第 3 章　命名方案的设计

3.0　概　　述

在前一章中，我们阐述了命名方案的一种抽象模型。当设计一个实际的命名方案时，许多工程性的考虑——约束、额外规定或迫切需求、环境压力——影响着设计。用户与计算机系统交互的主要方式之一是通过名字，系统命名方案的质量可能显著影响用户体验的质量。同样，由于名字是连接各个模块的黏合剂，命名方案的特性能够显著影响模块性对系统的作用。

本章探讨命名方案设计相关的工程性考虑因素。主要篇幅介绍各种影响模块性和可用性的命名考虑因素。对万维网中统一资源定位器（URL）的案例研究，阐明了命名模型以及在命名方案设计中出现的一些问题。最后，3.3 节探讨了实际命名方案的一些病态问题。

3.1　命名方案设计中的考虑因素

我们从讨论命名和模块性之间的相互作用开始。

3.1.1　模块化共享

用名字连接各个模块提供了极大的灵活性，但却引入了一种冒险：设计者有时需要处理预先存在的名字，可能是由设计者无法控制的其他人选择的。每当独立设计模块时，这一冒险都可能出现。如果为了使用一个模块，设计者就必须了解并避免模块内部为其组件所用的名字，那么我们就没能实现模块性的主要目标之一，即所谓模块化共享。模块化共享意味着，人们可以通过其名字来使用一个共享模块，而无须知道该模块所使用的其他模块的名字。

模块化共享的缺乏以名字冲突的形式显现：由于某种原因，两个或两个以上不同的值在相同上下文中竞争对同一名字的绑定。当集成两个（或更多个）独立构建的程序组、文档组、文件系统、数据库或者实际上任何组件的集合，这些组件使用相同的命名方案来实现面向集成的内部互连，名字冲突就可能会出现。名字冲突可能是个严重问题，因为修复它需要改变某些冲突名字的使用。实现这样的改变可能是棘手的或者是困难的，因为子系统的原作者不一定可以找来帮助查找、理解并改变冲突名字的使用。

实现模块化共享的明显方式是为每个子系统都提供自己的命名上下文，然后再设计出在多个上下文之间交叉引用的某种方法。使得交叉引用能够正确工作是个挑战。

例如，考虑图 3.1 所示的两组程序——文字处理器和拼写检查器——每组都包含通过名字连接到一起的多个模块，每组都有一个组件称为 INITIALIZE。过程 WORD_

PROCESSOR 的设计者希望使用 SPELL_CHECK 作为一个组件。如果设计者尝试通过在一个命名上下文中简单地绑定所有它们的名字来集成两组程序，如同图中所示（其中箭头显示了每个名字的绑定），那么将有两个模块竞争对名字 INITIALIZE 的绑定。我们遇到一个名字冲突。

于是，设计者转而尝试为每组程序创建单独的上下文，如图 3.2。这一步本身并没有完全解决问题，因为程序解释器现在需要一些规则，以便为名字的每次使用确定所用的上下文。例如，假设正在运行 WORD_PROCESSOR，并遇到名字 INITIALIZE。它如何知道应该在 WORD_PROCESSOR 的上下文而不是 SPELL_CHECK 的上下文中解析该名字？

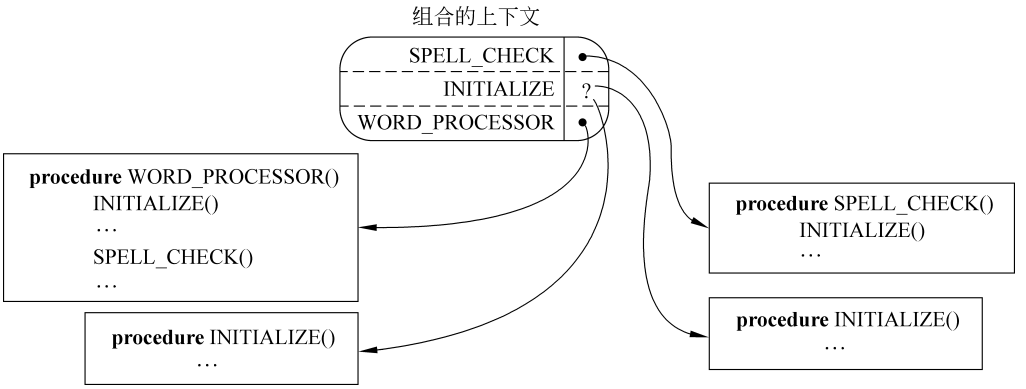

图 3.1 两套独立编写的程序通过只是合并其上下文而实现的过于简单的集成。过程 WORD_PROCESSOR 调用 SPELL_CHECK，但是 SPELL_CHECK 有一个组件具有与 WORD_PROCESSOR 的一个组件相同的名字。没有单独一组绑定可以正确工作

