够从协作图生成序列图也可以从序列图生成协作图。例如，借阅书刊的业务过程可以采用如下序列图来描述：

借阅书刊过程主要包括：管理员选择“借阅书刊”菜单，弹出对话框，管理员输入书刊信息和用户信息，系统查找数据库，是否存在该种物理书刊，如果不存在，显示提示信息，用例结束;是否存在借阅者信息，如果不存在，显示提示信息，用例结束;否则，管理员单击确认按钮后，该图书借阅给该借阅者，系统存储借阅信息到数据库。

用户界面的组织

用户界面布局图能够帮助组织系统页面、文件、服务的布局结构。在 UML 中，对于页面和文件的组织，可以使用构件图(Component Diagram) 或类图(Class Diagram) 建模型。本系统中使用类图对界面组织建模，页面结构以及各种业务服务被捆绑到不同的区域。

UML 业务建模实例分析[5]

在 UML 中，系统的体系结构使用部署图(DeploymentDiagram) 来完成。应用部署的规划对于规划整个 B/ S 系统是很有用的。它确定了一种有效的应用部署的规划组织方式,还可以作为一个模式在多个类似 B/ S 系统上应用。

在建模完成后，开发人员利用一些 UML Case 工具如 Rational ROSE 生成程序代码框架，并对代码框架进行修改和补充，形成完整代码;而且，还可根据代码逆向生成 UML 模型。这就较好地保证了模型与代码的一致性。

测试必须在整个项目周期中进行，对每个阶段都要用所建立的模型进行测试，这样才能保证开发的质量，减少开发的风险。

统一建模语言 UML 是国际软件工程领域具有划时代意义的重要成果，适用于以面向对象技术来描述任何类型的系统，而且适用于系统开发的不同阶段，从需求规格描述直至系统

完成后的测试和维护。软件系统的规模越来越大，复杂度不断提高，RUP 迭代式增量开发方式可以降低风险，同时可以适应需求变化的需要。

在本次 UML 实践之旅中，我们通过对图书馆管理系统的需求进行分析，将 UML 应用于系统开发的各个阶段，建立了系统的需求模型、静态模型和动态模型，同时遵循 Rationl 统一过程(RUP)的核心思想和基本原则，采用以用例为驱动、以体系构架为核心的迭代化面向对象分析和设计过程。

编号	简述	优先级
TA-1	借书	1
TA-2	还书	1
TA-3	读者申请办证	2
TA-4	查询读者	3
TA-5	图书注销	1
.	.	.

图 1：系统用例图

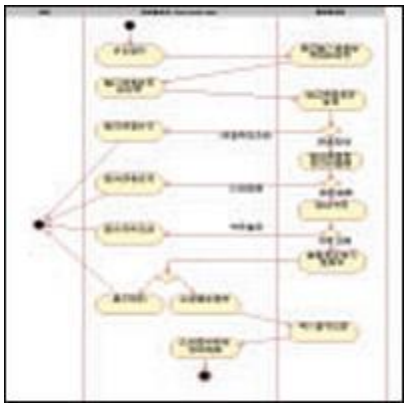

图 2：用况活动图

图 3：借书部分的类结构图

UML 业务建模实例分析[6]

UML 行为图

用况图（use case diagram）描述了一组用况和参与者（一种特殊的类）以及它们之间的

关系。

交互图（interaction diagram）是顺序图和协作图的统称。

顺序图(sequence diagram)是强调消息的时间次序的交互图。

协作图(collaboration diagram)是强调收发消息的对象的结构组织的交互图。

状态图显示了一个由状态，转换，事件和活动组成的状态机。

活动图显示了系统中从活动到活动的流。

UbuntuLinux 下 AzureusNAT 设置问题解决

在 Linux 下装了个 Azureus，NAT 端口测试的时候 15559 端口老是测试不成功，当然了，可能是端口没打开吧。其实我们也可以换成其它的端口。对于 Ubuntu 下 Azureus 出问题,一般有两种可能性比较大.一是程序运行不起来,这和 JDK 版本不兼容有关系,一般更新到 JDK6 就可以解决问题.另一是关于 NAT 配置问题

首先运行下列命令赋予 Azureus 一个端口:

iptables -I INPUT -p tcp --dport < your_port_number > -j ACCEPT

iptables -I INPUT -p udp --dport < your_port_number > -j ACCEPT

其中 your_port_number 是端口号,可以赋予 49125-65535 之间的任意值.然后创建一个文件

/etc/init.d/iptables_azureus

再写入下列信息:

(sleep 220

/sbin/iptables -I INPUT -p tcp --dport -j ACCEPT

/sbin/iptables -I INPUT -p udp --dport -j ACCEPT) &

上面取值 220,这是值足够大,(用来等待防火墙验证以及配置信息等等),你可以设置小点.

然后使该文件可执行并写入启动项。

chmod +x /etc/init.d/iptables_azureus

update-rc.d iptables_azureus start 51 S

这里一定要注意 51 S 后面还有一个点。

以上设置以后基本没多少问题，但是如果你的网络是通过路由器上网的话就不同了。

Telnet 协议基本知识

我们知道 Telnet 服务器软件是我们最常用的远程登录服务器软件，是一种典型的客户机/服务器模型的服务，它应用 Telnet 协议来工作。那么，什么是 Telnet 协议？它都具备哪些特点呢？

1 基本内容

Telnet 协议是 TCP/IP 协议族中的一员，是 Internet 远程登陆服务的标准协议。应用 Telnet 协议能够把本地用户所使用的计算机变成远程主机系统的一个终端。它提供了三种基本服务：

1）Telnet 定义一个网络虚拟终端为远的系统提供一个标准接口。客户机程序不必详细了解远的系统，他们只需构造使用标准接口的程序；

2）Telnet 包括一个允许客户机和服务器协商选项的机制，而且它还提供一组标准选项；

3）Telnet 对称处理连接的两端，即 Telnet 不强迫客户机从键盘输入，也不强迫客户机在屏幕上显示输出。

2 适应异构

为了使多个操作系统间的 Telnet 交互操作成为可能，就必须详细了解异构计算机和操作系统。比如，一些操作系统需要每行文本用 ASCII 回车控制符（CR）结束，另一些系统则需

要使用 ASCII 换行符（LF），还有一些系统需要用两个字符的序列回车-换行（CR-LF）；再比如，大多数操作系统为用户提供了一个中断程序运行的快捷键，但这个快捷键在各个系统中有可能不同（一些系统使用 CTRL+C，而另一些系统使用 ESCAPE）。如果不考虑系统间的异构性，那么在本地发出的字符或命令，传送到远地并被远地系统解释后很可能会不准确或者出现错误。因此，Telnet 协议必须解决这个问题。

为了适应异构环境，Telnet 协议定义了数据和命令在 Internet 上的传输方式，此定义被称作网络虚拟终端 NVT（Net Virtual Terminal）。它的应用过程如下：

对于发送的数据：客户机软件把来自用户终端的按键和命令序列转换为 NVT 格式，并发送到服务器，服务器软件将收到的数据和命令，从 NVT 格式转换为远地系统需要的格式；

对于返回的数据：远地服务器将数据从远地机器的格式转换为 NVT 格式，而本地客户机将将接收到的 NVT 格式数据再转换为本地的格式。

对于 NVT 格式的详细定义，有兴趣的朋友可以去查找相关资料。

3 传送远地命令

我们知道绝大多数操作系统都提供各种快捷键来实现相应的控制命令，当用户在本地终端键入这些快捷键的时候，本地系统将执行相应的控制命令，而不把这些快捷键作为输入。那么对于 Telnet 来说，它是用什么来实现控制命令的远地传送呢？

Telnet 同样使用 NVT 来定义如何从客户机将控制功能传送到服务器。我们知道 USASCII 字符集包括 95 个可打印字符和 33 个控制码。当用户从本地键入普通字符时，NVT 将按照其原始含义传送；当用户键入快捷键（组合键）时，NVT 将把它转化为特殊的 ASCII 字符在网络上传送，并在其到达远地机器后转化为相应的控制命令。将正常 ASCII 字符集与控制命令区分主要有两个原因：

1）这种区分意味着 Telnet 具有更大的灵活性：它可在客户机与服务器间传送所有可能的 ASCII 字符以及所有控制功能；

2）这种区分使得客户机可以无二义性的指定信令，而不会产生控制功能与普通字符的混乱。

4 数据流向

上面我们提到过将 Telnet 设计为应用级软件有一个缺点，那就是：效率不高。这是为什么呢？下面给出 Telnet 中的数据流向：

数据信息被用户从本地键盘键入并通过操作系统传到客户机程序，客户机程序将其处理后返回操作系统，并由操作系统经过网络传送到远地机器，远地操作系统将所接收数据传给服务器程序，并经服务器程序再次处理后返回到操作系统上的伪终端入口点，最后，远地操作系统将数据传送到用户正在运行的应用程序，这便是一次完整的输入过程；输出将按照同一通路从服务器传送到客户机。

因为每一次的输入和输出，计算机将切换进程环境好几次，这个开销是很昂贵的。还好用户的键入速率并不算高，这个缺点我们仍然能够接受。

5 强制命令

我们应该考虑到这样一种情况：假设本地用户运行了远地机器的一个无休止循环的错误命令或程序，且此命令或程序已经停止读取输入，那么操作系统的缓冲区可能因此而被占满，如果这样，远地服务器也无法再将数据写入伪终端，并且最终导致停止从 TCP 连接读取数据，TCP 连接的缓冲区最终也会被占满，从而导致阻止数据流流入此连接。如果以上事情真的发生了，那么本地用户将失去对远地机器的控制。

为了解决此问题，Telnet 协议必须使用外带信令以便强制服务器读取一个控制命令。我们知道 TCP 用紧急数据机制实现外带数据信令，那么 Telnet 只要再附加一个被称为数据标记(date mark)的保留八位组，并通过让 TCP 发送已设置紧急数据比特的报文段通知服务器便可以了，携带紧急数据的报文段将绕过流量控制直接到达服务器。作为对紧急信令的相应，服务器将读取并抛弃所有数据，直到找到了一个数据标记。服务器在遇到了数据标记后将返回

正常的处理过程。

6 选项协商

由于 Telnet 两端的机器和操作系统的异构性，使得 Telnet 不可能也不应该严格规定每一个 telnet 连接的详细配置，否则将大大影响 Telnet 的适应异构性。因此，Telnet 采用选项协商机制来解决这一问题。

Telnet 选项的范围很广：一些选项扩充了大方向的功能，而一些选项制涉及一些微小细节。例如：有一个选项可以控制 Telnet 是在半双工还是全双工模式下工作（大方向）；还有一个选项允许远地机器上的服务器决定用户终端类型（小细节）。

Telnet 选项的协商方式也很有意思，它对于每个选项的处理都是对称的，即任何一端都可以发出协商申请；任何一端都可以接受或拒绝这个申请。另外，如果一端试图协商另一端不了解的选项，接受请求的一端可简单的拒绝协商。因此，有可能将更新，更复杂的 Telnet 客户机服务器版本与较老的，不太复杂的版本进行交互操作。如果客户机和服务器都理解新的选项，可能会对交互有所改善。否则，它们将一起转到效率较低但可工作的方式下运行。所有的这些设计，都是为了增强适应异构性，可见 Telnet 的适应异构性对其的应用和发展是多么重要。

上面讨论了一些原理方面的东西，虽然我们在 Telnet 的使用过程中很难接触到这一层面，但我认为了解这些是有意义的，它会给我们带来许多启示。

SSLVPN 设备选购五大注意事项

SSL VPN 由于其强大的功能和实施的方便性应用越来越广泛，市场上的 SSL VPN 品牌也越来越多，如何选择适合自己的产品是需要用户仔细考虑的一个问题，本文从下面几个方面描述如何选择 SSL VPN 产品：

1.1 应用需求：

选择 VPN 是为了支持远程访问内部网络的应用，因此这一点也是最先需要考虑的一点，目前，大多数 SSL VPN 支持我们日常经常会用到的邮件系统、OA 系统、CRM/ERP 等等，但并不是所有的应用 SSL VPN 都能够提供支持，如动态端口的应用就只有部分 SSL VPN 能够提供支持。因此，在决定使用一款 SSL VPN 前一定要先确定是否能支持你的应用。

1.2 安全需求：

要构建一个安全的系统，不仅仅需要传输过程安全，还要提高系统安全性，以下几个方面是缺一不可的：

1)传输过程安全

传输的过程加密强度是确保内部数据不在传输过程中被黑客盗取的关键因素。传输过程加密强度越高，传输安全性就越有保障。目前，拥有 128 位加密以上的 SSL VPN 产品是比较适宜的，56 位 DES 加密相对强度低，选择时需要特别注意。

2)用户身份验证

用户名加密码的验证方式安全性相对较低，除了用户名和密码外，能提供其他的双因素验证方式的产品更加具有优势，如支持 PKI 体系等？

3)客户端设备的安全性：

客户端设备是否安装了个人防火墙、防病毒软件等。如果客户端设备不够安全，比如有木马程序，那么系统依然存在安全隐患。目前部分 SSL VPN 能够提供客户端环境检测，比如检测客户端是否安装了防火墙和防病毒软件。

4)完成访问后，客户端需要清除客户端机器的缓存

在移动用户完成远程访问后，是否就万事大吉了呢？当然不是，黑客或不法分子可以通过拷贝、复制驻留在客户端缓冲区内数据盗取企业机密。

5)服务端的日志跟踪

SSL VPN 服务器应该提供访问统计和跟踪功能，这样管理员能够根据日志随时掌握系统访问情况。

对于以上这些安全特性，SafeNet iGate SSL VPN 均能够提供支持。

1.SafeNet iGate 使用高强度的 128 位加密技术。

2.对于远程移动用户，iGate 能够结合 PKI 体系以及本地活动目录，值得一提的是，SafeNet 独有的 iKey 双因素身份认证 USB Key 与 iGate SSL VPN 完美结合，充分实现安全的双因素身份验证功能。

3.SafeNet iGate 支持客户端环境检测功能，SafeNet iGate 能够设定访问策略，当客户端不符合某个条件时，系统将禁止用户登陆。

4.为此，SafeNet iGate 在用户离线后可自动清除用户缓冲区的内容。另外，在拔除 iKey 后，访问也会自动中断。

5.SafeNet iGate 在用户界面上集成了日期查询功能，能够非常方便的进行日志跟踪。

1.3 易于管理和维护，使用操作性强

SSL VPN 的突出优势之一就在于移动性强、易用性强。但这些特性往往会增加管理难度。因此用户在选购 SSL VPN 时要重点考虑产品的管理性能。产品要做到界面简单，使用方便，灵活、细致地设置访问权限,采用基于用户/组/角色的认证机制，每个文件、网址或应用都可进行单独设置，使访问控制更易于管理。

SafeNet iGate 提供两个 Web 方式的管理 UI，一个是 Simple－UI，一个是 Classic－UI，把常用的设置和不常用的设置分别开来，这样大大降低了管理维护的复杂性。

1.4 性能

由于是集中系统，SSL 加速决定整个网络的吞吐量。如果 SSL 加速跟不上，远程接入就会比实际的 Internet 接入带宽低很多。有的 SSL VPN 产品采用专门的 SSL 加速硬件，从而提高了 VPN 的响应速度。另外，通过数据压缩技术，还对所有的传输数据进行压缩后再进行传输，这样就提高了整个网络的运行效率和实用性。

SafeNet iGate 配置硬件 SSL 加速卡，能够大大提高访问速度，另外 iGate 还提供数据压缩功能，能够大大增加网络吞吐量。

1.5 服务

除了上面提到的几点外，具有良好服务也至关重要。SSL VPN 还是一个在不断发展的技术，更新的可能会比较快，提供 SSL VPN 的厂家是否具有良好的产品服务质量、渠道响应速度和本地支持能力也非常重要。比如承诺免费或低费用升级，等等。

结束语

SSL VPN 的发展迎合了用户对低成本、高性价比远程访问的需求。现在，它已经广泛应用于各行各业。选购 SSL VPN 时，用户还要根据自身特点和不同的业务模式，选择适合自己的 SSL VPN 产品，再次强调，VPN 是正在发展的技术，更新换代可能会比较快，因此用户在选购时可以少考虑一些扩展性，多注重产品的实用性。毕竟，只有适合自己的，才是最理想的选择。

SQL 中 HAVING 从句的用法

用户在使用 SQL 语言的过程中可能希望解决的一个问题就是对由 sum 或其它集合函数运算结果的输出进行限制。例如，我们可能只希望看到 Store_Information 数据表中销售总额超过 1500 美圆的商店的信息，这时我们就需要使用 HAVING 从句。语法格式为：

```
SELECT "column_name1", SUM("column_name2")
FROM "table_name"
GROUP BY "column_name1"
HAVING (arithematic function condition)
```

（GROUP BY 从句可选）

由此，我们可以使用如下命令实现上述查询目的：

```
SELECT store_name, SUM(sales)
FROM Store_Information
GROUP BY store_name
HAVING SUM(sales) > 1500
```

查询结果显示为：

```
store_name SUM(Sales)
Los Angeles $1800
```

小注：

SQL 语言中设定集合函数的查询条件时使用 HAVING 从句而不是 WHERE 从句。通常情况下，HAVING 从句被放置在 SQL 命令的结尾处。

SQL 数据库设计经验

一个成功的管理系统，是由：[50% 的业务 + 50% 的软件] 所组成，而 50% 的成功软件又有 [25% 的数据库 + 25% 的程序] 所组成，数据库设计的好坏是一个关键。如果把企业的数据比做生命所必需的血液，那么数据库的设计就是应用中最重要的一部分。有关数据库设计的材料汗牛充栋，大学学位课程里也有专门的讲述。不过，就如我们反复强调的那样，再好的老师也比不过经验的教诲。所以我归纳历年来所走的弯路及体会，并在网上找了些对数据库设计颇有造诣的专业人士给大家传授一些设计数据库的技巧和经验。精选了其中的 60 个最佳技巧，并把这些技巧编写成了本文，为了方便索引其内容划分为 5 个部分：

第 1 部分 - 设计数据库之前
这一部分罗列了 12 个基本技巧，包括命名规范和明确业务需求等。

第 2 部分 - 设计数据库表
总共 24 个指南性技巧，涵盖表内字段设计以及应该避免的常见问题等。

第 3 部分 - 选择键
怎么选择键呢？这里有 10 个技巧专门涉及系统生成的主键的正确用法，还有何 时以及如何索引字段以获得最佳性能等。

第 4 部分 - 保证数据完整性
讨论如何保持数据库的清晰和健壮，如何把有害数据降低到最小程度。

第 5 部分 - 各种小技巧
不包括在以上 4 个部分中的其他技巧，五花八门，有了它们希望你的数据库开发工作会更轻松一些。

第 1 部分 - 设计数据库之前
考察现有环境
在设计一个新数据库时，你不但应该仔细研究业务需求而且还要考察现有的系统。大多数数据库项目都不是从头开始建立的；通常，机构内总会存在用来满足特定需求的现有系统（可

能没有实现自动计算）。显然，现有系统并不完美，否则你就不必再建立新系统了。但是对旧系统的研究可以让你发现一些可能会忽略的细微问题。一般来说，考察现有系统对你绝对有好处。

定义标准的对象命名规范

一定要定义数据库对象的命名规范。对数据库表来说，从项目一开始就要确定表名是采用复数还是单数形式。此外还要给表的别名定义简单规则（比方说，如果表名是一个单词，别名就取单词的前 4 个字母；如果表名是两个单词，就各取两个单词的前两个字母组成 4 个字母长的别名；如果表的名字由 3 个单词组成，你不妨从头两个单词中各取一个然后从最后一个单词中再取出两个字母，结果还是组成 4 字母长的别名，其余依次类推）对工作用表来说，表名可以加上前缀 WORK_ 后面附上采用该表的应用程序的名字。表内的列[字段]要针对键采用一整套设计规则。比如，如果键是数字类型，你可以用 _N 作为后缀；如果是字符类型则可以采用 _C 后缀。对列[字段]名应该采用标准的前缀和后缀。再如，假如你的表里有好多“money”字段，你不妨给每个列[字段]增加一个 _M 后缀。还有，日期列[字段]最好以 D_ 作为名字打头。

检查表名、报表名和查询名之间的命名规范。你可能会很快就被这些不同的数据库要素的名称搞糊涂了。假如你坚持统一地命名这些数据库的不同组成部分，至少你应该在这些对象名字的开头用 Table、Query 或者 Report 等前缀加以区别。

如果采用了 Microsoft Access，你可以用 qry、rpt、tbl 和 mod 等符号来标识对象（比如 tbl_Employees）。我在和 SQL Server 打交道的时候还用过 tbl 来索引表，但我用 sp_company （现在用 sp_feft_）标识存储过程，因为在有的时候如果我发现了更好的处理办法往往会保存好几个拷贝。我在实现 SQL Server 2000 时用 udf_ （或者类似的标记）标识我编写的函数。

工欲善其事, 必先利其器

采用理想的数据库设计工具，比如：SyBase 公司的 PowerDesign，她支持 PB、VB、Delphe 等语言，通过 ODBC 可以连接市面上流行的 30 多个数据库，包括 dBase、FoxPro、VFP、SQL Server 等，今后有机会我将着重介绍 PowerDesign 的使用。

获取数据模式资源手册

正在寻求示例模式的人可以阅读《数据模式资源手册》一书，该书由 Len Silverston、W. H. Inmon 和 Kent Graziano 编写，是一本值得拥有的最佳数据建模图书。该书包括的章节涵盖多种数据领域，比如人员、机构和工作效能等。其他的你还可以参考：[1]萨师煊　王珊 著　数据库系统概论（第二版）高等教育出版社　1991、[2][美] Steven M.Bobrowski 著 Oracle 7 与客户／服务器计算技术从入门到精通　刘建元等译 电子工业出版社，1996、[3]周中元　信息系统建模方法(下)　电子与信息化　1999 年第 3 期，1999

畅想未来，但不可忘了过去的教训

我发现询问用户如何看待未来需求变化非常有用。这样做可以达到两个目的：首先，你可以清楚地了解应用设计在哪个地方应该更具灵活性以及如何避免性能瓶颈；其次，你知道发生事先没有确定的需求变更时用户将和你一样感到吃惊。

一定要记住过去的经验教训！我们开发人员还应该通过分享自己的体会和经验互相帮助。即使用户认为他们再也不需要什么支持了，我们也应该对他们进行这方面的教育，我们都曾经面临过这样的时刻“当初要是这么做了该多好..”。

在物理实践之前进行逻辑设计

在深入物理设计之前要先进行逻辑设计。随着大量的 CASE 工具不断涌现出来，你的设计也可以达到相当高的逻辑水准，你通常可以从整体上更好地了解数据库设计所需要的方方面面。

了解你的业务
在你百分百地确定系统从客户角度满足其需求之前不要在你的 ER（实体关系）模式中加入哪怕一个数据表（怎么，你还没有模式？那请你参看技巧 9）。了解你的企业业务可以在以后的开发阶段节约大量的时间。一旦你明确了业务需求，你就可以自己做出许多决策了。
一旦你认为你已经明确了业务内容，你最好同客户进行一次系统的交流。采用客户的术语并且向他们解释你所想到的和你所听到的。同时还应该用可能、将会和必须等词汇表达出系统的关系基数。这样你就可以让你的客户纠正你自己的理解然后做好下一步的 ER 设计。

创建数据字典和 ER 图表
一定要花点时间创建 ER 图表和数据字典。其中至少应该包含每个字段的数据类型和在每个表内的主外键。创建 ER 图表和数据字典确实有点费时但对其他开发人员要了解整个设计却是完全必要的。越早创建越能有助于避免今后面临的可能混乱，从而可以让任何了解数据库的人都明确如何从数据库中获得数据。
有一份诸如 ER 图表等最新文档其重要性如何强调都不过分，这对表明表之间关系很有用，而数据字典则说明了每个字段的用途以及任何可能存在的别名。对 SQL 表达式的文档化来说这是完全必要的。

创建模式
一张图表胜过千言万语：开发人员不仅要阅读和实现它，而且还要用它来帮助自己和用户对话。模式有助于提高协作效能，这样在先期的数据库设计中几乎不可能出现大的问题。模式不必弄的很复杂；甚至可以简单到手写在一张纸上就可以了。只是要保证其上的逻辑关系今后能产生效益。

从输入输出下手
在定义数据库表和字段需求（输入）时，首先应检查现有的或者已经设计出的报表、查询和视图（输出）以决定为了支持这些输出哪些是必要的表和字段。举个简单的例子：假如客户需要一个报表按照邮政编码排序、分段和求和，你要保证其中包括了单独的邮政编码字段而不要把邮政编码糅进地址字段里。

报表技巧
要了解用户通常是如何报告数据的：批处理还是在线提交报表？时间间隔是每天、每周、每月、每个季度还是每年？如果需要的话还可以考虑创建总结表。系统生成的主键在报表中很难管理。用户在具有系统生成主键的表内用副键进行检索往往会返回许多重复数据。这样的检索性能比较低而且容易引起混乱。

理解客户需求
看起来这应该是显而易见的事，但需求就是来自客户（这里要从内部和外部客户的角度考虑）。不要依赖用户写下来的需求，真正的需求在客户的脑袋里。你要让客户解释其需求，而且随着开发的继续，还要经常询问客户保证其需求仍然在开发的目的之中。一个不变的真理是：“只有我看见了我才知道我想要的是什么”必然会导致大量的返工，因为数据库没有达到客户从来没有写下来的需求标准。而更糟的是你对他们需求的解释只属于你自己，而且可能是完全错误的。

第 2 部分 - 设计表和字段
检查各种变化
我在设计数据库的时候会考虑到哪些数据字段将来可能会发生变更。比方说，姓氏就是如此（注意是西方人的姓氏，比如女性结婚后从夫姓等）。所以，在建立系统存储客户信息时，我倾向于在单独的一个数据表里存储姓氏字段，而且还附加起始日和终止日等字段，这样就可以跟踪这一数据条目的变化。

采用有意义的字段名
有一回我参加开发过一个项目，其中有从其他程序员那里继承的程序，那个程序员喜欢用屏幕上显示数据指示用语命名字段，这也不赖，但不幸的是，她还喜欢用一些奇怪的命名法，其命名采用了匈牙利命名和控制序号的组合形式，比如 cbo1、txt2、txt2_b 等等。
除非你在使用只面向你的缩写字段名的系统，否则请尽可能地把字段描述的清楚些。当然，也别做过头了，比如 Customer_Shipping_Address_Street_Line_1，虽然很富有说明性，但没人愿意键入这么长的名字，具体尺度就在你的把握中。

采用前缀命名
如果多个表里有好多同一类型的字段（比如 FirstName），你不妨用特定表的前缀（比如 CusLastName）来帮助你标识字段。
时效性数据应包括“最近更新日期/时间”字段。时间标记对查找数据问题的原因、按日期重新处理/重载数据和清除旧数据特别有用。

标准化和数据驱动
数据的标准化不仅方便了自己而且也方便了其他人。比方说，假如你的用户界面要访问外部数据源（文件、XML 文档、其他数据库等），你不妨把相应的连接和路径信息存储在用户界面支持表里。还有，如果用户界面执行工作流之类的任务（发送邮件、打印信笺、修改记录状态等），那么产生工作流的数据也可以存放在数据库里。预先安排总需要付出努力，但如果这些过程采用数据驱动而非硬编码的方式，那么策略变更和维护都会方便得多。事实上，如果过程是数据驱动的，你就可以把相当大的责任推给用户，由用户来维护自己的工作流过程。

标准化不能过头
对那些不熟悉标准化一词（normalization）的人而言，标准化可以保证表内的字段都是最基础的要素，而这一措施有助于消除数据库中的数据冗余。标准化有好几种形式，但 Third Normal Form（3NF）通常被认为在性能、扩展性和数据完整性方面达到了最好平衡。简单来说，3NF 规定：
* 表内的每一个值都只能被表达一次。
* 表内的每一行都应该被唯一的标识（有唯一键）。
* 表内不应该存储依赖于其他键的非键信息。
遵守 3NF 标准的数据库具有以下特点：有一组表专门存放通过键连接起来的关联数据。比方说，某个存放客户及其有关定单的 3NF 数据库就可能有两个表：Customer 和 Order。Order 表不包含定单关联客户的任何信息，但表内会存放一个键值，该键指向 Customer 表里包含该客户信息的那一行。

更高层次的标准化也有，但更标准是否就一定更好呢？答案是不一定。事实上，对某些项目来说，甚至就连 3NF 都可能给数据库引入太高的复杂性。
为了效率的缘故，对表不进行标准化有时也是必要的，这样的例子很多。曾经有个开发餐饮分析软件的活就是用非标准化表把查询时间从平均 40 秒降低到了两秒左右。虽然我不得不这么做，但我绝不把数据表的非标准化当作当然的设计理念。而具体的操作不过是一种派生。所以如果表出了问题重新产生非标准化的表是完全可能的。

Microsoft Visual FoxPro 报表技巧
如果你正在使用 Microsoft Visual FoxPro，你可以用对用户友好的字段名来代替编号的名称：比如用 Customer Name 代替 txtCNaM。这样，当你用向导程序 [Wizards，台湾人称为‘精灵’] 创建表单和报表时，其名字会让那些不是程序员的人更容易阅读。

不活跃或者不采用的指示符

增加一个字段表示所在记录是否在业务中不再活跃挺有用的。不管是客户、员工还是其他什么人，这样做都能有助于再运行查询的时候过滤活跃或者不活跃状态。同时还消除了新用户在采用数据时所面临的一些问题，比如，某些记录可能不再为他们所用，再删除的时候可以起到一定的防范作用。

使用角色实体定义属于某类别的列[字段]

在需要对属于特定类别或者具有特定角色的事物做定义时，可以用角色实体来创建特定的时间关联关系，从而可以实现自我文档化。

这里的含义不是让 PERSON 实体带有 Title 字段，而是说，为什么不用 PERSON 实体和 PERSON_TYPE 实体来描述人员呢？比方说，当 John Smith, Engineer 提升为 John Smith, Director 乃至最后爬到 John Smith, CIO 的高位，而所有你要做的不过是改变两个表 PERSON 和 PERSON_TYPE 之间关系的键值，同时增加一个日期/时间字段来知道变化是何时发生的。这样，你的 PERSON_TYPE 表就包含了所有 PERSON 的可能类型，比如 Associate、Engineer、Director、CIO 或者 CEO 等。

还有个替代办法就是改变 PERSON 记录来反映新头衔的变化，不过这样一来在时间上无法跟踪个人所处位置的具体时间。

采用常用实体命名机构数据

组织数据的最简单办法就是采用常用名字，比如：PERSON、ORGANIZATION、ADDRESS 和 PHONE 等等。当你把这些常用的一般名字组合起来或者创建特定的相应副实体时，你就得到了自己用的特殊版本。开始的时候采用一般术语的主要原因在于所有的具体用户都能对抽象事物具体化。

有了这些抽象表示，你就可以在第 2 级标识中采用自己的特殊名称，比如，PERSON 可能是 Employee、Spouse、Patient、Client、Customer、Vendor 或者 Teacher 等。同样的，ORGANIZATION 也可能是 MyCompany、MyDepartment、Competitor、Hospital、Warehouse、Government 等。最后 ADDRESS 可以具体为 Site、Location、Home、Work、Client、Vendor、Corporate 和 FieldOffice 等。

采用一般抽象术语来标识“事物”的类别可以让你在关联数据以满足业务要求方面获得巨大的灵活性，同时这样做还可以显著降低数据存储所需的冗余量。

用户来自世界各地

在设计用到网络或者具有其他国际特性的数据库时，一定要记住大多数国家都有不同的字段格式，比如邮政编码等，有些国家，比如新西兰就没有邮政编码一说。

数据重复需要采用分立的数据表

如果你发现自己在重复输入数据，请创建新表和新的关系。

每个表中都应该添加的 3 个有用的字段

* dRecordCreationDate，在 VB 下默认是 Now()，而在 SQL Server 下默认为 GETDATE()
* sRecordCreator，在 SQL Server 下默认为 NOT NULL DEFAULT USER
* nRecordVersion，记录的版本标记；有助于准确说明记录中出现 null 数据或者丢失数据的原因

对地址和电话采用多个字段

描述街道地址就短短一行记录是不够的。Address_Line1、Address_Line2 和 Address_Line3 可以提供更大的灵活性。还有，电话号码和邮件地址最好拥有自己的数据表，其间具有自身的类型和标记类别。

过分标准化可要小心，这样做可能会导致性能上出现问题。虽然地址和电话表分离通常可以达到最佳状态，但是如果需要经常访问这类信息，或许在其父表中存放“首选”信息（比如 Customer 等）更为妥当些。非标准化和加速访问之间的妥协是有一定意义的。

使用多个名称字段

我觉得很吃惊，许多人在数据库里就给 name 留一个字段。我觉得只有刚入门的开发人员才

会这么做，但实际上网上这种做法非常普遍。我建议应该把姓氏和名字当作两个字段来处理，然后在查询的时候再把他们组合起来。
我最常用的是在同一表中创建一个计算列[字段]，通过它可以自动地连接标准化后的字段，这样数据变动的时候它也跟着变。不过，这样做在采用建模软件时得很机灵才行。总之，采用连接字段的方式可以有效的隔离用户应用和开发人员界面。

提防大小写混用的对象名和特殊字符
过去最令我恼火的事情之一就是数据库里有大小写混用的对象名，比如 CustomerData。这一问题从 Access 到 Oracle 数据库都存在。我不喜欢采用这种大小写混用的对象命名方法，结果还不得不手工修改名字。想想看，这种数据库/应用程序能混到采用更强大数据库的那一天吗？采用全部大写而且包含下划符的名字具有更好的可读性（CUSTOMER_DATA），绝对不要在对象名的字符之间留空格。

小心保留词
要保证你的字段名没有和保留词、数据库系统或者常用访问方法冲突，比如，最近我编写的一个 ODBC 连接程序里有个表，其中就用了 DESC 作为说明字段名。后果可想而知！DESC 是 DESCENDING 缩写后的保留词。表里的一个 SELECT * 语句倒是能用，但我得到的却是一大堆毫无用处的信息。

保持字段名和类型的一致性
在命名字段并为其指定数据类型的时候一定要保证一致性。假如字段在某个表中叫做“agreement_number”，你就别在另一个表里把名字改成“ref1”。假如数据类型在一个表里是整数，那在另一个表里可就别变成字符型了。记住，你干完自己的活了，其他人还要用你的数据库呢。

仔细选择数字类型
在 SQL 中使用 smallint 和 tinyint 类型要特别小心，比如，假如你想看看月销售总额，你的总额字段类型是 smallint，那么，如果总额超过了 $32,767 你就不能进行计算操作了。

删除标记
在表中包含一个“删除标记”字段，这样就可以把行标记为删除。在关系数据库里不要单独删除某一行；最好采用清除数据程序而且要仔细维护索引整体性。

避免使用触发器
触发器的功能通常可以用其他方式实现。在调试程序时触发器可能成为干扰。假如你确实需要采用触发器，你最好集中对它文档化。

包含版本机制
建议你在数据库中引入版本控制机制来确定使用中的数据库的版本。无论如何你都要实现这一要求。时间一长，用户的需求总是会改变的。最终可能会要求修改数据库结构。虽然你可以通过检查新字段或者索引来确定数据库结构的版本，但我发现把版本信息直接存放到数据库中不更为方便吗？。

给文本字段留足余量
ID 类型的文本字段，比如客户 ID 或定单号等等都应该设置得比一般想象更大，因为时间不长你多半就会因为要添加额外的字符而难堪不已。比方说，假设你的客户 ID 为 10 位数长。那你应该把数据库表字段的长度设为 12 或者 13 个字符长。这算浪费空间吗？是有一点，但也没你想象的那么多：一个字段加长 3 个字符在有 1 百万条记录，再加上一点索引的情况下才不过让整个数据库多占据 3MB 的空间。但这额外占据的空间却无需将来重构整个数据库就可以实现数据库规模的增长了。身份证的号码从 15 位变成 18 位就是最好和最惨痛的例

子。

列[字段]命名技巧
我们发现，假如你给每个表的列[字段]名都采用统一的前缀，那么在编写 SQL 表达式的时候会得到大大的简化。这样做也确实有缺点，比如破坏了自动表连接工具的作用，后者把公共列[字段]名同某些数据库联系起来，不过就连这些工具有时不也连接错误嘛。举个简单的例子，假设有两个表：
Customer 和 Order。Customer 表的前缀是 cu_，所以该表内的子段名如下：cu_name_id、cu_surname、cu_initials 和 cu_address 等。Order 表的前缀是 or_，所以子段名是：or_order_id、or_cust_name_id、or_quantity 和 or_description 等。
这样从数据库中选出全部数据的 SQL 语句可以写成如下所示：
Select * From Customer, Order Where cu_surname = "MYNAME" ;
and cu_name_id = or_cust_name_id and or_quantity = 1
在没有这些前缀的情况下则写成这个样子（用别名来区分）：
Select * From Customer, Order Where Customer.surname = "MYNAME" ;
and Customer.name_id = Order.cust_name_id and Order.quantity = 1
第 1 个 SQL 语句没少键入多少字符。但如果查询涉及到 5 个表乃至更多的列[字段]你就知道这个技巧多有用了。

第 3 部分 - 选择键和索引
数据采掘要预先计划
我所在的某一客户部门一度要处理 8 万多份联系方式，同时填写每个客户的必要数据（这绝对不是小活）。我从中还要确定出一组客户作为市场目标。当我从最开始设计表和字段的时候，我试图不在主索引里增加太多的字段以便加快数据库的运行速度。然后我意识到特定的组查询和信息采掘既不准确速度也不快。结果只好在主索引中重建而且合并了数据字段。我发现有一个指示计划相当关键——当我想创建系统类型查找时为什么要采用号码作为主索引字段呢？我可以用传真号码进行检索，但是它几乎就象系统类型一样对我来说并不重要。采用后者作为主字段，数据库更新后重新索引和检索就快多了。
可操作数据仓库（ODS）和数据仓库（DW）这两种环境下的数据索引是有差别的。在 DW 环境下，你要考虑销售部门是如何组织销售活动的。他们并不是数据库管理员，但是他们确定表内的键信息。这里设计人员或者数据库工作人员应该分析数据库结构从而确定出性能和正确输出之间的最佳条件。
使用系统生成的主键
这类同技巧 1，但我觉得有必要在这里重复提醒大家。假如你总是在设计数据库的时候采用系统生成的键作为主键，那么你实际控制了数据库的索引完整性。这样，数据库和非人工机制就有效地控制了对存储数据中每一行的访问。
采用系统生成键作为主键还有一个优点：当你拥有一致的键结构时，找到逻辑缺陷很容易。

分解字段用于索引
为了分离命名字段和包含字段以支持用户定义的报表，请考虑分解其他字段（甚至主键）为其组成要素以便用户可以对其进行索引。索引将加快 SQL 和报表生成器脚本的执行速度。比方说，我通常在必须使用 SQL LIKE 表达式的情况下创建报表，因为 case number 字段无法分解为 year、serial number、case type 和 defendant code 等要素。性能也会变坏。假如年度和类型字段可以分解为索引字段那么这些报表运行起来就会快多了。

键设计 4 原则
* 为关联字段创建外键。
* 所有的键都必须唯一。
* 避免使用复合键。
* 外键总是关联唯一的键字段。

别忘了索引
索引是从数据库中获取数据的最高效方式之一。95% 的数据库性能问题都可以采用索引技术得到解决。作为一条规则，我通常对逻辑主键使用唯一的成组索引，对系统键（作为存储过程）采用唯一的非成组索引，对任何外键列[字段]采用非成组索引。不过，索引就象是盐，太多了菜就咸了。你得考虑数据库的空间有多大，表如何进行访问，还有这些访问是否主要用作读写。
大多数数据库都索引自动创建的主键字段，但是可别忘了索引外键，它们也是经常使用的键，比如运行查询显示主表和所有关联表的某条记录就用得上。还有，不要索引 memo/note 字段，不要索引大型字段（有很多字符），这样作会让索引占用太多的存储空间。

不要索引常用的小型表
不要为小型数据表设置任何键，假如它们经常有插入和删除操作就更别这样作了。对这些插入和删除操作的索引维护可能比扫描表空间消耗更多的时间。
不要把社会保障号码（SSN）或身份证号码（ID）选作键
永远都不要使用 SSN 或 ID 作为数据库的键。除了隐私原因以外，须知政府越来越趋向于不准许把 SSN 或 ID 用作除收入相关以外的其他目的，SSN 或 ID 需要手工输入。永远不要使用手工输入的键作为主键，因为一旦你输入错误，你唯一能做的就是删除整个记录然后从头开始。
我在破解他人的程序时候，我看到很多人把 SSN 或 ID 还曾被用做系列号，当然尽管这么做是非法的。而且人们也都知道这是非法的，但他们已经习惯了。后来，随着盗取身份犯罪案件的增加，我现在的同行正痛苦地从一大摊子数据中把 SSN 或 ID 删除。

不要用用户的键
在确定采用什么字段作为表的键的时候，可一定要小心用户将要编辑的字段。通常的情况下不要选择用户可编辑的字段作为键。这样做会迫使你采取以下两个措施：
* 在创建记录之后对用户编辑字段的行为施加限制。假如你这么做了，你可能会发现你的应用程序在商务需求突然发生变化，而用户需要编辑那些不可编辑的字段时缺乏足够的灵活性。当用户在输入数据之后直到保存记录才发现系统出了问题他们该怎么想？删除重建？假如记录不可重建是否让用户走开？
* 提出一些检测和纠正键冲突的方法。通常，费点精力也就搞定了，但是从性能上来看这样做的代价就比较大了。还有，键的纠正可能会迫使你突破你的数据和商业/用户界面层之间的隔离。
所以还是重提一句老话：你的设计要适应用户而不是让用户来适应你的设计。
不让主键具有可更新性的原因是在关系模式下，主键实现了不同表之间的关联。比如，Customer 表有一个主键 CustomerID，而客户的定单则存放在另一个表里。Order 表的主键可能是 OrderNo 或者 OrderNo、CustomerID 和日期的组合。不管你选择哪种键设置，你都需要在 Order 表中存放 CustomerID 来保证你可以给下定单的用户找到其定单记录。
假如你在 Customer 表里修改了 CustomerID，那么你必须找出 Order 表中的所有相关记录对其进行修改。否则，有些定单就会不属于任何客户——数据库的完整性就算完蛋了。
如果索引完整性规则施加到表一级，那么在不编写大量代码和附加删除记录的情况下几乎不可能改变某一条记录的键和数据库内所有关联的记录。而这一过程往往错误丛生所以应该尽量避免。

可选键(候选键)有时可做主键
记住，查询数据的不是机器而是人。
假如你有可选键，你可能进一步把它用做主键。那样的话，你就拥有了建立强大索引的能力。这样可以阻止使用数据库的人不得不连接数据库从而恰当的过滤数据。在严格控制域表的数据库上，这种负载是比较醒目的。如果可选键真正有用，那就是达到了主键的水准。
我的看法是，假如你有可选键，比如国家表内的 state_code，你不要在现有不能变动的唯一

键上创建后续的键。你要做的无非是创建毫无价值的数据。如你因为过度使用表的后续键[别名]建立这种表的关联，操作负载真得需要考虑一下了。
别忘了外键
大多数数据库索引自动创建的主键字段。但别忘了索引外键字段，它们在你想查询主表中的记录及其关联记录时每次都会用到。还有，不要索引 memo/notes 字段而且不要索引大型文本字段（许多字符），这样做会让你的索引占据大量的数据库空间。

第 4 部分 - 保证数据的完整性
用约束而非商务规则强制数据完整性
如果你按照商务规则来处理需求，那么你应当检查商务层次/用户界面：如果商务规则以后发生变化，那么只需要进行更新即可。假如需求源于维护数据完整性的需要，那么在数据库层面上需要施加限制条件。如果你在数据层确实采用了约束，你要保证有办法把更新不能通过约束检查的原因采用用户理解的语言通知用户界面。除非你的字段命名很冗长，否则字段名本身还不够。
只要有可能，请采用数据库系统实现数据的完整性。这不但包括通过标准化实现的完整性而且还包括数据的功能性。在写数据的时候还可以增加触发器来保证数据的正确性。不要依赖于商务层保证数据完整性；它不能保证表之间（外键）的完整性所以不能强加于其他完整性规则之上。

分布式数据系统
对分布式系统而言，在你决定是否在各个站点复制所有数据还是把数据保存在一个地方之前应该估计一下未来 5 年或者 10 年的数据量。当你把数据传送到其他站点的时候，最好在数据库字段中设置一些标记。在目的站点收到你的数据之后更新你的标记。为了进行这种数据传输，请写下你自己的批处理或者调度程序以特定时间间隔运行而不要让用户在每天的工作后传输数据。本地拷贝你的维护数据，比如计算常数和利息率等，设置版本号保证数据在每个站点都完全一致。

强制指示完整性(参照完整性?)
没有好办法能在有害数据进入数据库之后消除它，所以你应该在它进入数据库之前将其剔除。激活数据库系统的指示完整性特性。这样可以保持数据的清洁而能迫使开发人员投入更多的时间处理错误条件。
关系
如果两个实体之间存在多对一关系，而且还有可能转化为多对多关系，那么你最好一开始就设置成多对多关系。从现有的多对一关系转变为多对多关系比一开始就是多对多关系要难得多。

采用视图
为了在你的数据库和你的应用程序代码之间提供另一层抽象，你可以为你的应用程序建立专门的视图而不必非要应用程序直接访问数据表。这样做还等于在处理数据库变更时给你提供了更多的自由。

给数据保有和恢复制定计划
考虑数据保有策略并包含在设计过程中，预先设计你的数据恢复过程。采用可以发布给用户/开发人员的数据字典实现方便的数据识别同时保证对数据源文档化。编写在线更新来“更新查询”供以后万一数据丢失可以重新处理更新。
用存储过程让系统做重活
解决了许多麻烦来产生一个具有高度完整性的数据库解决方案之后，我决定封装一些关联表的功能组，提供一整套常规的存储过程来访问各组以便加快速度和简化客户程序代码的开发。数据库不只是一个存放数据的地方，它也是简化编码之地。

使用查找
控制数据完整性的最佳方式就是限制用户的选择。只要有可能都应该提供给用户一个清晰的价值列表供其选择。这样将减少键入代码的错误和误解同时提供数据的一致性。某些公共数据特别适合查找：国家代码、状态代码等。

第 5 部分 - 各种小技巧
文档、文档、文档
对所有的快捷方式、命名规范、限制和函数都要编制文档。
采用给表、列[字段]、触发器等加注释的数据库工具。是的，这有点费事，但从长远来看，这样做对开发、支持和跟踪修改非常有用。
取决于你使用的数据库系统，可能有一些软件会给你一些供你很快上手的文档。你可能希望先开始在说，然后获得越来越多的细节。或者你可能希望周期性的预排，在输入新数据同时随着你的进展对每一部分细节化。不管你选择哪种方式，总要对你的数据库文档化，或者在数据库自身的内部或者单独建立文档。这样，当你过了一年多时间后再回过头来做第 2 个版本，你犯错的机会将大大减少。
使用常用英语（或者其他任何语言）而不要使用编码
为什么我们经常采用编码（比如 9935A 可能是‘青岛啤酒’的供应代码，4XF788-Q 可能是帐目编码）？理由很多。但是用户通常都用英语进行思考而不是编码。工作 5 年的会计或许知道 4XF788-Q 是什么东西，但新来的可就不一定了。在创建下拉菜单、列表、报表时最好按照英语名排序。假如你需要编码，那你可以在编码旁附上用户知道的英语。