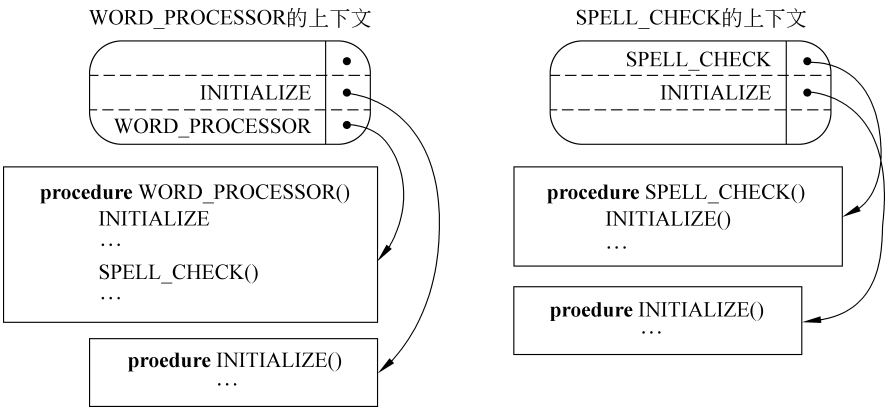

图 3.2 相同两组程序但是使用分离上下文的集成。为 SPELL_CHECK 拥有一个单独上下文，消除了这个名字冲突，但是程序解释器现在需要一些基础工作，选择一个上下文来覆盖另一个上下文

继续第 2 章中的命名模型以及电子邮件系统的例子，这个问题的一个直接解决办法是在 WORD_PROCESSOR 上下文中为 SPELL_CHECK 增加一个绑定并把明确的上下文引用连接到每个模块，如图 3.3。这一增加需要修补模块的表达，这种做法可能不太方便，如果一些模块属于其他人则甚至是不允许的。

图 3.4 表明另一种可能：增强程序解释器以记录最初发现每组程序的上下文。程序解

释器将使用对应上下文来解析在该程序中发现的所有名字。接下来，为了允许文字处理器通过名字调用拼写检查器，在 WORD_PROCESSOR 上下文中放置对 SPELL_CHECK 的一个绑定，如图中标号为 1 的实线所示（设想上下文现在是文件系统目录）。

这一额外的绑定产生了微妙的问题，可能会带来后期的惊讶。由于程序解释器在文字处理器上下文中找到 SPELL_CHECK，上下文选择规则（错误地）告诉它为 SPELL_CHECK 内部发现的名字使用该上下文，于是 SPELL_CHECK 将调用 INITIALIZE 的错误版本。一个解决办法是在文字处理器上下文中放置一个间接名字（图 3.2 中标号为 2 的虚线箭头），绑定到 SPELL_CHECK 自己的上下文中的 SPELL_CHECK 的名字。然后，解释器（假设它记录了实际发现每个程序的上下文）将正确解析在两组程序中发现的名字。