保存常用信息
让一个表专门存放一般数据库信息非常有用。我常在这个表里存放数据库当前版本、最近检查/修复（对 FoxPro）、关联设计文档的名称、客户等信息。这样可以实现一种简单机制跟踪数据库，当客户抱怨他们的数据库没有达到希望的要求而与你联系时，这样做对非客户机/服务器环境特别有用。

测试、测试、反复测试
建立或者修订数据库之后，必须用用户新输入的数据测试数据字段。最重要的是，让用户进行测试并且同用户一道保证你选择的数据类型满足商业要求。测试需要在把新数据库投入实际服务之前完成。

检查设计
在开发期间检查数据库设计的常用技术是通过其所支持的应用程序原型检查数据库。换句话说，针对每一种最终表达数据的原型应用，保证你检查了数据模型并且查看如何取出数据。

Microsoft Visual FoxPro 设计技巧
对复杂的 Microsoft Visual FoxPro 数据库应用程序而言，可以把所有的主表放在一个数据库容器文件里，然后增加其他数据库表文件和装载同原有数据库有关的特殊文件。根据需要用这些文件连接到主文件中的主表。比如数据输入、数据索引、统计分析、向管理层或者政府部门提供报表以及各类只读查询等。这一措施简化了用户和组权限的分配，而且有利于应用程序函数（存储过程）的分组和划分，从而在程序必须修改的时候易于管理。

SQLServer 中导入导出数据的三种方式(一)

在我们建立一个数据库时，并且想将分散在各处的不同类型的数据库分类汇总在这个新建的数据库中时，尤其是在进行数据检验、净化和转换时，将会面临很大的挑战。幸好 SQL Server 为我们提供了强大、丰富的数据导入导出功能，并且在导入导出的同时可以对数据进行灵活的处理。

在 SQL Server 中主要有三种方式导入导出数据：使用 Transact-SQL 对数据进行处理；调

用命令行工具 BCP 处理数据；使用数据转换服务(DTS)对数据进行处理。这三种方法各有其特点，下面就它们的主要特点进行比较。

一、使用方式的比较

1. 使用 Transact-SQL 进行数据导入导出

我们很容易看出，Transact-SQL 方法就是通过 SQL 语句方式将相同或不同类型的数据库中的数据互相导入导出或者汇集在一处的方法。如果是在不同的 SQL Server 数据库之间进行数据导入导出，那将是非常容易做到的。一般可使用 SELECT INTO FROM 和 INSERT INTO。使用 SELECT INTO FROM 时 INTO 后跟的表必须存在，也就是说它的功能是在导数据之前先建立一个空表，然后再将源表中的数据导入到新建的空表中，这就相当于表的复制（并不会复制表的索引等信息）。而 INSERT INTO 的功能是将源数据插入到已经存在的表中，可以使用它进行数据合并，如果要更新已经存在的记录，可以使用 UPDATE。

```
SELECT * INTO table2 FROM table1
--table1 和 table2 的表结构相同
INSERT INTO table2 SELECT * FROM table3
--table2 和 table3 的表结构相同
```

当在异构数据库之间的进行数据导入导出时，情况会变得复杂得多。首先要解决的是如何打开非 SQL Server 数据库的问题。

在 SQL Server 中提供了两个函数可以根据各种类型数据库的 OLE DB Provider 打开并操作这些数据库，这两个函数是 OPENDATASOURCE 和 OPENROWSET。它们的功能基本上相同，不同之处主要有两点。

(1) 调用方式不同。

OPENDATASOURCE 的参数有两个，分别是 OLE DB Provider 和连接字符串。使用 OPENDATASOURCE 只相当于引用数据库或者是服务（对于 SQL Server、Oracle 等数据库来说）。要想引用其中的数据表或视图，必须在 OPENDATASOURCE(...)后进行引用。

在 SQL Server 中通过 OPENDATASOURCE 查询 Access 数据库 abc.mdb 中的 table1 表

```
SELECT * FROM OPENDATASOURCE('Microsoft.Jet.OLEDB.4.0',
'Provider=Microsoft.Jet.OLEDB.4.0;Data Source=abc.mdb;Persist Security
Info=False')...
table1
```

OPENROWSET 相当于一个记录集，可以将直接当成一个表或视图使用。

在 SQL Server 中通过 OPENROWSETE 查询 Access 数据库 abc.mdb 中的 table1 表

```
SELECT * FROM OPENROWSET('Microsoft.Jet.OLEDB.4.0', 'abc.mdb';
'admin';'','SELECT * FROM table1')
```

SQLServer 中导入导出数据的三种方式(二)

(2)灵活度不同。

OPENDATASOURCE 只能打开相应数据库中的表或视图，如果需要过滤的话，只能在 SQLServer 中进行处理。而 OPENROWSET 可以在打开数据库的同时对其进行过滤，如上面的例子，在 OPENROWSET 中可以使用 SELECT*FROMtable1 对 abc.mdb 中的数据表进行查询，而 OPENDATASOURCE 只能引用 table1，而无法查询 table1。因此，OPENROWSET 比较 OPENDATASOURCE 更加灵活。

2.使用命令行 BCP 导入导出数据

很多大型的系统不仅仅提供了友好的图形用户接口，同时也提供了命令行方式对系统进行控制。在 SQLServer 中除了可以使用 SQL 语句对数据进行操作外，还可以使用一个命令行工具 BCP 对数据进行同样的操作。BCP 是基于 DB-Library 客户端库的工具。它的功能十分强大，BCP 能够以并行方式将数据从多个客户端大容量复制到单个表中，从而大大提高了装载效

率。但在执行并行操作时要注意的是只有使用基于 ODBC 或 SQLOLEDB 的 API 的应用程序才可以执行将数据并行装载到单个表中的操作。
BCP 可以将 SQLServer 中的数据导出到任何 OLEDB 所支持的数据库的，如下面的语句是将 authors 表导出到 excel 文件中。
bcppubs.dbo.authorsoutc:\temp1.xls
-c-q-S"GNETDATA/GNETDATA"-U"sa"-P"password"

SQLServer 网络数据库系统的设计和开发

用 XBase 开发的程序，在系统老化之后，不应盲目推翻重来，而是要利用新技术把原有资源转化，以适应现在的需求。XBase 型数据库比较适合转化成网络数据库。做网络数据库要有前台的应用程序和后台的网络数据库。前台程序的开发平台可以用 Delphi,Visual FoxPro 和 PowerBuilder。后台的网络数据库可以是 Novell 的 Sybase,WindowsNT 的 SQL Server, UNIX 的 Informix，以及 DB/2 for NT 等等。

Client/Server 模式的突出优点是数据集中化，跨系统信息共享,便于数据管理和维护，但对客户端维护困难的问题，我们在后面提出一个解决方法。客户端与主服务器通过网络以 TCP/IP 协议连接。客户端可以并发的存取主服务器上的数据。系统管理员可以在主服务器上监视客户的上线情况及数据库的使用情况，做到实时监控。Delphi 是 Borland(已经更名 Inprise)的产品，控件多，比较适合数据库开发。下面用 Delphi 3.0 C/S for Win95 做前台应用程序，后台采用 Windows NT 的 SQL SERVER 6.5(以下简称 SQL 服务器)做服务器来说明一个程序的开发过程及注意事项。

一、基本编程原理和步骤

基本原理如图所示

1 开发环境

我们在开发 Client/Server 应用程序时需要两台机器(服务器,一个客户机),但 Delphi 可以做到单机开发。Delphi 的可伸缩性很强，把应用程序由单层过渡到两层，只要简单地把连接的数据集由本机的数据库重新指向 SQL 服务器即可。像 PowerBuild 一样,Delphi 也有本地库，这就是 InterBase Server,它提供了一个单用户多实例的 SQL 服务器平台，可以做测试平台。我们在将数据库应用程序转移到对 Sybase 等远程数据库的访问之前，可以在 Local InterBase Server 平台建立和测试数据库应用程序。这样可暂时不考虑网络连接，专心致力于解决业务逻辑。当业务逻辑实现后，只要把 Database 控件中的 Aliasname 改成新的数据源即可将程序扩大到网络环境。大大提高了开发软件的效率，并且降低了开发难度。

2 后台建立数据库并创建连接

数据库建立包括安装系统，建库建表，对数据库写触发子和存储过程。

安装时要注意的是要把 SQL Server 的客户数留到 50 以上。否则服务器会在用户多了以后死锁。

SQL Server 是图形化的管理，库表的建立非常容易 。我们不提倡直接建立数据库、表格，输入数据。我们采取的方法是利用已有 DBF 数据库，在 Delphi 的 Database Desktop 把 DBF 转换成 SQL Server 中的表格。这样 XBase 开发的程序的资源就不会浪费了。对新表，用 FoxBase 建表格，输入数据后再转换。具体方法是：先用 ODBC 建立与 SQL Server 连接的数据源，用 Database Desktop 的菜单项 TOOLS－>UTILITIES－>COPY 的功能。其实这个工具可以进行异种数据库转换，在本身具有 ODBC 驱动程序或 Delphi 的 SQL LINK 支持的数据库之间任意进行转换。

3.前台程序的基本编程

Delphi 与后台数据库的连接可有两种途径，一是 ODBC，这是标准，兼容性很好。二是 Delphi 带的 SQL Link，它由 Delphi 自己开发，速度稍快，但我们在应用中发现 SQL Link 对有些数据库系统的支持并不是很稳定，如连 Informix 时，退出时就很容易死机。所以常以 ODBC

连接数据库。在控制面板的 ODBC 中需要设置数据源名称，服务器名称（如果不设置登录的数据库名称，就将登录到后台数据库给你这个用户的默认数据库）。

Delphi 是快速开发工具(RAD)中最容易上手的。编程一般经过三个步骤:

(1)注册 ODBC 数据源；

(2)配置 BDE；

(3)往 FROM 上放置 Query/Table,DataSource, DBgrid 控件,分别设置控件属性以实现与数据库的连接操作。具体说明如下:

Delphi 涉及数据库编程的控件有两类：

数据显示控件：用来显示数据库内的数据。DBGrid 用于全屏显示和编辑数据库表中的记录。数据连接控件：负责掌管数据库的连接。

Database 控件是为开发两层数据库应用程序时，设置登录数据库的有关参数。

Query 控件是用来传递 SQL 语句到服务器上得到一个数据集或执行一个动作。

Datasource 控件是连接数据显示控件和数据连接控件的桥梁。

Query 控件执行静态 SQL 语句的写法是：

Query1.SQL.Clear;

Query1.SQL.Add('SELECT ＊ FROM databasename ');

Query1.SQL.Open，

值得一提的是 Delphi 中还有种动态 SQL，可以嵌入变量参数，给编程带来极大方便。

Query1.SQL.Clear;

Query1.SQL.Add('SELECT ks FROM ks WHERE py like :py');

Query1.ParamByName('py').AsString:= Edit1.text＋'%'

Query1.Open;

Query 控件的执行方法有两种：Open，ExecSQL。Open 方式可以打开所联系的数据表格,得到一个数据集。ExecSQL 方式则只是运行 SQL 语句,并不将运行后的表格送往相连的数据显示控件，所以执行像 Update dataname Set ....,Delete ...FROM .. 的动作，就要用 ExecSQL。

当应用程序第一次访问 SQL 数据库，会触发一个自动连接过程。连接过程需要确认访问数据库的权限。如果你要在程序中接受口令，则必须把 Database 控件的 LoginPrompt 属性设为 False。

Database1.Params.Add('user name='＋myusername); // myusername 变量名

Database1.Params.Append('password='＋mypassword);

Database1.Connected:=true;

关于日期的处理，要不存在 2000 年问题，只要在 Form.OnCreate 事件中加上 ShortDateFormat:='mm/dd/dddd/yyyy'；mm ：有前导 0 的月份。dd：有前导 0 的日期，dddd:表示这一天是星期几，yyyy：四位年份。此外,在 Delphi 中的 BDE Aministrator 中的 Configuration－>System－>Formats 中有资料介绍，此方法的可行性尚未得到证实。

二、事务处理

在客户/服务器应用程序中，事务控制用来维护数据一致性。Delphi 中提供了事务的隐式和显式方法。隐式控制对写入数据库的数据的每一行都要进行事务控制，导致网络繁忙和程序性能下降；用显式控制能自定义开始、提交和终止事务的过程，因而网络开销小，性能高。

显式控制有两种方法：

1?利用 Database 控件

Database 控件用于事务控制的属性是 TransIsolation，方法有 StartTranstion、Commit 和 Rollback。标准写法如下：

Databasel.starttransaction;

Try

Query.SQL.Clear; //具体处理

```
Query.SQL.Add('update databasename set gz=gz＋1');
Query.ExecSQL;
Databasel.commit;
Except
Databasel.rollback; //遇异常，回滚
End;
```

2.直接利用远程 SQL 服务器的事务处理功能，把 SQL 语句通过 Query 控件传递到服务器上。当程序不用本地库时，可以使用这种方法，但要用在 BDE 将 SQLPassThroughMode 设置为 NOT SHARED。程序中我们可采用如下结构:

```
IF MessageDlg('确认改标志位?'', mtConfirmation, [mbYes, mbNo], 0)=mrYes then
Begin
DoSQL:=0; //用一个变量做标志
Query.SQL.Clear; // 改状态。
Query.SQL.Add('update .... Set .....)');
While DoSQL=0 do
Try
Query.ExecSQL;
DoSQL:=1;
Except
IF MessageDlg('改状态时发生共享冲突，要重试吗 ',
mtConfirmation, [mbYes, mbNo], 0) =mrNo Then Exit;
end;
end;
```

3 减小共享冲突的概率

在实际中还有一种情况也可能引起共享冲突，这就是缓存更新。应用程序往数据库中写数据，先放在本地的缓冲区里然后由 BDE 提交,这样在多用户情况下就可能发生数据同时由缓存向数据库提交的情况，引起共享冲突。所以对频繁修改的数据要取消缓存更新。

在数据库的应用中，利用临时表也能减小冲突的概率。利用 SQL 语句可以在服务器上建立临时表，临时表会在使用完后被系统自动删掉。

例如下面语句分别把查询结果和统计结果放入临时表：

```
SELECT ＊ FROM DatabaseName INTO temp temptable..
SELECT city_id,COUNT(＊) lzcount ,SUM(jour_amt) lzsum FROM db_connect
WHERE flag="2" GROUP BY city_id ORDER BY city_id INTO TEMP tempdatabase'
```

三、系统优化

1 服务器端

要尽可能的把客户端的工作向服务器上移植。在服务器上大量使用触发子预处理程序、完整性约束才能做到真正意义上的瘦客户端。Delphi 可以用 Query 控件传递 SQL 语句到服务器端而且可以动态地在服务器端创建存储过程,用 StoreProc 控件调用服务器上的存储过程。一些涉及大量记录的工作任务因此可以用存储过程改写。对唯一性字段（身份证号，凭证提交号）可以作为数据库主键，让重复数据不能入库,省去客户端对数据的检测。

下面的例子是利用触发子做数据库 RS 的更新日志功能：

```
CREATE TRIGGER updatetrigger ON dbo.rs
FOR update
AS
declare @oldvalue char(20)
SELECT @oldvalue =xm FROM deleted
declare @newvalue char(20)
SELECT @newvalue=xm FROM inserted
```

insert log values //往 LOG 表中写。
('rs 数据库','用户'+user_name( ),GETDATE( ),@oldvalue,@newvalue,'更新')
// User_name( ),GetDate( ) 是 SQL SERVER 系统函数

2 客户端
(1) 客户端禁用 Table 控件
举例：对数据库的一个字段进行求和，Table 控件的做法是把整个数据集传到客户机上，然后累加。而 Query 控件则是把 SQL 语句传到服务器上，由服务器统计出结果。两者相比 Query 在网络上只传回较少数据，而 Table 传输的数据较多。

(2)去除无用代码
有时程序中无用的代码占据大量空间。在程序调试过程中，可整理代码。方法是在调试时不用编译或保存，打开 File 菜单，选择 Save File As...，在保存文件对话框中选择 Cancel 即可去除无用代码。

(3)减少不必要的 I/O
要加快客户

SQLServer 连接基础知识

引言
该堆栈的顶部是 API 或对象库层。应用程序通过对象库公开的 API 函数或接口连接到 Microsoft? SQL Server。用于访问 SQL Server 的 API 示例包括 ODBC 和 DB-Library。用于访问 SQL Server 的对象库示例包括 OLE DB、ADO 和 ADO.NET。由于 ADO 最终使用 OLE DB 与服务器通信，因此 Windows 应用程序在与 SQL Server 通信时实际上只使用两个常用的对象库，即 OLE DB 和 ADO.NET。由于通过 ADO 或 ADO.NET 进行连接通常比通过 ODBC 进行连接更普遍（但 SQL Server 的查询分析器和企业管理器仍通过 ODBC 进行连接），因此本文将从 ADO/OLE DB 和 ADO.NET 的角度介绍 SQL Server 连接体系结构的客户端。如今，大多数应用程序均通过对象库（而非 ODBC 或类似 API）连接到 SQL Server。

ADO 和 OLE DB
OLE DB 客户端（也称作使用者）通过客户端提供程序与服务器以及其他后端程序进行通信。此提供程序是一组 COM 组件（一个或多个），用于将应用程序请求转换为网络进程间通信 (IPC) 请求。在使用 SQL Server 的情况下，最常用的 OLE DB 提供程序是 SQLOLEDB，它是 Microsoft 为 SQL Server 提供的 OLE DB 提供程序。SQLOLEDB 随附于 SQL Server 中，并作为 Microsoft 数据访问组件 (MDAC) 库的一部分安装。
为了使用 ADO 与 SQL Server 进行通信，应用程序首先使用 Connection 对象建立与服务器的连接。ADO 的 Connection 对象接受一个连接字符串，该字符串指定要使用的 OLE DB 提供程序以及传递给它的参数。如果应用程序使用 SQLOLEDB 提供程序连接到 SQL Server，则该字符串中将显示“SQLOLEDB”。
ADO 应用程序还可以通过 ODBC 连接到 SQL Server。为此，应用程序将使用适用于 ODBC 的 OLE DB 提供程序，并指定在其连接字符串中引用目标 SQL Server 的 ODBC 数据源。这种情况下，应用程序与 OLE DB 进行通信，同时 ODBC 的 OLE DB 提供程序调用相应的 ODBC API，以便与 SQL Server 进行会话。

ADO.NET
ADO.NET 应用程序通常使用 .NET Framework Data Provider for SQL Server 连接到 SQL Server。该本机提供程序使 ADO.NET 对象能够与 SQL Server 直接进行通信。通常，应用程序使用 SqlConnection 对象建立连接，然后使用 SqlCommand 对象向服务器发送命令，并接收服务器返回的结果。SqlDataAdapter 和 SqlDataReader 类通常与 SqlCommand 一起使

用，以便通过托管的代码应用程序与 SQL Server 进行交互。

通过 OleDbConnection 类，ADO.NET 应用程序还可以使用 SQLOLEDB OLE DB 提供程序与 SQL Server 进行交互。此外，它们可以通过 OdbcConnection 类使用 ODBC 访问 SQL Server。因此，仅通过托管代码，您就有三种不同的方法从应用程序访问 SQL Server。从故障排除的角度而言，了解这些方法是非常有用的，因为它可以帮助您将遇到的与连接相关的问题归结到特定的数据访问层或库。

客户端 Net-Library

该堆栈中的下一层是 Net-Library。Net-Library 在 API 或对象库（应用程序使用它与 SQL Server 进行通信）与网络协议（用于与网络交换数据）之间提供了一个通道。SQL Server 为所有主要的网络协议提供了 Net-Library。这些库以透明方式将客户端发出的请求发送到 SQL Server，并将服务器发出的响应返回给客户端。可以使用 SQL Server 的客户端网络实用程序配置适用于特定客户端的 Net-Library。支持的客户端协议包括 TCP/IP、命名管道、NWLink、多协议 (RPC) 和其他一些协议。

尤其值得一提的 Net-Library 是共享内存 Net-Library。顾名思义，该 Net-Library 使用 Windows 的共享内存功能在 SQL Server 客户端与服务器之间进行通信。显然，这意味着客户端与服务器必须位于同一台物理计算机上。

由于它能够绕过物理网络堆栈，因此共享内存 Net-Library 要比其他 Net-Library 快得多。对共享内存区域的访问受到同步对象的保护，因此客户端与服务器之间的通信速度主要受限于 Windows 对内核对象进行调度的能力，以及进程与共享内存区域之间进行数据复制的能力。

可以在连接时将某个时间段或（本地）指定为您的计算机名，来指示使用共享内存 Net-Library。也可以在连接时为计算机\实例名加上前缀 lpc:，来指示要使用共享内存 Net-Library。

注意，即使连接到同一台计算机上的 SQL Server，共享内存 Net-Library 也未必就是最佳的连接选项。在某些情况下，客户端与服务器之间的直接连接可能限制它的扩展性。与应用程序整体体系结构中的其他元素一样，应始终对给定技术解决方案进行全面的测试，然后才能判断它是否有良好的扩展性以及是否比其他方法更快。

连接

客户端进行连接时，SQL Server 的用户模式计划程序 (UMS) 组件将它指定给特定的计划程序。启动时，SQL Server 为系统上的每个 CPU 创建一个单独的 UMS 计划程序。当客户端连接到服务器时，这些客户端将指定给具有最少连接数的计划程序。连接后，客户端将不会更换计划程序 - 它将始终受到指定计划程序的控制，直到连接断开。

这对与服务器建立多个连接的应用程序很重要。如果应用程序性能较差，或无法在它的多个连接上平均分配工作，则在该应用程序的某些连接之间可能造成不必要的 CPU 资源争用，而其他连接实际上却处于空闲状态。

例如，应用程序与双处理器计算机上运行的 SQL Server 建立了四个连接，连接 1 和 3 隶属于处理器 0，连接 2 和 4 隶属于处理器 1。如果应用程序的大部分工作通过连接 1 和 3 执行，则这两个连接将争用 CPU 0，而 CPU 1 实际上可能仍处于空闲状态。这种情况下，应用程序只能断开某些连接或重新连接某些连接，并希望连接 1 和 3 隶属于不同的 CPU（连接时无法指定处理器隶属关系），或在它的连接上重新分配工作负荷，以便每个连接的工作负荷更加均衡。当然，后一种情况要远好于前一种情况。

连接内存

SQL Server 为客户端请求的每个连接保留三个数据包缓冲区。每个缓冲区的大小取决于 sp_configure 存储过程指定的默认网络数据包大小。如果默认网络数据包大小小于 8 KB，则这些数据包的内存将由 SQL Server 的缓冲池提供。否则，该内存将由 SQL Server 的 MemToLeave 区域分配。

值得一提的是，.NET Framework Data Provider for SQL Server 的默认网络数据包大小为 8KB，因此，与托管代码客户端连接关联的缓冲区通常由 SQL Server 的 MemToLeave 区域

提供。而典型的 ADO 应用程序却不同，它们的默认数据包大小为 4 KB，因此缓冲区将由 SQL Server 缓冲池分配。

事件

连接后的客户端请求通常分为两种广泛类别：语言事件和远程过程调用。尽管还存在其他类别，但大多数由 SQL Server 客户端发送到服务器的请求由以下两种类型之一构成：语言事件是从客户端发送到服务器的 一组 T-SQL。例如，如果调用 ADO Command 对象（其 CommandText 属性设置为 T-SQL 查询，CommandType 属性设置为 adCmdText）的 Execute 方法，则查询将作为语言事件提交给服务器。同样，如果将 CommandType 设置为 adCmdTable 并调用 Execute 方法，则 ADO 将生成一个内部查询（它将选择 CommandText 属性标识的表中的所有列），并将它作为语言事件提交给服务器。另一方面，如果将 CommandType 设置为 adStoredProc，则调用 Execute 将使 ADO 向服务器提交一个远程过程调用请求，以执行 CommandText 属性中列出的存储过程。

为何要关心将请求作为语言事件还是作为 RPC 提交给服务器呢？通常，这是因为 RPC 的功能更为出色，特别是在重复调用具有不同筛选值的同一查询时。尽管 SQL Server 可以自动将普通的语言事件请求参数化，但这种能力非常有限。它从不尝试自动将某些类型的查询参数化。这可能会导致基本相同的查询产生不同的执行，从而只因为这些不同的执行提供不同的值，而导致在服务器上白白浪费计划编译的成本。这通常不是您所希望的结果 - 您希望针对查询的第一次执行编译一个新的计划，然后将该计划重复用于具有不同参数的执行。

而 RPC 则通过显式参数化查询（而不是依赖服务器参数化查询）来支持计划重复使用。为过程的第一次执行生成一个计划后，随后的执行将自动重复使用该计划，即使它们提供的参数值不同。与通过语言事件调用存储过程相比，使用 RPC 调用存储过程不仅节省了计划编译所需的执行时间和 CPU 资源，还增强了 SQL Server 内存资源的利用率，因为它避免了冗余执行计划所浪费的内存。

在执行动态 T-SQL 时，通常首选 sp_executesql 而不是 EXEC() 也出于同样的原因。Sp_executesql 的工作方式是：使用指定的查询创建一个存储过程，然后使用提供的参数调用它。与 EXEC() 不同，sp_executesql 提供了一个允许您参数化动态 T-SQL 并支持计划重复使用的机制。使用 sp_executesql 执行的动态查询比使用 EXEC() 的查询能够在更大程度上避免不必要的编译和资源消耗。

TDS

从客户端发送到 SQL Server 的 RPC、语言事件和其他类型的请求被格式化为称作表格数据流 (TDS) 的 SQL Server 特定数据格式。TDS 是 SQL Server 客户端和服务器之间使用的“语言”。对于它的确切格式将不作介绍，但是，如果客户端要与 SQL Server 进行通信，就必须使用 TDS。

目前，SQL Server 支持三种版本的 TDS：TDS 8.0（适用于 SQL 2000 客户端）、TDS 7.0（适用于 SQL Server 7.0 客户端）和 TDS 4.2（适用于 SQL Server 4.2、6.0 和 6.5 客户端）。完全支持所有 SQL Server 2000 功能的版本只有 TDS 8.0。其他版本保持向后兼容。

服务器端 Net-Library

在服务器端，客户端请求最初由 SQL Server 为侦听特定网络协议而建立的侦听器接收。这些侦听器由服务器上的网络库以及服务器端的 Net-Library（在它们与服务器之间提供管道）构成。您可以使用 SQL Server 网络实用程序配置服务器侦听的协议。SQL Server 与客户端支持同样范围的网络协议（处理群集的情况除外）。对于群集化的 SQL Server，只有 TCP/IP 和命名管道可用。

SQL Server 为侦听客户端请求所使用的每个网络协议设置一个线程，并使用 Windows 的 I/O 完成端口机制等待和有效处理请求。从网络接收到 TDS 数据包时，Net-Library 侦听器将其重新汇编为它们的原始客户端请求，并将这些请求传递到 SQL Server 的命令处理层，即开放式数据服务 (ODS)。

将结果返回到客户端

服务器在准备将特定客户端请求的结果返回时，将使用最初接收请求时所用的网络堆栈。它通过服务器端 Net-Library 将结果发送到相应的网络协议，随后这些结果将通过网络以 TDS 格式返回到客户端。

在客户端上，客户端 Net-Library 将从服务器接收的 TDS 数据包从 IPC 层重新汇编，并将其继续转发到初始化该请求的 API 或对象库。

小结

尽管涉及了所有组件，但 SQL Server 客户端与服务器之间的往返过程却相当快 - 特别是在使用内存 Net-Library 时，亚秒响应时间非常普遍。构建和调整您自己的 SQL Server 客户端应用程序时，以下几个与数据相关的问题值得注意：

· 如果应用程序与 SQL Server 运行在同一台计算机上，则建议您使用共享内存 Net-Library（如果尚未使用它）。基于共享内存 Net-Library 的连接通常比其他类型的连接快很多。在注意上述内容的同时，还应：始终全面测试解决方案并将它与其他可行方案进行对比，这样才能判断它是否确实更好或更快。事实胜于雄辩。

· 由于客户端在第一次连接时将指定给特定的 UMS 计划程序，并只有在断开连接后，才会摆脱该计划程序的控制，因此确保在应用程序与服务器建立的连接上均衡分配工作负荷非常重要。工作负荷不均衡可导致不必要的 CPU 争用并降低资源使用率。

· 在服务器上配置的默认网络数据包大小以及客户端在连接时指定的网络数据包大小将直接影响它们在服务器上所需的内存量和分配内存的池。对服务器进行扩展性和速度配置时，应记住这一点。还应记住，默认情况下，ADO.NET 应用程序的网络数据包大小比 ADO 应用程序的更大。

· 通常，在向服务器发送请求时，应首选 RPC 而非语言事件。为此，应在使用的 ADO 或 ADO.NET 对象中设置相应的属性。

· 执行动态 T-SQL 时，应在可能的情况下使用 sp_executesql 代替 EXEC()。唯一例外的情况是，当使用 EXEC() 的功能将查询片断连接而成的动态查询字符串的大小超过单个本地变量的存储大小时（这种情况非常少见）。

· 当遇到客户端问题，并且怀疑它可能和连接服务器时所用的对象库或 API 有关时，可以使用的一个故障排除技巧就是更改所用的客户端机制，这样可以将问题归结为特定的组件。例如，假设您升级 MDAC 并开始在 SQL Server 错误日志中看到 17805 错误，这表明客户端 ADO 应用程序发送的 TDS 数据包的格式不正确。您可能尝试让应用程序转为使用 ODBC 的 OLE DB 提供程序，如果您可以较为容易地做到这一点，应看看该问题是否与 SQLOLEDB 提供程序有一定的关系。相反，如果基于 ADO 的应用程序一直通过 ODBC 进行连接，则可以切换到 SQLOLEDB，看看这是否能解决问题，或至少帮助您缩小问题的范围。

· 同样，在对连接问题进行故障排除时，更改正在使用的 Net-Library 有时会有所帮助。如果使用 TCP/IP，命名管道也许值得一试。例如，如果 DHCP 服务器出现问题，并且没有有效的 IP 地址，则您将无法使用 TCP/IP 连接到 SQL Server。通过切换到命名管道，可以快速地将问题归结为 TCP/IP 特定的因素上。另一方面，如果在切换 Net Library 后仍存在同样的问题，则可以排除 Net-Library 方面的问题。问题的原因可能是服务器已关闭，或在您与服务器之间的某处网络基础设施无法正常工作。最后，还可以容易地更改应用程序使用的 Net-Library，而不必更改应用程序本身，这样就为您提供一个帮助缩小问题范围的工具。尽管从长远角度而言，使用某一特定 Net-Library 并不可行，但让客户端临时使用它可以帮助您缩小连接相关问题的范围。

SOA 为支撑业务敏捷性提供了新的思路和方法

SOA 监管(SOA Governance)是 SOA 实施中的一个重要话题，但是很多人都搞不清楚其含义。我采访过很多人，也阅读过一些资料，才基本弄明白。总的感觉是，如果 直白地去讲 SOA 监管的问题，必然引进大量的新术语，一般开发者实在不容易听懂。如果能够举一个例子，那么大家就容易理解得多。恰好昨天在书上看到一个真 实的故事，很形象地说明了 SOA

监管的意义。所以不妨跟大家分享一下。这个故事是关于 Sun 的，当然这类事情实际上曾经发生在很多大型公司里。

在 90 年代后期，Sun 推出了一系列产品，包括 Java、Solaris 等，他们希望能够尽可能地鼓励用户去使用这些产品，但是当时网速太慢，通过 Internet 下载几百兆的软件根本不现实，于是 Sun 在网站上推出一个电子商务服务，下面我们不妨称之为服务 A，你只要通过信用卡付 10-20 美刀快 递费用，就可以免费获赠 Sun 的超值产品光盘。被叫去编写这个电子商务服务的程序员当时隶属与内部 IT 部门，他写了一个在线服务，用来完成信用卡付账交易。当然，这是一个“子服务”，我们不妨称其为服务 Z，这个在线服务 Z 运行在内网上，采用了今天看来都不落后的体系结构——直接通过 HTTP 传输加密的 XML 消息。很快，服务 A 对用户见面了，并且工作得很好。

不久之后，这个程序员被调到了 Java 开发组。当时 Sun 的 Java 网站提供一个类似 MSDN 的 Java 产品光盘订阅服务，下面不妨称之为服务 B，这个服 务每季度向订阅者寄送最新的 Java 产品光盘。当然，订阅者也要通过信用卡付订阅费。碰巧这项工作又交给了这位程序员来完成。他当然不愿意重写那个很麻烦 的信用卡结帐服务 Z，既然原来的那个服务是通过 HTTP 暴露在内网里的，何不复用之?他就简单地复用了这个信用卡结帐服务 Z，完成了任务。这样，在 90 年 代后半期，这位程序员就率先实现了企业服务的复用。而十年后，服务的复用正是今天 SOA 追求的目标之一。

这样就形成了一个有趣的局面，即服务 A 中包含一个子服务 Z，而服务 B 又依赖于服务 Z，Z 实际上成为了一个公共服务，但是这个秘密只有那个程序员和少数几个人知道，Sun 的经理们对此懵然不知。

几年之后，这位程序员离开了 Sun，随着他的离去，这个秘密变得更加不为人知。

随着互联网的发展，人们已经习惯于从网上直接下载软件，服务 A 已经变得越来越过时了。于是终于有一天，Sun 的一个经理决定，关闭服务 A。结果意想不到的事情发生了，随着 A 的关闭，服务 Z 也被关闭了，这就导致服务 B 全面崩溃，所有的订阅者都无法付款了。

这就是一个缺乏监管的情况下产生的典型事故。在传统的企业 IT 架构里，当系统仅仅是部门级烟囱系统时，软件模块之间的关系简单，监管不是一个很突出的问 题。而当各部门系统进行整合时，如果采用 EAI/ETL 方案，则也不大有监管的问题。只有在实施 SOA 的时候，把传统的烟囱系统打散成为一个个可复用的服 务时，监管的问题就突出了。SOA 监管的意图，就是要让各种服务以清晰有条理的方式组合协作起来，并清晰地度量每一个服务的开销，评估每一个服务的开发和 维护所需的技术，确定当服务失效时采取的必要措施。总之，就是要把服务管起来，让它们有组织有纪律的共同工作。如果没有一个监管的制度和计划，那么就会出 现这样的局面：服务与服务之间有什么关系?不知道。服务之间彼此是否依赖?不知道。这两个服务的功能是否重复?不知道。这个服务是否冗余?不知道。开发维 护这个服务需要什么技能?不知道。当用户量增加时，维持这一服务的 QoS 所需的硬件消耗怎么变化?不知道。当服务崩溃时，谁来接替?往谁那里打电话?是否 有手工流程紧急应对?不知道!一大堆无法无天的服务以谁也想不到的方式攒在一起，任何一个点风吹草动都有可能会天下大乱。这就是缺乏监管的 SOA 将发生的 局面。这样的 SOA，与其说是一个系统，不如说是一团乱麻，一场灾难。

因此，SOA 监管对 SOA 来说，不是可选的，而是必须的，甚至是决定 SOA 实施成败的关键。

SOA 架构十大设计原则服务共享和约架构

日前国外网站报道介绍了面向服务架构(SOA)的基本原则，提出了公共接口与内部实现要有明确界限等原则。虽然这些原则并不是绝对的真理，但可作为一个应用开发参考。

一、明确的边界

通过跨越定义明确的边界进行显式消息传递，服务得以彼此交互。有时候，跨越服务边界可能要耗费很大的成本，这要视地理、信任或执行因素而定。边界是指服务的公共接口与

其内部专用实现之间的界线。服务的边界通过 WSDL 发布，可能包括说明特定服务之期望的声明。

二、服务共享和约和架构

服务交互应当只以服务的策略、架构和基于合约的行为为基础。服务的合约通常使用 WSDL 定义，而服务聚合的合约则可以使用 BPEL 定义(进而，对聚合的每个服务使用 WSDL)。服务使用者将依靠服务的合约来调用服务及与服务交互。鉴于这种依赖性，服务合约必须长期保持稳定。在利用 XML 架构 (xsd:any) 和 SOAP 处理模型(可选标头)的可扩展性的同时，合约的设计应尽可能明确。

三、策略驱动

尽管它往往被认为是最不为人所了解的原则，但对于实现灵活的 Web 服务，它或许是最有力的。单纯依靠 WSDL 无法交流某些业务交互要求。可以使用策略表达式将结构兼容性(交流的内容)与语义兼容性(如何交流消息或者将消息交流给谁)分隔开来。

四、自治

服务是独立进行部署、版本控制和管理的实体。开发人员应避免对服务边界之间的空间进行假设，因为此空间比边界本身更容易改变。

五、采用可传输的协议格式，而不是 API

通常,服务提供商基于某种传输协议(例如 HTTP)提供服务,而服务消费者只能通过另一种不同的协议(比如 MQ)通信。因此，也许需要在服务提供商与消费者之间建立一座异步起动同步运行的连接桥梁,超越 HTTP 和 Java Messaging Service 消息服务(JMS)等协议.从技术角度讲，Java Messaging Service 消息服务(JMS)并不是一种传输协议,而是一组供应商中立(vendor-neutral)的通信 APIs。

六、面向文档

消息被构造为“纯文本的”XML 文档(换句话说，数据的格式只对 XML 有意义)。 消息通常用于传输业务文档，比如购买订单、发票和提单。这种交互类型与同步消息排队系统的兼容性很好，比如 MQ Series、MSMQ、JMS、TIBCO、IMS 等等。

七、松耦合

服务之间要求最小的依赖性，只要求它们之间能够相互知晓。

八、符合标准

当通过 Web 的服务实现时，最原始的(基本的)面向服务的架构(SOA)的模型仅仅提供了很低程度上的关于可靠性、安全性以及事务管理的标准化机制。第二代的技术条件和框架，如 WS-ReliableMessaging 规范、 WS-Security 规范和 WS-Coordination 规范 (与 WS-AtomicTransaction 规范和 WS-BusinessActivity 规范相联系)，它们试图以工业标准的方式定位存在的缺陷。

九、独立软件供应商

向 SOA 的转变正在深刻改变了经济现实。客户们会期待更合理的费用以及不必重新进行投资就能改进业务的能力。因此，独立软件供应商没有选择，只能使自己的业务更加灵活，以期让自己的客户也变得同样灵活。于是，面向服务不仅是简单的在现有的、紧耦合的、复杂的、不灵活的以及非组件化的业务功能上添加基于标准的接口。更重要的是，为了兑现 SOA 的承诺，独立软件供应商必须改变他们构建、打包、销售、交付、管理和支持自身产品的方式。

十、元数据驱动

开发元数据本身并不是元数据驱动应用程序的本意。使用元数据来驱动服务在系统边界的传播是一个更为正确的方法。

SOA 架构理念企业应用 SOA 最常见的 5 种模式

随着 SOA 技术的不断完善，SOA 理念与技术实践开始日趋深入。从国内整体市场看，越来越多的政府机构和企业已经跨过了对 SOA 的价值及重要性的认知阶段，开始从概念普及、局部尝试准备进入到大规模应用阶段。但是，如何基于国内的 SOA 需求环境让 SOA 真正落地，如何在各行业构建满足 SOA 特征的应用成为目前我国软件企业与服务提供商面临的一大难题。

笔者认为，只有 SOA 的应用模式被深刻理解，SOA 应用工作才会有据可依，大力推进 SOA 在中国的成功应用才会有坚实的基础。本文基于国内各行业的实际业务类型，重点研究了 SOA 的应用模式，并按照不同的侧面对此进行了总结。

SOA 作为软基础设施

从软基础设施的角度，SOA 的应用可以分为利用信息资源目录梳理业务活动和业务对象的应用模式，以及建立业务主题库的应用模式两类。利用信息资源目录梳理业务活动和业务对象的应用模式用于梳理业务以支撑基于 SOA 的应用； 建立业务主题库框架的应用模式则主要是阐述如何建立业务领域的主题库，基于这种应用模式可以建立多层次、分布式应用系统的基础库。

信息资源梳理的目的在于方便部门间的资源共享和业务协同，因此宜将政务部门的组织结构、部门职责作为梳理的起点，把各部门的业务活动作为信息资源梳理的脉络，把业务对象和业务流程作为信息资源的关键点，把业务活动之间的关系作为寻找和判断信息资源如何共享和协同的依据。

对于政务部门，信息资源主要包括基础信息、结构化业务数据、非结构化业务数据和应用资源等。此分类只是资源的外在表现形式，而如何利用这些资源形成完整的服务则是一项需要研究的内容。通过分析可以建立起相对完善的资源梳理和服务目录，完成以下工作：

1. 业务活动梳理、编目及查询；

2. 业务活动之间的关系梳理、编目及查询；

3. 基础信息维护、编目及查询；

4. 结构化业务数据维护、编目及查询；

5. 非结构化业务数据维护、编目及查询；

6. 应用资源类维护、编目及查询。

完成上述过程的梳理和资源分类只是一个起点，还要利用 SOA 理念，将梳理的结果以服务的形式体现出来，才能最终为业务应用提供最直接的帮助。事实上，梳理的过程和基于梳理结果建立服务的过程也就是搭建政务应用基础设施的过程，有了基础设施就有了业务应用的完整地图，业务流、数据流就可以按照指定的方式运行。

资源共享应用模式

从软基础设施的角度对电子政务领域的信息资源目录进行了梳理，建立了以组织架构、职责、业务活动、业务对象、业务流程为内容的目录体系，并基于这个目录体系形成了服务体系，就可以依据这些软基础设施进一步构建具体的资源和服务。这些资源可以通过服务的模式对外共享，任何需要这些资源的机构和个人都能拿到所需要的资源。

资源的有效共享依赖于三个方面： 一个是资源本身的描述，另一个是资源本身的实际存储方式，最后是资源的提供方式。

资源本身的描述和逻辑集中有赖于基于元数据的资源描述，逻辑集中就是将资源的描述

以目录的形式进行统一存储； 资源的物理存储方式依赖应用构建前期对数据的规划，此层的变动只会影响资源的物理层面特性，并不影响其服务的特性，因此原有的对应用层限制最大的数据层，通过目录的统一服务变得非常灵活而有弹性； 最后，资源的提供方式则是基于前两个方面的服务方案，资源共享以服务的形式体现。

业务协同应用模式

不同机构的业务办理都有可能依赖于其他业务，而业务本身的办理又通常需要资源的支撑，资源本身的负责方或许是本单位，也可能是其他单位，信息资源的共享应用模式解决了第二个问题，而第一个问题也通过对业务的梳理为业务协同建立了完整的指导。如何实现这些业务的协同是 SOA 在这种应用模式下的重点。在这种应用模式下，完成业务协同包括三个步骤：

第一步： 业务处理服务

业务处理服务源于对组织内或组织间业务活动的分析，组织内的业务处理服务可以直接基于业务活动抽象的用例来构造； 组织间的业务活动一部分来自于业务活动的分析，另一部分来自于资源共享的需求，近而依据这部分需求建立起共享的服务。

第二步： 业务流程服务

业务流程服务源于组织内各部门间或组织间的业务关系的分析，通过建立业务的前置关系、后置关系从而形成业务流程，依据业务活动间的关系建立起对外提供的业务服务。

第三步： 服务查询检索

服务查询检索主要是供外部用户明确了解组织提供了哪些服务、具体的服务内容是什么以及如何获取和使用这些服务。

服务查询检索依据信息资源目录，信息资源目录清晰梳理了客户的业务，但如何提供这些业务，则需要通过服务去实现。通过服务定义和服务描述，建立了关于服务的完整描述，使用者可以基于这些描述的任一方面对服务进行检索。

最后通过服务检索查询的功能开发定义明确的交互界面，用户可以通过交互界面查询定位所需的服务。

不同服务渠道的应用模式

服务的灵活性和可扩展性是 SOA 的主要特性之一，电子政务的一个重要特性是强调服务，因此整合不同服务渠道也是重点之一。

SOA 在应用与业务之间加入一个服务层，解决了原有的系统建设通常不会建立服务层完成系统间的调用，而是直接调用下层其他应用或者采用数据共享的方式，从而避免直接访问下层其他应用。另外，在大多数机构中，存在不同的应用和技术共存，由于这些应用提供的功能都是特定的，要在应用间共享信息最好的解决方案是转向一种面向服务的架构和 Web 服务，即在业务层之上加入一个服务层。

当数量众多的业务应用需要使用 Web 服务技术集成在一起的时候，可以进一步采用企业服务总线(ESB)的架构来管理这些可复用的应用组件，从而可以实现更加清晰地管理所有政务系统中所包含的可复用信息资产。

另外，SOA 还有第五种应用模式是基于虚拟数据中心的模式，也就是忽略数据在不同节点的部署而集中提供服务。如果要在单节点上提供虚拟数据中心，可以建立非分布式目录中心用于提供虚拟中心服务； 如果在多节点上建立虚拟数据中心，即跨节点的虚拟中心，需要建立分布式目录中心用于提供虚拟中心服务。

总之，通过 SOA 应用模式分类体系的研究，可以更好地帮助用户理解 SOA 的应用类型，并结合 SOA 架构的优势，确定业务下一步建设的方向。同时，指出传统软件开发方式的解决方案以及带来的局限性，明确 SOA 在这些方向的应用前景，并为基于 SOA 解决各种不同类型的问题提供独立于各种应用、领域、平台与标准的解决方案，从而真正起到指导电子政务各类应用实施的作用。

SOA 方法学和其他方法学的比较

广义上讲，SOA 方法学贯穿于 IT 生命周期的各个阶段和各个方面：IT 系统项目的规划，系统分析和设计，系统的实施，系统的部署和维护，以及整个过程中的监控和管理等。从实践的角度说，已经出现如下 SOA 方法学。

(1)面向服务的分析和设计(SOAD)。以服务为中心，根据业务需求发现服务、描述服务，并设计服务的实现。

(2)面向服务的开发过程。结合现有开发过程，规划以服务为中心的开发过程中的角色、职责、活动和工件。

(3)SOA 的成熟度分析和迁移路线图。以服务为中心，分析现有或目标系统的成熟度，并设计从现有成熟度迁移到目标成熟度的路线图。

(4)SOA 监管。设计组织和流程，确保 SOA 的设计原则在 IT 生命周期中得以贯彻，管理服务生命周期中的各种迁移的合理性等。

本章对 SOA 方法学的阐述主要集中在面向服务的分析和设计。首先介绍 SOA 方法学和主要的几种方法学的区别和联系，其次以 IBM 的 SOMA(Service Oriented Modeling and Architecture，面向服务的建模与架构)为例，介绍 SOA 分析和设计中的主要内容和方法。

1、SOA 方法学和其他方法学的比较

与 SOA 的设计原则类似，SOA 方法学并不是全新的方法学，它是现有方法学的继承和发展。一方面，原有的方法学并不能解决由于服务概念的引入带来的问题，如怎样发现服务，怎样定义服务;另一方面，服务是一个水平的概念，而不是一个垂直的概念，在服务分析和设计的过程中，需要处理服务和现有方法学产物的关系，如业务流程和服务，企业架构和 SOA，服务和对象等。因此服务的分析和设计最主要的职责在于发现服务、定义服务和实现服务，并指导如何和其他方法学结合完成这些职责。

流程建模(BPM)用于业务领域的分析和设计，如业务流程的定义、业务数据的定义等;企业架构(EA)和方案架构(SA)侧重在架构领域的分析和设计，如根据业务需求确定目前目标业务系统和 IT 系统，根据目标系统需求设计主要架构元素和它们之间的关系;面向对象的分析和设计(OOAD)则贯穿分析、设计和开发三个阶段，它主要分析细粒度的业务需求，如用例，分析和设计实现这些需求的类和对象，以及它们之间的关系。

1.BPM 和 SOA

业务流程建模是一个相当零散的领域，存在各种各样的方法和技术，有效的方法可以帮助企业对业务进行合理的划分，从而求得业务层面的灵活性。有些方法则侧重于流程建模本身，例如如何确定和定义业务流程中的业务活动、业务数据、业务规则、业务指标和业务事件等，但是 BPM 并不会帮助我们去发现和定义服务。从 SOA 的方法学来看，各种 BPM 的结果是面向服务的分析和设计的重要输入，如业务组件、业务流程和业务目标是服务发现的重要依据，而业务指标、业务数据、业务规则等是服务暴露的分析的重要依据。

2.EA 和 SOA

尽管和 BPM 一样，EA 是一个零散的领域，但是当前的 EA 主要侧重于定义跨越业务单元边界的系统框架，企业范围内系统的主要构成元素，这些元素间的关系，以及将这些元素有机组合在一起的参考架构。但是，各种 EA 技术都缺乏业务领域的蓝图指导企业架构的设计。从 SOA 方法学来看，一方面，面向服务的分析和设计通过和 BPM 结合将业务分解为各种类型的服务，可以作为企业业务的蓝图指导企业架构的设计;另一方面，企业架构设计的结果，如参考架构，又是服务实现的重要依据。

3.OOAD 和 SOA

面向对象的分析和设计告诉我们使用 Use Case 捕获需求，并设计类、对象及对象间交

互来满足 Use Case 定义的需求。但是面向对象的分析和设计往往只是局限在单个应用内部，它不会缺乏业务蓝图和企业架构蓝图的指导。从 SOA 方法学看，在原理层面上，OOAD 中的很多设计原则，如抽象、隔离关注等被 SOA 继承和发扬，并应用于服务的定义和实现中。而在操作层面上，服务模型为 OOAD 进行类和对象设计提供了业务蓝图和企业架构蓝图，与此同时，Use Case 作为对业务流程的补充说明被用于服务的发现和定义中。

Silverlight 应用程序的.NET 项目结构

今天先让我们从 Silverlight 应用程序的.NET 项目结构开始，看一下这样的一个项目中包含了些什么文件，这些文件分别是用来做什么的。了解这些内容有助于我们将来根据自己的需要制作更为复杂的项目内容。

在开始介绍之前，请务必安装 Silverlight 1.1 Alpha Refresh。

无论是 Silverlight 1.0 还是 1.1 版本，一个 Silverlight 项目总是包含以下几种文件：

1、嵌入 Silverlight 应用程序的 html 文件。

2、用来执行 Silverlight 应用程序载入工作的 JavaScript 文件。

3、定义和描述应用程序界面的 XAML 文件。

如果是.NET 的项目，界面描述中所定义的新类型都包含在程序集(assembly)文件中。

我们可以用 Microsoft Expression Blend 2 (目前最新的预览版本是 September Preview)或者 Visual Studio 2008 Beta2 (需安装好 Silverlight Tool Alpha)来创建一个新.NET 的项目。创建成功后，我们就可以看到一个 Silverlight 的.NET 项目包含了如下的目录结构：

References 目录

查看该目录，我们可以看到其中包含了许多(.dll)文件，这些文件定义了项目中所引用的各托管类型。如果我们要使用其他自定义的类型，就可能需要手动添加包含该类型定义的.dll 文件。若 Silverlight 插件中没有包含项目所引用的某些程序集文件，则应用程序用户需要下载这些程序集文件到本地。

Page.xaml

项目默认的主界面的 XAML 描述文件。

Page.xaml.cs

每个 xaml 文件之后都有一个对应的.NET 语言文件。由于我们创建的是 C#项目，所以其对应的文件为.cs 文件。这个文件对.xaml 文件中的所引用的.NET 类型做了定义。通过编译之后，项目会生成程序集文件，供.xaml 文件引用，且该程序集文件会随 Silverlight 应用程序一起下载到用户本地。

Silverlight.js

该文件包含了运行 Silverlight 应用程序之前所必需的各种操作，主要检查用户是否安装了符合版本需求的 Silverlight。如果没有安装，则 Silverlight 应用程序所在的页面将会显示图标，提示用户先安装插件。这里我们不建议 Silverlight 的开发设计人员改变该.js 文件的内容。

TestPage.html(Blend 2 中创建的文件名为 Default.html)

该文件为项目的默认主页，文件中引入了 Silverlight 的脚本文件来载入 Silverlight 应用程序。

TestPage.html.js(Blend 2 中的文件名为 Default_html.js)

该文件中定义了 TestPage.html 中调用的 CreatSilverlight()方法来载入 Silverlight 应用程序。

让我们再具体的看一看其中一些文件所包含的内容：

TestPage.html

以下是引用片段：

```
<html>
<head>
    <title>Silverlight Project Test Page </title>
    <script type="text/javascript" src="Silverlight.js"></script>
    <script type="text/javascript" src="TestPage.html.js"></script>
    <style type="text/css">
    <!--设置应用程序在 html 页面中显示的大小，也可以用百分比表示，设置为 100%则应用程序为自适应大小-->
        .silverlightHost { width: 640px; height: 480px; }
    </style>
</head>
<body>
    <!--以下的 ID 用来标示 DIV，尤其当页面总有多个 Silverlight 实例时，这个 ID 就成为了标示每个 Silverlight 应用程序的重要标志-->
    <div id="SilverlightControlHost" class="silverlightHost" >
        <script type="text/javascript">
            createSilverlight();
        </script>
    </div>
</body>
</html>
```

TestPage.html.js

以下是引用片段：

```
//creatSilverlight 函数用来执行载入 Silverlight 应用程序的操作及相关属性
function createSilverlight()
{
 Silverlight.createObjectEx({
  //指定了初始化载入的应用程序页面
  source: "Page.xaml",
  parentElement: document.getElementById("SilverlightControlHost"),
  id: "SilverlightControl",
  properties: {
   width: "100%",
   height: "100%",
   version: "1.1",
   enableHtmlAccess: "true"
  },
  events: {}
 });

 //  下面的函数是默认将键盘焦点集中在 Silverlight 应用程序上
    document.body.onload = function() {
     var silverlightControl = document.getElementById('SilverlightControl');
     if (silverlightControl)
     silverlightControl.focus();
    }
}
```

Page.xaml

以下是引用片段：

```
<!--Canvas 是一个包含了各种控件和元素的容器，每个 Silverlight 应用程序都有一个 XAML 的根文件，每个 XAML 文件的根都是一个 Canvas 容器，且只能有一个作为根元素的 Canvas 容器-->
<Canvas x:Name="parentCanvas"
      xmlns="http://schemas.microsoft.com/client/2007"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      <!--指定当 Load 事件发生时所要载入的方法-->
      Loaded="Page_Loaded"
      <!--指定该.xaml 文件中的托管类进入点，以及所涉及到的引用（程序集）的位置，一般项目编译生成的程序集文件都位于项目中的 ClientBin 文件夹中-->
      x:Class="SilverlightProjectStructure.Page;assembly=ClientBin/SilverlightProjectStructure.dll"
      Width="640"
      Height="480"
      Background="White"
      >
</Canvas>
```

注意：由于 Silverlight 1.1 还处于 Alpha 版本，.NET 语言中关于 Silverlight 的一些 API 可能会与将来的正式版本中有些出入，但是大多数基本的东西还是不变的。

SharePoint 工作流开发点滴(一些概念)

模板(Template),关联(Association)和实例(Instance)

模板:部署到站点集中的工作流功能(Feature),用来描述该功能所包含的程序集和表单等信息.

关联:将工作流模板与列表(List)或者内容类型(Content Type)联系起来,并向工作流提供初始值或参数.对应的表单叫做 Association.

实例:在列表或内容类型项上启动的工作流.对应的表单叫做 Initiation.

也就是说,实例是基于关联的,而关联又是基于模板的.一个列表或者内容类型可以拥有许多来自相同工作流模板的关联,但同一时刻同一关联只能启动一个工作流实例.

钝化(Dehydrated)

钝化指将工作流序列化(Serialized)并保存在数据库中.钝化后的工作流将从内存中清除.当工作流等待的事件发生时,工作流将反序列化(Deserialized)并被唤醒,然后继续它的流程.

事务性动作(Transacted Action)和批处理动作(Batched Action)直到工作流钝化后才会提交.例如,CreateTask 并没有马上创建任务,而是等到 OnTaskChanged 将工作流钝化之后才创建任务.所以在创建任务之后马上访问任务是错误的.

Method 和 Event Handle

Method 用来执行动作,例如 CreateTask 就是一个 Method 活动.

Event Handle 用来将工作流钝化,然后等待一定的事件被触发后唤醒工作流,例如 OnTaskChanged 就是一个 Event Handle 活动.

Method 的 Method Invoking 在 Method 要执行的动作之前执行,而 Event Handle 的 Method Invoking 却在 Event Handle 的事件触发之后执行.

Correlation Token

将若干相关联的活动映射到同一集合的标识符,例如给 CreateTask,OnTaskChanged 和 CompleteTask 指定相同的 Correlation Token,则这几个活动被关联到同一个任务.

在 SharePoint Workflow Actions 中,Correlation Token 大致按照下表来分组指定:

工作流	任务	修改
OnWorkflowActivated OnWorkflowItemChanged OhWorkflowItemDeleted SetState SendEmail UpdateAllTasks	CreateTask CreateTaskWithContentType UpdateTask DeleteTask CompleteTask RollbackTask OnTaskChanged OnTaskDeleted OnTaskCreated	EnableWorkflowModification OnWorkflowModified

InfoPath 表单

表单类型	宿主 ASPX	需要执行的动作	Event Handle	接收数据的属性
Association	CstWrkflIP.aspx	建立工作流模板和列表(或者内容类型)之间的关联	无	无
Initiation	IniWrkflIP.aspx	启动工作流实例	OnWorkflow Activated	SPWorkflowActivationProperty.InitiationData
Task	WrkTaskIP.aspx	更改	OnTaskChanged	OnTaskChanged.AfterProperties

Modification	ModWrkflIP.aspx	了任务 修改工作流	OnWorkflowMosified	OnWorkflowMosified.ContextData

Association 表单不接受数据,也不会存在相应的 Event Handle,因为此时工作流实例还没有启动,便没有钝化和唤醒一说,不过 Associatin 中的数据却可以在 SPWorkflowActivationProperty.AssociationData 中获取.

SharePoint 工作流开发点滴(我的任务不给你看)

一直以来都被 MOSS 的工作流权限问题所困扰.

我们虽然将任务分配给了某人,但事实上,所有在任务列表中有编辑权限的用户都可以编辑该任务.

而我们并不希望用户能够看到不属于自己的任务.

我曾经尝试过采用以下两种方法解决这一问题,未果.

1.更改任务列表的视图.

通常我们会把工作流任务分配给某人或者某组,所以我的想法是将任务列表的"我的任务"视图和"按我的用户组"视图合并为一个新的"所有我的任务"视图;

设置"所有我的任务"视图为默认视图;

删除其余视图;

取消用户创建视图的权限;

看起来不错,但是我连第一步都没能实现.

2.定制一个 EventHandler

记得园子里的一个朋友写过一篇通过开发 EventHandler 来控制工作流权限的文章.

思路是为任务列表定制一个 EventHandler,使任务列表在增加项目时可以自动更改该项目的权限.

通过定制 EventHandler 确实可以实现任务的权限分配,但是会引发另外的问题,譬如说,我的工作流中用 OnTaskChanged 活动来捕捉任务的变更.但是因为在任务创建之后 EventHandler 马上对其进行权限修改,所以导致工作流会发生一个小小的错误.

不过奇怪的是,虽然在文档库中显示该工作流发生了错误,但事实上流转还是正常的.

我一直固执的认为任务列表的权限应该是在 MOSS 中配置,而不应该在工作流内部做判断,后来,我终于发现我错了.

我们使用 CreateTask 活动来创建任务,CreateTask 活动有一个 HybridDictionary 类型的属性叫做 SpecialPermissions.

这个属性表示该任务的"特别权限",如果指定了"特别权限",那么创建的任务就不再继承任务列表的权限了.所以我们可以在创建任务之前为其指定这个"特别权限".

1.新建一个全局的 Contact 对象用来存储分配对象
Contactassignee=default(Contact);

2.在 onWorkflowActivated 的 Invoked 事件中从初始化表单中获取 assignee 对象
assignee=Contact.FromName(init.contact[0].DisplayName,workflowProperties.Web);

3.在属性面板中将 createTask 的 SpecialPermissions 属性绑定到新的 specialPermissions 对象 4.编写 createTask 的 MethodInvoking 方法

```
privatevoidcreateTask1_MethodInvoking(objectsender,EventArgse)
{
taskId=Guid.NewGuid();
taskProperties.AssignedTo=assignee.LoginName;
//判断分配对象是否用户
if(assignee.IsSPUser)
{
//为其添加"参与讨论"的权限.
specialPermissions.Add(assignee.LoginName,SPRoleType.Contributor);
}
//判断分配对象是否用户组
if(assignee.IsCollection)
{
try
{
SPGroupgroup=workflowProperties.Web.Groups[assignee.DisplayName];
//为组内每一个用户添加"参与讨论"的权限.
foreach(SPUseruseringroup.Users)
{
specialPermissions.Add(user.LoginName,SPRoleType.Contributor);
}
}
catch{}
}
}
```

因为默认的"参与讨论"权限就可以编辑工作流任务,所以为任务指定了"特别权限"之后,该任务将对不在 SpecialPermissions 中的用户不可见,这正是我们所需要的.

SharePoint 工作流开发点滴(添加外部工具,方便工作流开发)

在利用 ECM Starter kit Beta2 中的模板开发工作流的时候经常会切换到其它窗口,比如要为 InfoPath 表单模板生成类文件,要安装工作流等等.

可不可以把这些工作都集成到 Visual Studio.net 2005 IDE 中呢?答案是肯定的,利用 VS.net 2005 IDE 的外部工具功能,就可以轻松实现.

方法

点击工具菜单下的外部工具,将会弹出来一个非常容易理解的外部工具对话框,在这里可以进行外部工具的添加,删除和修改以及排序的工作.

下面来说明一下怎样利用外部工具更方便的开发工作流吧.

生成 GUID

在配置 Feature.xml 和 Workflow.xml 的时候,需要为这两个文件配置不同的 GUID ,VS.net 自带了一个生成 GUID 的小工具 GUIDGEN 并默认的把它添加到外部工具中,如果你像我一样发现并没有这个外部工具的话,打开外部工具对话框,填入以下信息:

标题	生成 GUID
命令	C:Program FilesMicrosoft Visual Studio 8Common7ToolsGUIDGEN.exe
初始目录	C:Program FilesMicrosoft Visual Studio 8Common7Tools

*如果你再次像我一样在初始目录中没有找到GUIDGEN.exe 的话,你可以到微软的下载中心去下载(为什么我如此不幸?).

生成表单模板类文件

如果你的工作流中用到了 InfoPath 表单模板,而且需要为此表单模板生成类文件的话(比

如要获取关联表单中的值),你可以添加一个这样的外部工具:

标题	生成表单模板类文件
命令	C:Program FilesMicrosoft Visual Studio 8SDKv2.0Binxsd.exe
参数	"myschema.xsd" /c /o:$(ProjectDir)
初始目录	$(ProjectDir)
使用输出窗口	√
提示输入参数	√

使用时,会弹出一个对话框,在参数栏的"之间填入包含模板架构文件(.xsd)的目录路径即可.

如:"c:tempdirmyschema.xsd" /c /o:$(ProjectDir)

(其中粗体字就是自己添加的目录路径)

安装工作流

标题	安装工作流
命令	$(ProjectDir)Install.bat
初始目录	$(ProjectDir)
使用输出窗口	√

只要配置好了 Install.bat 文件,直接调用此外部工具即可完成安装,而且可以在输出窗口中查看进度信息.

外部工具还有很多很有帮助的应用场景,就看你能不能想得出了.

SharePoint 工作流开发点滴(启动时失败的查错方法)

很多朋友在使用 Visual Studio 开发工作流时都遇到过工作流"启动时失败"的错误,我把我遇到这种情况时的查错方法和大家分享一下,希望对大家有所帮助,也希望大家可以告诉我更好的方法.

首先在 OnWorkflowActivated 方法中设置断点来调试,检查断点是否可以成功暂停,如果可以,则可以再检查一下 OnWorkflowActivated 中的逻辑代码.

如果断点处不停止就出现了"启动时失败"的错误,也就是说错误发生在工作流激活之前,这种情况就可以排除代码的错误,检查 infopath 表单和 workflow.xml 即可.

infopath 表单的配置较为麻烦,甚至连命名都是有规则的,需要多加注意.

另外,如果工作流包含关联表单(Association),那么初始化表单(Instantiation)中必须包含关联表单中定义的域.

如果还是找不出错误的来源,那么,记住工作流启动时的时间,检查 SharePoint 日志吧.日志的目录是:$:Program FilesCommon FilesMicrosoft Sharedweb server extensions12LOGS

最后再引用一句 Kaneboy 的签名:

"玩 SharePoint 就像是魔术,你不知道它为什么工作,也不知道它为什么不工作."

SharePoint 工作流开发点滴(工作流中的自定义类与内部错误)

最近在开发 SharePoint 工作流总是发生一个错误 :工作流开始之后便显示"已完成"或者开始之后报错"内部错误".

查看当时的日志,发现下面的段落:

02/06/2007 10:31:03.92 w3wp.exe (0x0758) 0x0F3C Windows SharePoint Services Workflow Infrastructure 72eo Unexpected DehydrateInstance: System.Runtime.Serialization.SerializationException: 在分析完成之前

就遇到流结尾。

在 System.Runtime.Serialization.Formatters.Binary.__BinaryParser.Run()

在 System.Runtime.Serialization.Formatters.Binary.ObjectReader.Deserialize(HeaderHandler handler, __BinaryParser serParser, Boolean fCheck, Boolean isCrossAppDomain, IMethodCallMessage methodCallMessage)

在 System.Runtime.Serialization.Formatters.Binary.BinaryFormatter.Deserialize(Stream serializationStream, HeaderHandler handler, Boolean fCheck, Boolean isCrossAppDomain, IMethodCallMessage methodCallMessage)

在 System.Runtime.Serialization.Formatters.Binary.BinaryFormatter.Deserialize(Stream serializationStream)

在 System.Workflow.ComponentModel.Activity.Load(Stream stream, Activity outerActivity, IFormatter formatter)...

02/06/2007 10:31:03.92* w3wp.exe (0x0758)

0x0F3C Windows SharePoint Services Workflow Infrastructure 72eo Unexpected ...

在 System.Workflow.ComponentModel.Activity.Load(Stream stream, Activity outerActivity)

在 System.Workflow.Runtime.Hosting.WorkflowPersistenceService.RestoreFromDefaultSerializedForm(Byte[] activityBytes, Activity outerActivity)

在 Microsoft.SharePoint.Workflow.SPWinOePersistenceService.LoadWorkflowInstanceState(Guid instanceId)

在 System.Workflow.Runtime.WorkflowRuntime.InitializeExecutor(Guid instanceId, CreationContext context, WorkflowExecutor executor, WorkflowInstance workflowInstance)

在 System.Workflow.Runtime.WorkflowRuntime.Load(Guid key, CreationContext context, WorkflowInstance workflowInstance) 在 System.Workflow.Runtime.WorkflowRuntime.GetWorkflow(Guid instanceId)

在 Microsoft.SharePoint.Workflow.SPWinOeHostServices.DehydrateInstance(SPWorkflowInstance wo...

02/06/2007 10:31:03.92* w3wp.exe (0x0758)

0x0F3C Windows SharePoint Services Workflow Infrastructure 72eo Unexpected ...rkflow)

02/06/2007 10:31:03.93 w3wp.exe (0x0758)

0x0F3C Windows SharePoint Services Workflow Infrastructure 88xr Unexpected WinWF Internal Error, terminating workflow Id# 472dae03-5465-4f04-876f-d4cc4caa902a

看里边最长的一段中文描述:"在分析完成之前就遇到流结尾",如果是 SharePoint 英文版,这段错误信息应该是"End of Stream encountered before parsing was completed".

也就是说 Workflow Runtime 根本就没有完整的分析完整个流程.

再看这句中文之前的英文:"DehydrateInstance: System.Runtime.Serialization.SerializationException".

原来工作流是在钝化实例的时候发生了序列化异常.

回想一下工作流的持久性,Workflow Runtime 会把空闲的工作流数据序列化为 XML 形式,然后把工作流实例从内存中清除,等到需要的时候再将其反序列化加载到内存.

会不会是因为我在工作流项目中添加了自定义类,而这个类又不支持序列化,所以导致工作流序列化失败?

在工作流中使用 InfoPath Initiation(或者 Association)表单时需要为其生成一个类,观察这个类,发现这个用 XSD 生成的类有如下特性来修饰:

[System.CodeDom.Compiler.GeneratedCodeAttribute("xsd", "2.0.50727.42")]

[System.SerializableAttribute()]

[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true)]

[System.Xml.Serialization.XmlRootAttribute(Namespace="http://schemas.microsoft.com/office/infopath/2003/myXSD/2007-01-30T13:00:28", IsNullable=false)]

下面来逐行分析一下:
[System.CodeDom.Compiler.GeneratedCodeAttribute("xsd", "2.0.50727.42")]
表示这个类是由 XSD 工具生成的.
[System.SerializableAttribute()]
表示这个类可以被序列化,我想关键就在这里.
[System.Diagnostics.DebuggerStepThroughAttribute()]
表示调试器会自动忽略被修饰的类内部的断点
[System.ComponentModel.DesignerCategoryAttribute("code")]
表示设计器的类别是"code"
[System.Xml.Serialization.XmlTypeAttribute(AnonymousType=true)]
表示系列化时生成的 XSD 架构是匿名类型

[System.Xml.Serialization.XmlRootAttribute(Namespace="http://schemas.microsoft.com/office/infopath/2003/myXSD/2007-01-30T13:00:28", IsNullable=false)]
设置序列化时根元素的命名空间
删去或修改我们不需要的特性(比如第一项),将其插入到我们的自定义类中,工作流就正常了.

SharePoint 工作流开发点滴(HelloWorldSequential 的注意事项)

HelloWorldSequential 是一个入门级的 SharePoint 工作流,基本上照着 MSDN 的教程一步一步来就可以了,我写这篇文字的主要目的是将其中一些重要的步骤列举出来,这些步骤的错误处理将直接导致工作流失败.
准备
HelloWorldSequential 的开发环境如下，请确保您的开发环境与此相同或相兼容:
已部署好的可用的 SharePoint Server 2007
.Net Framework 3.0（下载）
Visual Studio 2005 Extensions for Windows Workflow Foundation （下载）
ECM starter kit for Visual Studio 2005（下载）
Office InfoPath 2007 RTM 中文版
安装 ECM starter kit 之后如果出现项目模板丢失的情况请参考我的另外一篇文章《Visual Studio.net 2005 新建项目对话框中项目模版消失的解决方案》
添加 SharePoint Workflow Actions 到工具箱时选择 Microsoft.SharePoint.WorkflowActions 命名空间下的控件.
Visual Studio 项目
项目模板是 SharePoint Server 下的 SharePoint Sequential Workflow Library.
添加并配置工作流活动时请注意所有活动都有一个 Correlation Token 属性,值得注意的是工作流本身(Workflow),任务(Task)和修改(Modification)需要不同的 Correlation Token.
创建强命名的程序集,因为您的程序集是要加入 GAC 的.
InfoPath 表单
创建空白表单模板时要勾选仅启用浏览器兼容性功能.
在表单选项的安全和信任中,将表单的信任级别设置为域或者完全信任.

用于设置接收参数的 xml 文件中,接受参数总是以”ows_”开头.

修改组的名称,组的名称默认是 MyFields,如果您想要为此表单生成类文件,那么类的名字就是组的名字,所以建议修改默认的名称(尤其当您的工作流项目中包含多张需要生成类文件的表单时).

为简单起见,将表单发布到项目文件夹内,否则在部署时需要特别注意.

发布表单时将可访问路径留空,否则安装工作流时会失败

部署

Feature.xml 和 Workflow.xml 可以用插入代码段(Snippet)的方法生成内容,如果您发现您没有相关的代码段,请参照我的另外一篇文章《Visual Studio 2005 中代码段丢失的解决方案》.

Feature 的 ID 和 Workflow 的 ID 是两个不同的 GUID.

Install.bat 文件中, 请用可以作为文件夹名称的字符串来替换 MyFeature,最好不要包含空格,否则需要手动将 Install.bat 文件中的路径前后加引号.

调试

更改了 Feature.xml,Workflow.xml 和表单文件之后需要重新安装工作流并重启 IIS.

如果只是更改了程序集,只需要用新的程序集替换旧的,然后重启 IIS 就可以了.

ScottAmbler 谈如何编好的软件模型

我们期待自己成为一个优秀的软件模型设计者，但是，要怎样做，又从哪里开始呢？将下列原则应用到你的软件工程中，你会获得立杆见影的成果。

1. 人远比技术重要

你开发软件是为了供别人使用，没有人使用的软件只是没有意义的数据的集合而已。许多在软件方面很有成就的行家在他们事业的初期却表现平平，因为他
们那时侯将主要精力都集中在技术上。显然，构件（components），EJB（Enterprise Java Beans）和代理（agent）是很有趣的东西。但是对于用户来说，如果你设计的软件很难使用或者不能满足他们的需求，后台用再好的技术也于事无补。多 花点时间到软件需求和设计一个使用户能很容易理解的界面上。

2. 理解你要实现的东西

好的软件设计人员把大多数时间花费在建立系统模型上，偶尔写一些源代码，但那只不过是为了验证设计过程中所遇到的问题。这将使他们的设计方案更加可行。

3. 谦虚是必须的品格

你不可能知道一切，你甚至要很努力才能获得足够用的知识。软件开发是一项复杂而艰巨的工作，因为软件开发所用到的工具和技术是在不断更新的。而且，一个人也不可能了解软件开发的所有过程。在日常生活中你每天接触到的新鲜事物可能不会太多。但是对于从事软件开发的人来说，每天可以学习很多新东西(如果愿意的话)。

4. 需求就是需求

如果你没有任何需求，你就不要动手开发任何软件。成功的软件取决于时间（在用户要求的时间内完成）、预算和是否满足用户的需求。如果你不能确切知道用户需要的是-什么，或者软件的需求定义，那么你的工程注定会失败。

5. 需求其实很少改变，改变的是你对需求的理解

Object ToolSmiths 公司的 Doug 。Smith 常喜欢说："分析是一门科学，设计是一门艺术"。他的意思是说在众多的"正确"分析模型中只存在一个最"正确"分析模型可以完全满足解决某个具体问题的需要（我理解的意思是需求分析需要一丝不苟、精确的完成,而设计的时候反而可以发挥创造力和想象力 - 译者注）。 如果需求经常改动，很可能是你没有作好需求分析，并不是需求真的改变了。 你可以抱怨用户不能告诉你他们想得到什么，但是不要忘

记，收集需求信息是你工作。 你可以说是新来的开发人员把事情搞得一团糟，但是，你应该确定在工程的第一天就告诉他们应该做什么和怎样去做。如果你觉得公司不让你与用户充分接触，那只能说明公司的管理层并不是真正支持你的项目。你可以抱怨公司有关软件工程的管理制度不合理，但你必须了解大多同行公司是怎么做的。你可以借口说你们的竞争对手的成功是因为他们有了一个新的理念，但是为什么你没先想到呢？需求真正改变的情况很少，但是没有做好需求分析工作的理由却很多。

6. 经常阅读

在这个每日都在发生变化的产业中，你不可能在已取得的成就上陶醉太久。每个月至少读 2、3 本专业杂志或者 1 本专业书籍。保持不落伍需要付出很多的时间和金钱，但会使你成为一个很有实力的竞争者。

7. 降低软件模块间的耦合度

高耦合度的系统是很难维护的。一处的修改引起另一处甚至更多处的变动。你可以通过以下方法降低程序的耦合度：隐藏实现细节，强制构件接口定义，不使用公用数据结构，不让应用程序直接操作数据库（我的经验法则是：当应用程序员在写 S-QL 代码的时候，你的程序的耦合度就已经很高了）。耦合度低的软件可以很容易被重用、维护和扩充。

8. 提高软件的内聚性

如果一个软件的模块只实现一个功能，那么该模块具有高内聚性。高内聚性的软件更容易维护和改进。判断一个模块是否有高的内聚性，看一看你是否能够用一个简单的句子描述它的功能就行了。如果你用了一段话或者你需要使用类似"和"、"或"等连词，则说明你需要-将该模块细化。 只有高内聚性的模块才可能被重用。

9. 考虑软件的移植性

移植是软件开发中一项具体而又实际的工作，不要相信某些软件工具的广告宣传（比如 java 的宣传口号 write once run many？译者注）。即使仅仅对软件进行常规升级，也要把这看得和向另一个操作系统或数据库移植一样重要。 记得从 16 位 Windows 移植到 32 位 windows 的"乐趣"吗？当你使用了某个操作系统的特性，如它的进程间通信(IPC)策略，或用某数据库专有语言写了存储过程。你的软件和那个特定的产品结合度就已经很高了。好的软件设计者把那些特有的实现细节打包隐藏起来，所以，当那些特性该变的时候，你的仅仅需要更新那个包就可以了。

10. 接受变化

这是一句老话了：唯一不变的只有变化。你应该将所有系统将可能发生的变化以及潜在需求记录下来,以便将来能够实现（参见"Architecting for Change"， Thinking Objectively, May 1999）通过在建模期间考虑这些假设的情况，你就有可能开发出足够强壮且容易维护的软件。设计强壮的软件是你最基本的目标。

11. 不要低估对软件规模的需求

Internet 带给我们的最大的教训是你必须在软件开发的最初阶段就考虑软件规模的可扩充性。今天只有 100 人的部门使用的应用程序，明天可能会被有好几万人的组织使用，下月，通过因特网可能会有几百万人使用它。在软件设计的初期，根据在用例模型中定义的必须支持的基本事务处理，确定软件的基本功能。然后，在建造系统的时候再逐步加入比较常用的功能。在设计的开始考虑软件的规模需求，避免在用户群突然增大的情况下，重写软件。

12. 性能仅仅是很多设计因素之一

关注软件设计中的一个重要因素--性能，这好象也是用户最关心的事情。一个性能不佳的软件将不可避免被重写。 但是你的设计还必须具有可靠性，可用性，便携性和可扩展性。你应该在工程开始就应该定义并区分好这些因素，以便在工作中恰当使用。性能可以是，也可以不是优先级-最高的因素，我的观点是，给每个设计因素应有的考虑。

13. 管理接口

"UML User Guide"（Grady Booch，Ivar Jacobson 和 Jim Rumbaugh ,Addison Wesley, 1999）中指出，你应该在开发阶段的早期就定义软件模块之间的接口。这有助于你的开发人员全面理解软件的设计结构并取得一致意见，让各模块开发小组相对独立的工作。一旦模块的接口确定之后，模块怎样实现就不是很重要了。从根本上说，如果你不能够定义你的模块"从外部看上去会是什么样子"，你肯定也不清楚模块内要实现什么。

14. 走近路需要更长的时间

在软件开发中没有捷径可以走。缩短你的在需求分析上花的时间，结果只能是开发出来的软件不能满足用户的需求，必须被重写。在软件建模上每节省一周，在将来的编码阶段可能会多花几周时间，因为你在全面思考之前就动手写程序。你为了节省一天的测试时间而漏掉了一个 bug，在将来的维护阶段，可能需要花几周甚至几个月的时间去修复。与其如此，还不如重新安排一下项目计划。避免走捷径，只做一次但要做对（do it once by doing it right）。

15. 别信赖任何人

产品和服务销售公司不是你的朋友，你的大部分员工和高层管理人员也不是。大部分产品供应商希望把你牢牢绑在他们的产品上，可能是操作系统，数据库或者某个开发工具。大部分的顾问和承包商只关心你的钱并不是你的工程（停止向他们付款，看一看他们会在周围呆多长时间）。大部分程序员认为他们自己比其他人更优秀，他们可能抛弃你设计的模型而用自己认为更好的。 只有良好的沟通才能解决这些问题。要明确的是，不要只依靠一家产品或服务提供商，即使你的公司（或组织）已经在建模、文档和过程等方面向那个公司投入了很多钱。

16. 证明你的设计在实践中可行

在设计的时候应当先建立一个技术原型， 或者称为"端到端"原型。以证明你的设计是能够工作的。你应该在开发工作的早期做这些事情，因为，如果软件的设计方案是不可行的，在编码实现阶段无论采取什么措施都于事无补。技术原型将证明你的设计的可行性，从而，-你的设计将更容易获得支持。

17. 应用已知的模式

目前，我们有大量现成的分析和设计模式以及问题的解决方案可以使用。一般来说，好的模型设计和开发人员，都会避免重新设计已经成熟的并被广泛应用的东西。

18. 研究每个模型的长处和弱点

目前有很多种类的模型可以使用,如下图所示。用例捕获的是系统行为需求，数据模型则描述支持一个系统运行所需要的数据构成。你可能会试图在用例中加入实际数据描述，但是，这对开发者不是非常有用。同样，数据模型对描述软件需求来说是无用的。每个模型在你建模过程中有其相应的位置，但是，你需要明白在什么地方，什么时候使用它们。

19. 在现有任务中应用多个模型

当你收集需求的时候，考虑使用用例模型，用户界面模型和领域级的类模型。当你设计软件的时候，应该考虑制作类模型，顺序图、状态图、协作图和最终的软件实际物理模型。程序设计人员应该慢慢意识到，仅仅使用一个模型而实现的软件要么不能够很好地满足用户的需求，要么很难扩展。

20. 教育你的听众

你花了很大力气建立一个很成熟的系统模型，而你的听众却不能理解它们，甚至更糟一连为什么要先建立模型都不知道。那么你的工作是毫无意义的。教给你开发人员基本的建模知识；否则，他们会只看看你画的漂亮图表，然后继续编写不规范的程序。另外， 你还需

要告诉你的用户一些需求建模的基础知识。给他们解释你的用例(uses case)和用户界面模型，以使他们能够明白你要表达地东西。当每个人都能使用一个通用的设计语言的时候（比如UML-译者注），你的团队才能实现真正的合作。

21. 带工具的傻瓜还是傻瓜

你给我 CAD/CAM 工具，请我设计一座桥。但是，如果那座桥建成的话，我肯定不想当第一个从桥上过的人，因为我对建筑一窍不通。使用一个很优秀的 CASE 工具并不能使你成为一个建模专家，只能使你成为一个优秀 CASE 工具的使用者。成为一个优秀的建模专家需要多年的积累，不会是一周针对某个价值几千美元工具的培训。一个优秀的 CASE 工具是很重要，但你必须学习使用它，并能够使用它设计它支持的模型。

22. 理解完整的过程

好的设计人员应该理解整个软件过程，尽管他们可能不是精通全部实现细节。软件开发是一个很复杂的过程，还记得《object-oriented software process》第 36 页的内容吗？除了编程、建模、测试等你擅长工作外，还有很多工作要做。好的设计者需要考虑全局。必须从长远考虑如何使软件满足用户需要，如何提供维护和技术支持等。

23. 常做测试，早做测试

如果测试对你的软件来说是无所谓的，那么你的软件多半也没什么必要被开发出来。建立一个技术原型供技术评审使用，以检验你的软件模型。在软件生命周期中，越晚发现的错误越难修改，修改成本越昂贵。尽可能早的做测试是很值得的。

24. 把你的工作归档

不值得归档的工作往往也不值得做。归档你的设想，以及根据设想做出的决定；归档软件模型中很重要但不很明显的部分。给每个模型一些概要描述以使别人很快明白模型所表达的内容。

25. 技术会变，基本原理不会

如果有人说"使用某种开发语言、某个工具或某某技术，我们就不需要再做需求分析，建模，编码或测试"。不要相信，这只说明他还缺乏经验。抛开技术和人的因素，实际上软件开发的基本原理自 20 世纪 70 年代以来就没有改变过。你必须还定义需求，建模，编码，测试，配置，面对风险，发布产品，管理工作人员等等。软件建模技术是需要多年的实际工作才能完全掌握的。好在你可以从我的建议开始，完善你们自己的软件开发经验。以鸡汤开始，加入自己的蔬菜。然后，开始享受你自己的丰盛晚餐吧。

Rational 统一过程 RUP 贴近中小软件开发

对于中、小规模的软件项目，开发团队的规模不是很大。软件的开发周期也比较短。在这种情况下，完全照搬 RUP 并不完全适用。因此，裁剪 RUP 使其适合中、小型软件开发项目是非常有必要的。

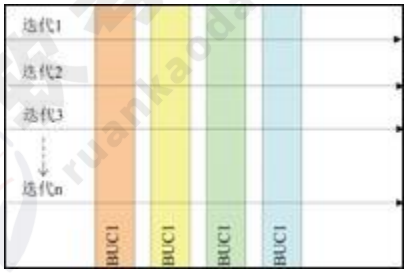

图 1：一个复杂的 BUC 的实现方法

Rational 统一过程(RUP)是 IBM 公司的一个软件过程产品。它几乎覆盖了软件开发过程中的所有方面。

J2EE 技术提供了一个基于组件的、多层分布式计算平台。在 J2EE 的应用系统的开发过程中，由于使用了中间件，开发人员可以把工作重点放在系统功能的建模、设计与实现上。此外，J2EE 技术结合了软件设计中的最佳实践(best practices)，如以架构为中心的软件体系结构、基于组件的架构等等。这一切都对现有的软件工程过程提出了新的挑战。所以，裁剪 RUP 并且使其在 J2EE 项目中起更大的作用是非常有意义的。

本文讲述了如何把 RUP 应用到小型项目团队开发 J2EE 应用系统的过程中，并且结合 J2EE 技术的特点从项目管理、架构设计、开发和测试等方面重点阐明了对 RUP 的裁剪。

项目管理

在 RUP 中，角色定义了个人或团队的行为和职责，包括分析设计人员、编程人员、测试人员、项目管理人员和辅助人员，一个人可以同时担当几个角色.一个角色也可以由几个人来共同承担。针对 J2EE 系统的开发和维护，J2EE 规范中也定义不同的角色，包括 J2EE 产品供应商、应用组件供应商、发布人员、系统管理员等等。

在实际的项目运行中.要根据软件开发组织的实际情况来确定角色的定义和分配。项目经理是必不可少的一个角色，通常是一个人来担任。项目经理代表整个项目与软件客户进行沟通和协商，并且制定软件开发计划等等。架构师也是一个必须的角色，通常由一名经验丰富的软件开发人员来担任。

在项目运行的前期，架构师负责设计软件架构和原型系统。在项目运行后期，架构师可以参与到具体的软件开发中。SQA 同样是必不可少的，通常是一名经验丰富的软件开发人员来担任。SQA 在整个项目的运行过程中负责监督和改进软件质量，包括制定系统测试方案、用户接受测试方案等等。开发人员是组成团队的主要力量，负责系统的设计、开发和测试。如果可能的话，团队中必须设立业务分析员的角色，负责商业建模等，通常由有特定行业经验的人来担任。

迭代开发计划

RUP 的精髓之一迭代式的开发，它是基于 Spiral 模型翻的。整个软件开发周期由很多个迭代组成，其中初始迭代最为重要。其它每个迭代都为了实现软件的部分功能。在完成所有迭代后，软件的所有功能都已实现并且通过测试。

初始迭代又叫作 0 迭代，它开始于项目的启动。结束于 RUP 初始阶段(inception phase)的完成。初始迭代在整个软件项目中起着十分重要的作用，这是因为在这个迭代中，项目团队和客户必须对软件项目的范围、成本、进度和应用系统的边界以及功能等达成一致的理解。

在初始迭代中，最重要的活动有明确项目的范围、商业需求和提出至少一个可用的软件架构方案。在明确项目范围的过程中，项目经理就项目的边界、产品、限制条件等与软件客户进行协商，从而达成一致认识。同时，在理解客户需求的基础上，项目经理或者业务分析员以需求说明书和功能说明书的形式把客户的需求记录下来。并且和客户达成一致理解。在此基础上，架构师提供至少一个合适的软件架构方案，并且完成原型系统。原型系统的目的不但是为了验证技术上的可行性，而且是为了给客户一个感性的认识，更好地完善对需求的理解。

需求说明书从客户的角度简要地描述了系统要具备的功能，它包含了很多商业用例。通常情况下，需求说明书还不能够全面地描述整个应用系统，所以软件开发组织还要从不同角度来描述系统的功能和特征，这就是功能说明书。功能说明书中包含了很多系统用例。功能说明书和需求说明书必须征求客户的意见，直到客户满意为止。

迭代计划是项目计划的一部分，指如何把要实现的系统分解成更小的子系统和如何在不同迭代中(除初始迭代之外)划分子系统，从而使每个迭代的目标明确，不同迭代之间的依赖关系达到最低。通常情况下，从逻辑上看，应用系统可以划分成多个 BUC，而每个 BUC 又可以进一步划分成 SUC;因此，可以从 BUC 的角度出发，根据相互之间的依赖程度来进行划分，把依赖程度低的 BUC 划分到不同的迭代中，从而确定每一个迭代的范围。一个复杂的 BUC 可以把它分解成独立的几个小 BUC 在几个迭代中来实现。

一个应用系统也是由很多组件组成的。一个或者几个组件组合起来可以实现一个 SUC 或者一个 BUC 的要求。在设计迭代计划的时候，要考虑到组件之间可能存在的约束关系。

基于 J2EE 的应用系统是基于组件架构的，因此，最小化迭代之间的依赖是一个最重要的衡量标准。

采用这种迭代办法后，每个迭代的范围限制在一个或者几个相互独立的 BUC 中。这样做的好处在于降低需求变化带来的风险。

风险管理

采用迭代式开发的一个很重要的原因是，项目的风险能够在早期的几个迭代中暴露出来。风险有两个基本的属性，一个是它发生的概率，还有一个是风险发生后对项目的影响。风险管理的目的是为了尽量降低风险发生时对项目的影响。

在风险管理中，首先要识别项目中存在的风险。其次根据风险发生的概率和风险发生后对项目的影响来分析存在的风险。通常采用量化风险的办法。给概率和影响分别赋予一定的数值，经过分析，把概率的数值和影响的数值相乘后的结果风险量化后的值。接着，对于量化后值比较高的风险制定相应的风险规避计划。在项目运行过程中，要不断地监督风险的变化。

架构设计

RUP 采用基于组件的软件架构和以架构为中心的开发方式。J2EE 技术强调基于组件的软件架构,能够很好地体现 RUP 的架构思想。根据 3D 方法可以把一个 J2EE 应用系统的架构从三维进行分析，分别是 Tier、Layer 和 Systematic Quality。在设计系统架构的时候，可以从这三个角度考虑。

Tier

从 Tier 层的角度进行考虑，一个 J2EE 应用系统的架构可以分为以下几个部分：客户端层、表示层、业务逻辑层、集成层、资源层，如图 2 所示。每层都是按系统中业务逻辑而划分的，它具有唯一的职责。每层与相邻层都是松散耦合的。

图 2：基于 Tier 层的 J2EE 应用系统架构

在实现的时候，需要结合项目的具体情况而定。基于 MVC 设计模式的 J2EE Web 应用系统中，客户一般访问 JSP。然后由 Control 层进行处理：如果需要进行复杂的业务逻辑处理并且已经有后台实现，业务逻辑使用 Facade 模式进行封装，形成统一的接口，业务逻辑层实现复杂的事务处理;如果需要访问资源层，再经过 DAO 层访问资源。

Layer

从 Layer 的角度进行考虑，一个 J2EE 应用系统的架构可以分为几个部分：最下层为操作系统、Java 虚拟机和网络，它们负责系统的底层操作和网络数据的传输;之上是 J2EE 服务层，一般由 J2EE 服务器(如 WebSphere，WebLogic 等)提供各种基础服务，如事务的管理(JTS)、命名目录服务(JNDI)、负载均衡(Load Balancing)、容错(failover)、安全(security)等;其次是通用业务层，它一般完成与具体业务无关的基本操作，由通用的组件来实现，如数据库处理组件、系统错误处理组件、字符处理和数值处理组件、日志(10g)处理、数据转化和编码维护等;最上层才是具体业务逻辑模块，它完成具体的业务逻辑。

图 3：基于 Layer 的 J2EE 应用系统架构

在实现的时候.底层一般是不需开发人员关心的操作系统和网络环境，并且不同 J2EE 服务器厂商都提供了相应 J2EE 服务层， 开发人员需要关心上面两层的实现。如果是 J2EEWeb 应用体系，应用服务层一般会使用 Struts 框架。log 服务一般选择 log4j 等。最上层才是具体业务模块。

Systematic Quality

这是指在软件架构中通过一定的办法或者使用一定的工具来达到系统要求的 QoS，一般指可扩展性、可移植性、可维护性、安全等等,而这些恰恰是 J2EE 架构本身所带来的好处。

实现和测试

实现是软件开发人员编写代码来完成每一个组件。测试是用来保证软件质量的重要手段。采用 RUP 的软件工程过程后，整个项目被划分成不同的迭代。每个迭代(除了初始迭代)的范围是一个或者多个独立的 BUC， 目标是编写代码实现 BUC 并且保证软件的质量。

在实现和测试的时候，集成(integration)是很重要的。这是因为整个软件开发过程分成多个迭代来完成，每个迭代(除初始迭代外)都是为了实现应用系统的一个部分。对于相邻的两个迭代。后者是在前者的基础上进行开发的，是实现功能上的一个增量。因此，相邻迭代之间需要功能上的集成。此外，每一个迭代都是由 BUC 组成的。从逻辑上来看，一个 BUC 是由一个或者多个 SUC 组成的。从实现上来看，每个 SUC 是由一个或者多个组件(component)组成的。因此，每一个迭代中都需要组件之间的集成，如图 4 所示。

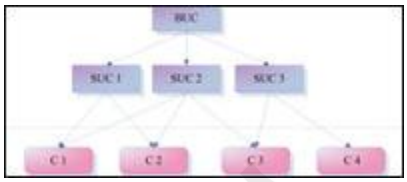

图 4：BUC 的组成结构

根据集成程度的不同，可以划分几个不同的开发集成和测试：

首先是 SUC 集成和单元测试。这个是粒度最小的集成，它把几个不同的组件集成起来，实现同一个 SUC。例如，SUC1 是通过集成 C1 和 C2 来实现的。同时，在集成完成后，进行相应的单元测试。

其次是 BUC 集成和集成测试。BUC 集成是把几个相关的组件集成起来,来实现它的功能。图 4 中 BUC 的实现需要集成 4 个组件。同时，在集成完成后进行相应的集成测试。

再次是迭代内集成和系统测试。迭代内集成从功能上来看，就是把这个迭代包含的所有 BUC 集成起来;从代码上看，是把所有和 BUC 相关的组件集成起来。同时，在集成完成后进行系统测试。系统测试分两步，首先是从功能上来测试每个 BUC，其次是测试不同 BUC 之间的依赖和约束。