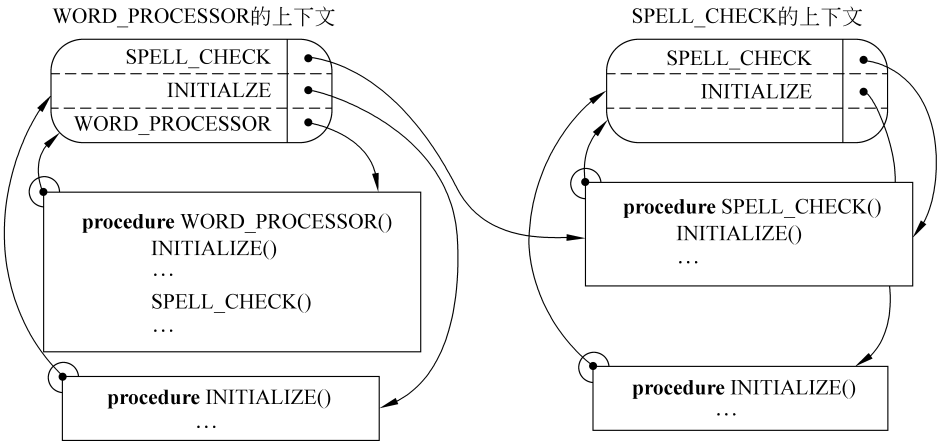

图 3.3　带有明确的上下文引用的模块化共享。添加到每个程序模块的小圆圈是上下文引用，告诉名字解释器哪个上下文用于该模块中发现的名字

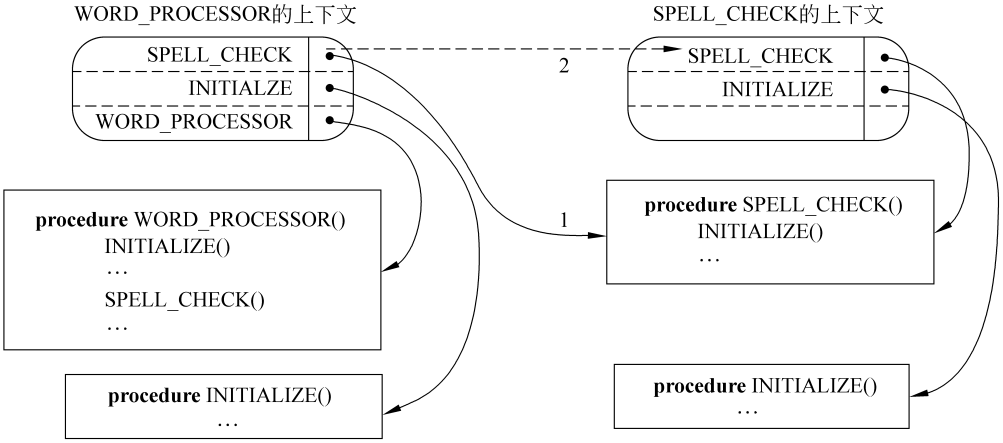

图 3.4　借助于独立上下文的集成。为 SPELL_CHECK 拥有一个单独上下文，消除了这个名字冲突，但是程序解释器仍旧需要一些基础工作，来选择一个上下文来重载另一个上下文。添加标号为 1 的带箭头的实线不能工作，但是标号为 2 的带箭头的虚线，即间接名字，工作得很好

记录上下文和使用间接引用（可能通过使用文件系统目录作为上下文）是司空见惯的，但有点特别。另一个把上下文引用连接到对象而无需修改对象表示的更加优雅的方式是，

把对象的名字不直接关联到对象本身，而是关联到一个包含原始对象加上其上下文引用的结构体。一些编程语言为过程定义实现了这样的结构体，称为“闭包”，把每个过程定义与在其范围内定义它的命名上下文连接。

使用静态范围和闭包的编程语言，为在大型应用程序不同部件的范围内模块化共享命名对象提供了一个更加系统化的方案，但对应机制很少在文件系统或者比如前面例子中的文字处理和拼写检查系统这样的合并应用中发现[1]。造成差异的一个原因是，一个程序通常包含对大量命名对象的多次引用，因此重要的是要组织好。另一方面，合并应用包括少数只具有几个交叉引用的大型组件，所以为模块化共享特殊设计的方案，似乎是足够的。

3.1.2 元数据与名字重载

对象的名字和应该与它关联的上下文引用是称为元数据的一类信息的两个例子，这类信息用于了解一个对象却不能在对象本身内部发现（或者即使它在里面也可能不容易找到）。图书馆书目记录是一组元数据，包括书名、作者、出版社、出版日期、购置日期、书的架上位置，都以一种标准格式存在。图书馆在处理元数据方面有丰富经验，但失败于系统化的组织元数据是在计算机系统中经常遇到的一个设计缺陷。

计算机系统内与对象关联的元数据的常见例子是：用户友好的名字，唯一的标识符，对象（可执行程序、文字处理文本、视频流等）的类型，创建、最后修改和上次备份的日期，备份副本的位置，其所有者名字，创建它的程序，以验证其完整性的加密质量校验和（称为见证，参见边栏 7.1（在线）），允许读取或更新对象的人员名单，以及对象的表示的物理位置。尽管不是普遍却是常见的一种元数据属性是关于一个对象的信息，它可能在没有改变对象本身的情况下被修改。