最后是迭代间集成和回归测试。对于相邻的两个迭代，从功能上来看，后者是前者基础上的一个增量。迭代间集成把这个增量准确地集成到应用系统上。同时，在集成完成后进行衰减测试。回归测试不但要测试功能增量的正确性。而且要测试增量发生后系统原来功能的正确性。

实例研究

笔者在 Trade Manager 项目中运用了上述的方法。TradeManager 是一个关于金融软件研究的项目，开发基于 J2EE 技术的金融订单管理系统。项目由 12 个人的团队来进行开发。团队成员分工明确，有项目经理、架构师、测试员和 SQA 等等。项目采用迭代式的开发方式。在初始迭代中，项目双方对项目范围、功能需求及架构达成一致，并签字同意。整个开发分为三个迭代阶段，根据功能点来划分，每个迭代分别实现交易前、交易中和交易后的功能。每个迭代的开发时间在六个星期。

这个软件采用 J2EE 的架构，如图 5 所示。其中 UI 和 Delegate 层在客户端，采用 Swing 技术来实现，是一个典型的肥客户端。Facade、Business Logic 和 DAO 在 J2EE 服务器端，采用 EJB 技术来实现，它与客户端的通讯是典型的 RMI/IIOp 协议，采用的服务器是 WebSphere。后台采用 Oracle 数据库来存放各种系统数据。同时，采用 SiteMinder 来实现系统的认证和授权。用 log4j 来实现 logging/auditing 功能。由于采用 WebSphere 集群技术，系统的可扩展性和高可用性得到了保证。

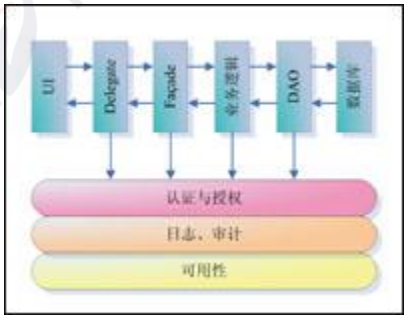

图 5：应用实例

在开发的三个迭代中(除初始迭代外)，相邻两个迭代进行衰减测试，由 SQA 和测试员来完成。每个迭代中，开发人员完成单元测试和集成测试，SQA 和测试人员完成系统测试。在做完三个迭代后，软件移交给客户进行 UAT。

RUP 适用于规模比较大的软件项目和大型的软件开发组织或团队。在实际中，软件项目团队根据自身客观条件的限制和技术的影响，应该对 RUP 进行必要的裁剪，从而让 RUP 更好地服务于软件开发过程。本文在结合 J2EE 技术特点的基础上，从不同方面讲述了如何定制 RUP，在项目管理中，应该正确定义团队角色、采用迭代式的开发方式和重视风险管理;在架构设计中，针对 J2EE 技术的特点。指出了从三个不同方面来设计软件架构;在开发和测试中，应该正确对待各个阶段的集成和测试。

Rambo 安全经验谈连载之：防护服务器的安全

任何网络的核心都在于服务器，包括数据库服务器、Web 服务器、DNS 服务器、文件和打印服务器等等。这些服务器为网络提供了大量的资源，通常，大多数重要的信息都存储在这些服务器上，这就意味着这些计算机对骇客来说是尤其诱人的，首当其冲，当然应该重点防护。

对于防护服务器来说，除了应用防护个人工作站的措施以外，还应该有些额外的重要措施。服务器上不会有人处理文档和电子表格，所以对服务器的防护不会像个人终端那样情况复杂。

首先，应该应用个人工作站的防护措施。打补丁，安装防病毒、防间谍软件，严格管理服务器的使用，除此之外还需要额外的防护措施。大多操作系统（如 Windows2003、Linux）都有日志功能，包括登陆日志、安装软件日志等等，应该确保所有安全相关行为都有日志，并定期对这些日志做检查。要知道服务器上的数据要比一般计算机上的数据有价值的多，正因为此，服务器上的数据一般都有定期备份。建议每天备份一次，但通常情况下，一礼拜备份一次也就够了。备份磁带应该放在不联网的地方（例如单位的保险库）、且防火的地方。对备份磁带进行安全管理与对服务器进行安全管理一样重要。对于任何计算机来说，所有不

需要的服务都应该关闭，对于服务器，可能要把不需要的服务和操作系统组件彻底卸载。但是执行之前要慎重，显然游戏和办公软件服务器并不需要，浏览器对于更新补丁来说是需要的。对于服务器来说还需要做的一点，大多服务器都有内建账户，例如 Windows 就有三个内建账户：administrator、guest、power guest。骇客猜测账户密码会首先从这些内建账户开始。事实上，互联网上有很多自动化的工具为骇客做这项工作。首先，应该自己创建一个账户不要显示账户的权限级别。例如，创建一个账户 basic_user，并禁用 administrator 账户，设置 basic_user 为管理员账户，并赋予相应的权限。（当然，这个账户是给具有管理员资格的人用的）这样做了以后，骇客就不会马上对这个账户感兴趣。要知道骇客总是想要管理员权限，为管理员账户做掩护对于防止骇客攻击来说是非常重要的。

对于 Windows 来说，还有一系列注册表设置可以提高计算机安全。使用 Cerberus，可以扫描出注册表设置的一些漏洞。什么样的设置会引起安全问题呢？通常要检查以下项目：

登陆：假如注册表设置在登陆时显示上一次登陆账户，那么黑客就少了一半工作要做。当黑客知道了密码，只需要破解密码就行了。

默认共享：一些文件及驱动器是默认共享的，打开这些共享是极不安全的。

在 Windows 组册表还有一些潜在的安全问题，Cerberus 不但能把这些问题扫描出来，还会提出解决这些问题的建议。

下面总结一下，对服务器的常规动作：

1、应用个人工作站常规防护措施。

2、定期做好备份，并做好灾备方案。

3、打开日志功能，定期做好审计。

4、为管理员账户加以伪装。

做好了工作站与服务器的安全防护措施，下面介绍一下如何防护整个网络，如果有兴趣的话请看下篇：防护企业网络。

PowerDesigner 数据与应用程序的完美融合

powerdesigner7．0 把物理数据模型、概念数据模型以及新增加的对象建模功能完美地结合在一起，为程序开发人员和数据库管理人员提供了一整套完善的应用程序建模解决方案。它不仅能够加快项目开发的进程，而且能够显著地提高软件的质量。

建议

如果用户打算购买或者升级到 powerdesigner7．0，则需要先投入一部分资金进行相关人员的培训。不过，一旦公司的技术人员熟练地掌握了这种工具，他们将大大地缩短项目的开发周期，有效地降低软件维护费用。

优点

●提供一系列工具来简化物理数据模型、概念数据模型和对象模型的构造过程
●优秀的报表功能
●自动生成 java 或者 powerbuilder 代码
●具有强大的逆向工程能力

缺点

●需要先对 it 人员进行培训
●支持有限的运行平台

费用

●physical architect 995 美元
●data architect 2995 美元
●developer 2995 美元

●object architect 4995 美元
●studio 6495 美元
●viewer 395 美元
运行平台
●windows 95／98／nt

任何一个参与过大型软件项目开发的人都知道，在开始构建一个应用程序之前，如果缺乏可靠完善的设计方案，其后果就像盖房子没有打好地基一样不堪设想；如果抱着节省投资或者缩短工期的想法而忽视了软件的设计和建模过程，那么，为了能够在软件开发的后期交付一个可靠的产品，你将不得不花费更多的资金来对它进行维护，这实在是一种得不偿失的做法。

sybase 公司的 powerdesigner7．0 是一个运行在 windows 平台上的系统建模工具，它可以使软件开发人员和数据库管理员协同工作，快速地建立起软件的系统模型，为开发一个稳定可靠的软件打下坚实的基础。尽管在购买 powerde signer7．0 初期，公司需要先花费一定的资金用于 it 人员的培训，但他们一旦熟练掌握了工具的使用方法，则能够在以后各个项目的开发过程中大大地缩短项目开发的周期，显著地提高软件的质量，有效地降低软件的维护费用，从而最终为公司节省大量投资。

powerdesigner7．0 提供了物理数据模型和概念数据模型的建模工具，新增加了对象建模功能，并提供一组实用的报表生成工具。用户还可以直接购买套装的建模工具包 powerdesignerstudio，它集成了所有的建模工具，具有描述数据流图的功能。如果我们并不需要所有的建模功能，也可以根据自己的使用情况购买某些特定的 powerdesigner7．0 工具。但是，不论我们有什么样的设计需求，powerdesigner7．0 都能很好地协助我们完成各种系统建模的工作。同市场上的其它同类工具相比，powerdesigner7．0 无论在功能上还是在易用性方面都毫不逊色。例如，它的数据建模工具不次于 ca 公司的产品 erwin，而它的对象建模功能也将对 rational 软件公司的拳头产品 rose 构成极大的威胁。

特性之一：
数据建模工具

对于数据库管理员来说，powerdesigner7．0 的数据建模工具将给他们的工作带来很大的帮助。在物理数据建模过程中，数据库管理员首先定义数据库模型、数据存取策略和索引参数等基本信息，然后他们根据这些信息就可以快速准确地完成数据库的创建工作了。

通过测试，我发现使用 powerdesign er7．0 提供的建模工具能够非常方便地创建一个新的物理数据模型。我还可以用它直接打开 powerdesigner6．0 格式的数据模型文件。此外，它还可以方便地从其它建模工具（如 erwin）创建的文件中引入数据模型。

而 powerdesigner7．0 灵活的数据库创建方式同样给我留下了很深的印象。我可以利用 powerdesigner7．0 来生成物理数据模型的 sql 脚本语言，然后直接在我的数据库中执行这些语句；同时，我还可以通过 odbc，根据物理数据模型直接创建数据库。

powerdesigner7．0 的物理数据建模工具不仅可以帮助用户灵活地创建数据库，而且对数据库的管理工作也有很大帮助。例如，用户可以把 powerdesigner7．0 的数据恢复功能作为数据库移植策略的一部分来使用。

特性之二：
逆向工程

对数据库进行逆向工程操作是 powerdesigner7．0 的另一特性。通过逆向工程，powerdesigner7．0 可以把一个设计好的数据库（包括触发器和存储过程）转化成一个物理数据模型。然后，我们可以通过修改这个模型来快速地创建一个新的数据库管理系统，从而大大地简化了把一个原有的数据库移植到一个新的数据库的过程。

在进行逆向工程操作时，powerdesigner7．0 还支持数据库的双向同步功能，也就是说，无论我们是修改物理数据模型，还是对数据库本身进行改动，这两者之间都能始终保持同步。我在测试过程中同时修改了物理数据模型及其对应的数据库，然后使用 compare／merge 工具对它们进行同步操作，其结果是对任何一部分的修改都同时反映在物理数据模型和数据库

中，从而达到了同步。

使用 powerdesigner7．0 的逆向工程操作，用户可以灵活地把物理数据模型转换成概念数据模型或者对象模型。在开发一个应用程序时，有的公司习惯于自顶向下的设计方法，从商业逻辑层逐渐过渡到实际的数据库；有的公司则更喜欢采用自底向上的设计方法，从数据库设计开始最终上升到商业逻辑层。powerdesigner7．0 灵活的设计能力则能够同时满足这两种不同用户的使用需求。

特性之三：

powerbuilder 代码

在创建了一个概念数据模型之后，powerdesigner7．0 可以根据它创建一个物理数据模型，然后把它应用到任何一种数据库产品中；用户也可以根据这个概念数据模型生成一个用 uml（unifiedmodelinglanguage）语言中的类图所表示的对象模型。

powerdesigner7．0 对象建模功能是众多新增功能中值得推荐的一个。在创建一个对象模型时，开发人员使用标准的 uml 类图来描述应用程序的结构以及这些类之间的商业逻辑。

在创建好一个对象模型之后，powerdesigner7．0 可以根据类图直接生成 java 代码或 sybase 的另一个开发工具———powerbuilder 代码。通过 codepreview 工具，我们可以在正式代码生成之前预览这些程序代码。如图 1 所示，如果你选择了生成 java 代码，powerdesign er7．0 将生成标准的 java 类或者 jav abeans。此外，我们还可以对原有的 java 源程序、二进制的 java 可执行文件以及 java 文档进行逆向工程操作，并可根据它们获得 powerdesigner7．0 中的类图。

如果你是一个 powerbuilder 程序员的话，你可以把 powerdesigner7．0 类图转换成非可视化的 powerbuilder 代码。通过 powerbuilder 的应用程序编程接口，这些新生成的代码能够自动地与 power builder 的库函数集成在一起。同 java 语言一样，用户也可以对 powerbuilder 代码进行逆向工程操作，根据代码来生成 powerdesigner7．0 中的类图。

其它

除了建模工具之外，powerdesigner7．0 还集成了一些非常实用的报表工具。如图 2 所示，使用这些工具，用户可以方便地创建单模型或者多模型的报表。powerdesigner 提供了很多报表模板来简化报表生成工作。同时，用户也可以创建适合自己的报告模板。

用户创建的报告可以包含在任何类型的 powerdesigner7．0 对象之中。报告完成后，他们可以选择以 rtf 格式或者 html 格式来保存这些报告。这样，他们可以方便地把它们包含在项目管理文档或者其它参考文档中，以便在将来编写用户帮助文件时方便地从中提取信息。

小结

如果你的公司希望大幅度地提高软件的开发质量，或者迫切需要加快项目开发进程的话，powerdesigner7．0 则是一项非常有益的投资。在项目开发的设计阶段把开发人员和数据库管理人员紧密联系在一起，为他们提供必要的工具来构建一个稳固的应用程序设计基础，这种严谨的软件开发方式必将在长期的软件开发、测试和维护过程中为公司节省大量的资金。

PDA 连接远程数据库的三种解决方案

在 Windows CE 5.0 或 Pocket PC 2003 或 Sarthpone 2003 中,可以使用下面三种方法访问远程数据.

第一种:

使用 Web Service 作为中介访问数据,在 PDA 中可以直接访问安装在访问器上的 Web Service,通过 Web Service 就可以访问远程数据库了.了解 Web Service 这种方法应该很简单.使用 Web Service 可以在速度上会稍显得有点慢.

第二种:

直接访问数据库

如果你使用得 Sql Server 2000 或 SQl Server 2005 都可以直接访问,如果你使用得 Access 数据库那就只能用上一种方法了.听 PDA 公司的人说 SQL Server 2000 要升级到 SP4,但好像我没有升级数据库也可以访问到.大家可以试试看. 如果你是在 Vs 2005 里开发,就先添加 System.Data.SqlClient 引用,如下面的一段代码(其实和桌面系统一样访问): SqlConnection conn = new SqlConnection("Server=10.116.192.7;DataBase=smcgz;User ID=salesmng;Password=j6f7j7g2;Persist Security Info=True;");

```
try
{
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "select Count(*) from Inventory";
cmd.Connection = conn;
conn.Open();
textBox1 .Text = cmd .ExecuteScalar ().ToString() ;
}
catch (SqlException ex)
{
MessageBox.Show(ex.Message.ToString ());
}
```

第三种:

还有只要是直接使用 Socket,如果你是写过网络程序的,这应该都知道怎么解决了,把写一个客户端安装在 PDA 上,在 PDA 上使用 Socket 连接服务器传递数据,当然还得写一个访问器端,开启监听接受客户端的数据,并向将从数据种查询到的数据发送到客户端.使用这种方法就比较复杂和麻烦.

P2P 网络软件设计缺陷助阵拒绝服务攻击

现有的 P2P 网络软件设计缺陷使得攻击者可以轻松的发起庞大的拒绝服务攻击，从而轻易使得互联网网站崩溃。

根据安全公司 Prolexic Technologies 的数据，在过去的三个月中，超过 40 个公司承受了成百上千个网络协议地址（IPs）的攻击，其中很多攻击都在每秒中产生了多于 1GB 的垃圾数据。该公司的首席科学家 Paul Sop 称，大量的网络地址使得那些基于诸如黑名单方式的路由器和防火墙失去了有效的防护能力。

“问题在于你能够用多快的速度来把”水“从”船“中舀出去”，Paul Sop 说，“如果你有某一分钟停了下来，那么你就会沉没”。

经典的 DOS 攻击一般通过数千个协作的电脑来发送数据包，从而使得网站服务崩溃，而最新的这种攻击方式由一些运行着一个叫做 DC++的 P2P 软件的电脑发起。这种软件基于一种名为 Direct Connect 的协议，从而使得文件可以在多个客户端之间进行直接的交换。

根据一名从 Sweden's Lund Institute of Technology 获取了学士学位的 DC++工程开发人员 Fredrik Ullner 的说法，“当文件共享网络发布时，用来寻找某个特定文件的目录被记录在某些被称为 hubs 的少数服务器上。较早版本的 hub 服务器软件存在一个漏洞，从而使得人们能够直接从另外一个服务器上获取信息。因此，攻击者可以控制他们的 hub 服务器，将大量的客户数据请求”转接“到受害者的网络服务器上。由于用户数目庞大，受害者的网络服务器很快就会崩溃。”

“这种针对 DC++拥有者和中心目录的攻击最早从 2005 年开始。最先的攻击瞄准了 DC++核心的 hubs 服务器目录：Hublist.org.恶意的 DC++用户最初开发了一种对单个 hub 进行洪水攻击的工具。而当 Hublist.org 将这些恶意用户的服务器移除了之后，他们就对 Hublist.org 进行了这类攻击。”Ullner 说。不仅仅是 Hublist.org，还有 DC++的主要项目网站 DCPP.net，都在攻击下无法访问，从而迫使开发者迁移到了 SourceForge.

“很不幸的是，这类攻击正在变得越来越普遍，”Ullner 在 SecurityFocus 的一封公开邮件

中提到。事实上，这项新的攻击技术是如此的有效，从而使得攻击者将他们的目标转向了其他的公司。

在三月份，很多的公司开始向 Prolexic 求助，希望能够避免这些极具破坏性的拒绝服务攻击。而在这些攻击中，超过 150，000 台电脑会打开多个连接，从而使得网站服务器被雪崩式的网络数据所埋葬。“被记录的最大一次攻击涉及到超过 300，000 台电脑。” Prolexic 公司的 Sop 说。

“我们有数百万个连接同时接入，”Sop 说，“我们可以很容易的辨别出这种攻击，但是由新 IPs 所发起连接的增长速度超出了我们的处理能力”。

“很多攻击是勒索行为的一部分。因为可以很简单将拒绝服务攻击转化成现金。而大约有四分之三的攻击是由商业间谍发动的。”Sop 说。

“这类攻击攻击所涉及到的金钱数目巨大，”Sop 说，“如果你在欧洲有着一个互联网公司，并且能够通过这类攻击解决掉你的竞争对手，那么为什么还把钱花在市场扩展上？”

好消息是，Prolexic 已经在周三宣称，他们已经找到了技术上的解决方案。

坏消息是，尽管在技术上已经能够很好的修补 DC++ hub 软件上的漏洞，但根据开发人员 Ullner 的说法，麻烦出在如何说服用户及时的打上补丁。事实上，正是由于人们对安全补丁的忽略，使得攻击者依然能够横行无忌。

最糟糕的是，即便是所有善意的 hub 管理员都将他们的系统升级到最新的版本，攻击者自己依然能够开设一个没有应用安全补丁的 hub，并在这个 hub 吸收到足够的用户时发动攻击。这使得安全人员很难能够对之进行控制。

MPEG－4 的主要技术概览

前言

MPEG－4 编码标准是目前最新的国际编码标准规范?熏本文就其主要的内容作了简单的概述?熏并在此基础上，着重介绍了具有特色的音频对象的编码和视频对象的编码。

1 多媒体传输集成框架

多媒体传输集成框架（ＤＭＩＦ）主要解决交互网络中、广播环境下以及磁盘中多媒体应用的操作问题，通过传输多路合成比特信息，建立客户端和服务器端的握手和传输。与过去不同的是，由于 MPEG－4 码流中，包括许多的 AV 对象，一般而言，这些 AV 对象都有各自的缓冲器，而不仅仅是视频缓冲器和音频缓冲器。

2 语法描述

MPEG－4 定义了一个句法描述语言来描述 AV 对象比特流表示和场景描述信息。这个句法描述语言是对Ｃ＋＋的扩展，不仅易于表达其 AV 对象特性，而且也易于软件仿真实现与模型验证。与 MPEG－4 相比，MPEG－1 和 MPEG－2 则采用一种类Ｃ语言的描述，MPEG－4 描述语言反映了面向对象技术来描述对象。

3 音频对象的编码

视频音频的压缩编码自然仍是 MPEG－4 的核心所在。不过，与以前的 MPEG－1、MPEG－2 不同的是：MPEG－4 不仅支持自然的声音（如语音和音乐），而且支持基于描述语言的合成声音，支持音频的对象特征。即一个场景中，同时有人声和背景音乐，它们也许是独立编码的音频对象。

3．1 自然声音编码

MPEG－4 研究比较了现有的各种音频编码算法，支持 2～64Ｋ的自然声音编码。如 8 ｋＨｚ采样频率的 2～4 ｋｂｉｔ／ｓ的语音编码，以及 8 或 16 ｋＨｚ采样频率 4～16 ｋｂｉｔ／ｓ的音频编码，一般采用参数编码；6～24 ｋｂｉｔ／ｓ的语音编码，一般采用码激励线性预测（ＣＥＬＰ）编码技术；16 ｋｂｉｔ／ｓ以上码率的编码，则可采用时频（Ｔ／Ｆ）变换编码技术。这些技术实质上借鉴了已有的音频编码标准，如Ｇ．723、Ｇ．728

以及 MPEG－1 和 MPEG－2 等。图 1 是ＭＰＧＥ4 的可伸缩自然音频编码器示意图，包括了 3 种编码技术。

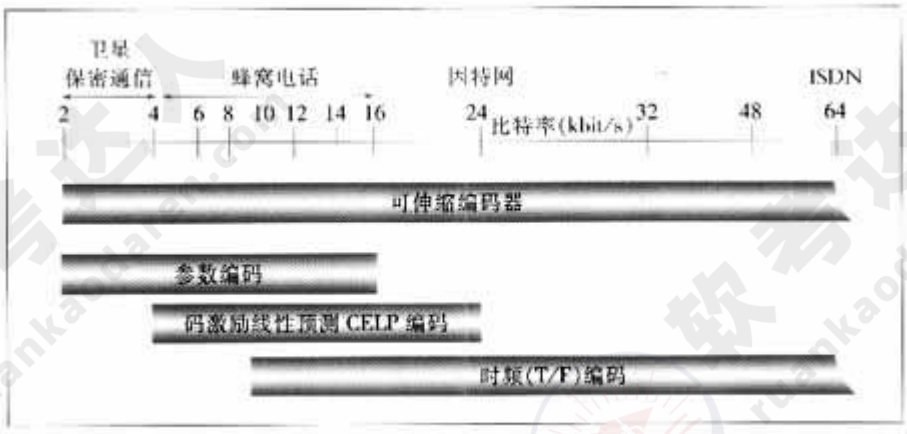

图 1　通用 MPEG-4 音频编码方框图

3．2 合成声音

在合成声音编码当中，MPEG－4 引入了 2 个极有吸引力的编码技术：文本到语音编码和乐谱驱动合成编码技术。这为网络上低比特率下交互的带有语音的游戏铺平了道路。事实上，合成声音编码技术即是一种基于知识库的参数编码。特别值得一提的是 MPEG－4 的乐谱驱动合成技术，在该技术中，解码器是由一种特殊的合成语言——结构化的音频管弦乐团语言（ＳＡＯＬ）驱动的。其中的“管弦乐团”是由不同的“乐器”组成的。当解码器不具有某一“乐器”时，MPEG－4 还允许解码器从编码器下载该“乐器”到解码器，以便正确恢复合成声音。可见，MPEG－4 不是提供一组角ＭＩＤＩ音乐标准中的“乐器”，而是提供了一个可随时扩充的“管弦乐团”，因此，其可“演奏”乐谱自然更加丰富多彩。

4 视觉对象的编码

同样，MPEG－4 也支持对自然和合成的视觉对象编码。合成的视觉对象如 2Ｄ、3Ｄ动画，人的面部表情动画等，这些合成图像单独编码，不仅可有效压缩，而且还便于操作。

对自然视觉对象的编码，仍是 MPEG－4 的重点。相对于静止图像，MPEG－4 采用零树小波算法（Ｚｅｒｏｔｒｅｅ ＷＡＶｅｌｅｔ ａｌｇｏｒｉｔｈｍ）以提供高压缩比，同时还提供多达 11 级的空间分辨率和质量的可伸缩性。

对于运动视频对象的编码，MPEG－4 采用了如图 2 所示的编码框图，以支持图像的编码。

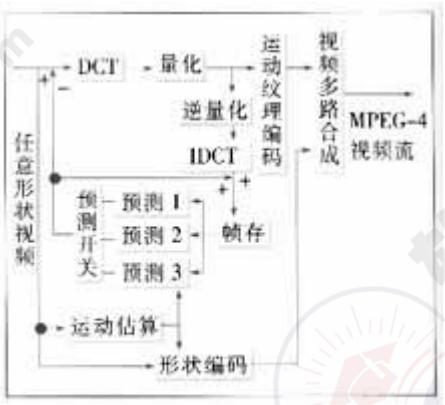

图 2　MPEG-4 视频编码方块图

可见，MPEG－4 为了支持基于对象的编码，引入了形状编码模块。为了支持高效压缩，

MPEG－4 仍然采用了 MPEG－1、MPEG－2 中的变换、预测混合编码框架。
对于一般的任意形状的视频对象，MPEG－4 编码后的码流结构见图 3。

图 3　通用 MPEG-4 视频编码

对于实时的极低比特率的应用，如可视电话，MPEG－4 视频编码采用极低比特率视频（ＶＬＢＶ）核进行编码，类似于ＩＴＵ的Ｈ．263 直接对矩形视频编码，而不采用形状编码模块。编码后的码流结构见图 4。

图 4　类似于 H.263 的 VLBV 核编码

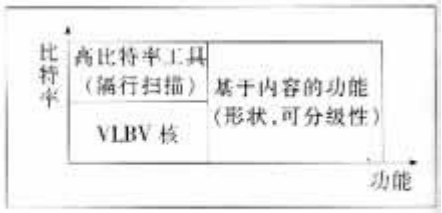

图 5　MPEG-4 的视频功能扩充

可见，MPEG－4 采取了向前兼容Ｈ．263，同时，也提供了一些高层特性，如基于内容的编码。其扩充的方式见图 5。
MPEG－4 支持有误码信道传输下的鲁棒性，提供了更好的同步和误码恢复机制。

5 场景描述
场景描述主要用于描述以上单个的 AV 对象如何在一个具体 AV 场景坐标下的组织与同步等问题。同时还有 AV 对象和 AV 场景的知识产权保护等问题。

6 MPEG－4 展望
MPEG－4 的应用将是广泛而深远的。这一新的标准将至少可以应用于以下场合：
ａ） 实时多媒体监控；
ｂ） 极低比特率下的移动多媒体通信；
ｃ） 基于内容存储和检索多媒体系统；
ｄ） Ｉｎｔｅｒｎｅｔ／Ｉｎｔｒａｎｅｔ上的视频流与可视游戏；
ｅ） 基于面部表情模拟的虚拟会议；
ｆ） ＤＶＤ上的交互多媒体应用；
ｇ） 基于计算机网络的可视化合作实验室场景应用；
ｈ） 演播室和电视的节目制作。

Linux 系统下检测 U 盘是否已连接的方法

Linux 的文件系统是异步的，也就是说写一个文件不是立刻保存到介质（硬盘，U 盘等）中，而是存到缓冲区内，等积累到一定程度再一起保存到介质中。如果没有 umount 就非法拔出 U 盘，程序是不知道的，fopen,fwrite 等函数都依然返回正确，知道操作系统要把写介

质的时候，才会提示 I/O 错误。可是很多数据都会因为这个不及时的错误报告而丢失。

事实上，USB 驱动程序在 U 盘插入和拔出时，都对系统配置文件做了修改。

例如 U 盘驱动程序会在插入或拔出时往 /proc/scsi/usb-storage-0/0 里面记上 Attached:Yes or No

通过查看这个文件就不难检测 U 盘是否插入或拔出了。

注意：U 盘是否插入的状态与是否挂载（mount）无关。

JDBC 中的中文处理

我们在做一个 JAVA 的应用，不可避免地要处理中文。经过艰苦的探索，目前有一些进展，找到了一些解决方法，但仍然面临着无法解决的问题。在此作一整理，希望对大家有所帮助，同时请各位高手帮忙考虑我们的问题。

背景：

JDK 1.15

VCafe 2.0

JPadPro

SERVER：

NT IIS

Sybase System 10

JDBC: Jconnect

CLIENT：

Browser: Netscape 4.04 + Patch

PWin95 & Pwin98 Beta3

CLASS 文件存放在 SERVER，由 BROWSER 运行 APPLET，APPLET 只起调入 FRAME 类主程序的作用。界面包括 Text field, Text Area,List, Choice 等。

一，取中文

用 JDBC 执行 SELECT 语句从 SERVER 取数据（中文）后，将数据用 APPEND 方法加到 TEXT AREA（TA），不能正确显示。但加到 LIST 中时，则大部分汉字可正确显示。

处理：将数据按“ISO-8859-1”格式转为字节数组，再按系统缺省编码格式（default character encoding）转为 STRING，即可在 TA 和 LIST 中正确显示。

程序段如下：

```
dbstr2 = results.getString(1);
//*********************************************************************
// After read result from Database server, Convert the result string.

dbbyte1 = dbstr2.getBytes("iso-8859-1");
dbstr1 = new String(dbbyte1);
//*********************************************************************
```

二，写中文到 DB

处理方式与以上相逆，先将 SQL 语句按 DEFAULT CHARACTER ENCODING 转为字节数组，再按 ISO-8859-1 转为 STRING，然后送执行，则中文信息可正确写入 DB。

```
sqlstmt = tf_input.getText();

//*********************************************************************************
// Before send statement to Database server, Convert sql statement.

dbbyte1 = sqlstmt.getBytes();
```

```
sqlstmt = new String(dbbyte1,"iso-8859-1");
//*****************************************************************************

_stmt = _con.createStatement();
_stmt.executeUpdate(sqlstmt);
。。。。。。
```

问题：
以上方法当本地客户机上存在 CLASSPATH 指向 JDK 的 CLASSES。ZIP 时（称为 A 情况），可正确运行。
但如果客户机只有 Browser，没有 JDK 和 CLASSPATH 时（称为 B 情况），则汉字无法正确转换。
我们的分析：
1,
经过测试，在 A 情况下，程序运行时系统的 default character encoding = "GBK" or "GB2312".
在 B 情况下，程序启动时，Browser 的 JAVA CONSOLE 中出现如下信息：
can´t find resource for sun.awt.windows.awtLocalization_zh_CN

然后系统的
default characterencoding = "8859-1".

2,
如果在转换字符串时不采用 default character encoding，而是直接采用“GBK”或“GB2312”，则在 A 情况下仍然可正常，在 B 情况下，系统出现错误：UnsupportedEncodingException。

3，
在本地客户机上，我把 JDK 的 CLASSES。ZIP 解压后，放在另一个目录中，CLASSPATH 只包含该目录。然后逐步删除目录中的 CLASS 文件，一边运行测试程序，最后发现在一千多个 CLASS 文件中，只有一个是不可缺少的，该文件是：
sun.io.CharToByteDoubleByte.class
我将该文件拷到 SERVER 端和其它的类放在一起，并在程序的开头 IMPORT 它，仍然在 B 情况下无法正常。

4，
在 A 情况下，如果在 CLASSPTH 中去掉 sun.io.CharToByteDoubleByte.class，则程序运行时，测得 default character encoding 为“8859-1”，否则为 GBK 或 GB2312。

5，
分析 BROWSER 程序 NETSCAPE 目录下的文件 /program/java/classes/java40.jar, 发现其中没有包括 sun.io.CharToByteDoubleByte.class,
不知这是需要升级，还是有其它方法可以解决？ 盼望各位高手指导！
Email: sailor@mailserv.stu.edu.cn
现在我们取得的一点小小进展，在转换字符串时不采用 default character encoding，而是直接采用“GBK”或“GB2312”，在情况 A 和 B 底下，从 DB 取数据都没有问题，但是写中文到 DB 也采用“GBK”或“GB2312”时，情况 B 仍是出错的。
当我们使用老外公司开发的 jdbc 第四类 driver 获取数据库中文信息时，常会出现乱码现象，如????D.

解决办法 1：
使用 interface ResultSet 的方法 getBytes()得到一 byte[]，然后由此 byte[]数组产生一新的
String,可获得正确的汉字，但此方法有一定的局限性，在某些 driver 上可以实现，如 weblogic 公司
开发的 fastforward 产品。另此种方法不规范，根据 sun jdbc 的标准 varchar 和 var 推荐用 getString()
方法来获取。

解决办法 2:
使用 interface ResultSet 的方法 getString(),这时我们得到的 String 一定是乱码，如何解决，
String temp = result.getString (s);
if (temp != null) {
byte[] b = temp.getBytes ("8859_1");
temp = new String (b);
此时的 temp 一定是正确的中文，，，，，此种方法我在 sybase 公司开发的 jconnect4 上实验成功，在 fastforward 上也成功。

java 基本内容概述

1 .对象的概念

Bruce Eckel 在《Java 编程思想》里面说，万物皆对象。

对象从词法上来讲，是一个名词，通俗一点说是一个东西。比如说，房子、汽车、动植物等，实实在在存在的，看的见摸的着的。还有一类是比较抽象的，比如说用户信息等。现实世界中，每个大的对象都是有若干个小的对象组成，小的对象最终有原子这个对象组成。程序也一样，它虽然是抽象的，但也是现实世界的描述。

以前计算机刚发明的时候，编程很困难，程序员要编写机器能懂的东西，所以都是过程式的语言，最出名的莫过于 C 了。随着计算机软硬件的发展，人们编写计算机能懂的程序不是一件难事了，但是描述现实世界的业务逻辑越来越复杂，程序代码动辄上千上万几十万几百万，逻辑结构非常复杂，要想看让别人看懂很难。而别人看不懂就无法维护，自己过段时间也不一定能完全理解。所以编程语言越来越注重人本身的体验。面向对象就是比较接返现实世界的一种描述方式，它出现目的完全是为了编程人员开发程序更方便、维护更方便。

2 .程序的控制流程（if、while、for、switch）

世界很复杂，本质很简单。

道家认为世界只有阴阳。生物种类那么多，最终都是由 4 个碱基对组成。计算机世界那么复杂，最终也归结为 0 和 1。

程序发展了这么多年，基本的逻辑控制流程还是不外乎这三种形式：条件、循环、选择。正应了那句话，简单就是美，简单就是永恒。

if 语句示例：
```
if name==null then
System.out.println("input failed");
else if name.length()==0 then
System.out.println("input name is invalid");
else
System.out.println("Input successfully");
```
while 循环示例：
```
while(rs.hasNext)
{
name=rs.getString("name");
```

```
……
if(name!=null && name.equals("sinboy"))
break;
}

for 示例：
for(int i=0;i<100;i++){
……
}
Jdk1.5 的新写法，比如遍历迭代器可以这样简化：
for(String s:sList){
System.out.println("s:"+s);
}
switch 示例：
switch(i){
case 1:System.out.println("1");break;
case 2:System.out.prntln("2");break;
default:System.out.println("other");
}
```

3.程序的作用范围（public ,protected ,private)

在 Java 里面，属性、方法、类都可声明它的作用范围，目的是为了封装，引起不必要的麻烦。比如我们去自动取款机取钱，我们只要插入银行卡，输入正确的密码，即可正常取钱。我们不需要关心自动取款机到底是怎么知道把数钱并最终把钱从取钞口给我们吐来的。

public:公有的，谁都可以使用，就象城市的公交车、大马路一样，属于公共资源，每个人都可以合法使用。在 Java 中，公有的东西，在同一个类、不同类、不同包中都可以调用。如果是子类继承，也同样为公有的。

protected:受保护的，比如象小区内的健身设备，原则上是小区内的居民才可以使用，外面的人是不准使用的。同一个类、同一个包中可以访问。子类继承？

private :私有的。象我们每个人的内裤，是绝对不充许别人用的：）。私有的东西，只能在同一个类内被使用。

default:

static:静态的，即不需要生成实例之后再去调用它。

final:常量

4.继承

继承是面向对象语言的一个重要特征。通过继承，能够象父与子、子与孙一样把一组相关的对象通过”血源关系”联系在一起。子从父继承一切公有属性，同时它还通扩展一些新的东西，或把父类的一些东西重新定义。这叫继承和发扬。在 Java 里面继承关系为树，即每个子只有一个父，但一个父可能有多个子。

继承的好处是什么呢？

可以把共性的东西抽象出来。

可以隐藏具体的实现，提高程序的灵活性。

5.多态

比如一个方法，名字叫 query(int from,int end),同时还可以定义成同样的名字参数不一样，表示的意思还是一样的，比如都是查询数据库的数据，但查询条件可能有多种多样。多态的好处是为做同样的事情定义了多种操作方式，增强的程序的灵活性、方便性。

6.接口和内隐类

接口和抽象类有点类似，都不能直接生成对象。接口规定了一个类要实现的动作，比如汽车这个接口，规定了不管是张三的宝马小汽车还是李四的解放在货车都必须能依靠燃烧燃

料牵引发动机提供运动的动力，必须能向前走、向左、向右、后退等。接口定义了游戏规则，类去按照这些规则去玩游戏。

内隐类是嵌套在一个类中的私有类，它只能在这个类里被访问，有点象临时变量的意思。

7.容器(Array,Set,Map,List)

在程序中要想保存对象，就要用到容器了。在 Java 中大概有几种，数组、Set、Map、List。

数组，比如 String[] names

Set

Map

List

8.异常处理

在程序的执行过程中，总有突发的或意想不到的事情发生，为了应用这些意外情况，我们需要做一些异常处理。就象国家为了预防某种突发事件而设置的紧急预案一样。我们的异常处理也是要在出现突发的、不可预知的事情时采用的一种手段，来保证程序的顺利执行而不是突然崩溃。

在 Java 当中，异常处理是在 try,catch 块中完成的。try 中是可能发性异常的代码，catch 中是异常处理程序。

典型的异常处理例子：

```
try{
File file=new File("test.txt");
FileOutputStream out=new FileOutputstream(file);
out.write(bytesdata);
out.flush();
}
catch(IOException e){
//异常处理……
}
```

9.Java I/O

10.分布式计算（网络编程、JDBC、Servlets、JSP、EJB）

J2EE1.4 的多层 Web 框架技术

Sun 的 Java 2 Enterprise Edition(J2EE)平台已经成为使用最广泛的 Web 程序设计技术，最近几年，J2EE Web 程序的开发已经成为信息系统的关键。

J2EE1.4 的多层 Web 框架技术

J2EE 为多层 Web 应用系统提供了容器平台。在这里，容器概念实际是指应用服务器提供的特定功能的软件模块，用户所开发的程序构件要在容器内运行，构件和容器的关系有些像计算机插件和主板的关系；程序构件在部署时被安装在容器里，容器是能提供基本功能的底层平台，它们之间通过接口进行通信；一般 Web 程序开发者只要开发出满足其需要的程序构件并能安装在容器中就够了，程序构件的安装过程包括设置各个构件在 J2EE 应用服务器中的参数以及设置 J2EE 应用服务器本身。这些设置决定了在底层由 J2EE 服务器提供的多种服务(譬如安全、交易管理、JNDI 查寻和远程调用等)。

J2EE 应用框架使同样的程序构件在一个 Web 程序之内能够根据其部署的方式实现不同的功能。例如，同样的 Enterprise JavaBean 可以采用不同等级的数据库数据存取安全设置，J2EE 容器还负责管理某些基本的服务，譬如构件的生命周期、数据库连接资源共享、数据持久性(data persistency)。

J2EE 1.4 应用平台由以下几种类型的程序容器(container)组成：Enterprise JavaBeans(EJB)容器负责所有 EJB 的运行，EJB 根据功能可以分为 session bean(通常称为会话 bean，称之为会话期间 bean 更确切)，entity bean(实体 bean)，message-drivenbean(消息驱动 bean)。这一层主要负责数据处理以及和数据库或其他 Java 程序的通信，它对应多层结构的业务层和数据访问层，Web 容器管理所有 JSP，JSTL 和 servlet 等 Web 构件的运行，这些构件主要负责程序和 Web 的通信，这一层对应多层结构中的表示层。应用客户端容器负责所有 Web 程序在客户端构件的运行；Applet 容器可以看作特殊的应用客户端容器。它负责在 Web 浏览器和 Java 插件(Java Plug-in)上运行 Java Applet 程序(Applet 是一种简化并具有安全保护的 Java 小程序)，应用客户端容器和 Applet 程序容器基本对应多层结构中的用户接口层；每种容器内都使用相关的各种 Java Web 编程技术，这些技术包括三类：
J2EE 各种不同的应用构件(如 Servlet，JSP，EJB)，它们构成了应用的主体。

J2EE 平台提供的应用服务(如 JDBC，JTS，JNDI)，这些服务保证并促进构件的良好运行。

J2EE 的应用通信技术(如 RMI，JMS，JavaMail)在平台底层实现机器和程序之间的信息传递。

1、构件技术

"构件"这一概念是指在应用程序中能发挥特定功能的软件单位。简单地说，就是几种特定的 Java 程序，这些程序有固定的格式和编写方法，它们的功能和使用方式在一定程度上被标准化了；最基本的 Java 构件是在 Java 标准版(Java 2 Standard Edition)中的 JavaBean，它是按照特定格式编写的 Java 类文件。JavaBeans 包括实例变量(Instance Variable（)和 get()，set()的方法来访问实例变量的数据。这种格式大大简化了程序设计。J2EE 的构件在 JavaBeans 基础上进行了拓展。由于 Web 编程比较复杂，J2EE 提供了更多应用构件，主要包括三类：客户端的 Applet 和程序客户；Web 容器内的 JSP，Servlet，JSTL 等构件；企业 Java Beans 容器内的 EJB 构件和资源连接构件。

1）J2EE 的客户

J2EE Web 应用可以和多种客户端程序连接。这些客户主要包括以下部分：Web 客户端包括动态生成的网页(包含各种各样的类型标注语，如 HTML，XML 等)，以及在客户机上运行的 Web 浏览器。Web 浏览器(如 Internet Explorer，Netscape)以标准格式显示从服务器传递来的网页。不管 J2EE 应用服务器早怎样生成这些网页的，它们被传递给浏览器时已经是 HTML 或 XML 格式，浏览器只是正确地显示给用户。所以，有时 Web 客户端被称为"瘦客户"，瘦客户不承担复杂的数据检索和计算任务，这些复杂而耗时的操作在 J2EE 服务器端 Web 容器和 EJB 容器内进行。这样保证了"客户-服务器"结构的优势，降低了 Web 流量。

Applet 是基于 Java 的小型客户端构件。它一般在 Web 浏览器上运行，通过 HTTP 协议和服务器进行通信。从服务器传给浏览器的网页可能包括嵌入的 Applet 程序；这些 Applet 程序在浏览器所安装的 Java 虚拟机(Java virtual machine)上执行。这要求客户机的浏览器事先安装 Java Plug-in 和有关安全许可文件。

Web 客户端程序和 Applet 各有优点。Web 客户程序(就是用网页)更简单和普遍，因为它不需要安装 Java Plug-In，也无须操心客户程序运行的安全问题。并且，Web 客户程序使编程和网页设计分离，这样保证了程序设计的模块化。会作漂亮的网页的人员因而不需要了解 Java 编程语法。Applet 更适合复杂的客户界面。

由于 Applet 更接近 Java 编程，许多专业人员更擅长利用 Java 丰富的功能(API)来进行客户端编程。如果使用得法，Applet 界面会更快速和灵活。比如，美国一家非常受欢迎的股票交易公司 Datek 就推出了使用 Applet 为界面的网上股票交易工具，无须用户点击图标和按键，它能够随时更新股票的交易分析数据和曲线。

客户应用程序是指在客户机上运行的 J2EE 程序。它为用户提供了丰富的界面(如 JavaSwing，AWT)和复杂的操作。客户应用程序直接访问在服务器 EJB 容器内的 EJB 程序。当然，J2EE 客户应用程序也可像 Applet 客户那样以 HTTP 连接和服务器的 Servlet 通信。与 Applet 不同的是，客户应用程序一般需要在客户端进行安装，而 Applet 是在 Web 上下载，无须专门安装。一般来说，客户应用程序适合在企业内部网中使用，Applet 适合在 WWW 上使用。

Java Web Start 客户是基于 JFC/Swing API 的，适合 J2EE 程序使用的客户应用，它基于比较新的 Java Web Start 技术。这种技术主要提供了网上的快捷程序下载和安装方式。Java Web Start 兼有 Applet 和客户应用程序的优点，但设置起来有些复杂。

无线客户基于移动信息设备定型技术(Mobile Information Device Profile)。Java 微型版(Java 2 Micro Edition)提供了 MIDP 的 API 和有限连接设备配置(Connected Limited Device Configuration)技术。这些技术可以使无线设备(如手机，PDA)同 J2EE 程序进行通信。

2）Web 构件

Web 构件是在 J2EE Web 容器上运行的软件程序。Web 容器主要支持多层结构的表示层。它的功能是在 HTTP 协议上对 Web 请求(request)进行响应(response)。这些所谓响应其实就是动态生成的网页。用户每在浏览器上点击一个链接或图标，实际上是通过 Web 向服务器发出请求。J2EE 平台的 Web 构件对这些请求进行处理后回复给客户相应的 HTML 或 XML 文件。

J2EE Web 构件包括 servlet，Java Server Page(JSP)和 Java Server Pages Standard Tag Library(JSTL)。

Servlet 是 Java 动态处理 HTTP 请求和生成网页的类(class)。每个 servlet 就是一个在 J2EE 应用服务器 Web 容器(又称 Web 服务器)里的程序构件。这种构件有效地利用了 Web 服务器的 HTTP 通信功能。Web 服务器负责将 Web 请求传递给 servlet。

Web 服务器内部根据用户要求的统一资源定位器(URL，即通常所说的网址)查找到对应的 servlet，然后将 servlet 处理生成的 HTML 或 XML 文件以 HTTP 形式反馈给客户。Web 程序开发人员主要编写 servlet 类程序无须关心 Web 服务器的运作细节，编写 servlet 的程序员用 servlet API 进行以下工作：初始化和结束 servlet；连接 servlet 的运行环境；接收或传递 Web 请求，发送 Web 反应；维护和管理客户会话(session)；和其他 Web 构件协同工作；使用过滤器对 Web 请求和响应进行处理；实现 Web 安全管理。

JSP 可以说是 servlet 的变形，它像是文本格式的 servlet，它的写法有些像写网页，这样就为应用开发者(特别是不熟悉 Java 语言的)提供了方便，JSP 在 Web 容器内会被自动编译为 servlet，编写 JSP 比编写 servlet 程序更简洁；一个 JSP 文件包括两类成分：生成 HTML 或 XML 模板和处理动态内容的 JSP 元素。JSP 开发者如果只改变网页外观，他们只要对 JSP 内的模板进行编辑而不用改动 JSP 元素。JSP 元素主要用于生成动态内容或调用底层 EJB 构件，Servlet 编程将二者混在一起，而 JSP 就清楚多了，JSTL 将常用的 JSP 功能封装成为简单的标签(tag)。熟悉 HTML 网页编程的人知道，网页是由各种有标签的文字组成的，各种标签(如表格、字体)的写法基本固定。JSTL 采用了相似的概念设计 JSP。例如，设计者不必自己用 JSP 写一个能连接数据库的 JSP 文件，可以用现成的 JSTL 标签来进行数据库连接。这种 JSTL 标签是最优化和标准化的，任何种类的 Web 服务器都会支持，这样就省去了不少 JSP 的麻烦；JSTL 目前提供基本的 JSP 功能，仍在不断扩充中。

3）企业 JavaBeans(EJB)构件

EJB 容器用于实现企业业务操作的程序，它在多层结构中处于业务层和数据访问层。这里我们引入"业务逻辑"这个概念。在 J2EE 编程中，业务逻辑指特殊企业领域对数据的处理需求，譬如银行业务、零售或财务等，简单说就是企业程序中的数据结构和算法。业务逻辑因企业的业务性质而异，它由 EJB 构件在 J2EE Web 程序中实现，EJB 构件能够从客户端或 Web 容器中收到数据并将处理过的数据传送到企业信息系统来存储，EJB 还能够从数据库检索数据并送回到客户端；由于 EJB 依赖 J2EE 容器进行底层操作，使用 EJB 构件编写的程序具有良好的扩展性和安全性。

J2EE 1.4 版有三种 EJB 构件：session bean(会话 bean)，entity bean(实体 bean)和 message-drivenbean(消息驱动 bean)。

会话 bean 主要用来描述程序的业务逻辑。一个会话 bean 代表 Web 应用程序和客户的一次会话过程(一次"会话")。在程序运行过程中，当 Web 应用的客户(如网上购物的消费者，银行系统使用者)执行完操作之后，会话 bean 和它所使用的数据会被删除(即不在数据库保存)。会话 bean 主要是为客户进行与业务逻辑相关的数据操作，如计算交易金额、存取数据等。会话 bean 可以是无状态的(stateless)或有状态的(stateful)。无状态是指不管任何用户每

次调用其方法，会话 bean 都作同样响应。有状态是指会话 bean 需要维护和记录不同方法之间的构件状态，这种分类主要适用不同的数据操作。

实体 bean 是用于表示和维护 Web 应用的数据实体的构件。简单地说，数据实体就是程序所使用的数据库中的数据对象。一个实体 bean 代表存放在数据库的一类数据对象。它是数据库内数据在 EJB 容器里的翻版。实体 bean 与会话 bean 不同，如果一个客户终止使用服务或 J2EE 应用服务器被关闭，EJB 容器会保证实体 bean 的数据保存到数据库内。这就是所谓数据持久性(data persistence)。实体 bean 根据其实现数据持久性的方法分为 bean-managed persistence 和 container-managed persistence 两类。Bean-managed persistence 指实体 bean 本身管理对数据库的访问，这要求编程者自己写一些数据库操作指令(如 SQL)。Container-managed persistence 指对数据库的访问由 EJB 容器负责；编程者只要定义相关设置，而不需要写数据库操作指令。虽然 container-managed persistence 更简单，但是有些复杂的数据操作还是需要 bean-managed persistence 来完成。

消息驱动 bean 实现了客户和服务器更松散的方法调用，利用消息服务器有其特定的优势，一个消息驱动 bean 能让客户和服务器之间进行异步(asynchronous)通信，服务器并不要求立刻响应；当 Java 消息服务器(Java message server)收到从客户端发来的消息时，消息驱动 bean 被激活，客户并不像使用会话 bean 那样直接调用消息驱动 bean，这样客户不必要知道消息驱动 bean 中具体有什么方法可以调用。

2、服务技术

J2EE Web 程序服务器提供了方便编程的各种服务技术，这些技术是一般 Web 应用需要用到但 Web 编程者不需要自己开发的，例如命名服务(naming service)、部署服务(deployment service)、数据连接(JDBC)、数据事务(data transaction)、安全服务(security service)和连接框架(connector architecture)，在 Web 应用中一般通过调用现成的 API 来使用这些技术。

1）命名技术(JNDI)

J2EE 命名服务提供应用构件(包括客户、EJBbeans、servlet、JSP 等)程序命名环境。在传统的面向对象编程中，如果一个类 A 要调用另一个类 B，A 需要知道 B 的源程序然后在其中 new 一个 B 的实例。当一方程序改变时，就要重新编译，而且类之间的连接比较混乱。Java Naming and Directory Interface(JNDI)(命名和目录接口)简化了高级 Web 程序类之间的查找调用。它提供了应用的命名环境(naming environment)。这就像一个公用电话簿，Web 构件在命名环境注册登记，并且通过命名环境查找所需其他构件。

JNDI API 提供了 Web 构件进行标准目录操作的方法，譬如将对象属性和 Java 对象联系在一起，或者通过对象属性来查找 Java 对象。由于 JNDI 已经被标准化，程序可以通过使用 JNDI 来访问其他通用的命名服务，包括常用的 Web 命名协议 LDAP，NDS，DNS 和 NIS。这促进了 J2EE Web 程序与其他平台系统的整合。

2）数据连接技术(JDBC)

Java Data-Base Connection(JDBC)API 使 J2EE 平台和各种关系数据库之间连接起来。JDBC 技术提供 Java 程序和数据库服务器之间的连接服务，同时它能保证数据事务的正常进行。另外，JDBC 提供了从 Java 程序内调用 SQL 数据检索语言的功能；J2EE 平台使用 JDBC 2.0 以上的 API 以及 JDBC 2.0 拓展 API，这些 API 提供了高级的数据连接功能。

3）数据事务技术

数据事务(data transaction)用于保证数据读写时不会出乱。当程序进行数据库操作时，要么成功完成，要么一点也不改变数据库数据。最怕的是把数据改了一半程序出错，那样程序和数据就会出错。所以，数据事务有一?quot;不可分微粒"的概念，就是指一次数据事务过程不能间断，J2EE 的数据事务服务保证应用程序的数据读写进程互相不干扰。

如果一个数据操作能整个完成，它就会被批准；否则，应用程序服务器就当什么都没做。应用程序开发者不用自己实现这些功能，这样使数据操作简化了，数据事务技术使用 JTA 的 API，它可以在 EJB 层或 Web 层实现。

4）安全技术

J2EE 提供了严密的安全措施，它用于保证程序资源只能由获准的用户来使用。这一般分为两步。首先是验证(authentication)，即个体必须由验证确定其身分。典型的做法是使用者提供验证数据(譬如用户名和密码)。我们称能被验证的个体为"本体"(principal)，本体可能是个人用户或其他程序。第二步是授权(authorization)。当一位被验证通过的本体设法访问程序资源时，系统要根据安全策略确定是否该本体有权限进行这样的操作。

J2EE 的容器提供两种安全方法：声明性(declarative)和程序性(programmatic)。声明性安全技术指在程序之外设定安全机制的参数。也就是编程者在程序配置描述文件里指定如何使用安全技术。相以地程序性安全技术在程序内用指令设定安全机制。这一般是指在 Web 构件或 EJB 构件的程序里调用 Java 的安全技术 API。由于声明性安全设定不需要改动程序源代码，在 J2EE 程序中，一般使用声明性安全技术比较简单。

5）连接框架技术

J2EE 连接框架技术(connector architecture)是一组用于连接 J2EE 平台到企业信息系统(EIS)的标准 API。企业信息系统是一个广义的概念，它指企业处理和存储信息数据的程序系统，譬如企业资源计划(ERP)、大型机数据事务处理以及数据库系统等。由于很多系统已经使用多年，它们不一定是标准的数据库或 Java 程序。这些现有的信息系统又称为遗产系统(legacy systems)，例如 SAP、CICS 和非关系数据库等系统。J2EE 连接框架技术解决了现有企业信息系统与 EJB 容器和构件的集成，为此，J2EE 连接框架技术定义一套扩展性强、安全的数据交互机制。这使 J2EE Web 程序能和其他类型的系统进行通话。这种技术主要是规定了 J2EE 程序服务器和遗留系统之间的资源适配器，这样使两者能够相互读懂对方的数据。

6）Web 服务技术

Web 服务技术是通过互联网进行远程应用服务和计算的新技术，被称为新一代的 Web 应用技术。在 1.4 版本之后，它已经成为 J2EE 平台的一部分，Web 服务通过基于 XML 的开放标准使企业之间进行信息连接，企业使用基于 XML 的 Web 服务描述语言(WSDL)来描述他们的 Web 服务(比如银行转账、价格查询等)；通过互联网，系统之间可以使用 Web 服务注册(如 UDDI)来查找被登记的服务目录，这样实现了真正在 Internet 上的信息查询和交换。Java 的 Web 服务主要提供系列于 XML 和 Web 服务协议有关的 API(如 JAXM，JAXP，JAXR，JAXRPC)等；在最新的 J2EE 1.4 平台内，Sun 将引进部分 Web 服务的功能。

IT 项目管理向沟通要效率

项目沟通管理是现代项目管理知识体系中的九大知识领域之一。项目沟通管理把成功所必须的因素——人、想法和信息之间提供了一个关键连接。涉及项目的任何人都应准备以项目"语言"发送和接收信息并且必须理解他们以个人身份参与的沟通怎样影响整个项目。项目管理中，因为沟通是一个是个软指标，沟通所起的作用不好量化，而沟通对项目的影响往往也是隐形的。然而，沟通对项目的成功，尤其是 IT 项目的成功太重要了。本文结合 IT 项目的特点谈谈 IT 项目中的沟通的特殊意义以及沟通对项目实施效率的影响。

一、IT 项目中沟通的特殊意义

对于项目来说，要科学地组织、指挥、协调和控制项目的实施过程，就必须进行信息沟通。沟通对项目的影响往往是潜移默化的，所以，在成功的项目中人们往往感受不到沟通所起的重要作用，在失败项目的痛苦反思中，却最能看出沟通不畅的危害。没有良好的信息沟通，对项目的发展和人际关系的改善，都会存在着制约作用。沟通失败是 IT 项目求生路上最大的拦路虎。常常能听到的典型例子是某某集团耗资几千万的 ERP 项目最终弃之不用，原因是开发出的软件不是用户所需要的，没提高用户的工作效率反而增加了工作量，不难看出，造成这种尴尬的局面的根本原因是沟通失败。当一个项目组付出极大的努力，而所做的工作却得不到客户的认可时，是否应该冷静地反思一下双方之间的沟通问题？软件项目开发中最

普遍现象是一遍一遍的返工，导致项目的成本一再加大，工期一再拖延，为什么不能一次把事情做好？原因还是沟通不到位。

通常的项目管理教材将项目沟通的重要性归结为四点：

（1）决策和计划的基础。 项目班子要想作出正确的决策，必须以准确、完整、及时的信息作为基础。通过项目内、外部环境之间的信息沟通，就可以获得众多的变化的信息，从而为决策提供依据。

（2）组织和控制管理过程的依据和手段。在项目班子内部，没有好的信息沟通，情况不明，就无法实施科学的管理。只有通过信息沟通，掌握项目班子内的各方面情况，才能为科学管理提供依据，才能有效地提高项目班子的组织效能。

（3）建立和改善人际关系是必不可少的条件。信息沟通、意见交流，将许多独立的个人、团体、组织贯通起来，成为一个整体。信息沟通是人的一种重要的心理需要，是人们用以表达思想、感情与态度，寻求同情与友谊的重要手段。畅通的信息沟通，可以减少人与人的冲突，改善人与人、人与班子之间的关系。

（4）项目经理成功领导的重要手段。项目经理是通过各种途径将意图传递给下级人员并使下级人员理解和执行。如果沟通不畅，下级人员就不能正确理解和执行领导意图，项目就不能按经理的意图进行，最终导致项目混乱甚至项目失败。因此，提高项目经理的沟通能力，与领导过程的成功性关系极大。

除以上四点外，根据 IT 项目的特点，IT 项目沟通的重要性还体现在下面两点：

（5）信息系统本身是沟通的产物。软件开发过程实际上就是将手工作业转化成计算机程序的过程。不像普通的生产加工那样有具体的有形的原料和产品，软件开发的原料和产品就是信息，中间过程间传递的也是信息，而信息的产生、收集、传播、保存正式沟通管理的内容。可见，沟通不仅仅是软件项目管理的必要手段，更重要的，沟通是软件生产的手段和生产过程中必不可少的工序。

（6）软件开发的柔性标准需要沟通来弥补。软件开发不像加工螺钉、螺母，有很具体的标准和检验方法。软件的标准柔性很大，往往在用户的心里，用户好用是软件成功的标准，而这个标准在软件开发前很难确切地、完整地表达出来。因此，开发过程项目组和用户的沟通互动是解决这一现实问题的唯一办法。

因为以上特点，可以说，在 IT 行业，沟通的成败决定整个项目的成败，沟通的效率影响整个项目的成本、进度，沟通不畅的风险是 IT 项目的最大风险之一。

二、沟通对项目实施效率的影响

沟通对项目实施效率的影响往往是间接的、不易觉察和量化的。不少项目管理者认为项目管理九大知识领域中的沟通是一个软指标，很难考核一个项目组成员的沟通能力。下面从几个与沟通有直接或间接关系因素讨论沟通对 IT 项目实施效率的影响。

项目复杂程度与实施效率

沟通路径所消耗掉的工作量多少取决于软件项目本身的复杂度和耦合度。原 IBM 在马里兰州盖兹堡的系统技术主管 Joel Aron，在他所工作过的 9 个大型项目的基础上，对程序员的实施效率进行了研究。他根据程序员和系统部分之间的交互划分这些系统，得到了如下的实施效率：

非常少的交互	10,000 指令每人年
少量的交互	5,000
较多的交互	1,500

一般说来，底层软件（操作系统、编译器、嵌入式系统、通信软件）的接口复杂度要比应用软件（MIS、操作维护软件、管理软件）要高得多。

在估算软件开发项目工作量时要充分考虑任务的类别和复杂程度，因为抽象的、接口复杂的系统开发过程中沟通消耗必然大。还有，有深厚行业背景的软件，要考虑开发人员为熟悉行业知识所要付出的沟通消耗。

团队规模与实施效率

IBM 360 操作系统之父的 F.P.布鲁克在他著的《人月神化》中提到：需要协作沟通的人员的数量影响着开发成本，因为成本的主要组成部分是相互的沟通和交流，以及更正沟通不当所引起的不良结果（系统调试）。

人与人之间必需通过沟通来解决各自承担任务之间的接口问题，如果项目有 n 个工作人员，则有 n×（n -1）/ 2 个相互沟通的路径。假设一个人单独开发软件，年实施效率为 10000 行代码，而每一条沟通路径上每年消耗掉的工作量可折合 500 行代码，则团队规模和沟通消耗以及实施效率存在以下关系：

团队规模 n	沟通路径数 n×（n -1）/ 2	沟通消耗（LOC/人年） 沟通路径数×500	实施效率（LOC/人年） 10000－沟通消耗/n
1	0	0	10000
4	6	3000	9250
6	15	7500	8750
10	45	22500	7750

由此可知，一个人单独开发一个软件，人均效率最高，只可惜大部分软件规模和时间要求都不允许一个人单独开发，而团队开发的沟通消耗却呈二次方增长。所以，项目团队应该尽可能精简，以较少的人在最可能允许的时间内完成任务是相对高效的。

团队的组织方式与实施效率

不难看出，通过减少沟通消耗、提高沟通效率能够提高项目团队工作效率。良好的团队组织可以减少不必要交流和合作的数量，是提高团队效率的关键措施。

减少交流的方法是明确的个人分工和接口定义。卡内基－梅隆大学的 D.L.Parnas 认为，编程人员仅了解自己负责的部分，而不是整个系统的开发细节时，工作效率最高。

一种行之有效的方法是改变沟通的结构和方式。比如上面的例子中，一个 10 人的项目团队，沟通路径有 10×(10-1)/2=45 条，这种计算基于一种假设，即团队中成员间的关系是对称的，各人在团队中的沟通地位完全对等，沟通方式是全通道式的。同样一个项目，

由一位系统架构师将系统分为三个相对独体的子系统，构架师负责子系统间的接口定义；然后将其余 9 人分为三个小组，每个小组负责一个子系统，小组组长和架构师相互沟通子系统间的接口，小组间的交流通过架构师组织进行；每个小组内部采用全通道式的沟通方式。那么，这样的一个组织方式沟通路径只有 9 条，沟通效率是全通道式组织方式的五倍。

当然，这种方法的先决条件是有一个对整个项目总体把握很好的软件架构师以及精确完整地定义所有接口。

团队的默契度与实施效率

很显然，团队的默契程度对软件实施效率影响很大。一个经过长期磨合、相互信任、形成一套达默契的做事方法和风格的团队，可能省掉很多不必要的沟通，相反，初次合作的团队因为团队成员各自的背景和风格不同、成员间相互信任度不高等原因，要充分考虑沟通消耗。

软件企业人员流动率高的特点导致团队凝聚力和默契度的锤炼比较困难。而凝聚力和默契度的需要长期的、大量的内部沟通和交流才能逐步形成，由此不难理解持续良好的沟通和交流是一个团队的无形资产，自然，稳定、默契的开发团队形成一个软件企业的核心竞争力

的道理。

还有一点不容忽视，那就是软件开发这种以人脑为主要工具的创造性很强的作业，开发人员的心情和兴奋度对个人工作效率影响很大，而一个人置身于氛围良好、合作默契的团队中心情一般较好，这种良好的氛围所能带来的能量是不可估量的。

三、高效沟通技巧

一个高素质的团队组织者和协调管理者所发挥的作用往往对项目的成败起决定作用，一个优秀的项目经理必然是一个善于沟通的人。沟通研究专家勒德洛（Ludlow,R.）提到：高级管理人员往往花费 80%的时间以不同的形式进行沟通，普通管理者约花 50%的时间用于传播信息 。可见，沟通的效率直接影响管理者的工作效率。介绍沟通技巧的文章很多，这里只提几种提高沟通效率的技巧。

沟通要有明确目的

沟通前，经理人员要弄清楚作这个沟通的真正目的是什么？要对方理解什么？漫无目的的沟通就是通常意义上的唠嗑，也是无效的沟通。确定了沟通目标，沟通的内容就围绕沟通要达到的目标组织规划，也可以根据不同的目的选择不同的沟通方式。

善于聆听

沟通不仅仅是说，而是说和听。一个有效的听者不仅能听懂话语本身的意思，而且能领悟说话者的言外之意。只有集中精力地听，积极投入判断思考，才能领会讲话者的意图，只有领会了讲话者的意图，才能选择合适的语言说服他。从这个意义上上讲，“听”的能力比“说”的能力更为重要。

渴望理解是人的一种本能，当讲话者感到你对他的言论很感兴趣时，他会非常高兴与你进一步加深交流。所以，有经验的聆听者通常用自己的语言向讲话者复述他所听到的，好让讲话者确信，他已经听到并理解了讲话者所说的话。

避免无休止的争论

沟通过程中不可避免地存在争论。软件项目中存在很多诸如技术、方法上的争论，这种争论往往喋喋不休，永无休止。无休止的争论当然形不成结论，而且是吞噬时间的黑洞。终结这种争论的最好办法是改变争论双方的关系。争论过程中，双方都认为自己和对方在所争论问题上地位是对等的，关系是对称的。从系统论的角度讲，争论双方形成对称系统，而对称系统是最不稳定的，而解决问题的方法在于变这种对称关系为互补关系。比如，一个人放弃自己的观点或第三方介入。

项目经理遇到这种争议中一定要发挥自己的权威性，充分利用自己对项目的决策权。

使用高效的现代化工具

电子邮件、项目管理软件等现代化工具的确可以提高沟通效率，拉近沟通双方的距离，减少不必要的面谈和会议。IT 项目的项目经理，更应该很好地运用之。

IPTV 与数字电视的异同

一般所说的 IPTV 与数字电视，既有相似点，又有区别。烽火网络公司的技术专家从以下方面阐述了二者的异同。

1．技术体系

IPTV 系统又叫交互电视，它的系统结构主要包括流媒体服务、节目采编、存储及认证计费等子系统，主要存储及传送的内容是以 MP-4 为编码核心的流媒体文件，基于 IP 网络传输，通常要在边缘设置内容分配服务节点，配置流媒体服务及存储设备，用户终端可以是 IP 机顶盒＋电视机，也可以是 PC。

有线数字电视的广播网采取的是 HFC 网络体系，与传统的模拟有线电视网络体系架构

相同，而开展新型的交互式业务情况下（如 VOD），网络体系会有所不同。有线数字电视 VOD 系统主要包括 VOD 服务、节目采编、存储及认证计费系统，主要存储及传送的内容是 MP-2TS 流，采用 IPOVERDWDM 技术，基于 DVDIP 光纤网传输，与 IPTV 的分布式架构不同，有线数字电视 VOD 系统采用的是集中式的服务架构，在 HFC 分前端并不需要配置用于内容存储及分发的视频服务器，只需要放置 DWDM 接收机及 GAM 调制等设备即可，大大降低了系统的运营成本及管理复杂度，用户终端是数字机顶盒＋电视机。目前国内已经基本形成数字电视产业链，出现了众多的数字电视机顶盒制造商、前端设备制造商、系统集成商。

2．业务内容

IPTV 有很灵活的交互特性，因为具有 IP 网的对称交互先天优势，其节目在网内，可采用广播，组播，单播多种发布方式。可以非常灵活地实现电子菜单、节目预约、实时快进、快退、终端帐号及计费管理、节目编排等多种功能。另外基于 Inter 网的其它内容业务也可以展开，如网络游戏、电子邮件、电子理财等。

有线数字电视采用广播方式，如果要实现视频点播必须将原来的 HFC 广播网络进行双向改造。一般情况下，只开通有关生活资讯的交互频道。如果数字电视要支持视频点播时，也是通过 CALBEMODEM 进入的 IP 网络来支持，实际上就是 IPTV，杭州的数字 IPTV 系统正是这个模式。从提供的内容服务上看，有线数字电视不如 IPTV。

3．主要优势

IPTV 的主要卖点是交互，及 Inter 网内业务的扩充。IPTV 还可以非常容易地将电视服务和互联网浏览、电子邮件，以及多种在线信息咨询、娱乐、教育及商务功能结合在一起，在未来的竞争中处于优势地位。

数字电视的卖点主要为高清的图像质量。

4．用户群

两者的市场用户群都是家庭用户，只是一个依托有线，一个依托宽带，给用户带来的利益类似。

5．发展前景

在很长一段时间内会出现两者并存的状况。发展数字电视是国家早就计划的政策，IPTV 是在众多的电视节目中增加一个节目频道，并不代替有线数字电视。因为 IPTV 的实时性广播有一定的使用成本，所以完全用 IPTV 代替掉有线或卫星电视意义并不大。但是，因为 IPTV 的众多吸引人们的功能，它作为一个独立的节目频道还是十分有生命力。从信息产业发展角度看，IPTV 还是三网合一的最大切入点。

IPSec 基础(一)—IPSec 服务

IPSec 协议不是一个单独的协议，它给出了应用于 IP 层上网络数据安全的一整套体系结构，包括网络认证协议 Authentication Header（AH）、封装安全载荷协议 Encapsulating Security Payload（ESP）、密钥管理协议 Internet Key Exchange （IKE）和用于网络认证及加密的一些算法等。IPSec 规定了如何在对等层之间选择安全协议、确定安全算法和密钥交换，向上提供了访问控制、数据源认证、数据加密等网络安全服务。

一、安全特性

IPSec 的安全特性主要有：

·不可否认性 "不可否认性"可以证实消息发送方是唯一可能的发送者，发送者不能否认发送过消息。"不可否认性"是采用公钥技术的一个特征，当使用公钥技术时，发送方用私钥产生一个数字签名随消息一起发送，接收方用发送者的公钥来验证数字签名。由于在理论上只有发送者才唯一拥有私钥，也只有发送者才可能产生该数字签名，所以只要数字签名通过验证，发送者就不能否认曾发送过该消息。但"不可否认性"不是基于认证的共享密钥技术的

特征，因为在基于认证的共享密钥技术中，发送方和接收方掌握相同的密钥。

·反重播性 "反重播"确保每个 IP 包的唯一性，保证信息万一被截取复制后，不能再被重新利用、重新传输回目的地址。该特性可以防止攻击者截取破译信息后，再用相同的信息包冒取非法访问权（即使这种冒取行为发生在数月之后）。

·数据完整性 防止传输过程中数据被篡改，确保发出数据和接收数据的一致性。IPSec 利用 Hash 函数为每个数据包产生一个加密检查和，接收方在打开包前先计算检查和，若包遭篡改导致检查和不相符，数据包即被丢弃。

·数据可靠性（加密） 在传输前，对数据进行加密，可以保证在传输过程中，即使数据包遭截取，信息也无法被读。该特性在 IPSec 中为可选项，与 IPSec 策略的具体设置相关。

·认证 数据源发送信任状，由接收方验证信任状的合法性，只有通过认证的系统才可以建立通信连接。

二、基于电子证书的公钥认证

一个架构良好的公钥体系，在信任状的传递中不造成任何信息外泄，能解决很多安全问题。IPSec 与特定的公钥体系相结合，可以提供基于电子证书的认证。公钥证书认证在 Windows 2000 中，适用于对非 Windows 2000 主机、独立主机，非信任域成员的客户机、或者不运行 Kerberos v5 认证协议的主机进行身份认证。

三、预置共享密钥认证

IPSec 也可以使用预置共享密钥进行认证。预共享意味着通信双方必须在 IPSec 策略设置中就共享的密钥达成一致。之后在安全协商过程中，信息在传输前使用共享密钥加密，接收端使用同样的密钥解密，如果接收方能够解密，即被认为可以通过认证。但在 Windows 2000 IPSec 策略中，这种认证方式被认为不够安全而一般不推荐使用。

四、公钥加密

IPSec 的公钥加密用于身份认证和密钥交换。公钥加密，也被称为"不对称加密法"，即加解密过程需要两把不同的密钥，一把用来产生数字签名和加密数据，另一把用来验证数字签名和对数据进行解密。

使用公钥加密法，每个用户拥有一个密钥对，其中私钥仅为其个人所知，公钥则可分发给任意需要与之进行加密通信的人。例如：A 想要发送加密信息给 B，则 A 需要用 B 的公钥加密信息，之后只有 B 才能用他的私钥对该加密信息进行解密。虽然密钥对中两把钥匙彼此相关，但要想从其中一把来推导出另一把，以目前计算机的运算能力来看，这种做法几乎完全不现实。因此，在这种加密法中，公钥可以广为分发，而私钥则需要仔细地妥善保管。

五、Hash 函数和数据完整性

Hash 信息验证码 HMAC（Hash message authentication codes）验证接收消息和发送消息的完全一致性（完整性）。这在数据交换中非常关键，尤其当传输媒介如公共网络中不提供安全保证时更显其重要性。

HMAC 结合 hash 算法和共享密钥提供完整性。Hash 散列通常也被当成是数字签名，但这种说法不够准确，两者的区别在于：Hash 散列使用共享密钥，而数字签名基于公钥技术。hash 算法也称为消息摘要或单向转换。称它为单向转换是因为：

1）双方必须在通信的两个端头处各自执行 Hash 函数计算；

2）使用 Hash 函数很容易从消息计算出消息摘要，但其逆向反演过程以目前计算机的运算能力几乎不可实现。

Hash 散列本身就是所谓加密检查和或消息完整性编码 MIC（Message Integrity Code），通信双方必须各自执行函数计算来验证消息。举例来说，发送方首先使用 HMAC 算法和共享密钥计算消息检查和，然后将计算结果 A 封装进数据包中一起发送；接收方再对所接收的消息执行 HMAC 计算得出结果 B，并将 B 与 A 进行比较。如果消息在传输中遭篡改致使 B 与 A 不一致，接收方丢弃该数据包。

有两种最常用的 hash 函数：

·HMAC-MD5 MD5（消息摘要 5）基于 RFC1321。MD5 对 MD4 做了改进，计算速度比 MD4 稍慢，但安全性能得到了进一步改善。MD5 在计算中使用了 64 个 32 位常数，最终生成一个 128 位的完整性检查和。

·HMAC-SHA 安全 Hash 算法定义在 NIST FIPS 180-1，其算法以 MD5 为原型。 SHA 在计算中使用了 79 个 32 位常数，最终产生一个 160 位完整性检查和。SHA 检查和长度比 MD5 更长，因此安全性也更高。

六、加密和数据可靠性

IPSec 使用的数据加密算法是 DES--Data Encryption Standard（数据加密标准）。DES 密钥长度为 56 位，在形式上是一个 64 位数。DES 以 64 位（8 字节）为分组对数据加密，每 64 位明文，经过 16 轮置换生成 64 位密文，其中每字节有 1 位用于奇偶校验，所以实际有效密钥长度是 56 位。 IPSec 还支持 3DES 算法，3DES 可提供更高的安全性，但相应地，计算速度更慢。

七、密钥管理

·动态密钥更新

IPSec 策略使用"动态密钥更新"法来决定在一次通信中，新密钥产生的频率。动态密钥指在通信过程中，数据流被划分成一个个"数据块"，每一个"数据块"都使用不同的密钥加密，这可以保证万一攻击者中途截取了部分通信数据流和相应的密钥后，也不会危及到所有其余的通信信息的安全。动态密钥更新服务由 Internet 密钥交换 IKE（Internet Key Exchange）提供，详见 IKE 介绍部分。

IPSec 策略允许专家级用户自定义密钥生命周期。如果该值没有设置，则按缺省时间间隔自动生成新密钥。

·密钥长度

密钥长度每增加一位，可能的密钥数就会增加一倍，相应地，破解密钥的难度也会随之成指数级加大。IPSec 策略提供多种加密算法，可生成多种长度不等的密钥，用户可根据不同的安全需求加以选择。

·Diffie-Hellman 算法

要启动安全通讯，通信两端必须首先得到相同的共享密钥（主密钥），但共享密钥不能通过网络相互发送，因为这种做法极易泄密。

Diffie-Hellman 算法是用于密钥交换的最早最安全的算法之一。DH 算法的基本工作原理是：通信双方公开或半公开交换一些准备用来生成密钥的"材料数据"，在彼此交换过密钥生成"材料"后，两端可以各自生成出完全一样的共享密钥。在任何时候，双方都绝不交换真正的密钥。

通信双方交换的密钥生成"材料"，长度不等，"材料"长度越长，所生成的密钥强度也就越高，密钥破译就越困难。 除进行密钥交换外，IPSec 还使用 DH 算法生成所有其他加密密钥。

IPSec 基础（二）——IPSec 协议类型

IPSec 提供了两种安全机制：认证和加密。认证机制使 IP 通信的数据接收方能够确认数据发送方的真实身份以及数据在传输过程中是否遭篡改。加密机制通过对数据进行编码来保证数据的机密性，以防数据在传输过程中被窃听。IPSec 协议组包含 Authentication Header（AH）协议、Encapsulating Security Payload（ESP）协议和 Internet Key Exchange（IKE）协议。其中 AH 协议定义了认证的应用方法，提供数据源认证和完整性保证；ESP 协议定义了加密和可选认证的应用方法，提供可靠性保证。在实际进行 IP 通信时，可以根据实际安全需求

同时使用这两种协议或选择使用其中的一种。AH 和 ESP 都可以提供认证服务，不过，AH 提供的认证服务要强于 ESP。IKE 用于密钥交换（将在以后部分讨论）。

一、Authentication Header（AH）协议结构

AH 协议为 IP 通信提供数据源认证、数据完整性和反重播保证，它能保护通信免受篡改，但不能防止窃听，适合用于传输非机密数据。AH 的工作原理是在每一个数据包上添加一个身份验证报头。此报头包含一个带密钥的 hash 散列（可以将其当作数字签名，只是它不使用证书），此 hash 散列在整个数据包中计算，因此对数据的任何更改将致使散列无效--这样就提供了完整性保护。

AH 报头位置在 IP 报头和传输层协议报头之间，见图一。 AH 由 IP 协议号" 51"标识，该值包含在 AH 报头之前的协议报头中，如 IP 报头。AH 可以单独使用，也可以与 ESP 协议结合使用。

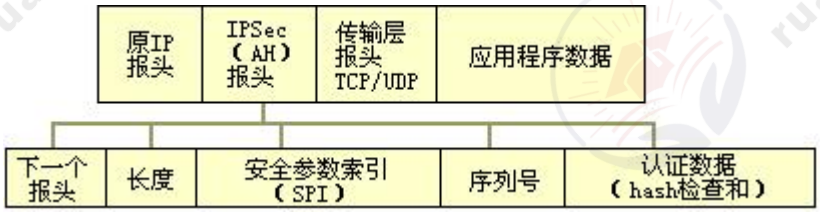

图 1 AH 报头

AH 报头字段包括：

·Next Header（下一个报头）： 识别下一个使用 IP 协议号的报头，例如，Next Header 值等于"6"，表示紧接其后的是 TCP 报头。

·Length（长度）： AH 报头长度。

·Security Parameters Index (SPI，安全参数索引)： 这是一个为数据报识别安全关联的 32 位伪随机值。SPI 值 0 被保留来表明"没有安全关联存在"。

·Sequence Number（序列号）：从 1 开始的 32 位单增序列号，不允许重复，唯一地标识了每一个发送数据包，为安全关联提供反重播保护。接收端校验序列号为该字段值的数据包是否已经被接收过，若是，则拒收该数据包。

·Authentication Data（AD，认证数据）： 包含完整性检查和。接收端接收数据包后，首先执行 hash 计算，再与发送端所计算的该字段值比较，若两者相等，表示数据完整，若在传输过程中数据遭修改，两个计算结果不一致，则丢弃该数据包。

数据包完整性检查：

如图二所示，AH 报头插在 IP 报头之后，TCP，UDP，或者 ICMP 等上层协议报头之前。一般 AH 为整个数据包提供完整性检查，但如果 IP 报头中包含"生存期（Time To Live）"或"服务类型（Type of Service）"等值可变字段，则在进行完整性检查时应将这些值可变字段去除。

图 2 AH 为整个数据包提供完整性检查

二、Encapsulating Security Payload（ESP）协议结构

ESP 为 IP 数据包提供完整性检查、认证和加密，可以看作是"超级 AH"， 因为它提供机密性并可防止篡改。ESP 服务依据建立的安全关联（SA）是可选的。然而，也有一些限制：

·完整性检查和认证一起进行。

·仅当与完整性检查和认证一起时，"重播（Replay）"保护才是可选的。

·"重播"保护只能由接收方选择。

ESP 的加密服务是可选的，但如果启用加密，则也就同时选择了完整性检查和认证。因为如果仅使用加密，入侵者就可能伪造包以发动密码分析攻击。

ESP 可以单独使用，也可以和 AH 结合使用。一般 ESP 不对整个数据包加密，而是只加密 IP 包的有效载荷部分，不包括 IP 头。但在端对端的隧道通信中，ESP 需要对整个数据包加密。

如图三所示，ESP 报头插在 IP 报头之后，TCP 或 UDP 等传输层协议报头之前。ESP 由 IP 协议号"50"标识。

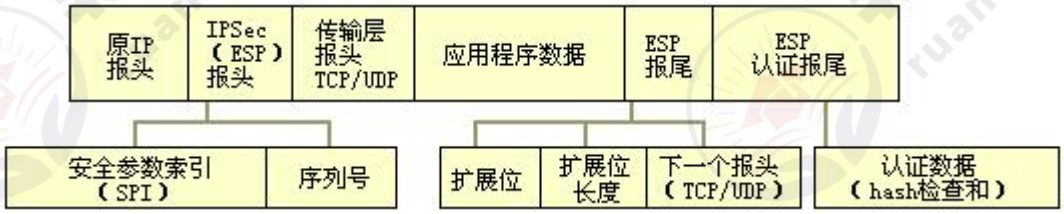

图 3 ESP 报头、报尾和认证报尾

ESP 报头字段包括：

·Security Parameters Index (SPI，安全参数索引)：为数据包识别安全关联。

·Sequence Number（序列号）：从 1 开始的 32 位单增序列号，不允许重复，唯一地标识了每一个发送数据包，为安全关联提供反重播保护。接收端校验序列号为该字段值的数据包是否已经被接收过，若是，则拒收该数据包。 ESP 报尾字段包括：

·Padding（扩展位）：0-255 个字节。DH 算法要求数据长度（以位为单位）模 512 为 448，若应用数据长度不足，则用扩展位填充。

·Padding Length（扩展位长度）：接收端根据该字段长度去除数据中扩展位。

·Next Header（下一个报头）：识别下一个使用 IP 协议号的报头，如 TCP 或 UDP。

ESP 认证报尾字段：

·Authentication Data（AD，认证数据）： 包含完整性检查和。完整性检查部分包括 ESP 报头、有效载荷（应用程序数据）和 ESP 报尾。见图四。

图 4 ESP 的加密部分和完整性检查部分

如上图所示，ESP 报头的位置在 IP 报头之后，TCP，UDP，或者 ICMP 等传输层协议报头之前。如果已经有其他 IPSec 协议使用，则 ESP 报头应插在其他任何 IPSec 协议报头之前。ESP 认证报尾的完整性检查部分包括 ESP 报头、传输层协议报头，应用数据和 ESP 报尾，但不包括 IP 报头，因此 ESP 不能保证 IP 报头不被篡改。ESP 加密部分包括上层传输协议信息、数据和 ESP 报尾。

三、ESP 隧道模式和 AH 隧道模式

以上介绍的是传输模式下的 AH 协议和 ESP 协议，ESP 隧道模式和 AH 隧道模式与传输模式略有不同。

在隧道模式下，整个原数据包被当作有效载荷封装了起来，外面附上新的 IP 报头。其中"内部"IP 报头（原 IP 报头）指定最终的信源和信宿地址，而"外部"IP 报头（新 IP 报头）中包含的常常是做中间处理的安全网关地址。

与传输模式不同，在隧道模式中，原 IP 地址被当作有效载荷的一部分受到 IPSec 的安全保护，另外，通过对数据加密，还可以将数据包目的地址隐藏起来，这样更有助于保护端对端隧道通信中数据的安全性。

ESP 隧道模式中签名部分（完整性检查和认证部分）和加密部分分别如图所示。ESP 的签名不包括新 IP 头。

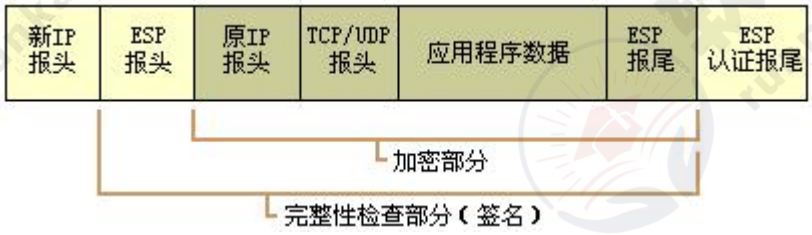

图 5 ESP 隧道模式

下图标示出了 AH 隧道模式中的签名部分。AH 隧道模式为整个数据包提供完整性检查和认证，认证功能优于 ESP。但在隧道技术中，AH 协议很少单独实现，通常与 ESP 协议组合使用。

图 6 AH 隧道模式

IPSec 基础（三）——密钥交换和密钥保护

Internet 密钥交换（IKE）

两台 IPSec 计算机在交换数据之前，必须首先建立某种约定，这种约定，称为"安全关联"，指双方需要就如何保护信息、交换信息等公用的安全设置达成一致，更重要的是，必须有一种方法，使那两台计算机安全地交换一套密钥，以便在它们的连接中使用。见图七。

图七　Internet 密钥交换

Internet 工程任务组 IETF 制定的安全关联标准法和密钥交换解决方案--IKE（Internet 密钥交换）负责这些任务，它提供一种方法供两台计算机建立安全关联 (SA)。SA 对两台计算机之间的策略协议进行编码，指定它们将使用哪些算法和什么样的密钥长度，以及实际的密钥本身。IKE 主要完成两个作用：

·安全关联的集中化管理，减少连接时间
·密钥的生成和管理

一、什么是 SA?

安全关联 SA（Security Association）是单向的，在两个使用 IPSec 的实体（主机或路由器）间建立的逻辑连接，定义了实体间如何使用安全服务（如加密）进行通信。它由下列元素组成：1）安全参数索引 SPI；2）IP 目的地址；3）安全协议。

SA 是一个单向的逻辑连接，也就是说，在一次通信中，IPSec 需要建立两个 SA，一个用于入站通信，另一个用于出站通信。若某台主机，如文件服务器或远程访问服务器，需要同时与多台客户机通信，则该服务器需要与每台客户机分别建立不同的 SA。每个 SA 用唯一的 SPI 索引标识，当处理接收数据包时，服务器根据 SPI 值来决定该使用哪种 SA。

二、第一阶段 SA（主模式 SA，为建立信道而进行的安全关联）

IKE 建立 SA 分两个阶段。第一阶段，协商创建一个通信信道（IKE SA），并对该信道进行认证，为双方进一步的 IKE 通信提供机密性、数据完整性以及数据源认证服务；第二阶段，使用已建立的 IKE SA 建立 IPsec SA。分两个阶段来完成这些服务有助于提高密钥交换的速度。第一阶段协商（主模式协商）步骤：

1．策略协商，在这一步中，就四个强制性参数值进行协商：
1）加密算法：选择 DES 或 3DES
2）hash 算法：选择 MD5 或 SHA
3）认证方法：选择证书认证、预置共享密钥认证或 Kerberos v5 认证
4）Diffie-Hellman 组的选择

2．DH 交换

虽然名为"密钥交换"，但事实上在任何时候，两台通信主机之间都不会交换真正的密钥，它们之间交换的只是一些 DH 算法生成共享密钥所需要的基本材料信息。DH 交换，可以是公开的，也可以受保护。在彼此交换过密钥生成"材料"后，两端主机可以各自生成出完全一样的共享"主密钥"，保护紧接其后的认证过程。

3．认证 DH 交换需要得到进一步认证，如果认证不成功，通信将无法继续下去。"主密钥"结合在第一步中确定的协商算法，对通信实体和通信信道进行认证。在这一步中，整个待认证的实体载荷，包括实体类型、端口号和协议，均由前一步生成的"主密钥"提供机密性和完整性保证。

三、第二阶段 SA（快速模式 SA，为数据传输而建立的安全关联）

这一阶段协商建立 IPsec SA，为数据交换提供 IPSec 服务。第二阶段协商消息受第一阶段 SA 保护，任何没有第一阶段 SA 保护的消息将被拒收。

第二阶段协商（快速模式协商）步骤：

1．策略协商，双方交换保护需求：
·使用哪种 IPSec 协议：AH 或 ESP
·使用哪种 hash 算法：MD5 或 SHA
·是否要求加密，若是，选择加密算法：3DES 或 DES 在上述三方面达成一致后，将建立起两个 SA，分别用于入站和出站通信。

2．会话密钥"材料"刷新或交换

在这一步中，将生成加密 IP 数据包的"会话密钥"。生成"会话密钥"所使用的"材料"可以和生成第一阶段 SA 中"主密钥"的相同，也可以不同。如果不做特殊要求，只需要刷新"材料"后，生成新密钥即可。若要求使用不同的"材料"，则在密钥生成之前，首先进行第二轮的 DH 交换。

3．SA 和密钥连同 SPI，递交给 IPSec 驱动程序。

第二阶段协商过程与第一阶段协商过程类似，不同之处在于：在第二阶段中，如果响应超时，则自动尝试重新进行第一阶段 SA 协商。

第一阶段 SA 建立起安全通信信道后保存在高速缓存中，在此基础上可以建立多个第二阶段 SA 协商，从而提高整个建立 SA 过程的速度。只要第一阶段 SA 不超时，就不必重复第一阶段的协商和认证。允许建立的第二阶段 SA 的个数由 IPSec 策略属性决定。

四、SA 生命期

第一阶段 SA 有一个缺省有效时间，如果 SA 超时，或"主密钥"和"会话密钥"中任何一个生命期时间到，都要向对方发送第一阶段 SA 删除消息，通知对方第一阶段 SA 已经过期。之后需要重新进行 SA 协商。第二阶段 SA 的有效时间由 IPSec 驱动程序决定。

密钥保护

一、密钥生命期

生命期设置决定何时生成新密钥。在一定的时间间隔内重新生成新密钥的过程称为"动态密钥更新"或"密钥重新生成"。密钥生命期设置决定了在特定的时间间隔之后，将强制生成新密钥。例如，假设一次通信需要 1 万秒，而我们设定密钥生命期为 1 千秒，则在整个数据传输期间将生成 10 个密钥。在一次通信中使用多个密钥保证了即使攻击者截取了单个通信密钥，也不会危及全部通信安全。密钥生命期有一个缺省值，但"主密钥"和"会话密钥"生命期都可以通过配置修改。无论是哪种密钥生命期时间到，都要重新进行 SA 协商。单个密钥所能处理的最大数据量不允许超过 100 兆。

二、会话密钥更新限制

反复地从同一个的"主密钥"生成材料去生成新的"会话密钥"很可能会造成密钥泄密。"会话密钥更新限制"功能可以有效地减少泄密的可能性。 例如，两台主机建立安全关联后，A 先向 B 发送某条消息，间隔数分钟后再向 B 发送另一条消息。由于新的 SA 刚建立不久，因此两条消息所用的加密密钥很可能是用同一"材料"生成的。如果想限制某密钥"材料"重用次数，可以设定"会话密钥更新限制"。譬如，设定"会话密钥更新限制"为 5，意味着同一"材料"最多只能生成 5 个"会话密钥"。

若启用"主密钥精确转发保密（PFS）"，则"会话密钥更新限制"将被忽略，因为 PFS 每次都强制使用新"材料"重新生成密钥。将"会话密钥更新限制"设定为 1 和启用 PFS 效果是一样的。如果既设定了"主密钥"生命期，又设定了"会话密钥更新限制"，那么无论哪个限制条件先满足，都引发新一轮 SA 协商。在缺省情况下，IPSec 不设定"会话密钥更新限制"。

三、Diffie-Hellman（DH）组

DH 组决定 DH 交换中密钥生成"材料"的长度。密钥的牢固性部分决定于 DH 组的强度。IKE 共定义了 5 个 DH 组，组 1（低）定义的密钥"材料"长度为 768 位；组 2（中）长度为 1024 位。密钥"材料"长度越长，所生成的密钥安全度也就越高，越难被破译。

DH 组的选择很重要，因为 DH 组只在第一阶段的 SA 协商中确定，第二阶段的协商不再重新选择 DH 组，两个阶段使用的是同一个 DH 组，因此该 DH 组的选择将影响所有"会话密钥"的生成。

在协商过程中，对等的实体间应选择同一个 DH 组，即密钥"材料"长度应该相等。若 DH 组不匹配，将视为协商失败。

四、精确转发保密 PFS（Perfect Forward Secrecy）

与密钥生命期不同，PFS 决定新密钥的生成方式，而不是新密钥的生成时间。PFS 保证无论在哪一阶段，一个密钥只能使用一次，而且，生成密钥的"材料"也只能使用一次。某个"材料"在生成了一个密钥后，即被弃，绝不用来再生成任何其他密钥。这样可以确保一旦单个密钥泄密，最多只可能影响用该密钥加密的数据，而不会危及整个通信。

PFS 分"主密钥"PFS 和"会话密钥"PFS，启用"主密钥"PFS，IKE 必须对通信实体进行重新认证，即一个 IKE SA 只能创建一个 IPsec SA，对每一次第二阶段 SA 的协商，"主密钥"PFS 都要求新的第一阶段协商，这将会带来额外的系统开销。因此使用它要格外小心。

然而，启用"会话密钥"PFS，可以不必重新认证，因此对系统资源要求较小。"会话密钥"PFS 只要求为新密钥生成进行新的 DH 交换，即需要发送四个额外消息，但无须重新认证。PFS 不属于协商属性，不要求通信双方同时开启 PFS。"主密钥"PFS 和"会话密钥"PFS 均可以各自独立设置。

Flash 加载外部文件的各种方法与技巧

Flash 可以通过帧、按扭、影片剪辑来调用外部文件。调用的外部文件包括：外部文本文件、外部程序文件、外部*.swf 文件、外部图片文件、外部音乐文件、外部脚本文件等。

现在我们将 Flash 加载外部文件的各种方法与技巧总结如下。

一、用 loadVariables 调用外部文本文件

文本文件开头要以 flash 中的动态文本框的变量名开头，如 msg="....."这样的形式，要和编辑的 Flash 文件放在同一目录下。

具体调用方式如下：

1.用工具箱中的文本工具，选择动态文本，给动态文本一个变量名，如:msg,在编辑区拖出一个文本框

2.制作两个按扭（一个调用，一个清除）拖放到场景中。

3.在调用按扭上添加 AS 代码：

```
on(release){//松开鼠标后执行下面的代码；
   loadVariables("msg.txt",msg);//调用和你编辑的 Flash 处于同一目录下的 msg.txt 文本文件到动态文本 msg 中；
   System.useCodepage=true;//使外部文件的中文字符能够正确显示；
}
```

4.清除按扭上的 AS 代码：

```
on(release){//松开鼠标后清楚动态文本框中的内容；
      msg="";
}
```

Ctrl+Enter 测试.