在文件系统中维护元数据的一个策略是，在记录文件物理位置并提供读取和更新元数据方法的同一文件系统结构中为元数据保留存储空间。这一策略很有吸引力，因为它允许不关心元数据的应用程序很容易地忽略它。这样，编译器可以读取一个输入文件而不必明确地识别并忽略文件主人的名字或者文件上次备份日期，然而自动备份应用程序可以使用元数据访问方法来检查这两个域。2.5.1 节中描述的 UNIX 文件系统，通过在 inode 中存储元数据来使用这一策略。

计算机文件系统几乎总是支持对专门描述文件的元数据建行管理为关于每个文件的专门元数据的管理提供支持，这些元数据比如它的物理位置、大小和访问权限。但它们为除文件名以外的用户提供的元数据很少有任何准备。因为这一局限性，人们经常发现文件名被与作为引用的名字使用有很小关系或毫无关系的那些元数据重载[2]。这一命名方案甚至可能把语法规则加到允许的名字上，以支持带有元数据的重载。名字重载的一个典型例子是

---

1 20 世纪 70 年代，IBM 进行了一次雄心勃勃的尝试，设计一种将所有这些概念连入硬件的命名架构，记载于 George Radin 和 Peter R. Schneider 所写的一篇技术报告中：An architecture for an extended machine with protected addressing，IBM 波基普西实验室技术报告 TR 00.2757，1976 年 5 月。尽管该体系结构本身从来没有推向市场，但一些思想后来出现在 IBM System/38 和 AS/400 计算机系统中。

2 使用重载这个词来描述携带元数据的名字，与使用同一个词来描述在一门编程语言中代表一些不同操作的符号，是尽管相似却又截然不同的。

以扩展名结尾的文件名，扩展名标识文件类型，如文本、文字处理文档、电子表格、二进制应用程序或电影。其他例子如图 3.5 所示。物理地址是名字重载的另一个例子，因为非常普遍，下一节探讨了它的特殊属性。没有任何重载的名字称为单纯名字。应用于单纯名字有意义的操作只有 COMPARE、RESOLVE、BIND 和 UNBIND。人们不能通过运用解析操作从中提取元数据。

名字	重载这一名字的一些事情
solutions.txt	solutions　文件内容; txt　文件格式
solutions.txt.backup 2	backup 2　这是第二个备份副本
businessplan 10–26–2007.doc	10–26–2007　文件创建时间
executive summary v4	v4　版本号
image079.large.jpg	079　文件在一个序列中的位置; large　图像大小
/disk–07/archives/Alice/	disk–07　保存文件的物理设备; Alice　用户标识
OSX.10.5.2.dmg	OSX　程序名字; 10.5.2　程序版本
IPCC_report_TR–4	IPCC　作者; TR–4　技术报告序列标识符
cse.pedantic.edu	cse =　系名; pedantic =　大学名; edu =　注册名
ax539&ttiejh!90rrwl	没有（明显的）重载

图 3.5　重载名字和单纯名字的一些例子

另一方面，重载名字能够以两种不同方式使用：

（1）作为一个标识符，使用 COMPARE、RESOLVE、BIND 和 UNBIND。

（2）作为提取重载元数据的源。

路径名特别容易受重载的影响。因为它们通过一系列上下文来描述路径，给人的诱惑是使用到对象的物理位置的路由信息来重载它们。

名字的重载可能是无害的，但也导致了模块化设计及抽象原则的破坏。 问题通常以脆弱名字的形式出现。例如，一个文件移动到新的物理位置，即使文件的标识和内容没有改变，有必要更改文件名字，此时名字脆弱性出现。例如，假设一个计算平方根的库程序，而恰巧存储在 disk05 上并被命名为/disk05/library/sqrt。如果 disk05 后来变得太满，该库不得不迁至 disk06，该程序的路径名改变成/disk06/library/sqrt，人们不得不追查和修改旧名称的每一次使用。名字脆弱性是万维网地址停止工作的原因之一。3.2 节中的案例研究更加详细地探讨这一问题。