当然代码也可以写在帧上:(调用按扭的实例名是 bt1,清除按扭的实例名是 bt2)

第一帧上加如下代码：

```
stop();
_root.bt1.onRelease=function(){
loadVariables("msg.txt",msg);
System.useCodepage=true;
}
_root.bt2.onRelease=function(){
_root.msg="";
}
```

Ctrl+Enter 测试.

进阶：让调用的外部文本文件能够滚动

1.用工具箱中的文本工具，选择动态文本，给动态文本一个变量名，如:msg,在编辑区拖出一个文本框

2.新建一层,制作两个按扭（一个调用，一个清除）拖放到此层中

3.新建一层,在动态文本框的右侧用矩形工具画一个竖条,高度和动态文本一样;再制作一个向上的方向"箭头"按 F8 转为按扭元件;复制一个"箭头"按扭,垂直镜象,使这两个按扭和竖条的两端对齐.

4.调用按扭上的 AS 代码：

```
on(release){//松开鼠标后执行下面的代码：
   loadVariables("msg1.txt",msg);//调用 msg1.txt 文本文件到动态文本框 msg 中；
   System.useCodepage=true;//使外部文件的中文字符能够正确显示；
}
```

清除按扭上的 AS 代码：

```
on(release){
   _root.msg="";//清除动态文本框中的内容;
}
```

向上按扭上的 AS 代码：

```
on(press){//在按扭的感应区上每按一次鼠标执行下面的代码;
   _root.msg.scroll=_root.msg.scroll-1;//文本向下滚动一行;
}
```

向下按扭上的 AS 代码：

```
on(press){//在按扭的感应区上每按一次鼠标执行下面的代码;
   _root.msg.scroll=_root.msg.scroll+1;//文本向上滚动一行;
}
```

Ctrl+Enter 测试.

当然代码也可以写在帧上:(调用按扭的实例名是 bt1,清除按扭的实例名是 bt2,向上按扭的实例名是:up,向下按扭上的实例名是 down)

在第一帧上加如下代码：

```
_root.bt1.onRelease=function(){//松开鼠标后执行下面的代码；
   loadVariables("msg1.txt",msg);//调用 msg1.txt 文本文件到动态文本框 msg 中；
   System.useCodepage=true;//使外部文件的中文字符能够正确显示；
}
_root.bt2.onRelease=function(){//松开鼠标后执行
   _root.msg="";//清除动态文本框中的内容;
}
_root.up.onPress=function(){//在向上按扭上每按一次鼠标执行下面的代码;
   _root.msg.scroll=_root.msg.scroll-1;//文本向下滚动一行;
}
_root.down.onPress=function(){//在向下按扭上每按一次鼠标执行下面的代码;
_   root.msg.scroll=_root.msg.scroll+1;//文本向上滚动一行;
}
```

Ctrl+Enter 测试。

二、用 loadMovie 调用外部*.swf 文件

（一）调用外部*.swf 文件加载到影片剪辑中

外部*.swf 文件要和编辑的 Flash 文件放在同一目录下

1.新建立一个空的影片剪辑 mymc,把它放在场景中，实例名是：mymc.

2.新建一层,制作两个按扭（一个调用，一个清除）拖放到此层中

3.调用按扭上的 AS 代码：

```
on(release){//鼠标离开按扭后执行下面的代码；
   loadMovie("flash8.swf","mymc");//加载外部的"flash8.swf"文件到"mymc"空影片剪辑中；
   mymc._x=70;//加载影片的 X 轴坐标；
   mymc._y=20;//加载影片的 Y 轴坐标；
   mymc._xscale=70;//加载影片的宽度；
   mymc._yscale=70;//加载影片的高度；
}
```

清除按扭上的 AS 代码：

```
on(release){//鼠标离开按扭后执行下面的代码
   unloadMovie(mymc);//删除用 loadMovie 加载的*.swf 文件；
}
```

Ctrl+Enter 测试

（二）调用外部*.swf 文件并加载到时间轴上

外部*.swf 文件要和编辑的 Flash 文件放在同一目录下

1.制作两个按扭（一个调用，一个清除）拖放到场景中

2.调用按扭上的 AS 代码：

```
on(release){//鼠标离开按扭后执行下面的代码
   loadMovie("flash8.swf",1);//加载外部的"flash8.swf"文件到场景中，层深为 1;
}
```

清除按扭上的 AS 代码：

```
on(release){//鼠标离开按扭后执行下面的代码
   unloadMovie(1);//删除层深为 1 的用 loadMovie 所加载的"flash8.swf"文件
}
```

Ctrl+Enter 测试 。

当然二和三的代码都可以写在帧上。

三、用 loadMovie 调用外部图片，加载到影片剪辑中

将外部图片必须和正在编辑的 Flash 文件放在同一目录下

1.制作两个按扭（一个调用，一个清除）拖放到场景中

2.制作一个空的影片剪辑，拖到场景中，实例名是：mymc;

3.时间轴上第一帧上的 AS 如下：

```
i=0;//定义一个变量 i，并且赋初值为 0；
```

调用按扭上的 AS 代码：

```
on(release){//鼠标离开按扭后执行下面的代码；
   i++;
   if(i>9){//因为外面这有 9 张图，当变量大于 9 的时候，让变量为 1，这样能够使加载的图片是连续的，即，每点一次按扭，就换一张图，等到换到第 9 张图，再点按扭，则循环到第一张图（j1.jpg 到 j9.jpg,因为没有 j0.jpg,如果 i=0，则会提示没有找到 j0.jpg);
      i=1;
   }
   loadMovie(("j"+i)+".jgp",mymc);//从 j1.jpg 开始加载图片到影片剪辑 mymc 中；
   mymc._x=110;//以下设置加载图片的属性
   mymc._y=35;
   mymc._xscale=130;
   mymc._yscale=130;
}
```

清除按扭上的 AS 代码：

```
on(release){
   unloadMovie(mymc);//删除掉用 loadMovie 加载到影片剪辑的图片；
}
```

Ctrl+Enter 测试。当然 AS 代码也可以写在帧上。

四、用 mySound.loadSound 调用外部声音文件，加载到场景中

外部声音文件必须和正在编辑的 Flash 文件放在同一目录下

1.制作两个按扭（一个调用，一个清除）拖放到场景中

2.调用按扭上的 AS 代码：

```
on(release){//鼠标离开按扭后执行下面的代码；
   mySound=new Sound();//建立一个新的声音对象 mySound;
    mySound.loadSound("zaihuni.mp3〞 ,true);// 加载外部的*.mp3 声音文件到 mySound 对象中，并且按流的方式播放（参数为 false 时，是以装载完后播放)；
}
```

清除按扭上的 AS 代码：

```
on(release){
   mySound.stop();//当按下清除按扭后，停止声音的播放
}
```

Ctrl+Enter 测试。

五、用 loadMovieNum()函数调用所有外部文件

（一）loadMovieNum()函数详解

用法：loadMovieNum("url",level [, variables])

功能：在播放原来加载的 SWF 文件的同时将 SWF 文件或 JPEG 文件加载到 Flash Player 中的某个级别。

参数：该函数有 3 个参数：url、target、variables。variables 是可选参数。

1.参数 url：要加载的 SWF 文件或 JPEG 文件的绝对或相对 URL（路径）。该函数的此参数和上面 loadMovie 中的此参数用法完全一致，这里不再作解释。

2.参数 level：一个整数，指定 SWF 文件将加载到 Flash Player 中的哪个级别。加载的时候，可以这样来写：

```
loadMovieNum("01.swf", 1);
 loadMovieNum("02.swf", 2);
 loadMovieNum("03.swf", 3);
```

loadMovieNum 加载后的控制可以这样使用：

```
_level1._x=10 ;
 _level2.aa._alpha=50 ;
 _level3.aa.bb._width=110;
 _level4.mysound.stop();// 加载到级别为 4 的对象 MC:mysound 停止（播放音乐）
```

需要注意的的是，每一个级别只能同时存在一个 SWF 或 JPEG 文件。如果两个 SWF 或 JPEG 文件的级别相同，那么后者将替换掉前者。级别不同的_level，级别大的将覆盖掉级别小的，即：数字大的将处于数字小的之上(如上例：03.swf 处在 02.swf 和 01.swf 上方，02.swf 处在 01.swf 上方) 。

注意：如果将 SWF 文件加载到级别 0，则 Flash Player 中的每个级别均被卸载，并且级别 0 将替换为该新文件。处于级别 0 的 SWF 文件为所有其它加载的 SWF 文件设置帧频、背景色和帧大小。如：

```
loadMovieNum("00.swf", 0); //以下均不显示，这样只有一个 00.swf
loadMovieNum("01.swf", 1);
loadMovieNum("02.swf", 2);
loadMovieNum("03.swf", 3);
```

3.参数 variables：可选参数，指定发送变量所使用的 HTTP 方法。该参数必须是字符串 GET 或 POST。如果没有要发送的变量，则省略此参数。GET 方法将变量追加到 URL 的末尾，它用于发送少量的变量。POST 方法在单独的 HTTP 标头中发送变量，它用于发送大量的变量。

（二）loadMovieNum()的卸载外部文件调用

使用 unloadMovieNum() 可删除用 loadMovieNum() 加载的 SWF 文件或图像。

用法: unloadMovieNum(level)

参数: level 所加载影片的级别 (_levelN)。

如：

```
on(release){
        unloadMovieNum (1200);//如下:unloadMovieNum (_level1200);不能卸载
        loadMovieNum ("sje.swf",1200)
        }
```

（三）lloadMovieNum()的定位

从前面我们已经知道了，函数 loadMovieNum 是将 SWF 或 JPEG 文件加载到_level（级别）的。并且是用_level1._x、_level2._x、_level1.aa._x 之类的来定位的。比如我们要将 02.swf 加载到主场景坐标系的(50,100)中，代码如下：

```
    loadMovieNum ( “02.swf" , 1 ); //极有可能不能如愿定位
_level1. _x = 50 ;
_level1. _y = 100 ;
```

但是在测试的时候大家会发现加载进来的 02.swf 并没有如我们预期的那样出现在主场景坐标系的(50,100)位置，这是为什么呢？

原来，在没有使用 loadMovieNum 以前，所有的图形、MC 等等都是处于_level0 里的，我们通常使用的_root 其实就是_level0。你可以作个测试：trace(_root==_level0)，你会发现返回值是“true"。

那么，我们用 loadMovieNum 加载 SWF 或 JPEG 文件的时候，程序就要对你指定的_level 进行创建，随后再紧接着载入 SWF 或 JPEG 文件。如果像刚才代码中写的那样，不判断_level 是否存在而调用其属性，势必无功而反。

解决的方法也很简单，就是用一个循环来判断指定的_level 是否存在。一旦_level 产生，那么它的种种属性自然可以调用了。代码如下：

```
loadMovieNum ( “02.swf" , 1 ); //如愿定位
onEnterFrame = function () {
    if ( _level1 ) {
        with ( _level1 ) {
            _x = 50 ;
            _y = 100 ;
        }
    delete onEnterFrame ;
    }
};
```

或者：(上面的代码较好些)

```
loadMovieNum ( “02.swf" , 1 );
    function go () {
        if ( _level1 ) {
            with ( _level1 ) {
                _x = 50 ;
                _y = 100 ;
            }
        clearInterval ( fps ); //清除对 setInterval() 的调用
    }
}
fps = setInterval ( go,100 );//每隔一定的时间，就调用函数、方法或对象
```

再或者大家可以使用 Macromedia Flash MX 2004 中的 MovieClipLoader 类来判断加载状态。

注意：深度 depth 值越大，其加载的内容越往后，即越在下层，处于下方，被遮盖；这与级别标识符_levelN 相反，在_levelN 中 N 值越大，其加载的内容越往前，即越在上层，处于上方。

（四）小技巧：用 lloadMovieNum()保护你的作品——使用虚假的文件后缀

其实加载外部文件的扩展名不一定非得以 .swf 命名，虽然加载的影片也在 IE 的临时文件夹中，但已是自定义的文件格式了，这样可以起到保护作用。

例如：

```
loadMovieNum("feng.exe",0);//加载 feng.swf 将扩展名改为—> feng.exe 加载
/*
loadMovieNum("feng.txt",0); //加载 feng.swf 将扩展名改为—> feng.txt 加载
loadMovieNum("feng.doc",0); //加载 feng.swf 将扩展名改为—> feng.doc 加载
loadMovieNum("feng.abcde",0); //加载 feng.swf 将扩展名改为 —> feng.abcde 加载
loadMovieNum("feng",0); //加载 feng.swf 将扩展名删除—> feng 加载
```

```
*/
```

EOS 构件的层次结构

EOS 构件引用分三个层次，构件包，构件和构件逻辑。

构件包 可以包含一组构件，是用来对 EOS 构件进行分类，是 EOS 构件最大复用单位。

构件 是 EOS 构件的主要部分，构件可以包含一组构件逻辑，分页面构件，展现构件，流程构件，业务构件，数据构件，运算构件 6 种.

构件逻辑 是构件的具体内容，粒度最细，所以也称为原子构件，每个构件逻辑都提供数据输入输出接口。

运算构件

EOS 中最底层的构件，又叫原子构件，用于完成特定的业务计算和程序集成的相关构件称为运算构件，运算构件中包含的运算逻辑是标准的 Java Method。

业务构件

用于完成多个运算逻辑的逻辑流程的相关构件称为业务构件，是通过 EOS 开发环境开发的多个运算逻辑的图形化逻辑流程。

展现构件

用于控制业务流转与页面交互的构件称为展现构件，展现构件是通过 EOS 开发环境开发的图形化组装业务构件、页面构件，来表达一个完整的功能。

页面构件

用于完成用户交互界面生成及页面集成的相关构件称为页面构件，页面构件可以是标准 J2EE 展现层的构件，如 Jsp, Tag 等

数据构件

用于完成数据模型定义，管理和数据操作的相关构件称为数据构件。

流程构件

用于完成某个包含人工和自动的业务活动流程的构件称为流程构件，是通过 EOS 开发环境开发的工作流的图形化描述 。

EclipseForm 设计指南之定制布局

Eclipse Form 提供了 2 个新的布局，TableWrapLayout 和 ColumnLayout。

（1）TableWrapLayout

·问题：如果将上例中超链接的文本设置的足够长

link.setText("This is an example of a form that is much longer and will need to wrap.");

即使设置了 SWT.WRAP，文本内容不会自动 WRAP，这是因为体内容的布局是 GridLayout

·Eclipse Form 提供替代的布局 TableWrapLayout：类似于 GridLayout，但是具有象 HTML 表格一样自动 WRAP 功能

·下面是解决超链接文本自动 WRAP 的例子：

```
public void createPartControl(Composite parent) {
toolkit = new FormToolkit(parent.getDisplay());
form = toolkit.createScrolledForm(parent);
form.setText("Hello, Eclipse Forms");
Composite body = form.getBody();
TableWrapLayout layout = new TableWrapLayout();
body.setLayout(layout);
HyPerlink link = toolkit.createHyperlink(body, "Click here.", SWT.WRAP);
link.addHyperlinkListener(new HyperlinkAdapter() {
public void linkActivated(HyperlinkEvent e) {
System.out.println("Link activated!");
}
});
layout.numColumns = 2;
link.setText
("This is an example of a form that is much longer and will need to wrap.");
TableWrapData td = new TableWrapData();
td.colspan = 2;
link.setLayoutData(td);
Label label = toolkit.createLabel(body, "Text field label:");
Text text = toolkit.createText(body, "");
td = new TableWrapData(TableWrapData.FILL_GRAB);
text.setLayoutData(td);
text.setData(FormToolkit.KEY_DRAW_BORDER, FormToolkit.TEXT_BORDER);
Button button = toolkit.createButton(body,
"An example of a checkbox in a form", SWT.CHECK);
td = new TableWrapData();
td.colspan = 2;
button.setLayoutData(td);
toolkit.paintBordersFor(body);
}
```

·下面是程序变化的地方：

1) TableWrapLayout 替代 GridLayout

2) 使用 TableWrapData 来提供布局数据信息

3) 设置的属性使用 colspan、rowspan 等来源于 HTML 表格单元的属性

·要注意的是：需要自动 WRAP 的控件，需要设置成 SWT.WRAP 风格

（2）ColumnLayout

·ColumnLayout 是 Eclipse Form 提供的另一个定制布局

·ColumnLayout 的布局方式是从上到下，从左到右

·在变化 Form 的宽度时，会自动调整控件列数以适应 Form 的宽度

·ColumnLayout 的设置很简单，通常只要设置列数的范围（缺省是 1-3）

Delphi 源程序格式书写规范

1.规范简介
本规范主要规定 Delphi 源程序在书写过程中所应遵循的规则及注意事项。编写该规范的目的是使公司软件开发人员的源代码书写习惯保持一致。这样做可以使每一个组员都可以理解其它组员的代码，以便于源代码的二次开发记忆系统的维护。

2.一般格式规范
2.1 缩进
缩进就是在当源程序的级改变时为增加可读性而露出的两个空格。缩进的规则为每一级缩进两个空格。不准许使用 Tab。因为 Tab 会因为用户所作的设置不同而产生不同的效果。当遇到 begin 或进入判断、循环、异常处理、with 语句、记录类型声明、类声明等的时侯增加一级， 当遇到 end 或退出判断、循环、异常处理、with 语句、记录类型声明、类声明等的时侯减少一级。例如：

```
if TmpInt <> 100 then
  TmpInt := 100;
```

2.2 Begin..End
begin 语句和 end 语句在源程序中要独占一行，例如:

```
for I := 0 to 10 do begin //不正确的用法
end;
for I := 0 to 10 do       //正确的用法
begin
end;
```

2.3 空格
在操作符及逻辑判断符号的两端添加空格，例如：I := I + 1;，a and b 等，但添加括号时不需要空格。例如：if (a > b) then //错误的用法
If (a > b) then //正确的用法
又例如：procedure Test(Param1: integer; Param3: string);

3. Object Pascal 语法书写格式规范
3.1 保留字
Object Pascal 语言的保留字或关键词应全部使用小写字母。

3.2 过程和函数
3.2.1 命名及格式
过程和函数的名称应全部使用有意义的单词组成，并且所有单词的第一个字母应该使用大写字母。例如：

```
procedure formatharddisk;//不正确的命名
procedure FormatHardDisk;//正确的命名
```

设置变量内容的过程和函数，应使用 Set 作为前缀，例如：

```
procedure SetUserName;
```

读取变量内容的过程和函数，应使用 Get 作为前缀，例如：

```
function GetUserName: string;
```

3.2.2 过程和函数的参数
3.2.2.1 命名
统一类型的参数写在同一句中:

```
procedure Foo(Param1, Param2, Param3: Integer; Param4: string);
```

3.2.2.2 命名

所有参数必须是有意义的；并且当参数名称和其它属性名称重了的时候，加一个前缀‘A’，例如：

```
procedure SomeProc(AUserName: string; AUserAge: integer);
```

3.2.2.3 命名冲突

当使用的两个 unit 中包括一个重名的函数或过程时，那么当你引用这一函数或过程时，将执行在 use 子句中后声明的那个 unit 中的函数或过程。为了避免这种‘uses-clause-dependent’需要在引用函数或过程时，写完整函数或过程的出处。例如：

```
SysUtils.FindClose(SR);
Windows.FindClose(Handle);
```

3.3 变量

3.3.1 变量命名及格式

首先所有变量必须起有意义的名字，使其它组员可以很容易读懂变量所代表的意义，变量命名可以采用同义的英文命名，可使用几个英文单词，但每一单词的首字母必须大写。例如：

```
var
  WriteFormat:：string；
```

同时对于一些特定类型可采用一定的简写如下：

指针类型

P

纪录类型

Rec

数组类型

Arr

类

Class

循环控制变量通常使用单一的字符如：i, j, 或 k。 另外使用一个有意义的名字例如：UserIndex ，也是准许的。

3.3.2 局部变量

在过程中使用局部变量遵循所有其它变量的命名规则。

3.3.3 全局变量

尽量不使用全局变量，如必须使用全局变量则必须加前缀‘g’，同时应在变量名称中体现变量的类型。例如：

```
  gprecUserCount: point;//名称为 UserCount 的全局变量,其类型为指向一结构的指针
```

但是在模块内部可以使用全局变量。所有模块内全局变量必须用‘F’为前缀。如果几个模块之间需要进行资料交换，则需要通过声明属性的方法来实现。例如：

```
type
  TFormOverdraftReturn = class(TForm)
  private
{ Private declarations }
FuserName: string;
FuserCount: Integer;
Procedure SetUserName(Value: string);
Function GetUserName: string;
  public
{ Public declarations }
property UserName: string read GetUserName write SetUserName;
property UserCount: Integer read FuserCount write FuserCount;
  end;
```

3.4 类型

3.4.1 大小写协议

保留字的类型名称必须全部小写。Win32 API 的类型通常全部大写，对于其它类型则首字母大写，其余字母小写，例如：

```
var
  MyString: string;   // reserved word
  WindowHandle: HWND; // Win32 API type
  I: Integer;    // type identifier introduced in System unit
```

3.4.2 浮点类型

尽量不使用 Real 类型，他只是为了和旧的 Pascal 代码兼容，尽量使用 Double 类型。Double 类型是对处理器和数据总线做过最优化的并且是 IEEE 定义的标准数据结构。当数值超出 Double 的范围时，使用 Extended 。但 Extended 不被 Jave 支持。但使用其它语言编写的 DLL 时可能会使用 Single 类型。

3.4.3 枚举类型

枚举类型的名字必须有意义并且类型的名字之前要加前缀'T'。枚举类型的内容的名字必须包含枚举类型名称的简写，例如：

```
TSongType = (stRock, stClassical, stCountry, stAlternative, stHeavyMetal, stRB);
```

3.4.4 数组类型

数组类型的名字必须有意义并且类型的名字之前要加前缀'T'。如果声明一个指向数组类型的指针必须在该类型的名字之前加前缀'P'，例如：

```
type
  PCycleArray = ^TCycleArray;
  TCycleArray = array[1..100] of integer;
```

3.4.5 记录类型

记录类型的名字必须有意义并且类型的名字之前要加前缀'T'。如果声明一个指向数组类型的指针必须在该类型的名字之前加前缀'P'，例如：

```
type
  PEmployee = ^TEmployee;
  TEmployee = record
    EmployeeName: string
    EmployeeRate: Double;
  end;
```

3.5 类

3.5.1 命名及格式

类的名字必须有意义并且类型的名字之前要加前缀'T'。例如：

```
type
  TCustomer = class(TObject)
```

类实例的名字通常是去掉'T'的类的名字。例如：

```
var
  Customer: TCustomer;
```

3.5.2 类中的变量

3.5.2.1 命名及格式

类的名字必须有意义并且类型的名字之前要加前缀'F'。所有的变量必须是四有的。如果需要从外部访问此变量则需要声明一属性

3.5.3 方法

3.5.3.1 命名及格式

同函数和过程的命名及格式。

3.5.3.2 属性访问方法

所有的属性访问方法必须出现在 private 或 protected 中。属性访问方法的命名同函数和过

程的命名另外读方法(reader method)必须使用前缀‘Get’. 写方法(writer method)必须使用前缀‘Set’。写方法的参数必须命名为‘Value’，其类型同所要写的属性相一致。例如：

```
TSomeClass = class(TObject)
private
  FSomeField: Integer;
protected
  function GetSomeField: Integer;
  procedure SetSomeField( Value: Integer);
public
  property SomeField: Integer read GetSomeField write SetSomeField;
end;
```

3.6 属性

3.6.1 命名及格式

同其用操作的，出去前缀‘F’的类的变量的名称相一致 。

3.7 文件

3.7.1 项目文件

3.7.1.1 项目目录结构

程序主目录--Bin（应用程序所在路径）

-Db（本地数据库所在路径）

-Doc（文档所在路径）

-Hlp（帮助文件所在路径）

-Backup（备份路径）

-Tmp（临时文件路径）

3.7.1.2 命名

项目文件必须使用一个有意义的名字。例如： Delphi 中系统信息的项目文件被命名为 SysInfo.dpr。

3.7.2 Form 文件

3.7.2.1 命名

同 Form 的名称相一致：例如：Form 的名称为 FormMain 则 Form 文件的名称就为 FormMain.frm。

3.7.3 Data Module 文件

3.7.3.1 命名

data module 文件的命名应该有意义，并且使用‘DM’作为前缀。例如： 用户 data module 被命名为‘DMCustomers.dfm’。

3.7.4 Remote Data Module 文件

3.7.4.1 命名

remote data module 文件的命名应该有意义，并且使用‘RDM’作为前缀。例如：用户 remote data module 被命名为‘RDMCustomers.dfm’。

3.7.5 Unit 文件

3.7.5.1 普通 Unit

3.7.5.1.1 Unit 文件命名

unit 文件的命名应该有意义，并且使用‘unit’作为前缀。例如： 通用 unit 被命名为‘UnitGeneral’。

3.7.5.2 Form Units

3.7.5.2.1 命名

Form unit 文件的名字必须和 Form 的名称保持一致。例如：主窗体叫 FormMain.pas 则 Form Unit 文件的名字为：UnitFormMain。

3.7.5.3 Data Module Units

3.7.5.3.1 命名

Data Module unit 文件的名字必须和 Data Module 的名称保持一致。例如：主 Data Module 叫 DMMain.pas 则 Data Module Unit 文件的名字为：UnitDMMain。

3.7.5.4 文件头

在所有文件的头部应写上此文件的用途，作者，日期及输入和输出。例如：

{

修改日期：

作者：

用途：

本模块结构组成：

}

3.7.6 Forms 和 Data Modules Forms

3.7.6.1 Form 类

1. Form 类命名标准

Forms 类的命名应该有意义，并且使用'TForm'作为前缀。例如： About Form 类的名字为:

TAboutForm = class(TForm)

主窗体的名字为

TMainForm = class(TForm)

2. Form 类实例的命名标准

Form 的类实例的名字应同期掉'T'的 Form 类的名字相一致。例如：

Type Name

Instance Name

TaboutForm

AboutForm

TmainForm

MainForm

TCustomerEntryForm

CustomerEntryForm

3.7.6.2 Data Modules Form

3.7.6.2.1. Data Module Form 命名标准

Data Modules Forms 类的命名应该有意义，并且使用'TDM'作为前缀。例如：

TDMCustomer = class(TDataModule)

TDMOrders = class(TDataModule)

3.7.6.2.2. Data Module 实例命名标准

Data Module Form 的类实例的名字应同期掉'T'的 Data Module Form 类的名字相一致。例如：

Type Name

Instance Name

TCustomerDataModule

CustomerDataModule

TordersDataModule

OrdersDataModule

3.8 控件

3.8.1 控件实例的命名

控件的实例应使用去掉'T'该控件类的名称作为前缀，例如：

输入用户姓名的 Tedit 的名字为：EditUserName。

3.8.2 控件的简写

控件的名称可使用以下简写，但所用简写于控件名称之间药添加'_'：

3.8.2.1 Standard Tab

mm TMainMenu

pm TPopupMenu
mmi TMainMenuItem
pmi TPopupMenuItem
lbl TLabel
edt TEdit
mem TMemo
btn TButton
cb TCheckBox
rb TRadioButton
lb TListBox
cb TComboBox
scb TScrollBar
gb TGroupBox
rg TRadioGroup
pnl TPanel
cl TCommandList
3.8.2.2 Additional Tab
bbtn TBitBtn
sb TSpeedButton
me TMaskEdit
sg TStringGrid
dg TDrawGrid
img TImage
shp TShape
bvl TBevel
sbx TScrollBox
clb TCheckListbox
spl TSplitter
stx TStaticText
cht TChart
3.8.2.3 Win32 Tab
tbc TTabControl
pgc TPageControl
il TImageList
re TRichEdit
tbr TTrackBar
prb TProgressBar
ud TUpDown
hk THotKey
ani TAnimate
dtp TDateTimePicker
tv TTreeView
lv TListView
hdr THeaderControl
stb TStatusBar
tlb TToolBar
clb TCoolBar
3.8.2.4 System Tab
tm TTimer
pbTPaintBox
mp TMediaPlayer
olec TOleContainer
ddcc TDDEClientConv

ddci TDDEClientItem
ddsc TDDEServerConv
ddsi TDDEServerItem
3.8.2.5 Internet Tab
csk TClientSocket
ssk TServerSocket
wbd TWebDispatcher
pp TPageProducer
tp TQueryTableProducer
dstp TDataSetTableProducer
nmdt TNMDayTime
nec TNMEcho
nf TNMFinger
nftp TNMFtp
nhttp TNMHttp
nMsg TNMMsg
nmsg TNMMSGServ
nntp TNMNNTP
npop TNMPop3
nuup TNMUUProcessor
smtp TNMSMTP
nst TNMStrm
nsts TNMStrmServ
ntm TNMTime
nudp TNMUdp
psk TPowerSock
ngs TNMGeneralServer
html THtml
url TNMUrl
sml TSimpleMail
3.8.2.6 Data Access Tab
ds TDataSource
tbl TTable
qry TQuery
sp TStoredProc
db TDataBase
ssn TSession
bm TBatchMove
usql TUpdateSQL
3.8.2.7 Data Controls Tab
dbg TDBGrid
dbn TDBNavigator
dbt TDBText
dbe TDBEdit
dbm TDBMemo
dbi TDBImage
dblb TDBListBox
dbcb TDBComboBox
dbch TDBCheckBox
dbrg TDBRadioGroup
dbll TDBLookupListBox
dblc TDBLookupComboBox
dbre TDBRichEdit

dbcg TDBCtrlGrid
dbch TDBChart
3.8.2.8 Decision Cube Tab
dcb TDecisionCube
dcq TDecisionQuery
dcs TDecisionSource
dcp TDecisionPivot
dcg TDecisionGrid
dcgr TDecisionGraph
3.8.2.9 QReport Tab
qr TQuickReport
qrsd TQRSubDetail
qrb TQRBand
qrcb TQRChildBand
qrg TQRGroup
qrl TQRLabel
qrt TQRText
qre TQRExpr
qrs TQRSysData
qrm TQRMemo
qrrt TQRRichText
qrdr TQRDBRichText
qrsh TQRShape
qri TQRImage
qrdi TQRDBMImage
qrcr TQRCompositeReport
qrp TQRPreview
qrch TQRChart
3.8.2.10 Dialogs Tab
OpenDialog TOpenDialog
SaveDialog TSaveDialog
OpenPictureDialog TOpenPictureDialog
SavePictureDialog TSavePictureDialog
FontDialog TFontDialog
ColorDialog TColorDialog
PrintDialog TPrintDialog
PrinterSetupDialog TPrintSetupDialog
FindDialog TFindDialog
ReplaceDialog TReplaceDialog
3.8.2.11 Win31 Tab
dbll TDBLookupList
dblc TDBLookupCombo
ts TTabSet
ol TOutline
tnb TTabbedNoteBook
nb TNoteBook
hdr THeader
flb TFileListBox
dlb TDirectoryListBox
dcb TDriveComboBox
fcb TFilterComboBox
3.8.2.12 Samples Tab
gg TGauge

cg TColorGrid
spb TSpinButton
spe TSpinEdit
dol TDirectoryOutline
cal TCalendar
ibea TIBEventAlerter
3.8.2.13 ActiveX Tab
cfx TChartFX
vsp TVSSpell
f1b TF1Book
vtc TVTChart
grp TGraph
3.8.2.14 Midas Tab
prv TProvider
cds TClientDataSet
qcds TQueryClientDataSet
dcom TDCOMConnection
olee TOleEnterpriseConnection
sck TSocketConnection
rms TRemoteServer
mid TmidasConnection

4．修改规范

本规则所做的规定仅适用于已经纳入配置管理的程序。在这类修改中，要求保留修改前的内容、并标识出修改和新增的内容。并在文件头加入修改人、修改日期、修改说明等必要的信息。

4．1 修改历史记录

对源文件进行经过批准的修改时，修改者应在程序文件头加入修改历史项。在以后的每一次修改时，修改者都必须在该项目中填写下列信息：

修改人
修改时间
修改原因
修改说明即如何修改

4．2 新增代码行

新增代码行的前后应有注释行说明。
// 修改人，修改时间，修改说明
新增代码行
// 修改结束

4．3 删除代码行

删除代码行的前后用注释行说明。
//修改人，修改时间，修改说明
//要删除的代码行（将要删除的语句进行注释）
//修改结束

4．4 修改代码行

修改代码行以删除代码行后在新增代码行的方式进行。
//修改人，修改时间，修改说明
//修改前的代码行

```
//修改结束
    //修改后的代码行
        修改后的代码行
//修改结束
```

C 语言的代码规范探讨

1.C 语言书写规范
1.1 符号命名规则
1.1.1 符号名包括模块名、常量名、标号名、子程序名等。这些名字应该能反映它所代表的实际东西，具有一定的意义，使其能够见名知义，有助于对程序功能的理解。命名采用匈牙利命名法。规则如下：
(1)所有宏定义、枚举常数和 const 变量，用大写字母命名。在复合词里用下划线隔开每个词。
(2)复合词中每个单词的第一个字母大写。除了规则 5.1.1.1 以外，避免使用下划线。
(3)类、类型定义和枚举型名的第一个字母大写。
(4)函数名是复合词的，第一个词采用全部小写，随后每个单词采用第一个字母大写，其它字母小写方式；如果是单个词的，采用全部小写方式。
(5)循环变量可采用 i, j, k 等，不受上述规则限制。
(6) 类的成员变量应采用 m_开头。
(7) 全局变量词头为 g_ 。
(8) 临时变量词头为 tmp_ 。
(9) 对结构体内的变量命名，遵循变量的具体含义命名原则
(10)用小写字母的前缀表示变量的类型，前缀的下一个字母用大写。
表 1
词 头 类 型 词 头 类 型
ch char l long
i integer u unsigned
b boolean p pointer
f float lp long pointer
d double s string
st structure sz ASCII string
by byte n short int
H handle x,y 分别为 x,y 坐标
dw DWORD fn function

表 2
词 头 变 量 名 词 头 变 量 名
task task sig signal
sb binary semaphores wd watchdog
sm mutual exclusion
semaphores tm timer
sc counting semaphores msg message
pipe pipe
例：

```
#define ARRAY_SIZE 24 /*规则 5.1.1.1*/
int g_iFlag;
classMyClass /*规则 5.1.1.3*/
{
};
void someFunc( ) /*规则 5.1.1.2 和 5.1.1.4*/
{
```

.2.
Q/ECC/BJ 010—2001
int nArray[ARRAY_SIZE];
unsigned char uchByte;
char szName[ ];
char *pszName = szName;
}
(11)有些词头（如 p 和 u）可以和其它词头组合。
例：WDOG_ID wdId;
WDOG_ID g_wdId; /*全局 watchdog Id，故以 g_开头*/
1.1.2 名字的长度一般不要过长或过短。过长的名字会增加工作量，使程序逻辑流程变得模糊；过短的名字无法表达符号的实际意义。约定长度范围：3-31；

1.2 数据和函数说明
1.2.1 数据说明次序应当规范化，使数据属性容易查找，也有利于测试、排错和维护。说明的先后次序应固定，应按逻辑功能排序，逻辑功能块内建议采用下列顺序：整型说明、实型说明、字符说明、逻辑量说明。
1.2.2 如果设计了一个复杂的数据结构，应当通过注释对其变量的含义、用途进行说明。
1.2.3 在函数的声明中使用异常声明。
如：void f() throw(toobig, toosmall, divzero);
在声明一个函数时，将它所抛出的异常列出，便于函数的使用者了解可能会发生哪些异常。

1.3 程序注释
1.3.1 程序注释是程序员与日后的程序读者之间通信的重要手段之一，注释分为文件注释、函数注释和功能注释。
1.3.2 正规程序的注释应注意：
——注释行的数量占到整个源程序的 1/3 到 1/2。
1.3.3 文件注释位于整个源程序的最开始部分，注释后空两行开始程序正文。它包括：
——程序标题。
——目的、功能说明。
——文件作者、最后修改日期等说明。
例：
./***
（空一行）
标题: Demo.c
功能: 测试 VxWorks 的各种系统调用.
说明:
该程序测试各种 VxWorks 的系统调用函数。包括任务（taks）的创建、挂起及任务间通过信号灯实现同步，通过消息队列进行通讯。
程序创建了两个任务：一个高优先级的任务和一个低优先级的任务。两个任务间通过一个二进制的信号灯进行同步，通过消息队列进行通讯。
当前版本： x.x
修改信息： 2000.06.05 John， Initial Version
2000.07.05 Tom， Bug xxxx fixed
***/
(空 2 行，开始程序正文)

1.3.4 函数注释通常置于每函数或过程的开头部分，它应当给出函数或过程的整体说明对于理解程序本身具有引导作用。一般包括如下条目：
——模块标题。
——有关本模块功能和目的的说明。

——调用格式
——接口说明：包括输入、输出、返回值、异常。
——算法。如果模块中采用了一些复杂的算法。
例：
file://（注释开头应和上一函数空两行）
（注释开头与上一函数最后一行间隔两行）
/***
标题：assignmentComplete
功能：BSC=>MSC 消息生成函数，生成 assignment_complete 指配完成消息(BSMAP 消息) .
格式：
int assignmentComplete(int iCellId, int iServiceChannnelNum, char *pszMSGData) throw(exception1, exception2)
输入：
int iCellId: MS 所在的小区识别
iCellId 取值：0x00-——0xff .4.
Q/ECC/BJ 010—2001
int iServiceChannnelNum：MS 所占的业务信道号码
输出：
char * pszMSGData：指配完成消息数据
返回值: 0x00 正常
异常：exception1 异常情况 1, exception2 异常情况 2
***/
(注释后直接开始程序正文，不空行。)
1.3.5 功能性注释嵌在源程序体中，用于描述其后的语句或程序段做什么工作，也就是解释下面要做什么，或是执行了下面的语句会怎么样。而不要解释下面怎么做，因为解释怎么做常常与程序本身是重复的。
例：
/*把 amount 加到 total 中*/
total = amount + total;
这样的注释仅仅是重复了下面的程序，对于理解它的工作并没有什么作用。而下面的注释，有助于读者理解。
/*将每月的销售额 amount 加到年销售额 total 中*/
total = amount + total;

1.4 函数编写应尽可能短小精悍，一般不超过两屏，以便于调试和理解。

1.5 语句结构
为保证语句结构的清晰和程序的可读性，在编写软件程序时应注意以下几个方面的问题：
——在一行内只写一条语句，并采用空格、空行和移行保证清楚的视觉效果。
——每一个嵌套的函数块，使用一个 TAB 缩进（可以设定为 4 个空格），大括号必须放在条件语句的下一行，单独成一行，便于匹对：
如，有一段程序如下：
for(i=1;iif(a[j]应写为
for(i=1; i
{
t=1;
for(j = i+1; j
{
if(a[i]<A[J])
t=j;
if(t!=1)

{ .5.
Q/ECC/BJ 010—2001
work=a[t];
a[t]=a[i];
a[i]=work;
}
}
}
——文件之中不得存在无规则的空行，比如说连续十个空行。
一般来讲函数与函数之间的空行为 2-3 行；
在函数体内部，在逻辑上独立的两个函数块可适当空行，一般为 1-2 行。
——程序编写首先应考虑清晰性，不要刻意追求技巧性而使得程序难以理解。
——每行长度尽量避免超过屏幕宽度，应不超过 80 个字符。
——除非对效率有特殊要求，编写程序要作到清晰第一，效率第二。
——尽可能使用函数库。
——尽量用公共过程或子程序去代替重复的功能代码段。要注意，这个代码应具有一个独立的功能，不要只因代码形式一样便将其抽出组成一个公共过程或子程序。
——使用括号清晰地表达算术表达式和逻辑表达式的运算顺序。如将 x=a*b/c*d 写成 x=(a*b/c)*d 可避免阅读者误解为 x=(a*b)/(c*d)。
——避免不必要的转移。
——避免采用过于复杂的条件测试。
——避免过多的循环嵌套和条件嵌套。
——建议不要使用 *=，^=, /=等运算符。
——一个函数不要超过 200 行。一个文件应避免超过 2000 行。
——尽量避免使用 go to 语句。
——避免采用多赋值语句，如 x = y = z ;
——不鼓励采用?:操作符，如 z = (a>b)?a:b;
——不要使用空的 if else 语句。如
if(cMychar >= ‘A’)
if(cMychar <= ‘Z’)
printf(“This is a letter \n”);
else
printf(“This is not a letter \n”);
else 到底是否定哪个 if 容易引起误解。可通过加{}避免误解。
——尽量减少使用“否定”条件的条件语句。如：
把 if(!((cMychar<’0’) || (cMychar>’9’)))
改为 if((cMychar>=’0’) && (cMychar<=’9’))

CtoC++新手指南

C++技术固然是很时髦的，许多 C 用户都想在尽可能短的时间内为自己贴上 C++的标签。介绍 C++的书很多，但只有那些已经侥幸入门的用户才偶尔去翻翻，仍有不少在 C++门口徘徊的流浪汉。
本文只针对 C 用户，最好是一位很不错的老用户(譬如他在遇到最简单的问题时都尝试着使用指针)，通过一些 C 和更好的 C++(本文用的是 Borland C++3.1 版本)例程介绍有关 C++的一些知识，让读者朋友们“浅入深出”，轻轻松松 C to C++!
一、标签！标签！
快快为你的程序贴上 C++的标签，让你看起来很像个合格的 C++用户……

1.注释(comment)

C++的注释允许采取两种形式。第一种是传统 C 采用的/*和*/，另一种新采用的则是//，它表示从//至行尾皆为注释部分。读者朋友完全可以通过//使你的代码带上 C++的气息，如 test0l:

```
//test01.cpp
#include <iostream.h>
//I'm a C++user!
//…and C is out of date.

void main()
{
cout<<"Hello world!\n"; //prints a string
}
```

Hello-world!