这一观察的一般性版本是，重载制造了保持名字不变的目标和修改重载信息的需要之间的一种紧张。通常，使用一个名字的模块需要只要该模块存在，名字就保持不变。为此，使用重载必须谨慎并理解名字将被如何使用。

最后，在模块化系统中，重载的名字在到达真正知道如何解释该重载的模块之前，可能通过几个模块来传递。一个名字被认为是对一个模块不透明的，如果该名字没有该模块知道如何解释的重载。单纯名字可以被看做对除 RESOLVE 以外的所有模块是不透明的。

还有元数据重载的更加微妙的形式。如果是用户的头脑而不是计算机系统执行元数据提取，重载可能不那么明显。例如，在互联网主机名“CityClerk.Reston.VA.US”中，上下文标识符“Reston.VA.US”，也可以看做一个真实地方，美国名为弗吉尼亚州莱斯顿的城镇

的标识符。这个名字的每个部分被用来命名两个不同的现实世界的事物：名字“莱斯顿”同时标识一座城镇和一个“名字-值”对的表，该表作为可以从中查找市政部门名字的上下文。因为它有帮助记忆的价值，人们发现借助重载的重用是有帮助的，假设这一重用被相当精确和一致的执行（另一方面，如果有人把芝加哥的一个万维网服务命名为“SaltLakeCity.net”，看到这个名字的人可能会错误地认为它实际上是位于盐湖城中）。

3.1.3 地址：定位对象的名字

在计算机系统中，一个地址是物理位置或者映射到物理位置的虚拟位置的名字。计算机系统是由真实的物理对象构造的，因此地址的例子比比皆是：寄存器号码、物理和虚拟内存地址、处理器号码、磁盘扇区号、可移动媒体卷号、I/O 通道号码、通信链路标识符、网络接入点地址、显示器上的像素位置，名单似乎无穷无尽。

地址并非单纯名字。地址特征可以刻画为，它是以这样一种方式被重载，地址解析提供了一个指向某个虚拟或真实坐标系统中命名对象的位置的向导。如同其他重载名字，地址可以通过两种方法来使用，在这一实例中：

（1）作为一个带有普通命名操作的标识符。

（2）作为一个定位器。

因此，“列奥纳多·达·芬奇”是一个曾经绑定到一个自然人、现在绑定到对列奥纳多的记忆的标识符。此标识符可以被用来比较，以避免当两人都前往佛罗伦萨时与列奥纳多·迪·比萨相混淆[3]。今天，标识符有助于避免混淆了他们的著作。同时，“列奥纳多·达·芬奇”也是一个定位符；它表明，如果你要检查列奥纳多的出生记录，你应该查阅名为芬奇的城镇的档案。

因为对许多物理设备的访问是几何的，地址通常是以这样一种方式从整数的紧凑集合中选择出来的，地址邻接对应着物理邻接，像“加 1”或从另一个地址中减去一个地址的算术操作有一个有用的物理意义。

例如，寻道臂通过计算它所经过的磁道数来找到磁盘上的 1079 号磁道，磁盘臂调度器看磁道地址差别来决定执行寻道的最好顺序。再比如，内存芯片包含一组数据位，每个数据位都有一个唯一的整数地址。当读写特定地址的请求到达芯片时，该芯片解析该地址的单独数据位的路由给选择器，选择器指导来自或者发到存储的预期数据位的信息流。

有时对地址应用算术操作是不合适的，即使它们是从整数紧凑集中选择的。例如，电话号码（技术上称为“目录号码”）是被重载了以地区代码和交换代码组成的路由信息的整数，但是有着连续地址的两个地区代码物理上不是必然邻接的。同样，有着连续目录号码的两个电话号码物理上也不是必然邻接的（在过去数十年，在电话交换设备内部存在连续目录号码的物理邻接性，但这一邻接性是如此限制人，以至于它被通过引入间接层作为电话交换设备的一部分的方式放弃了）。

在地址中发现的重载的位置信息会导致名字脆弱性。当对象移动时，它的地址、进而

3 实际上，他们不可能都在同一时间参观佛罗伦萨。数学家列奥纳多·迪·比萨（也称为斐波纳契）生活在艺术家列奥纳多·达·芬奇之前的 3 个世纪。

它的名字发生改变。为此，系统设计人员通常遵循电话交换系统的例子：他们采用“借助间接性来分离模块”的设计原理来隐藏地址。间接层的添加提供了一种由一些外部可见但稳定的名字到一个对象移动到新位置时可以轻易改变的地址绑定。理想情况下，地址永远不必向直接操纵的对象的解释层之上暴露。因此，举例来说，一台有通信端口的个人电脑的用户，可以编写程序为该端口使用如 COM1 的名字，而不是如 4D7Chex 这样的十六进制地址，该地址在端口卡被替换时可能改成 4D7Ehex。