如果你尝试着在 test0l. exe 中找到这些高级的注释，很简单，它们不会在那里的。

2. cincout

你可能从 test0l 中嗅出什么味儿来了，在 C++中，其次的贵族是 cout，而不是很老土的 printf ( )。左移操作符'<<'的含义被重写，称作“输出操作符”或“插入操作符”。你可以使用'<<'将一大堆的数据像糖葫芦一样串起来，然后再用 cout 输出:

```
cout << "ASCII code of "<< 'a' << " is:" <<97;
```

ASCII code of a is:97

如何来输出一个地址的值呢?在 C 中可以通过格式控制符”%p”来实现，如:

```
printf ("%p， &i);
```

类似地，C++也是这样:

```
cout << & i;
```

但对字符串就不同啦!因为:

```
char * String="Waterloo Bridge";
cout << String; //prints ‘Waterloo Bridge'
```

只会输出 String 的内容。但方法还是有的，如采取强制类型转换:

```
cout<<(void *)String;
```

cin 采取的操作符是'>>',称作“输入操作符”或“提取操作符”。在头文件 iostream.h 中有 cin cout 的原型定义，cin 语句的书写格式与 cout 的完全一样:

```
cin>>i; //ok
cin>>&i; //error. Illegal structure operation
```

看到了?别再傻傻地送一个 scanf()常用的'&'地址符给它。

C++另外提供了一个操纵算子 endl，它的功能和'\n'完全一样，如 test0l 中的 cout 语句可改版为:

```
cout << ”Hello world!”<
```

3.即时声明

这是笔者杜撰的一个术语，它的原文为 declarations mixed with statements，意即允许变量的声明与语句的混合使用。传统 C 程序提倡用户将声明和语句分开，如下形式:

```
int i=100;
float f; //declarations
i++;
f=1.0/i; //statements
```

而 C++抛弃这点可读性，允许用户采取更自由的书写形式:

```
int i=100;
i++;
float f =1. 0/i;
```

即时声明常见于 for 循环语句中:

```
for(int i = 0; i < 16; i++)
for(int j = 0; j < 16; j++)
putpixel(j i Color[i][j]);
```

这种形式允许在语句段中任点声明新的变量并不失时机地使用它(而不必在所有的声明结束之后)。

特别地，C++强化了数据类型的类概念，对于以上出现的”int i=1 j=2;”完全可以写成：

```
int i(1) j (2);
```

再如:

```
char * Stringl("Youth Studio.”);
char String2[]("Computer Fan.“);
```

这不属于“即时声明”的范畴，但这些特性足以让你的代码与先前愚昧的 C 产品区别开来。

4.作用域(scope)及其存取操作符(scope qualifier operator)

即时声明使 C 语言的作用域的概念尤显重要，例如以下语句包含着一条错误，因为 ch 变量在 if 块外失去了作用域。

```
if(ok)
char ch='!';
else
ch='?'; //error. access outside condition
```

作用域对应于某一变量的生存周期，它通常表现为以下五种:

块作用域

开始于声明点，结束于块尾，块是由{}括起的一段区域

函数作用域

函数作用域只有语句标号，标号名可以和 goto 语句一起在函数体任何地方

函数原型作用域

在函数原型中的参量说明表中声明的标识符具有函数原型作用域

文件作用域

在所有块和类的外部声明的标识符(全局变量)具有文件作用域

类作用域

类的成员具有类作用域

具有不同作用域的变量可以同名，如 test02:

```
//test02.cpp
#include <iostream.h>
int i=0;
void main()
{
cout << i << ' '; //global 'int i' visible
{
float i(0.01); //global 'int i' overrided
cout<< i << ' ';
}
cout<<i<<endl; //global 'int i' visible again
}
```

```
0 0.01 0
```

编译器并未给出错误信息。

作用域与可见性并不是同一概念，具有作用域不一定具有可见性，而具有可见性一定具有作用域。

在 test02 中，float i 的使用使全局 int i 失去可见性，这种情形被称作隐藏(override)。但这并不意味着 int i 失去了作用域，在 main()函数运行过程中，int i 始终存在。

有一种办法来引用这丢了名份的全局 i，即使用 C++提供的作用域存取操作符::，它表示引用的变量具有文件作用域，如下例程:

```
//test03.cpp
#include <iostream.h>
enum {boy girl};
char i = boy;
void main()
{
{
float i(0.01);
cout << "i=" << i << endl;
::i=girl; //modify global 'i'
}
cout << "I am a " << (i ? "girl." : "boy.");
}
i=0.01
I am a girl.
```

在上例中，通过::操作符，第 8 行语句偷偷地改写了 i 所属的性别。更妙的是，::之前还可以加上某些类的名称，它表示引用的变量是该类的成员。

5. new delete

许多 C 用户肯定不会忘记 alloc()和 free()函数族，它们曾经为动态内存分配与释放的操作做出了很大的贡献，如:

```
char *cp = malloc(sizeof(char));
int *ip=calloc(sizeof(int) 10);
free(ip);
free(cp);
```

C++允许用户使用这些函数，但它同时也提供了两个类似的操作符 new 和 delete，它们分别用来分配和释放内存，形式如下:

```
p = new TYPE;
delete p;
```

因此以上的 cp 操作可改版为:

```
char*cp=new char;
delete cp;
```

new delete 操作符同样亦可作用于 C 中的结构变量，如:

```
struct COMPLEX*cp = new struct COMPLEX;
delete cp;
```

当不能成功地分配所需要的内存时，new 将返回 NULL。对于字符型变量可以如下初始化:

```
char ch('A'); //char ch='A'
```

对应地，new 可以同时对变量的值进行初始化，如:

```
char p=new char ('A'); //cp='A'
```

new 不需要用户再使用 sizeof 运算符来提供尺寸，它能自动识别操作数的类型及尺寸大小，这虽然比 malloc)函数聪明不了多少，但起码使用起来会比它方便得多。当然，正如 calloc()函数，new 也可以用于数组，形式如下:

```
p = new TYPE[Size] ;
```

对应的内存释放形式:

```
delete [] p;
```

同理首例中 ip 操作可以改版为:

```
int * ip=new int[10];
delete [] ip;
```

用 new 分配多维数组的形式为:

```
p = new TYPE [c0] [c1]... [cN];
```

从来没有太快活的事情，例如以下使用非法:
int***ip2=(int***)new int[m] [n][k]; //error. Constant expression required
int***ip 1=(int***)new int[m][2][81; //ok
C++最多只允许数组第一维的尺寸(即 c0)是个变量，而其它的都应该为确定的编译时期常量。
使用 new 分配数组时，也不能再提供初始值:
char*String =new char[20] ("Scent of a Woman"); //error: Array allocated using 'new' may not have an initializer

6.引用(reference)
(1)函数参数引用
以下例程中的 Swap()函数对数据交换毫无用处:
```
//test04. cpp
#include <iostream.h>
void Swap(int va int vb)
{
int temp=va;
va=vb;
vb=temp;
cout << "&va=" << &va << "&vb=" << &vb << endl;
}
void main()
{
int a(1) b(2);
cout << "&a=" << &a << "&b=" << &b << endl;
Swap(a b);
cout << "a=" << a << " b=" << b << endl;
}
```
&a=0x0012FF7C&b=0x0012FF78
&va=0x0012FF24&vb=0x0012FF28
a=1
b=2c 语言对参数的调用采取拷贝传值方式，在实际函数体内，使用的只是与实参等值的另一份拷贝，而并非实参本身(它们所占的地址不同)，这就是 Swap()忙了半天却什么好处都没捞到的原因，它改变的只是地址 0x0012FF24 和 0x0012FF28 处的值。当然，可采取似乎更先进的指针来改写以上的 Swap ()函数:
```
//test05. cpp
#include <iostream.h>

void Swap(int * vap int * vbp)
{
int temp = *vap;
*vap = *vbp;
*vbp = temp;
cout << "vap=" << vap << "vbp=" <<vbp << endl;
cout << "&vap=" << &vap << "&vbp=" << &vbp << endl;
}
void main()
{
int a(1) b(2);
int * ap = &a * bp = &b;
cout << "ap=" << ap << "bp=" << bp << endl;
cout << "&ap=" << &ap << "&bp=" << &bp << endl;
Swap(ap bp);
```

```
cout << "a=" << a << "b=" << b <<endl;
}
ap=0x0012FF7Cbp=0x0012FF78
&ap=0x0012FF74&bp=0x0012FF70
vap=0x0012FF7Cvbp=0x0012FF78
&vap=0x0012FF1C&vbp=0x0012FF20
a=2b=1
```

在上例中，参数的调用仍采取的是拷贝传值方式，Swap()拷贝一份实参的值(其中的内容即 a b 的地址)，但这并不表明 vapvbp 与实参 apbp 占据同一内存单元

对实际数据操作时，传统的拷贝方式并不值得欢迎，C++为此提出了引用方式，它允许函数使用实参本身(其它一些高级语言，如 BASIC FORTRAN 即采取这种方式)。以下是相应的程序:

```
//test06. cpp
#include <iostream.h>
void Swap(int & va int & vb)
{
int temp=va;
va=vb;
vb=temp;
cout << "&va=" << &va << "&vb=" << &vb << endl;
}
void main()
{
int a(1) b(2);
cout << "&a=" << &a << "&b=" << &b << endl;
Swap(a b);
cout << "a=" << a << "b=" << b << endl;
}
&a=0x0012FF7C&b=0x0012FF78
&va=0x0012FF7C&vb=0x0012FF78
a=2b=1
```

很明显，a b 与 vavb 的地址完全重合。

对 int&的写法别把眼睛瞪得太大，你顶多只能撇撇嘴，然后不动声色地说:“就这么回事!加上&就表明引用方式呗!”

(2)简单变量引用

简单变量引用可以为同一变量取不同的名字，以下是个例子:

```
int Rat;
int & Mouse=Rat;
```

这样定义之后，Rat 就是 Mouse(用中文说就是:老鼠就是老鼠)，这两个名字指向同一内存单元，如:

```
Mouse=Mickey; //Rat=Mickey
```

一种更浅显的理解是把引用看成伪装的指针，例如，Mouse 很可能被编译器解释成:*(& Rat)，这种理解可能是正确的。

由于引用严格来说不是对象(?!)，在使用时应该注意到以下几点:

①引用在声明时必须进行初始化;

②不能声明引用的引用;

③不能声明引用数组成指向引用的指针(但可以声明对指针的引用);

④为引用提供的初始值必须是一个变量。

当初始值是一个常量或是一个使用 const 修饰的变量，或者引用类型与变量类型不一致时，编译器则为之建立一个临时变量，然后对该临时变量进行引用。例如:

```
int & refl = 50; //int temp=50 &refl=temp
```

float a=100.0;
int & ref2 = a; / / int temp=a&ref2=temp

(3)函数返回引用
函数可以返回一个引用。观察程序 test07:
//test07.cpp
#include <iostream.h>
char &Value (char*a int index)
{
return a[index];
}
void main()
{
char String[20] = "a monkey!";
Value(String 2) = 'd';
cout << String << endl;
}
a donkey!
这个程序利用函数返回引用写出了诸如 Value (String 2) ='d'这样令人费解的语句。在这种情况下，函数许用在赋值运算符的左边。允函数返回引用也常常应用于作符重载函操数。
7.缺省参数(default value)
从事过 DOS 环境下图形设计的朋友(至少我在这个陷阱里曾经摸了两年时间)肯定熟悉 initgraph()函数，它的原型为:
void far initgraph(int far *GraphDriver int far*GraphMode char far*DriverPath);
也许你会为它再定做一个函数:
void InitGraph(int Driver int Mode)
{
initgraph(& Driver &Mode “”);
}
一段时间下来，你肯定有了你最钟情的调用方式，例如你就喜欢使用 640 * 480 * 16 这种工作模式。
既然如此，你完全可以将函数 InitGraph ( )声明成具有缺省的图形模式参数，如下:
void InitGraph(int Driver = VGA int Mode = VGAHI);
这样，每次你只需简单地使用语句 InitGraph ();即可进入你所喜爱的那种模式。当然，当你使用 InitGraph (CGA CGAHI);机器也会毫不犹豫地切入到指定的 CGAHI 模式，而与正常的函数没有两样。
这就是缺省参数的用法!为了提供更丰富的功能，一些函数要求用户提供更多的参数(注意到许多 Windows 程序员的烟灰缸旁边都有一本很厚很厚的 Windows 函数接口手册)，而实际上，这些参数中的某几项常常是被固定引用的，因此，就可以将它们设定为缺省参数，例如以下函数:
void Putpixel(int y inint xt Color=BLACK char Mode =COPY_PUT);
将可能在((x y)处以 Color 颜色、Mode 模式画一个点，缺省情况下，颜色为黑色，写点模式为覆盖方式。
以下对函数的调用合法:
Putpixel (100 100); // Putpixel(100 100 BLACK COPY _PUT)
PutPixel (100 100 RED); // PutPixel(100 100 RED COPY_ PUT)
PutPixel(100 100 RED XOR_PUT);
而以下调用形式并不合法:
Putpixel();
Putpixel (100) ;
Putpixel(100 100 XOR_PUT);
前两种形式缺少参数，因为 x、y 值并没有缺省值;第三种形式则天真地以为编译器会将其处

理成:
PutPixel (100 100 BLACK XOR_PUT);
并且不会产生任何二义性问题，不幸的是，C++并不赞成这样做。
作为一条经验，缺省参数序列中最容易改变其调用值的应尽量写在前面，最可能使用其缺省值的(即最稳定的)置于后端。如果将以上函数原型声明成如下形式:
void Putpixel(int Color = BLACK char Mode = COPY_PUT int x=100 int y=100);
包括你自己，也不会喜欢它。

CMMI5 在项目中的精简应用

CMMI5 在小型项目中的成本过高，根据自己对 CMMI5 的实施体会与在实际项目中的应用，在项目实施的过程中精简了 CMMI5 的实施流程和部分文档，这个精简的流程在项目实施的过程中既可以确保流程规范与质量信赖又可以节约项目成本。以下跟大家分享一下 CMMI5 在项目中的精简应用：

一、需求与规范的管理

1、 由测试负责人（或专门的需求分析负责人）统一接收来自移动总的行业网关相关规范和新需求，测试负责人浏览规范获知大意后回复邮件，将规范和新需求转发给开发经理、项目经理、相关的开发人员和测试人员，同时 commit 到 CVS；

2、 测试负责人（或专门的需求分析负责人）、项目经理仔细阅读规范与需求后，对规范和新需求进行研究，并就难点和疑点进行讨论，整理出重点内容，并将重点内容发给开发经理、项目经理、相关的开发人员和测试人员，同时 commit 到 CVS；

3、 开发经理、项目经理、测试负责人、需求分析负责人、相关的开发人员与测试人员开会对规范、需求和重点内容进行讨论，确定需求的具体含义以及最终实现的需求和功能点；

4、 项目经理根据规范、需求和开会讨论结果编写《需求规格说明书》与《功能列表》，测试负责人（或专门的需求分析负责人）对文档进行检查并修改完善，然后 commit 到 CVS；

5、 测试负责人（或专门的 PPQA）确认所有相关文档经过了评审并都已经 commit 到 CVS。

二、项目计划与测试计划

1、 由开发经理组织项目计划讨论会，在讨论会上各开发负责人对自己所负责的模块所需要的工作量进行评估，根据工作量和工程需求初步确定总体开发计划、测试计划和发布时间；

2、项目经理根据估算工作量和工程需求编写项目计划，使用 CMMI5 总体测试计划模板并对其进行适当的裁剪和补充，编写适合本项目的项目计划；

3、测试负责人根据项目计划与发布时间编写测试计划，使用 CMMI5 总体测试计划模板并对其进行适当的裁剪和补充，编写适合本项目的测试计划；

4、项目计划与测试计划编写完成后发送给开发经理、项目经理、相关的开发人员和测试人员，开发经理、项目经理、相关的开发人员和测试人员阅读项目计划、测试计划后将建议和意见以邮件的形式反馈给项目经理与测试负责人，项目经理与测试负责人收集大家的邮件分别对项目计划与测试计划进行修改完善，同时回复邮件说明项目计划与测试计划修改情况，如果存在争议则召开一个小型会议对异议进行讨论，修改后的项目计划、测试计划 commit 到 CVS；

5、测试负责人（或专门的 PPQA）确认所有相关文档经过了评审并都已经 commit 到 CVS。

三、开发设计与评审

1、 项目经理构思系统设计，项目组开发成员一起讨论系统的设计，对设计形成较为清晰的思路；

2、 项目经理负责编写概要设计文档，与开发经理、开发团队成员与测试负责人一起讨论概要设计；

3、 概要设计完成后，项目经理编写详细设计文档、数据库设计文档和编码规范，各模块负责人负责编写所负责的模块进行详细设计；

4、 设计文档编写完成后，发邮件通知开发经理、项目经理、测试负责人、相关开发人员和测试人员；

5、 开发经理、项目经理、测试负责人、相关开发人员和测试人员对所提交的概要设计文档、详细设计文档进行审查，将建议和意见以邮件的形式反馈给模块负责人；

6、 模块负责人收集邮件中的修改建议并对设计文档进行修改，同时回复邮件说明详细设计修改情况，修改后的详细设计 commit 到 CVS；

7、如果对设计存在争议或出现明显不合理的设计，召开一个小型会议对异议进行讨论，有效解决设计所出现的分歧；

8、 测试负责人（或专门的 PPQA）对开发最终修改的详细设计计划进行检查，并确认所有文档都已经 commit 到 CVS。

注：在大型的项目中，必须先完成概要设计后再完成详细设计，在小项目或需求中可做适当剪裁概要设计与详细设计合在一起完成。

四、测试方案与评审

1、在项目的设计阶段，测试负责人根据规范文档、功能列表和概要文档编写总体测试方案与性能测试方案；

2、测试方案编写完成后，发邮件通知开发经理、项目经理、相关开发人员和测试人员；

3、开发经理、项目经理、测试负责人、相关开发人员和测试人员对所提交的测试方案进行审查，开发经理和项目经理对测试方案进行总体性的审查，而各模块负责人则负责相关模块或功能的测试方案的审查，将建议和意见以邮件的形式反馈给测试负责人；

4、测试负责人收集邮件中的修改建议并对测试方案进行修改，同时回复邮件说明测试方案修改情况，修改后的测试方案 commit 到 CVS；

5、测试负责人（或专门的 PPQA）对最终修改的测试方案进行检查，并确认所有文档都已经 commit 到 CVS。

五、编码实现与单元测试

1、在产品详细设计完成后，开发工程师依据设计进行编码工作；

2、编码完成后，开发工程师编写单元测试案例并进行单元测试，单元测试完成后提交单元测试报告；

3、项目经理根据项目实际情况对开发工程师编写的代码组织 Code Review，记录相关问

题；

4、产品模块单元测试完成后，开发之间进行产品联调测试，并修改所发现问题以及提交联调测试报告；

5、产品初步完成后，在提交测试前进行一次产品演示，参加人员包括开发经理、项目经理、测试负责人、开发工程师、测试工程师、售前工程师与售后工程师，在演示的过程中对产品提出改进建议；

6、各模块负责人对 Code Review 以及产品展示所发现的问题进行修改，相关的代码与文档 commit 到 CVS；

7、项目经理对编码完成后的系统进行确认，确保提交测试的系统是可运行的，测试负责人（或专门的 PPQA）确认所有文档和代码都已经 commit 到 CVS。

六、测试设计与评审

1、 在项目编码阶段，测试方案编写完成后，测试负责人或相关测试人员根据测试方案、规范文档、功能列表和详细设计进行测试用例设计；

2、 测试案例设计的类型包括功能测试，边界测试，异常测试，性能测试，压力测试等，在用例设计中，除了功能测试案例外，应尽量考虑边界、异常、性能的情况，以便发现更多的隐藏问题；

3、 在编写测试案例的过程中，对于存在疑问的地方或测试重点，主动与开发负责人或项目经理沟通讨论，一方面有助于设计完善的测试案例，另一方面也有助于开发进一步清晰编码思路；

4、 测试用例编写完成后，发邮件给开发经理、项目经理、相关开发人员和测试人员；

5、 开发经理、项目经理、相关开发人员和测试人员对所提交的测试案例进行审查，开发经理与项目经理对测试案例进行总体性的检查，各模块负责人则负责检查自己所负责的测试案例，将建议和意见以邮件的形式反馈给测试负责人；

6、测试负责人收集大家的邮件对测试案例进行修改完善，同时回复邮件说明修改情况，如果存在争议则召开一个小型会议对异议进行讨论，修改后的测试案例 commit 到 CVS；

7、 测试用例编写完成之后需要不断完善，软件产品新增功能或更新需求后，测试案例必须配套修改更新；在测试过程中发现设计测试案例时考虑不周，需要对测试案例进行修改完善；在软件交付使用后客户反馈的软件缺陷，而缺陷又是因测试案例存在漏洞造成，也需要对测试案例进行完善；

8、 测试负责人（或专门的 PPQA）对最终修改测试案例进行检查，并确认所有文档都已经 commit 到 CVS。

七、测试实施

1、代码提交前一天准备相关的测试环境（如服务器或数据库等），代码提交后测试人员向 Build Master 申请打包，并搭建正式测试环境，为了不做到测试以及确保产品可以跨平台，每个测试人员各自搭建一个测试环境，每个平台至少要有一个以上的测试人员负责；

2、测试环境搭建好后进行烟雾测试，如果烟雾测试通过则继续详细的功能测试，否则中断测试并返回给开发；

3、测试人员按照预定的测试计划和测试方案逐项对测试案例进行测试，在测试过程中发现的任何与预期目标不符的现象和问题都必须详细记录下来，填写测试记录，在必要的时候协助开发追踪与修改所发现的问题；如果在测试的过程中发现重大的 bug 或因为某些 bug 导致测试不能继续，测试中断并返回给开发；

4、每个测试阶段测试结束后，由测试负责人总结测试情况，对测试结果进行分析和下一阶段测试测试计划与可能引进的 bug 数量进行预测，并提交“测试阶段分析报告”，并发送给开发经理、项目经理、相关测试人员和开发人员；

5、开发经理对测试阶段分析报告中存在的问题采取恰当的措施和调整相关资源，确保下一阶段的开发与测试计划顺利进行；

6、开发对 bug 进行修改；

7、开发对 bug 修改后测试人员进行回归测试，经过修改的软件可能仍然包含着错误，甚至引入了新的错误，因此，对于修改以后的程序和文档，按照修改的方法和影响的范围，必须重新进行有关的测试；

8、产品的功能比较完善后，进行产品的性能压力测试，并根据测试结果进行性能调优；

9、确认测试，在软件发布前，对产品进行确认测试；

10、当测试产品达到测试计划所制定的产品质量目标和测试质量目标，整理产品发布包和编写相关文档，确认发布包和文档完整后进行产品发布。

八、产品发布

当测试产品达到测试计划所制定的产品质量目标和测试质量目标，整理产品发布包和编写相关文档，在发布前对照功能列表进行一次全面的确认测试，确认发布包和文档完整后进行产品发布。对于新产品来说，必要的文档必须包括：（1） 产品安装操作手册;（2） 产品白皮书;（3） 产品管理维护手册;（4） 用户操作手册;（5） 总体测试报告（6）性能测试报告。

九、版本控制

在测试过程中，软件的打包统一由 Build Master 完成。新版本软件发布之后，马上对代码进行质量控制：（1） Build Master 给新版本的源代码打一个 cvs tag，方便代码回滚 check out。比如，发布版本为 IAGW1.0.0，则给该软件源代码也打一个与发布版本相同名字的 tag IAGW1.0.0。这样做的一个好处是，在目前的软件的基础上做了修改并发布新的版本后，如果需要 check out 某个版本的源代码，则可以通过这个版本的 tag 来 check out，代码的修改可以在该版本上进行。（2） Build Master 对新发布的软件源代码进行 cvs lock，不允许开发人员在软件发布之后 commit 源代码，直到有新版本需求修改再给开发人员开放 commit 权限。这样做的好处是避免开发人员随意修改和 commit 源代码，确保源代码服务器上的源代码版本与当前最新的发布版本一致。

十、自动测试

产品稳定后，进行自动测试工具开发，对于稳定的功能使用自动测试工具进行测试，新增的功能使用手工测试，使用自动测试+手工测试的模式，可以大大提供测试效率。

十一、小结：应用推广思路与体会

整体思路是：首先对项目进行需求分析，有效的需求分析方法是需求分析人员、项目经理、开发经理与测试负责人分别阅读规范与原始需求，特别是需求分析负责人与项目经理，

需要对需求进行深入的分析研究，然后开会讨论，消除对需求的误解与遗漏，讨论结束后编写功能列表说明文档与需求规格说明书并评审；对于规范中不明确的问题集中后由测试负责人（或需求分析负责人）直接与移动总规范负责人直接交流，确保不会因为规范的理解不正确导致项目实现与需求不一致。需求分析完成后，编写项目计划书与测试计划书；项目计划、测试计划编写前先开会讨论，由模块负责人估算工作量，能确定的问题和时间安排都在讨论中确定下来，然后根据工作量和工程需求制定项目计划和测试计划。开发在编码前需要进行概要设计和详细设计，开发工程师在编码前对系统的总体设计架构、各自所负责的模块有一个清晰的设计思路，经评审后确认模块的设计是否合理；开发在编码完成后在提交测试前必须进行单元测试与联调测试，提交给测试的软件是一个可运行的产品。测试工作中，在项目设计或编码阶段，测试负责人对项目进行测试设计，指导测试实施有依可循，在编写案例的过程中会遇到很多与流程和细节处理相关的问题，与开发一起讨论也有助于提前发现问题与完善代码；在测试实施阶段，测试人员记录所发现的问题，并协助开发及时解决，在测试过程中所遇到的问题，测试负责人进行记录和分析，在每个阶段完成后提交经分析后的测试阶段报告，在软件测试阶段报告中总结分析了测试过程中所发现的问题并对这些问题提出解决建议，在后续的开发与测试中进行改进与调整，确保项目能够按时保质发布。为了节约资源，计划或设计都是以邮件的形式进行评审；对于存在严整分歧的问题，组织一个小型会议进行讨论有效解决问题，小型讨论会是解决问题的一种有效途径，任何问题都可以通过 face-to-face 的交流达到共识。软件的管理和版本管理则由 Build Master 负责，确保软件得到良好的控制。在整个项目实施的过程中，需要有一个 PPQA 对流程进行检查与监督。

这个精简的实施流程，不但确保了软件的质量，而且实施成本较低，在团队实施中非常容易推广。 在整个流程中，测试负责人除了负责测试相关任务以外，同时承担了需求管理、流程跟踪、协调沟通等工作（当然，也可由项目经理或开发经理等担任），在其中由测试推动项目开发与实现，在开发成员之间、开发与测试之间搭了一座沟通的桥梁，这样的一个协调与推动促进了项目的顺利完成，适合于五至二十的小型团队。不过这种测试与开发的模式，对测试负责人的要求很高，不但要求测试负责人具有很强的责任心与沟通协调能力，而且还需要具有很高的业务分析能力和 CMMI5 实施经验。

Casl 汇编语言辅导（上）

一、Casl 汇编语言语法介绍

学习一个汇编语言需要掌握 3 个要点：CPU 的可编程寄存器结构、寻址方式及指令系统、伪指令。

1、COMETCPU 的可编程寄存器

COMETCPU 字长 16 位，采用从左到右的编号。bit0 在最左边（最高位），bit15 在最右边（最低位）

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

1）16 位通用寄存器五个：

GR0、GR1、GR2、GR3、GR4

通用功能：各种算术逻辑运算

特殊功能：除 GR0 外都可作变址寄存器（地址指针）XR，GR0 可看成累加器。

GR4 兼作堆栈指针（SP）

2）指令计数器 PC 一个（16 位）

存放正在执行的那条指令的第 1 个字的地址（一条指令占二个字），指令结束时，PC 中存放下一条指令的地址 （一般为原指令地址 +2 ）。

3）状态寄存器 FR 一个（二位）

运算结果 FR0 FR1

大于 0 0

等于 0 1

小于 1 0
可以把 FR0 看成 SF（符号位），FR1 看成 ZF（零位位）
除了算术逻辑运算指令（包括移位指令）外，LEA 指令也会影响 FR

2、符号指令写法与寻址方式
OP GR，ADR[，XR]
其中 OP 为操作码；GR 为五个通用寄存器 GR0～GR4 之一；ADR 为一个 16 位的地址码；XR 为四个变址寄存器 GR1～GR4 之一，[ ]表示可以省略。
1）直接寻址：当 [，XR] 省略时，为直接寻址。
2）变址寻址：有效地址 E = ADR +（XR），当 ADR = 0 时，为寄存器间接寻址。
3）立即数寻址：Casl 没有立即数寻址。但在 LEA 指令中，当 [，XR] 省略时，可作立即数传送。没有立即数运算指令。
4）寄存器寻址：Casl 也没有寄存器寻址（对源操作数）。但 LEA 指令当 ADR = 0 时，可作寄存器寻址（只用 于数据传送）。

3、Casl 指令系统
1）取数指令 LD：内存→寄存器
LD GR，ADR[，XR]
2）存数指令 ST：寄存器→内存
ST GR，ADR[，XR]
3）加法 ADD、减法 SUB、逻辑与 AND、逻辑或 OR、逻辑异或 EOR 指令：
ADD ┐
SUB │
AND ├ GR，ADR[，XR]
OR │
EOR ┘
内存 OP 寄存器→寄存器
4）算术比较 CPA：
两个带符号数比较，结果不回送，只影响标志位。
CPA GR，ADR[，XR]
5）逻辑比较 CPL：两个无符号数比较，结果不回送，只影响标志位。
CPL GR，ADR[，XR]
6）算术左移 SLA、算术右移 SRA：把操作数看成带符号数。对寄存器操作数进行移位（GR 的第 0 位——符号位不变。右移时空出的其余位补与第 0 位相同的 1 或 0 ；左移时空出的位补 0 ），位数由有效地址 E 决定。
SLA ┐ GR，ADR[，XR]
SRA ┘
7）逻辑左移 SLL、逻辑右移 SRL：把操作数看成无符号数。对寄存器操作数进行移位（不管左右移，GR 空出的位补 0 ），位数由有效地址 E 决定。
SLL ┐ GR，ADR[，XR]
SRL ┘
8）取有效地址指令 LEA：E→寄存器
LEA GR，ADR[，XR]

该指令有几个特殊用途：
【例 1】LEA GR0，41H 把立即数 41H 送给 GR0
【例 2】LEA GR0，0，GR1 把寄存器 GR1 的内容送给 GR0
【例 3】LEA GR1，1，GR1 寄存器 GR1 的内容加 1（相当于 8086 汇编中的 INC 指令）
【例 4】LEA GR1，-1，GR1 寄存器 GR1 的内容减 1（相当 8086 汇编中的 DEC 指令）
【例 5】LEA GR1，N，GR1 寄存器 GR1 的内容加 N（相当于立即数加法）

【例 6】LEA GR1，-N，GR1 寄存器 GR1 的内容减 N（相当于立即数减法）

9）无条件转移指令 JMP 与条件转移指令 JPZ（不小于转）、JMI（小于转）、JNZ（不等于转）、JZE（等于转）

JMP ┐
JPZ │
JMI ├ ADR[，XR]
JNZ │
JZE ┘

●当地址码中缺 [XR] 时，所有转移指令为直接转移（ ADR 即为转移地址）
当地址码中有 [XR] 时，所有转移指令为间接相对接转移
●JPZ 与 JMI 根据符号位 FR0 作为判断条件
JNZ 与 JZE 根据零位位 FR1 作为判断条件

10）进栈 PUSH 与出栈 POP：

（1）进栈指令 PUSH：

PUSH ADR[，XR]
(SP)-1→SP，E→(SP)

（2）出栈指令 POP：

POP GR
((SP))→GR，(SP)+1→SP

注意：出栈指令的目的寄存器是 GR0～GR4，而进栈指令的源操作数不包括 GR0。

11）子程序调用 CALL 与返回 RET 指令

（1）子程序调用指令 CALL：

CALL ADR[，XR]
(SP)-1→SP，(PC)+2→(SP)，E→(PC)

（2）子程序返回指令 RET：

RET
((SP))→PC，(SP)+1→SP

4、伪指令

1）START：程序开始

2）END：程序结尾

3）常量定义指令 DC：

此伪指令与其它汇编语言中的 DB 指令似。利用 DC 指令可定义：

（1）定义十进制常数：

十进制常数名 DC n
其中 -32768 < n ≤65535

（2）定义字符串常数：

字符串常数名 DC '字符串'

（3）定义十六进制常数：

十六进制常数名 DC #h
其中 0000 ≤ h ≤FFFF

（4）定义地址：

地址常数 DC LABLE
其中 LABLE 是程序中的地址标号

因为 Casl 没有立即数运算指令,所以需要与立即数进行算术逻辑运算时,都要将立即数定义为内存常数进行运算。

4）变量定义指令 DS：用来保留指定字数的存储区域

[LABLE] DS n

其中 n 是十进制常数（≥0），当 n=0 时，存储区域不存在，但标号 LABLE 仍然有效，即代表下一字的地址。

5）宏指令：IN、OUT、EXIT

Casl 中有进行输入、输出及结束程序等宏指令，而没有定义输入、输出符号指令，这类处理由操作系统完成。

程序中出现宏指令时，Casl 生成调用操作系统的指令串，但是，生成的指令串字数不定。执行宏指令时，GR 的内容保持不变，而 FR 的内容不确定。

（1）输入宏指令 IN：

[LABLE] IN ALABLE，NLABLE

宏指令 IN 从输入装置上输入一个记录，纪录中的信息（字符）依次按字符数据的形式存放在标号为 ALABLE 开始的区域内，已输入的字符个数以二进制数形式存放在标号为 NLABLE 的字中，纪录之间的分隔符号不输入。

（2）输出宏指令 OUT：

[LABLE] OUT ALABLE，NLABLE

宏指令 OUT 将存放在标号为 ALABLE 开始的区域中的字符数据作为一个记录向输出装置输出，输出的字符个数由标号为 NLABLE 的字的内容指定。输出时，若要纪录间的分隔符号，由操作系统自动插入输出。

（3）宏指令 EXIT ：

[LABLE] EXIT

宏指令 EXIT 表示程序执行的终止，控制返回操作系统。

二、基本程序结构

1、顺序程序（略）

2、分枝程序

1）简单分支程序

将比较指令（或其它能使标志位发生变化的指令）和条件转移指令结合，可实现分支程序。简单分支程序有两种形式：

在汇编语言中，采用图 a 的结构容易出错，例如把 GB0 中的十六进制数转换成 ASCII 码可分成两段，0～9 的 ASCII 码是 30H～39H ，即加 30H ；A～F 的 ASCII 码是 41H～45H ，即加 37 H。两种结构的程序如下：

图 a 结构 图 b 结构

```
CPL GR0,C10 CPL GR0,C10
JMI L1 JMI L1
ADD GR0,C37 ADD GR0,C7
JMP L2 L1 ADD GR0,C30
L1 ADD GR0,C30 …
L2 … C10 DC 10
C10 DC 10 C7 DC 7
C37 DC #37 C30 DC #30
C30 DC #30
```

若采用图 a 的结构，很容易把 JMP L2 漏掉，变成 A～F 的 ASCII 码加了 67H。

2）多岔分支程序

可以用多条比较指令和条件转移指令实现多岔分支。

但用散转表实现多岔分支则程序更为简练，其思路是在散转表中存放各个分支程序的入口地址，用查表方法将入口地址放入变址寄存器，然后用 JMP 指令或 CALL 指令的间接寻址方

式使程序转到此入口。

【例 1】根据键盘输入命令转入各命令处理程序

```
SB START
AGAIN IN ALABLE，NLABLE
；输入一个字符串
LD GR1，NLABLE ；字符串长度
LEA GR1，0，GR1
JZE AGAIN ；若字符串长度 =0，重输
LD GR1，ALABLE ；将第一个字符放到 GR1 中
LEA GR1，-65，GR1 ；减去"A"的 ASCII 码
JMI AGAIN ；若该字符<"A"，重输
CPA GR1，C4
JPZ AGAIN ；若该字符>"D"，重输
LD GR2，ENTRY，GR1 ；将散转表中的一项地址放入 GR2
CALL 0，GR2 ；转入地址所指子程序
JMP AGAIN
ALABLE DS 5 ；输入字符串缓冲区
NLABLE DS 1 ；输入字符串长度
C4 DC 4 ；常数 5
ENTRY DC ASUBR ；子程序 A 入口地址
DC BSUBR ；子程序 B 入口地址
DC CSUBR ；子程序 C 入口地址
DC DSUBR ；子程序 D 入口地址
  END
```

在散转表中也可存放转移到各个分支程序入口的转移指令，然后用变址寻址方式的 JMP 指令或 CALL 指令使程序转到此入口。

【例 2】

```
SB START
AGAIN IN ALABLE，NLABLE
；输入一个字符串
LD GR1，NLABLE ；字符串长度
LEA GR1，0，GR1
JZE AGAIN ；若字符串长度 =0，重输
LD GR1，ALABLE ；将第一个字符放到 GR1 中
LEA GR1，-65，GR1 ；减去"A"的 ASCII 码
JMI AGAIN ；若该字符<"A"，重输
CPA GR1，C4
JPZ AGAIN ；若该字符>"D"，重输
SLL GR1，1 ；散转表每项占 2 字
JMP ENTRY，GR1
ALABLE DS 5
NLABLE DS 1
C4 DC 4
ENTRY JMP ASUBR
JMP BSUBR
JMP CSUBR
JMP DSUBR
END
```

3、循环程序

循环程序也是用转移指令实现，循环的退出一般用循环计数器实现，也可用其它条件来控制退出。

1）用循环计数器控制

【例 1】将 SD 处的 100 个数搬到 TD 处（搬家程序）

```
SB START
LEA GR1，100 ； 循环计数器
LEA GR2，0 ； 偏移地址指针
LOOP LD GR0，SD，GR2 ； 源操作数→GR0
ST GR0，TD，GR2 ； GR0→目的操作数
LEA GR2，1，GR2 ； 修正偏移地址指针
LEA GR1，-1，GR1 ； 搬完否？
JNZ LOOP
EXIT
SD DS 100
TD DS 100
END
```

当源操作数和目的操作数有重迭时，若源操作数在目的操作数的前面，则必须采用从下往上的搬法，即先搬最后一个，再一个一个往前搬。

【例 2】将 SD 处的 100 个数搬到 TD 处(有重迭)

```
SB START
LEA GR1，100 ； 循环计数器
LEA GR2，99 ； 偏移地址指针
LOOP LD GR0，SD，GR2 ； 源操作数→GR0
ST GR0，TD，GR2 ； GR0→目的操作数
LEA GR2，-1，GR2 ； 修正偏移地址指针
LEA GR1，-1，GR1 ； 搬完否？
JNZ LOOP
EXIT
SD DS 1
TD DS 100
END
```

2）循环次数可能为 0 的循环程序

前面的例子无法实现循环次数可能为 0 的循环程序（如用连加实现两个数相乘的算法），因为循环计数器预置为 0 时，实际的循环次数是 65536。

【例 2】将 A 和 B 两个整数相乘和放到 C 中（不考虑溢出）

```
SB START
LD GR1，B ； 乘数作循环计数器
LEA GR0，0 ； 乘积清 0
LOOP LEA GR1，0，GR1
JZE L1 ； 若乘数为 0，退出循环
ADD GR0，A ； 被乘数加到乘积中
LEA GR1，-1，GR1 ； 加完否？
JMP LOOP
L1 ST GR0，C ； 乘积→C
EXIT
A DS 1
B DS 1
```

```
C DS 1
END
```

3）循环次数不定的循环程序

不用循环计数器，而用其它方法控制循环的退出。

【例 4】测试字符串 STR 的长度，并将它保存到 L 中

```
SB START
LEA GR1，0 ；字符串长度计数器（兼地址指针）清 0
LOOP LD GR0，STR，GR1 ；取一字符
CPL GR0，FEND
JZE L1 ；若是结束符，退出循环
LEA GR1，-1，GR1 ；长度计数器+1
JMP LOOP
L1 ST GR0，L ；长度→L
EXIT
L DS 1
FEND DC '$'
STR DC 'This is a sample$'
END
```

4）多重循环

【例 5】冒泡排序程序。

一组有符号数 al（i=1，2，…，100），存放在 Al 开始的连续单元中。下面一程序将这组数在原来的内存区中由大到小重新排序。

```
SB START
LEA GR1，99
ST GR1，CN ；外循环计数器初值
L1 LEA GR1，0 ；外循环指针
LEA CR2，1 ；内循环指针兼计数器初值
LEA GR3，0 ；交换标志置 0
L2 LD GR0，A1，GR1
CPA GR0，A1，GR2
JPZ L3 ；内循环体，若 Ai≥Ai+1，不动
LD GR4，A1，GR2 ；若 Ai<Ai+1，交换
ST GR4，A1，GR1
ST GR0，A1，GR2
LEA GR3，1 ；交换标志置 1
L3 LEA GR1，1，GR1 ；内循环调整
LEA GR2，1，GR2
CPL GR2，CN
JZE L2 ；内循环控制
JMI L2
LEA CR3，0，CR3
JZE L4 ；本次内循环交换标志为 0，排序结束
LD GR4，CN ；外循环计数减 1
LEA GR4，-1，GR4
JZE L4 ；排序结束
ST GR4，CN
JMP L1
L4 EXIT
```

```
A1 DS 100
CN DS 1
END
```

这是一个二重循环结构的程序。程序中采用冒泡排序的方法，首先从第 1 个单元开始，依次把相邻两个元素比较，即第 1 个单元内的数与第 2 个单元内的数比较，第 2 个单元内的数与第 3 个单元内的数比较，…第 99 个单元与第 100 个单元的数相比较。每次把较大者放在前面单元，较小者放在后面单元。这样第一次遍历做了 n-1 次比较，最后最小的元素放在第 100 个单元。

第二次遍历再从头开始，依次将相邻两个元素进行比较，把 n-1 个元素遍历一次需做 n-2 次比较，这样，第 99 个单元存放全部元素中次小元素。

如此反复，小的元素往下，大的元素向上犹如汽泡上浮，因此这种排序的方法给它一个形象的名字，叫做冒泡算法。在程序中还设 GR3 为交换标志，若本次内循环结束交换标志为 0，说明没有交换。即所有的 Ai≥Ai+1，排序结束。

4、子程序

所谓子程序是指完成一定功能的指令序列，在主程序的若干地方可以用 CALL 指令对它调用，子程序结束时通过 RET 指令返回主程序。在子程序中，如果没有特别说明，则子程序中所要使用到的寄存器要保护。在子程序的开始用 PUSH 指令将子程序中要使用的寄存器保护，在返回前用 POP 指令恢复。主程序与子程序可以处在同一个程序中，此时，主程序对子程序的调用称为程序内的调用，CASL 中也允许主程序与子程序不在同一个程序中，这样的调用就称为程序间调用。

在子程序中一般不将 GR4 作为通用寄存器使用，因为 GR4 作为栈顶指针。如果其它通用寄存器不够用，要使用 GR4，则先将 GR4 中的栈顶指针值保存，然后使用 GR4，GR4 使用完后，立即恢复栈指针的值。

1）参数传递

主程序调用子程序时，通常要向子程序传递参数（入口参数）。子程序结束，也会有数据送回主程序（返回参数）。当参数较少时，主程序将参数存于寄存器（GR0～GR3）传递给子程序；也可放于内存给子程序；当传递参数较多时，通常存于内存（参数表），并将参数表的首地址存于寄存器传递给子程序；也可以通过堆栈传递参数值。

【例 1】寄存器传递参数

```
MAIN START
LD GR0，A ；被加数送 GR0
LEA GR1，B ；加数送 GR1
CALL SUBA
ST GR0，C ；结果回送
……
EXIT
SUBA ADD GR0，0，GR1
RET
A DC 56
B DC 89
C DS 1
D DC 186
E DC 567
F DS 1
END
```

【例 2】内存传递参数

```
MAIN START
```

```
LD GR0，A ；被加数送 GR0
LD GR1，B
ST GR1，BUF ；加数送 BUF 内存单元
CALL SUBA
ST GR0，C ；结果回送
……
EXIT
SUBA ADD GR0，BUF
RET
BUF DS 1
A DC 56
B DC 89
C DS 1
D DC 186
E DC 567
F DS 1
END
```

【例 3】参数表传递参

```
MAIN START
LEA GR1，LIST ；参数表首址送 GR1
CALL SUBA
EXIT
SUBA LD GR0，0，GR1 ；被加数
ADD GR0，1，GR1 ；与加数相加
ST GR0，2，GR1 ；结果回送
RET
LIST DC 56
DC 89
DS 1
END
```

【例 4】堆栈传递参数

```
MAIN START
LD GR1，A
PUSH 0，GR1 ；被加数压入堆栈
LD GR1，B
PUSH 0，GR1 ；加数压入堆栈
LEA GR1，C
PUSH 0，GR1 ；结果地址压入堆栈
CALL SUBA
LEA GR4，3，GR4 ；堆栈指针退回
EXIT
SUBA LD GR0，3，GR4 ；被加数
ADD GR0，2，GR4 ；与加数相加
LD GR1，1，GR4 ；结果地址
ST GR0，0，GR1 ；结果回送
RET
A DC 56
B DC 89
C DS 1
END
```

2）子程序的返回

子程序通过 RET 指令返回主程序。RET 指令的功能是将栈顶的返回地址置入 PC。一般情况，应该保证返回时的堆栈指针 SP（GR4）与调用时一致。以保证正确返回。但 CASL 中，GR4 兼作通用寄存器，变址器和栈顶指针，因此在子程序中可以通过 GR4 修改返回地址，这样子程序返回时，就不一定返回到主程序的调用点下面的一条指令。在早期的程序员 CASL 试题中常常利用这一技巧。

3）现场保护

通常情况下，子程序都应该保护它所使用过的寄存器。因 PUSH 指令不能保护 GR0，所以 GR0 用 ST 指令保护，用 LD 指令恢复。

【例 4】无符号乘法。

有两个无符号数 N1，N2，下面一子程序实现 N1×N2 返回。调用子程序时 GR1 存放参数 N1、N2 和相乘结果的存放区域的首地址，如下图所示。设相乘过程不会产生溢出。

```
GR1+0 N1
+1 N2
+2  结果
MULT START
ST GR0，SAVE
PUSH 0，GR2 ；保护寄存器
PUSH 0，GR3
LEA GR2，0 ；部分积清零
LEA GR3，16 ；循环计算器
LD GR0，1，GR1 ；取出乘数 N2
LOOP SLL GR2，1 ；部分积左移一位
SLL GR0，0 ；测乘数 N2 最高位是否为 1
JPZ L1 ；若为 0，跳过这一位
ADD GR2，0，GR1 ；若为 1，部分积加上被乘数
L1 SLL GR0，1 ；乘数左移一位，算下一位
LEA GR3，-1，GR3 ；16 位算完否？
JNZ LOOP
ST GR2，2，GR1 ；存放乘积
LD GR0，SAVE ；恢复寄存器的值
POP GR3
POP GR2
RET
SAVE DS 1
END
```

象手算乘法一样，由于两个数是二进制数，所以唯一的问题是乘 0 还是乘 1；显然，乘 0 结果为 0，而乘 1 则结果与开始的数(被乘数)相同。因此在二进制乘法中每一步都可以简化成下列运算：

如果乘数中现有位为 1，则将被乘数加到部分积上，为 0 则不加。剩下的问题是部分积和被乘数每次相加要进行移位，保证每次相加都正确对准。乘法过程可分成以下几步：

(1)积清 0，设置计数器初值 16。

(2)积左移 1 位。

(3)乘数最高位是否为 1。

(4)若为 1，则被乘数加积。

(5)乘数左移 1 位，计数器减 1，若不为 0 转(2)。

程序中 GR0 存乘数，GR2 存积，GR3 为计数器

三、汇编语言常用子程序

1、拆字与拼字：

【例 1】将 GR0 中的四位 BCD 码从高到低依次存放到 GR2 所指的四个内存单元中。

```
START
LEA GR3,4 ； 循环计数器
L1 ST GR0,REG ； 保护其余几位 BCD 码
AND GR0,C000F ； 屏蔽高 3 位，留下最低 1 位 BCD 码
ST GR0,3,GR2 ； 将此位 BCD 码存放到 GR2 所指第四个内存单元
LD GR0,REG ； 恢复其余几位 BCD 码
SRL GR0,4 ； 将已处理过的此位 BCD 码移出
LEA GR2.-1,GR2 ； 地址指针减 1
LEA GR3,-1,GR3 ； 循环计数器减 1
JNZ L1 ； 未处理完，继续
RET
C000F DC #000F ； 十六进制常数，屏蔽高 3 位 BCD 码用
REG DS 1 ； 暂存单元
END
```

【例 2】将存放在 GR2 所指的四个内存单元中的四位 BCD 码依从高到低顺序压缩到 GR0 中 。

```
START
LEA GR0,0 ； GR0 清 0
LEA GR3,4 ； 循环计数器
L1 SLL GR0,4 ； 将已处理过的 BCD 码移到高位
LD GR1,0,GR2 ； GR1 用作临时工作寄存器
AND GR1,C000F ； 屏蔽高 12 位
ST GR1,0,GR2 ； 对内存单元中的 BCD 码预处理
ADD GR0,0,GR2 ； 将已处理过的此位 BCD 码加到 GR0 低位
LEA GR2.1,GR2 ； 地址指针指向下一位 BCD 码
LEA GR3,-1,GR3 ； 循环计数器减 1
JNZ L1 ； 未处理完，继续
RET
C000F DC #000F ； 十六进制常数，屏蔽高 12 位二进制数
END
```

2、数字与 ASCII 码之间的相互转换：

十进制数字 0～9 的 ASCII 码是 30H～39H ，所以只要将十进制数（BCD 码）加 30H 就是对应的 ASCII 码。

十六进制数转换成 ASCII 码可分成两段，0～9 的 ASCII 码是 30H～39H ，即加 30H ；A～F 的 ASCII 码是 41H～45H ，即加 37 H。

【例 1】将 GR0 中的四位 BCD 码化成 ASCII 码从高到低依次存放到字符串变量 STR 中。

```
START
LEA GR2,3 ； 相对于 STR 首址的地址指针
LEA GR3,4 ； 循环计数器
L1 ST GR0,REG ； 保护其余几位 BCD 码
AND GR0,C000F ； 屏蔽高 3 位，留下最低 1 位 BCD 码
ADD GR0,C30 ； 转换成 ASCII 码
ST GR0,STR,GR2 ； 将 ASCII 码存放到 GR2 所指第四个内存单元
LD GR0,REG ； 恢复其余几位 BCD 码
SRL GR0,4 ； 将已处理过的此位 BCD 码移出
```

```
LEA GR2.-1,GR2 ； 地址指针减 1
LEA GR3,-1,GR3 ； 循环计数器减 1
JNZ L1 ； 未处理完，继续
RET
C000F DC #000F ； 十六进制常数，屏蔽高 3 位 BCD 码用
C30 DC #30 ； 十六进制常数 30
STR DS 4
REG DS 1 ； 暂存单元
END
```