当一个名字因为正作为一个不是由间接层隐藏的地址使用而必须改变时，事情变得更加复杂，可能会开始出问题。至少有 4 个备选方案已经在命名方案中采用：

- 搜索并改变所有旧地址的使用。最好的情况，这个方案也是一个麻烦。在一个大的或地理上分散的系统，它可能是相当痛苦的。搜索通常忽略了名字的一些使用，而那些用户，在他们下一次试图使用名字时，要么收到一个对于仍然存在的对象的令人费解的未发现响应，或者更糟糕，发现旧地址现在指向不同的对象。出于这个原因，这个方案可以与下一个方案结合。
- 规定名字的使用者必须进行一次基于属性的对象搜索，如果他们收到一个没有找到的响应或者检测到该地址已经被绑定到不同对象上。如果搜索找到正确对象，新地址替换旧地址，至少对于该用户来说是这样。不同用户将不得不做另一次搜索。
- 如果命名方案提供同义词或者间接名字，增加绑定，以便使得旧地址和新地址都继续标识该对象。如果地址短缺而必须重复使用，这个方法则没有吸引力。
- 如果名字是绑定到一个活动实体，如接收邮件的邮局服务，在旧地址放置一个如邮件转发员的活动中间人。

这些备选方案可能都是没有吸引力的。更好的方法几乎总是设计师用来把地址隐藏到一个间接层之后。3.3.2 节提供了这个问题和采用间接性的解决方案的一个例子。习题 2.1 探索了电话系统中关于称为呼叫转移功能的一些有趣的间接性相关的命名问题。

人们可能会建议通过只使用单纯名字，即没有重载的名字的方式避免名字脆弱性问题。用这种方法的麻烦是，这样使得定位对象比较困难。当最底层名字没有携带重载的寻址元数据，解决一个物理对象的名字的唯一方法是通过搜索所有名字的一个枚举。如果上下文是小的并且本地的，这种技术可能是可以接受的。如果情况是全世界和广泛分布的，名称解析变得相当有问题。例如，考虑铁路车辆定位问题，给定只有一个唯一序列号印在它的一侧。如果由于某种原因，你知道车辆在一个特定的壁板上，搜索可能是简单的，但如果车辆可能在大陆上的任何地方，搜索是一项艰巨的设想。

3.1.4　生成唯一的名字

在一个唯一标识符的名字空间，某一协议需要确保所有名字真的都是独一无二的。通常的做法是，命名方案为新创建的对象生成名字，而不是依靠创建者提出一个唯一的名字。生成唯一的名字的一个简单方案是，发放连续的整数或足够精度的时间戳值。边栏 3.1 显示了一个例子。产生唯一的名字的另一个方案是，从一个足够大的名字空间中随机选择名字。这样做是为了使意外两次选择相同名字（称为碰撞的一种名字冲突形式）的概率微乎其微。这个方案的麻烦是，有限状态机创建真正的随机性是很难的，所以意外造成名字碰

撞的机会要远远高于人们依据名字空间大小所预测的机会。人们必须采用仔细的设计，例如，通过使用高品质的伪随机数生成器并用如系统启动时创建的时间戳这样一个唯一的输入作为它的种子。这样设计的一个例子是 Apollo DOMAIN 操作系统中使用的命名系统，为了向系统用户提供一个高度的透明性，它为所有对象提供了跨局域网的唯一标识符。更多细节，见进一步阅读推荐 3.2.1。

避免产生名字冲突还有一种方法，就是为具有二进制表示并且命名时已存在的对象，选择对象内容作为其唯一的名字。这种方法为具有相同内容的两个对象分配相同的名字。但是，在一些应用中这可能是一个特性，它提供了一种发现不想要的复制副本存在的方法。该名字很可能是相当长的，因此更实际的方法是使用称为哈希的内容较短版本作为名字。例如，人们可以通过一个输出是一个中等固定长度位串的加密转换函数来处理所存储文件的内容，并使用该位串作为名字。安全哈希算法（SHA，在边栏 11.8[在线]中描述）的一个版本，对任何大小的输入，都产生长度是 160 位的输出。如果转变函数是足够高质量的，两个不同文件几乎肯定会以不同名字而告终。