【例 2】将 GR0 中的 16 位二进制数化成四位十六进制数 ASCII 码从高到低依次存放到字符串变量 STR 中。

```
START
LEA GR2,3 ； 相对于 STR 首址的地址指针
LEA GR3,4 ； 循环计数器
L1 ST GR0,REG ； 保护其余几位二进制数
AND GR0,C000F ； 屏蔽高 12 位，留下最低 4 位二进制数
CPL GR0,C10 ； < 10 否？
JMI L2 ； < 10 跳过加 7 ，只加 30H
ADD GR0,C7 ； ≥ 10，加 30H 前先加上 7
L2 ADD GR0,C30 ； 加上 30H
ST GR0,STR,GR2 ； 将 ASCII 码存放到 GR2 所指第四个内存单元
LD GR0,REG ； 恢复其余几位二进制数
SRL GR0,4 ； 将已处理过的此 4 位二进制数移出
LEA GR2.-1,GR2 ； 地址指针减 1
LEA GR3,-1,GR3 ； 循环计数器减 1
JNZ L1 ； 未处理完，继续
RET
C000F DC #000F ； 十六进制常数，屏蔽屏蔽高 12 位二进制数
C30 DC #30 ； 十六进制常数 30
C10 DC 10 ； 十进制常数 10
C7 DC 7 ； 常数 7
STR DS 4
REG DS 1 ； 暂存单元
END
```

【例 3】将字符串 STR 中的四位十六进制数的 ASCII 码化成 16 位二进制数放到 GR0 中 。

```
START
LEA GR0,0 ； GR0 清 0
LEA GR2,0 ； 相对于 STR 首址的地址指针
LEA GR3,4 ； 循环计数器
L1 SLL GR0,4 ； 将已处理过的十六进制数移到高位
LD GR1,STR,GR2 ； GR1 用作临时工作寄存器
AND GR1,C00FF ； 屏蔽高 8 位
SUB GR1,C30 ； 减去 30H
CPL GR1,C0A ； < 10 否？
JMI L2 ； < 10 ，完成转换
SUB GR1,C7 ； ≥ 10，再减去 7
L2 ST GR1,STR,GR2 ； 将 STR 中的 ASCII 码转换成十六进制数
```

```
ADD GR0,STR,GR2 ；将此位十六进制数加到 GR0 低位
LEA GR2.1,GR2 ；地址指针指向下一位 ASCII 码
LEA GR3,-1,GR3 ；循环计数器减 1
JNZ L1 ；未处理完，继续
RET
C00FF DC #00FF ；十六进制常数，屏蔽高 8 位用
C30 DC #30 ；十六进制常数 30
C0A DC #0A ；十六进制常数 0A
C7 DC 7 ；常数 7
STR DS 4
END
```

3、利用加减法及移位指令做乘法：

1）左移指令可将操作数乘 2 的整数次方（2、4、8、16）；右移指令可将操作数除以 2 的整数次方。

若操作数是无符号数，用逻辑移位指令；若操作数是有符号数，用算术移位指令。

【例 1】将 GR0 中的二进制数乘以 8。

```
SLL GR0,3
```

【例 2】将 GR0 中的带符号二进制数除以 4。

```
SRA GR0,2
```

2）将移位指令和加减法结合起来可完成乘数不是 2 的整数次方的乘法运算。

【例 1】将 GR0 中的二进制数乘以 10。

```
START
SLL GR0,1
ST GR0,REG
SLL GR0,2
ADD GR0,REG
RET
REG DS 1
END
```

【例 2】将 GR0 中的二进制数乘以 7。

```
START
ST GR0,REG
SLL GR0,3
SUB GR0,REG
RET
REG DS 1
END
```

4、二进制数与十进制数的转换

1）二化十：

将二进制数转换为十进制数的一种常用算法是将被转换的二进制数依次被 10i（对 16 位二进制数，i 为 4、3、 2、1、0）除，所得的商即为该十进制数位的值，其余数再被下一个 10i 除。一般用减法代替除法，即一边减 10i，一边计数器加 1，直到不够减再进行下一位 10i-1。以求得十进制数的各位数。

例如：一个相当于十进制数 34635 的二进制数，可先用 10000 去减，可减 3 次，得万

位数是 3；再用 1000 去减，得千位数是 4；……

【例 1】将 GR0 中的二进制数转换为十进制数的 ASCII 码放入字符串 STR 中。

```
START
LEA GR1，0 ；减数表及字符串指针
LEA GR2，5 ；循环计数器
L1 LEA GR3，48 ；该十进制位的数码预置 0 的 ASCII 码
L2 LEA GR3，1，GR3 ；数码位的 ASCII 码加 1
SUB GR0，SNO，GR1 ；操作数减去 10i
JPZ L2 ；够减，继续
ADD GR0，SNO，GR1 ；不够减，操作数及数码位的 ASCII 码恢复
LEA GR3，-1，GR3
ST GR3，STR，GR1 ；转换好的该位 ASCII 码存入结果字符串
LEA GR1，1，GR1 ；地址指针加 1
LEA GR2，-1，GR2 ；循环计数器减 1
JNZ L1 ；未结束，继续下一位
RET
SNO DC 10000
DC 1000
DC 100
DC 10
DC 1
STR DS 5 ；转换结果字符串
END
```

1）十化二：

将十进制数转换为二进制数的算法基础是下面公式：

N = (Dn-1*10n-1+Dn-2*10n-2+……+D1*101+D0*100

= ((…((Dn-1*10+Dn-2)*10+……+D1)*10+D0)*10

可以用循环程序实现此公式，*10 可用移位及加法指令完成。

【例 2】将存放在字符串 STR 中的五位十进制数（<65536）的 ASCII 码转换成二进制数放到 GR0 中 。

```
START
LEA GR0,0 ；转换结果寄存器清 0
LEA GR2,5 ；循环计数器
LEA GR1,0 ；地址指针（偏移量）
L1 SLL GR0,1 ；转换结果*10，先乘以 2
ST GR0,REG ；暂存 2*X
SLL GR0,2 ；2*X*4=8*X
ADD GR0,REG ；8*X + 2*X
LD GR3,STR,GR1 ；取一位 ASCII 码
AND GR3,C000F ；将 ASCII 码变成 BCD 码
ST GR3,REG ；结果暂存
ADD GR0,REG ；将新的一位 BCD 码加到转换结果中
LEA GR1,1,GR1 ；地址指针加 1
LEA GR2,-1,GR2 ；循环计数器减 1
JNZ L1 ；未结束，继续下一位
RET
C000F DC #000F ；十六进制常数，屏蔽高 12 位二进制数
STR DC '35475'
REG DS 1 ；暂存单元
```

```
END
```

5、求累加和

【例 1】将变量 NUMBER 中的 5 个二进制数累加后放入变量 SUM 中。

```
START
LEA GR2，NUMBER ；地址指针
LEA GR3，5 ；循环计数器
LEA GR0，0 ；累加和清 0
L1 ADD GR0，0，GR2 ；累加
LEA GR2，1，GR2 ；地址指针指向下一个二进制数
LEA GR3，-1，GR3 ；计数器减 1
JNZ L1 ；未完，继续
ST GR0，SUM ；累加结束，累加和送入 SUM 单元
RET
NUMBER DS 5
SUM DS 1
END
```

【例 2】将自然数 1～16 累加后放入变量 SUM 中。

```
START
LEA GR1,0 ；地址指针
LEA GR0，0 ；累加和清 0
L1 LEA GR1,1,GR1 ；自然数加 1
ST GR1,SUM ；加数(自然数)送入内存暂存
ADD GR0，SUM ；累加
CPA GR1,A
JNZ L1 ；未完，继续
ST GR0，SUM ；累加结束，累加和送入 SUM 单元
EXIT
A DC 16
SUM DS 1
END
LEA GR1,0
LEA GR0,0 ;累加和(被加数)
CF LEA GR1,1,GR1
;
ADD GR0,C
JNZ CF
ST GR0,B
EXIT
A DC 16
```

6、利用递归方法求 5 的阶乘

```
START ；主程序
LEA GR1，5 ；入口参数 5（阶乘）→GR1
CALL FACT ；调用 FACT 子程序
EXIT
FACT LEA GR1，0，GR1
JNZ IIA ；GR1 未到 0，继续递归
LEA GR1，1 ；GR1=0，退出递归
RET
```

```
IIA PUSH 0，GR1 ；保护这一级乘数
LEA GR1，-1，GR1 ；乘数减 1
CALL FACT ；继续调用 FACT 子程序自己
POP GR1 ；逐级退出递归，恢复本级乘数
LEA GR0，0
CALL MUL ；乘上本级乘数
RET
MUL ADD GR0，RESUL
LEA GR1，-1，GR1
JNZ MUL
ST GR0，RESUL
RET
RESUL DC 1
END
```

Casl 汇编语言辅导（下）

三、试题解释

1、2002 年试题四 [程序说明]

本程序将 80 个 ASCII 编码的数字字符转换成 BCD 码(二十进制码)，并将每四个 BCD 码压缩在一个字中。见下面图示。

数字字符数据地址 ASCII

+0 0033H '3'

+1 0036H '6'

+2 0038H '8'

+3 0032H '2'

... ...

压缩后的数据地址 BCD 码

YS 3682H

... ...

程序中约定原始数字字符存放在 SJ 开始的连续存区中，转换和压缩结果存放在 YS 开始的连续存区中。

[程序]

```
Y START 1
LEA GR1，0 2
__(1)__ 3
S0 LEA GR3，4 4
S1 LD GR0，SJ，GR2 5
__(2)__ 6
ST GR0，WK 7
__(3)__ 8
OR GR4，WK 9
LEA GR3，-1，GR3 10
JNZ S2 11
ST GR4，YS，GR1 12
LEA GR1，1，GR1 13
__(4)__ 14
S2 LEA GR2，1，GR2 15
CPL GR2，C80 16
__(5)__ 17
WL EXIT 18
```

```
SJ DS 80 19
WK DS 1 20
CF DC #000F 21
C80 DC 80 22
YS DS 20 23
END
```

从程序说明中可知要将 80 个 ASCII 编码的数字字符转换成 BCD 码，并将每四个 BCD 码压缩在一个字中。必须有双重循环：内循环将每四个 BCD 码压缩在一个字中，外循环完成 80 个 ASCII 码转换（20 个内循环）。从第 4 行 LEA GR3，4 可看出，GR3 是内循环计数器，S1 是内循环开始标号。

从第 2 行、第 5 行及第 15 行可知，GR2 是源地址指针（在第 2 行赋初值），GR1 是目的地址指针，应该在第 3 行赋初值。从而得到__(1)__空应为 LEA GR2，0。

从第 7 行及第 9 行可知，第 7～9 行是将转换好的 BCD 码拼装到压缩字中。GR4 是压缩字、WK 是放转换好的 BCD 码的暂存单元。由此得__(2)__是将十进制数的 ASCII 码转换成 BCD 码，即 AND GR0，CF。而将 BCD 码拼装到压缩字前，应该将压缩字中原来的 BCD 码左移 4 位，所以__(3)__空应为 SLL GR4，4。

第 10～17 行应该是内外循环的控制语句，也是本程序的难点。

从第 10、11 行可看出，第 15～17 行应该内循环尚未结束的处理，但第 16、17 行却是判断外循环结束的语句。因此可知，此程序把内外循环的控制语句合在一起了。__(5)__空应是 JNZ S1 或 JMI S1，注意，应该是跳转到内循环的起点 S1，而不应该是 S0。因为此处内循环尚未结束，不能给内循环计数器 GR3 重新赋值。

第 12～14 行应该是内循环结束的处理，即把拼装好的压缩字送回到目的地址（第 12、13 行）。但__(4)__空不应该是 JMP S0，因为这样将会越过判断外循环结束的语句第 16、17 行。所以__(4)__空应该是非跳转语句，而从__(5)__（JNZ S1）看，内循环计数器 GR3 应重新赋值，所以__(4)__空应该是 LEA GR3，4。

标号 S0 与 WL 是没用的。另外压缩字 GR4 也不需赋初值清 0，因为 4 次移位，每次移 4 位，原来不管是什么数都移出到外面了。

2、2001 年试题三[程序说明]

子程序 DEHZ 用来对 HZ 编码的字串做解码处理。

HZ 编码是海外华人创造的一种将含有高位为 1 的汉字双字节字串转换成易于在网络中传输的 ASCII 字符串的变换方式。编码过程中,被转换字符串中的原汉字子字符串各字节高位作清零处理,使之成为 ASCII 子字符串,并在其前后两端分别添加 ~{ 和 ~} 作为标记；而对于原 ASCII 子字符串,则将其中的 ~ 改写为 ~~,其余字符不变。

DEHZ 解码子程序则是 HZ 编码的复原过程。复原 ASCII 子字符串过程中遇有 ~~ 字符则改写为一个 ~,遇有 ~{ 则将其后直至 ~} 标记前的各字节高位置 1,复原为汉字子字符串,同时删除其前后标记。~的后续字符不属于以上情况均作为错误处理。

调用该子程序时, GR1 存放原始字符串首地址, GR2 存放还原后的目标字符串首地址。工作寄存器 GR3 用作处理汉字子字符串的识别标志,进入子程序时应初始化为处理 ASCII 子字符串。程序按照 CASL 语言的标准约定,字符串的每个字符只占用一个存储字的低八位。原始字符串和目标字符串均以 0 作为结束标志。

[程序]

```
START 1
DEHZ PUSH 0,GR3 2
PUSH 0,GR2 3
PUSH 0,GR1 4
LEA GR3,0 5
LOOP __(1)__ 6
CPA GR0,MARK0 7
JNZ GOON 8
LEA GR1,1,GR1 9
```

```
LD GR0,0,GR1 10
CPA GR0,MARK0 11
__(2)__ 12
CPA GR0,MARK1,GR3 13
JNZ ERROR 14
__(3)__ 15
LEA GR1,1,GR1 16
JMP LOOP 17
ERROR OUT ERS1R,ERLEN 18
JMP EXIT 19
GOON __(4)__ 20
ST GR0,0,GR2 21
LEA GR2,1,GR2 22
LEA GR1,1,GR1 23
CPA GR0,VO 24
__(5)__ 25
EXIT P0P GR1 26
P0P GR2 27
P0P GR3 28
RET 29
V1 DC 1 30
V0 DC O 31
DC #0080 32
MARK0 DC '~ ' 33
MARK1 DC '{ }' 34
ERSTR DC 'ERROR!' 35
ERLEN DC 6 36
END 37
```

[解]

寄存器作用：
GR1：源字符串地址指针，调用该子程序时,存放源字符串首地址。
GR2：目标字符串地址指针，调用该子程序时,存放目标字符串首地址。
GR3：用作处理汉字子字符串的识别标志。0 表示 ASCII 码（初始值），1 表示汉字。
GR0：工作寄存器，存放待处理的字符。（从第 7 行 "CPA GR0,MARK0" 看出）
分析：
1）从第 7 行 "CPA GR0,MARK0" 看出，GR0 中存放待处理的字符。所以第 6 行（__(1)__）必定是一条取数指令："LD GR0，0，GR1" ，即把源字符串地址指针所指的字符取到 GR0 中。

2）从第 7 行及第 8 行可知，第 9 行到第 17 行是处理碰到“~”的情况，即判断后一字符是否是“~”、“{”及“}”。若都不是，则出错。
3）从第 11 行可知，要判断是否连续两个“~”情况，而后面是继续比较，所以第 12 行（__(2)__）应该是一条“JZE”指令，而且是转移到 GOON ，即把“~”存放到目标字符串中。

4）第 13 行“CPA GR0,MARK1,GR3”应该是比较“{”及“}”：在中文状态比较“}”（结束），在西文状态比较“{”（开始）。所以这里用的是变址寻址，即由 GR3 的值是 0 或 1，决定是比较“{”还是“}”。

5）第 12 行（__(3)__）应该是改变汉字子字符串的识别标志 GR3 的指令，即原来是 0 的变成 1，原来是 1 的变成 0，异或指令可以达到此目的。将 GR3 与常数 1 相异或，因 Casl 没有没有立即数运算指令，只能和常数 V1 异或：EOR GR3，V1。

6）第 20 行（__(4)__）是 GR0 中存放的字符送到目标字符串前的处理工作：若是西文状态（ GR3 的值是 0 ），保持原样；若是中文状态（ GR3 的值是 1 ），字节最高位置 1,复原为汉字子字符串，将 GR0 的内容与十六进制数 0080 相或，即能达到目的。所以这条指令应是：OR GR0，V0，GR3 。

7）第 24 行将 GR0 与 0 比较，是判断字符串结束标志 0 。若非 0（未结束），继续处理。故第 25 行（__(5)__）应是一条条件转移指令：JNZ LOOP 。

3、2000 年试题四[程序说明]
（1）本子程序根据每位职工的基本工资（非负值）和他完成产品的超额数或不足数计算该职工的应发工资。

（2）主程序调用时，GR1 中给出子程序所需参数的起始地址，参数的存放次序如下表：
GR1 a1
b1
c1
a2
b2
c2
…
an
bn
cn
-1（结束标志）
其中 ai 为职工 i 的基本工资；bi 为职工 i 的完成产品的超额数或不足数；ci 为职工 i 的应发工资数（i = 1、2、…、n）。
bi 以原码形式存放（大于零为超额，小于零为不足），基本工资与计算所得的应发工资以补码形式存放。

（3）应发工资的计算规则为：
●恰好完成定额数（此时 bi = 0），应发工资即为基本工资。
●每超额 4 件，在基本工资基础上增加 10 元（不到 4 件，以 4 计算，例如超额数为 10 时，增加 30 元）。
●每不足 4 件，在基本工资基础上减 5 元（不到 4 件，以 4 计算，例如不足数为 5 时，减 10 元）。
[程序]
START 1
BEG
PUSH 0，GR1 2
PUSH 0，GR2 3
PUSH 0，GR3 4
L1 __(1)__ 5
LEA GR0，0，GR2 6
JMI FINISH 7
LD GR3，1，GR1 8
LEA GR2，0，GR3 9
AND GR2，C7FFF 10
JZE L3 11
SRL GR3，15 12
LEA GR2，-1，GR2 13
L2 __(2)__ 14

LEA GR2，-4，GR2 15
JPZ L2 16
L3 __(3)__ 17
__(4)__ 18
__(5)__ 19
FINISH POP GR3 20
POP GR2 21
POP GR1 22
RET 23
C7FFF DC #7FFF 24
BONUS DC 10 25
DC -5 26
END 27
[解]

寄存器作用：
GR1：地址指针
GR2：临时工作单元。先放 ai，后放 bi（绝对值）。
GR3：bi 符号
GR0：ci---应发工资
分析：
1）从第 6 行 "LEA GR0，0，GR2" 及第 7 行 "JMI FINISH" 可知 GR0 开始时应是 ai，GR2 也应是 ai，（从 LEA 指令功能分析）。所以第 5 行（1）应该是取数指令：
LD GR2，0，GR1

2）从第 8 行 "LD GR3，1，GR1" 及第 9 行 "LEA GR2，0，GR3" 可知 GR2 及 GR3 放的都是 bi（超额数或不足数），而从第 10 行 "AND GR2，C7FFF"（注意：C7FFF 是 16 进制常量的标号（第 24 行）），可知 GR2 存放其绝对值。而且在该值为 0 时直接结束该职工处理（第 11 行 "JZE L3"）。

3）从第 12 行 "SRL GR3，15" 可知 GR3 存放 bi 的符号（超额为 0，不足为 1）

4）从第 25、26 两行可知 BONUS 是每个超额或不足单位（4 件）的增加或扣除金额。从而得出最关键的第 14 行 （2） 应为 "ADD GR0，BONUS，GR3"。第 15、16 行指出这一加或减（GR3=1 时，源操作数为负）是一循环过程，一直到 GR2<0。为防止 bi 为 4 的整数倍时多加减一次，在第 13 行中先将 GR 减 1。
5）第 17、18、19 行（L3）依次是该职工的应发工资回送、修改地址指针（指向下一职工）和跳到处理程序开始（L1）：
ST GR0，2，GR1
LEA GR1，3，GR1
JMP L1

4、1999 年试题四〖程序 4.1〗[程序 4.1 说明]
本子程序是对 15 位二进位串，求它的奇校验位，生成 16 位二进制位串，使 16 位二进制位串有奇数个 1。
进入此子程序时，15 位二进制位串在 GR1 的第 1 位至第 15 位，并假定 GR1 的第 0 位是 0，求得的奇校验位装配在 GR1 的第 0 位上。
[程序 4.1]
START 1
BEG PUSH 0，GR2 2
PUSH 0，GR3 3

LEA GR3，1 4
__ (1) __ 5
L1 SLL GR2，1 6
__ (2) __ 7
LEA GR3，1，GR3 8
L2 JZE L3 9
JMP L1 10
L3 __ (3) __ 11
ST GR3，WORK 12
ADD GR1，WORK 13
POP GR3 14
POP GR2 15
RET 16
WORK DS 1 17
END 18
[分析]
1）从说明中已知，被转换的二进位串（一个字）放在 GR1 中。

2）第 6 行 "SLL GR2，1" 这条指令是处理奇偶校验用的，因此 GR2 也应该是工作单元，初始值为被处理数，故第 5 行（ （1） ）应该是 "LEA GR2，0，GR1"。

3）从第 4、5 行看，GR3 是一个计数器（统计值为 1 的位的个数），初始值为 1，即当 GR2 一个 1 也没有时，其值为 1（奇校验）。

4）第 6 行 "SLL GR2，1" 将被处理数左移一位，需要判断最高位是否为 1，若是，计数器加 1，否则跳过这条指令。因最高位也是符号位，所以可用 "JPZ L2"（ （2） ）。

5）第 9 行（L2）"JZE L3"是移位结束条件，即移到结果为 0 时结束。

6）第 11、12、13 行，是在计数器值为奇数（即实际 1 的个数为偶数）时把被处理字 GR1 最高位变成 1。而计数器 GR3 为奇数即其最低位为 1，因此需把 GR3 的最低位变成最高位，所以第 11 行（ （3） ）应该是：SLL GR3，15

5、1999 年试题四〖程序 4.2〗[程序 4.2 说明]
子程序 SUM 是将存贮字 A 起的 n(n>0) 个字求和，并将结果存于存贮字 B 中。
调用该子程序时，主程序在 GR1 中给出存放子程序所需参数的起始地址。参数的存放次序如下图：
（GR1）+0 A
+1 n
+2 B
[程序 4.2]
START
SUM LD GR2，0，GR1
LD GR3，1，GR1
LEA GR0，0
L5 ADD GR0，0，GR2
LEA GR2，1，GR2
_ (4) _
JNZ L5
L3 __(5)
ST GR0，0，GR3

RET
END
[分析]
1）GR1 为参数表起始地址
2）GR2 为数组地址指针，起始值为 A
3）GR3 为计数器，初始值为数组长度 n。
4）GR0 为累加和工作单元
5）（ （4） ）应该是计数器减 1：LEA GR3，-1，GR3
6）（ （5） ）应把结果单元地址 B 赋给 GR3：LD GR3，2，GR1

5、1998 年试题四[程序说明]
本程序是统计字符串中数字字符"0"至"9"的出现次数。
字符串中的每个字符是用 ASCII 码存贮。一个存贮单元存放两个字符,每个字符占 8 位二进位。
程序中，被统计的字符串从左至右存放在 STR 开始的连续单元中，并假定其长度不超过 200，字符串以'·'符作为结束。NCH 开始的 10 个单元存放统计结果。

```
START MIN 1
MIN LEA GR2，9 2
LEA GR0，0 3
L1 _ (1) _ 4
LEA GR2，－1，GR2 5
JPZ L1 6
LEA GR4，0 7
LEA GR1，0 8
L2 LD GR2，STR，GR1 9
EOR GR4，C1 10
JNZ RL 11
_ (2) _ 12
RL SRL GR2，8 13
LEA GR3，0，GR2 14
SUB GR3，C9 15
JM1 L3 16
JNZ L4 17
L3 LEA GR3，0，GR2 18
SUB GR3，CO 19
JM1 L5 20
LEA GR2，1 21
_(3)_ 22
_(4)_ 23
L4 LEA GR4，0，GR4 24
JNZ L2 25
_(5)_ 26
JMP L2 27
L5 SUB GR2，C 28
JNZ L4 29
EXIT 30
C1 DC 1 31
C DC '·' 32
C0 DC '0' 33
C9 DC '9' 34
STR DS 200 35
```

NCH DS 10 36
END 37
[解]
1）第 2～8 行（L2 以前）是初始化程序，其中第 2～6 行是把计数器存放单元 NCH 开始的十个单元清零。地址指针是 GR2（递减），故（1）为：ST GR0，NCH，GR2

2）从第 8、9 行看出 GR1 是地址指针（相对于 STR）。GR2 是工作单元（要处理的字符）

3）因一个字放两个字符，故 GR4 用作高低字节标志。起始值为 0，先处理高字节，第 10 行指令"EOR GR4，C1"一方面判断是否第一次（结果非 0），并将 GR4 置 1。
第一次处理高字节，用逻辑右移指令将高 8 位内容移到低 8 位（高 8 位置 0）。
第二次处理低字节，用先逻辑左移再逻辑右移指令将高 8 位内容置 0，故（2）为：
SLL GR2，8

4）在处理过程又用 GR3 作临时工作单元，即把 GR2 内容送给 GR3 再处理。处理时先判是否>"9"（不计数）。然后减以 "0" ，使 GR3 变成 0～9。
5）计数处理是在第 21、22、23 三行中完成。使 NCH 开始的 10 个单元中与 GR3 对应的那个单元加 1。因加法指令的目的操作数只能是寄存器，所以先给 GR2 送 1（第 21 行），再将 NCH 对应单元内容加到 GR2 中，再将 GR2 内容送回 NCH 对应单元（采用 GR3 变址寻址）。故 （3） 及 （4） 为："ADD GR2，NCH，GR3"及"ST GR2，NCH，GR3"。

6）在一个字的第二次处理后（用第 24、25 行判断），要修改字符串的地址指针 GR1（加 1）。故 （5） 为："LEAGR1，1，GR1"。

6、1997 年试题四[程序说明]
本子程序将一个非负二进整数翻译成五位十进整数字符。
进入子程序时，在 GR0 中给出被翻译的非负二进整数，在 GR2 中给出存放五位十进整数数字字符的起始地址。
十进制数字字符用 ASCII 码表示。当结果小于五位时，左边用空白符替换；当二进整数为零时，在（GR2）+4 中存放 0 的 ASCII 码。
数字字符 0 至 9 的 ASCII 码是 48 至 57，空白符的 ASCII 码是 32。
[程序]
START 1
LEA GR1，0 2
LEA GR3，32 3
L1 ____(1)____ 4
JPZ L2 5
ST GR3，0，GR2 6
LEA GR2，1，GR2 7
LEA GR1，1，GR1 8
LEA GR4，-4，GR1 9
JNZ L1 10
L2 ___(2)___ 11
L3 ___(3)___ 12
JMI L4 13
SUB GR0，SNO，GR1 14
LEA GR3，1，GR3 15
___(4)___ 16
L4 ST GR3，0，GR2 17
LEA GR2，1，GR2 18
LEA GR1，1，GR1 19

___(5)___ 20
JNZ L2 21
RET 22
SNO DC 10000 23
DC 1000 24
DC 100 25
DC 10 26
DC 1 27
END 28
[解]
这是一个典型的二化十汇编语言题例，其算法是将被转换的二进制数依次被 10i（i 为 4、3、

2、1、0）除，所得的商即为该十进制数位的值，其余数再被下一个 10i 除。一般用减法代替除法，即一边减 10i，一边计数器加 1，直到不够减再进行下一位 10i-1。
1）寄存器分配：GR0：被转换数；GR2：存放五位十进整数数字字符的起始地址。
GR1：数位计数器（兼作 SNO 内存数组的下标）
GR3：在初始化时放空格的 ASCII 码（48），在转换时作某一位的数码计数器（初始值为 0 的 ASCII 码 48）

2）SNO 内存变量依次存放 104、103、102、101、100 。

3）第 2～9 行为初始化程序，在 GR0<10i 时，对应的十进整数数字字符单元放空格（当结果小于五位时，左边用空白符替换），此过程一直进行到 GR0≥10i 或 GR1= 4（个位）。因此____(1)____应为 "CPL GR0，SNO，GR1"。

4）L2 开始进行除法（减法）。GR3 作某一位的数码计数器。从 L4 可看出，该计数值直接放到结果单元 [GR2]，而按题意所放的是 ASCII 码，所以其初始值应为 0 的 ASCII 码 48。因此___(2)___为:
LEA GR3，48 5）根据算法，GR0≥10i 才做减法,故____(3)____还是 "CPL GR0，SNO，GR1"。

6) ___(4)___是 "JMP L3"，即继续做这一位的减法，直至 GR0<10i。

7）L4 后 3 行是某一位结束处理：结果送到地址指针 GR2 所指的存放单元；地址指针 GR2 加 1；SNO 内存数组的下标 GR1 加 1。

8）___(5)___应该是判断除法是否做到个位结束。即下标 GR1=5，因此这一句为：
LEA GR3，-5，GR1

7、1996 年试题四[程序说明]
子程序 OFFSET 用二分法，查找无符号整数 M 在一个长度为 N 的有序（升序）无符号整数列表 NTABLE 中的位置。程序中标号为 LOW 和 UP 的两个存储字分别用作存放查找空间的上下限。
进入子程序时，在 GR1 中中给出存放子程序所需参数的起始地址。参数的存放次序如下图：
(GR1)+0 M
1 N
2 NTABLE 的首址
从子程序返回时，GR0 中存放查找结果，即 M 在此有序表中的位置序数，如表中找不到 M，则 GR0 中返回 0，其它寄存器的内容保持不变。
[程序]
START 1
OFFSET PUSH 0，GR2 2

PUSH 0，GR3 3
LD GR0，0，GR1 4
LEA GR2，0 5
ST GR2，LOW 6
___(1)___ 7
___(2)___ 8
ST GR2，UP 9
LOOP ADD GR2，LOW 10
SRL GR2，1 11
LEA GR3，0，GR2 12
___(3)___ 13
___(4)___ 14
JZE FOUND 15
JPZ INCLOW 16
LEA GR2，-1，GR2 ； M<NTABLE（K） 17
ST GR2，UP 18
JMP CMPLU 19
INCLOW LEA GR2，1，GR2 ； M> NTABLE(K) 20
ST GR2，LOW ； K+1→LOW 21
___(5)___ 22
CMPLU CPL GR2，LOW 23
___(6)___ 24
___(7)___ 25
FOUND LEA GR0，1，GR2 26
POP GR3 27
POP GR2 28
RET 29
LOW DS 1 30
UP DS 1 31
END 32
[解]
二分法查找的基本思想是对任意一段查找空间 [LOW，UP]（有序）中的的表元，试探位置 K=(LOW+UP)/2 上的成分 NTABLE(K) 与 M 进行比较，其可能结果有三种：
1）NTABLE（K）= M，找到，结束查找。
2）NTABLE（K）< M，下一查找空间为[K+1，UP]。
3）NTABLE（K）> M，下一查找空间为[LOW，K-1]。
初始查找空间为 LOW=0，UP=N-1。
程序中空格___(1)___和___(2)___前面的两条指令是将查找空间的上限 LOW 中 0，二在它之后的指令是将 GR2 中的值存于查找空间的下限 UP 中。因此这两个空格是把下限初值 N-1 送给 GR2。由于进入子程序时，N 存放在（GR1）+1 中，所以这两条指令为：
LD GR2，1，GR1
LEA GR2，-1，GR2
从标号 LOOP 开始的循环是求试探位置 K，根据 NTABLE(K) 和 M 比较结果，分别处理三种不同的情况，直至查到或查找空间为 0 。
考察空格___(3)___和___(4)___前面的指令，可得 K 在 GR2 和 GR3 中，在执行___(3)___和___(4)___两条指令后，有三种转向，因此这两条指令是将 GR0 中的 M 与 NTABLE(K)比较。而从程序说明中以知，NTABLE(0) 地址在 GR1+2。故 NTABLE(K) 的地址应为 GR2 或 GR3 与（GR1+2）相加（绝对地址）。但 GR2 在后面要作相对地址 K 用，所以只能是 GR3 与（GR1+2）相加。所以空格___(3)___和___(4)___为：
ADD GR3，2，GR1
CPL GR0，0，GR3

执行上述两条指令后，若不相等则要调整查找空间，在继续查找前，先应判断查找空间是否为 0，在程序中是用号为 CMPLU 的指令实现，显然 GR2 内应是查找空间的下限 UP。故___(5)___的答案为:
LD GR2，UP
当查找空间不为 0 时（UP>LOW），应继续查找，所以___(6)___的解答为:
JPZ LOOP
子程序返回时,GR0 中存放查找结果，在表中找到 M 时，GR0 中存放 M 在表中的位置序数，在程序中用 "FOUND LEA GR0，1，GR2" 实现（这里 GR2 中是试探位置，与位置序数差 1 ）。若表中找不到 M，GR0 中要放 0，所以___(7)___处应填 "LD GR2，-1"。

C++指针使用方法解惑

C++指针使用方法解惑
“void ClearList(LNode * & HL)”
仔细看一下这种声明方式，确实有点让人迷惑。
下面以
void func1（ MYCLASS *&pBuildingElement ）;
为例来说明这个问题。在某种意义上，"*"和"&"是意思相对的两个东西，把它们放在一起有什么意义呢？。为了理解指针的这种做法，我们先复习一下 C/C++编程中无所不在的指针概念。我们都知道 MYCLASS*的意思：指向某个对象的指针，此对象的类型为 MYCLASS。 Void func1（MYCLASS *pMyClass）;
// 例如： MYCLASS* p = new MYCLASS;
func1（p）;
上面这段代码的这种处理方法想必谁都用过，创建一个 MYCLASS 对象，然后将它传入 func1 函数。现在假设此函数要修改 pMyClass： void func1（MYCLASS *pMyClass）
{
DoSomething（pMyClass）;
pMyClass = // 其它对象的指针
}
第二条语句在函数过程中只修改了 pMyClass 的值。并没有修改调用者的变量 p 的值。如果 p 指向某个位于地址 0x008a00 的对象，当 func1 返回时，它仍然指向这个特定的对象。（除非 func1 有 bug 将堆弄乱了，完全有这种可能。）
现在假设你想要在 func1 中修改 p 的值。这是你的权利。调用者传入一个指针，然后函数给这个指针赋值。以往一般都是传双指针，即指针的指针，例如，CMyClass**。

MYCLASS* p = NULL;
func1（&p）;
void func1（MYCLASS** pMyClass）;
{
*pMyClass = new MYCLASS;
……
}
调用 func1 之后，p 指向新的对象。在 COM 编程中，你到处都会碰到这样的用法--例如在查询对象接口的 QueryInterface 函数中：
interface ISomeInterface {
HRESULT QueryInterface（IID &iid, void** ppvObj）;
……
};
LPSOMEINTERFACE p=NULL;
pOb->QueryInterface（IID_SOMEINTERFACE, &p）;

此处，p 是 SOMEINTERFACE 类型的指针，所以&p 便是指针的指针，在 QueryInterface 返回的时候，如果调用成功，则变量 p 包含一个指向新的接口的指针。 [
Page]

如果你理解指针的指针，那么你肯定就理解指针引用，因为它们完全是一回事。如果你象下面这样声明函数：

```
void func1（MYCLASS *&pMyClass）;
{
pMyClass = new MYCLASS;
……
}
```

其实，它和前面所讲得指针的指针例子是一码事，只是语法有所不同。传递的时候不用传 p 的地址&p，而是直接传 p 本身：

```
MYCLASS* p = NULL;
 func1（p）;
```

在调用之后，p 指向一个新的对象。一般来讲，引用的原理或多或少就象一个指针，从语法上看它就是一个普通变量。所以只要你碰到*&，就应该想到**。也就是说这个函数修改或可能修改调用者的指针，而调用者象普通变量一样传递这个指针，不使用地址操作符&。

至于说什么场合要使用这种方法，我会说，极少。MFC 在其集合类中用到了它--例如，CObList，它是一个 Cobjects 指针列表。

```
Class CObList : public Cobject {
……
// 获取/修改指定位置的元素
Cobject*& GetAt（POSITION position）;
Cobject* GetAt（POSITION position） const;
};
```

这里有两个 GetAt 函数，功能都是获取给定位置的元素。区别何在呢？

区别在于一个让你修改列表中的对象，另一个则不行。所以如果你写成下面这样：

```
Cobject* pObj = mylist.GetAt（pos）;
```

则 pObj 是列表中某个对象的指针，如果接着改变 pObj 的值： pObj = pSomeOtherObj；这并改变不了在位置 pos 处的对象地址，而仅仅是改变了变量 pObj。但是，如果你写成下面这样： Cobject*& rpObj = mylist.GetAt（pos）;

现在，rpObj 是引用一个列表中的对象的指针，所以当改变 rpObj 时，也会改变列表中位置 pos 处的对象地址--换句话说，替代了这个对象。这就是为什么 CObList 会有两个 GetAt 函数的缘故。一个可以修改指针的值，另一个则不能。注意我在此说的是指针，不是对象本身。这两个函数都可以修改对象，但只有*&版本可以替代对象。

在 C/C++中引用是很重要的，同时也是高效的处理手段。所以要想成为 C/C++高手，对引用的概念没有透彻的理解和熟练的应用是不行的。

C++建立数据库 VCL

随着数据库的广泛应用，数据库编程已经成为程序设计中发展迅猛的一支。C++ Builder 在数据库开发方面具有的强大功能是无可比拟的，你甚至可以不写一行程序就生成漂亮的数据库程序。

下面对 C++Builder 中的几个数据库 VCL 的使用技巧做一下介绍：

一、DBGrid 控件

1.设置 DBGrid 的字段显示宽度属性

为了在 DBGrid 中建立较小的列，你必须建立一个显示标题，它等于或小于字段值。例如，你希望建立一个只有三个字符宽的列，你的列标题显示必须只有三个字符或更少。

2.改变 DBGrid 的显示字段及日期显示格式

(1)双击 DBGrid 对应的 Table1,进入字段编辑器。

(2)点右键出现选单选“Add Fields… ,出现添加字段对话框，选择要添加的字段(该字段将在运行时由 DBGrid 显示)然后点 OK 按钮。

(3)假设添加了“日期”字段，点该字段，在属性表中的：DisplayLabel 中填入你希望 DBGrid 显示的字段名。如果原来字段名是英文的，这里用中文名后 DBGrid 将显示中文名。在 DisplayFormat 中填入：yyyy-mm-dd，以后日期将按 1999-05-28 格式显示。

二、Tquery 控件

Tquery 控件是数据库编程中非常重要的一个控件，它负责通过 BDE 与数据库建立联系，通过 SQL 语句方便的建立查询。Query 必须建立相应的 SQL 才能生效。

Tquery 的参数设置如下：

(1)在 SQL 属性中：Select * from 表名 where 字段名=变量名

跟在“ 后面的是变量。这样写后，在参数属性中就可以修改该变量的数据类型等。

(2)对变量的赋值：

Query1-〉 Active=false;

Query1-〉 Params-〉 Items[0]-〉 AsString=Edit1-〉 Text;

Query1-〉 Active=true;查找符合变量的记录

(3)用 DBGrid 显示结果

DBGrid 的 DataSource 与 DataSource1 连接，而 DataSource1 的 DataSet 与 Tquery1 连接。

三、应用示例

通过 Query 控件嵌入 SQL 语句建立的查询比 Table 更简单、更高效。

用一个简单的代码来说明如何建立查询程序：

例如，要建立一个检索表 1 中书名为 book1 的程序则在表单上放置 DBGrid,DataSource,Query 三个控件加入以下代码：

```
DBGrid1-〉 DataSource=DataSource1;
DataSource1-〉 DataSet=Tqery1;
Query1-〉 Close();
Query1-〉 SQL-〉 Clear();
Query1-〉 SQL-〉 Add(〞Select * From 表 Where (书名=′ book1′ 〞);
Query1-〉 ExecSQL();
Query-〉 Active=true;
```

你就可以在生成的表格中看到所有名称为 book1 的记录。

ASP.NET 多频道网站架构实现方法(1)

主体架构

各频道分别位于不同的 Web Project(具有独立的二级域名)，并将所有的业务逻辑以及数据访问功能封装成 Class Library，所有频道共用这个 Class Library。

下面详细介绍实现方法。

假设网站有三个频道，新闻、论坛以及博客，对应的二级域名为"news"、"forum"、"blog"。除此之外，还需要另外定义两个域名，分别用于网站首页以及用户注册、登陆功能(基于 Passport 机制，本文后面将作详细介绍)，对应域名为"homepage"、"passport"。

1.配置各频道 URL

a.配置 hosts 文件

用文本编辑器打开 hosts 文件(位于 c:\windows 或 winnt\system32\drivers\etc\)，该文件中存放初始的域名解析信息。当我们在浏览器中请求某个 URL 时，系统首先在 hosts 文件中

查找相应域名，如果找到则跳转至指定 IP,如果没找到，则进一步提交 DNS 进行域名解析。

配置很简单，格式形如"[IP][空格][域名]"，每条数据对应一行。下面为配置内容：

192.168.1.2 www.mysite.com
192.168.1.2 passport.mysite.com
192.168.1.3 news.mysite.com
192.168.1.5 forum.mysite.com
192.168.1.9 blog.mysite.com

你可能已经注意到了，各频道对应于不同的 IP，这正是该架构的开发灵活性所在。各频道(Web Project)可以创建于不同的开发者电脑。通过将配置内容同步到各台电脑，可以方便的在各频道间进行页面浏览，就像这些频道位于你自己的电脑一样!采用这种方式可以极大降低开发耦合性，每个频道都是一个独立的模块，一个频道中的 Bug 不会影响到另一个频道。

b.配置 Web.Config

考虑到各频道二级域名有可能进行调整，将相应配置信息存放于 Web.Config 文件是一个好办法。同样的，该配置信息必须同步到各 Web Project。下面为配置内容：

```
＜add key="SiteDomainName" value="mysite.com"/＞
＜add key="HomepageSiteURL" value="http://www.mysite.com/homepage/"/＞
＜add key="PassportSiteURL" value="http://passport.mysite.com/passport/"/＞
＜add key="NewsSiteURL" value="http://news.mysite.com/news/"/＞
＜add key="ForumSiteURL" value="http://forum.mysite.com/forum/"/＞
＜add key="BlogSiteURL" value="http://blog.mysite.com/blog/"/＞
＜add key="LocalSiteURL" value="/blog/"/＞
```

各配置项说明如下

SiteDomainName:站点域名，形如"mysite.com"、"mysite.com.cn"、"mysite.net"等。该配置项的使用方法将在后文介绍。

LocalSiteURL:当前频道根路径，也就是 Web Project 所在网站或虚拟目录的路径，以"/"开头。该配置项主要用于频道内部的引用，比如图片引用、页面链接等。

其余配置项:用于频道间的引用，比如频道导航、功能调用等。

2.创建 Model 部件

在 MVC 模式组成中，Model 部件包括所有的业务逻辑*作，其中也包含数据访问*作。

本方案将 Model 部件拆分成对象实体、对象*作以及数据访问三部分，封装成三个 Class Library。

由于 Class Library 设计本身就是一个很大的话题，本文就不再祥述了，有兴趣的话可以参考一些相关资料。

ASP.NET 多频道网站架构实现方法(2)

经验分享：

上述的 Model 部件拆分方式适用于业务功能比较复杂的大型项目，要求团队内部有着明确、细化的分工合作。但如???? ?? ?o扯? 果面对的是中小型项目，该方式很有可能成为开发效率的瓶颈。这主要是由项目特点决定的，中小型项目业务功能相比大型项目没有那么复杂，开发人员数量也比较有限，往往一个人要负责整个模块的开发。在这种情况下，架构层次过于繁多，每次修改一个层时，其他相关层也得跟着同步修改，这样反而影响了开发效率。

3.实现 Passport 机制

很多网站都采用 Session 来存放个人信息，比如登录信息，并以次作为用户登录与否的判断依据。但 Session 有一个缺陷，就是无法在多个 Web 应用中共享，一个 Web 应用生成

的 Session 只能由他自己使用。哪种方法可以在多个 Web 应用中实现数据共享呢?答案是 Cookie。Cookie 将信息存放于客户端,并在需要时发送回服务器端。

Passport，即通行证，是目前普遍采用的一种用户身份认证机制，简单来说就是一次登录，全站通行。这也正是我们的要求。

这里讨论的通行证机制基于 Cookie,实现也比较方便。其中的关键点是 Cookie 的 Domain 属性设置,Domain 属性表示 Cookie 信息回发的目标域,也就是接收 Cookie 的域,接收 Cookie 的域必须与发送 Cookie 的域一致，否则无效。比如：发送域为"blog.mysite.com"，则接收域可以设为"blog.mysite.com"或"mysite.com",而"news.mysite.com"和"blog.yoursite.com"为无效接收域。要想让所有频道都能接收到 Cookie，必须将 Domain 属性设置为不带二级域名前缀的形式，如"mysite.com"、"mysite.com.cn"、"mysite.net"等。

登录成功后向客户端发送相应 Cookie，其中可以包括一些全局信息，比如用户编号、用户名等。用户退出时删除相应 Cookie，特别要注意的是，删除 Cookie 时也要设置正确的 Domain 属性。

关于该 Passport 机制，还有两个问题值得讨论：

a.Cookie 的过期时间
有两种方案可以采用，一种是默认方式，即不设置 Cookie 的 Expires 属性,采用这种方案时，Cookie 存放于内存中，在浏览器关闭前 Cookie 将一直存在，也就是一直处于登录状态。这种方式主要用于对信息安全要求不是很高的网站，比如娱乐休闲类网站;另一种是指定明确的过期时间,一般情况下会将用户最后一次访问网站的时间加上一个超时时间段作为过期时间,有点类似于 asp 中的 session 超时机制,这种方式主要用于对安全性要求比较高的网站，比如网上银行、电子邮箱等。

b.Cookie 的信息安全
由于 Cookie 是以明文方式传递数据，不可避免的存在安全隐患，因此对重要数据的加密是非常有必要的。加密可以采用可逆算法，比如 DES。

4.创建 Web Project
前文已提过，Web Project 的创建比较灵活，既可以创建于不同的开发者电脑，也可以创建于同一台电脑。这主要取决于开发团队规模。

5.部署
分别部署各频道，设置二级域名，将 Web.Config 中的相关配置改为生产环境的实际数据。
其中比较繁复的工作就是各频道中相同部分的部署，比如说网站头部(Logo、导航栏等)，网站底部(版权声明、联系方式等)，图片，CSS，JavaScript 等。当然也可以把这些公用资源单独部署于一个频道中，以供其他频道调用，但这样做就破坏了各频道松耦合的特性，如果用于存放公用资源的频道出了问题，那其余频道也将无法正常使用。

结束
本文讨论了 asp.net 中多频道网站架构的一种实现方法，由于涉及到的内容较多，无法一一展开，但对其中的重点部分还是多加了点笔墨，希望对你有用

4G 内存下 LinuxMtrr 表不正确的解决方法

这个会导致 nvidia 的驱动不能加速 2d，解决方案一般就是重写 mtrr 表。

```
echo “disable=2〞 >| /proc/mtrr
echo “disable=1〞 >| /proc/mtrr
echo “disable=3〞 >| /proc/mtrr
echo “disable=4〞 >| /proc/mtrr
echo “disable=0〞 >| /proc/mtrr
echo “base=0×00000000 size=0×80000000 type=write-back” >|
/proc/mtrr
echo “base=0×80000000 size=0×40000000 type=write-back” >|
/proc/mtrr
echo “base=0xC0000000 size=0×10000000 type=write-back” >|
/proc/mtrr
echo “base=0×100000000 size=0×20000000 type=write-back” >|
/proc/mtrr
echo “base=0×120000000 size=0×10000000 type=write-back” >|
/proc/mtrr
```