边栏 3.1　根据时间戳生成一个独特名字

一些银行系统为每个事务生成一个独特的字符串名字。一个典型的名字生成方案是，读取数字时钟以获得一个时间戳并把时间戳转换成字符串。典型的时间戳可能包含自 2000 年 1 月 1 日以来的微秒数。一个 50 位的时间戳将在大约 35 年之后重复，这针对银行的目的可能是足够的。假设 2007 年 4 月 1 日下午 1:35 的时间戳是

00010111110110101101001100111001100010111010011001

要转换这个位串为字符串，把它分为 5 位的块，并把每块作为对 32 个字母数字字符的一个表的索引进行解释。这些 5 位的块是：

00010–11111–01101–01101–00110–01110–01100–01011–10100–11001

接下来，将位块作为索引号重新解释：

2　31　13　13　6　16　12　11　20　25

然后，在这个 32 个字母数字字符表中查找这些数字：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
B	C	D	F	G	H	J	K	L	M	N	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5	6	7	8	9	0

结果是 10 个字符的独特名字“D0RRJ-UQPY4”。你可能在网上银行系统执行的事务中已经看到过类似的独特名字。

以命名对象的内容为基础的任何命名方案的主要问题是，名字是重载的。当有人修改了它的名字是从原来的内容构造的对象，产生的问题是，是否要改变其名字。这个问题在不允许对象被修改的保护存储系统中不会出现，因此哈希生成的唯一的名字，有时在这些系统中使用。

唯一的标识符和所生成的名字也可以在唯一标识符名字空间以外的地方使用。例如，当一个程序为一个临时文件需要一个名字，它可以指定一个生成的名字并把该文件放于用户的工作目录中。在这种情况下，名字生成器的设计挑战是设计出一个不会与由人选择的

或者其他自动名字生成器生成的已有名字中的名字发生冲突的算法。3.3.1 节给出了一个失败于满足这一挑战的系统的例子。

在一个大的、地理上分散的系统中提供唯一的名字，需要仔细的设计。一种方法是创建一个分层的命名方案。这种方法利用了一个重要的层次结构特征：代表性。

例如，互联网的一个目标是允许在一个计算机接入点的全球名字空间中创建几亿不同的唯一的名字。如果试图通过让在国际电信联盟的人协调名字指定的方式来满足这一目标，巨大数量的名字指定几乎肯定会导致长时间的拖延以及形如意外发生名字冲突的错误。

相反，一些中央权威机构指定名字“edu”或“uk”，并把以该后缀结尾的命名职责委托给其他人，在“edu”的情况下，委托一位专家来分配大学名字。该专家接受教育机构的请求，例如，分配名字“pedantic”并因而并把以后缀“.pedantic.edu”为结尾的命名职责委托给 Pedantic 大学的网络工作人员。这名工作人员分配名字“cse”给计算机科学与工程系，进一步把以后缀“.cse.pedantic.edu”为结尾的命名职责委托给该系某个人。该系的网络管理员可以，在贴在墙上的名单或者一个小的在线数据库的帮助下，分配如“ginger”的本地唯一的一个名字，同时可以相信完全合格的名字“ginger.cse.pedantic.edu”也是全球唯一的。

唯一标识符名字空间的一个不同例子是商业以太网的寻址方案。每个以太网接口都有一个唯一的 48 位介质访问控制（MAC）地址，通常制造商将其设置在硬件中。为了允许这一分配做得独一无二，但又没有一个全世界共有的单一中央登记处，有一个 MAC 地址的浅的层次结构。一个标准制定的权力机构给每个以太网接口制造商分配一块 MAC 地址，所有这些地址都开始于相同的前缀。制造商能够以方便的任何方式来自行分配该块内的 MAC 地址。如果制造商使用光了一个块内的所有 MAC 地址，它向中央权力机构申请另一个块，它可能有一个与相同制造商先前使用的前缀没有关系的前缀。

这一策略的一个后果，在大型网络中尤其显著，是一个以太网接口的 MAC 地址不提供对于物理定位接口卡有用的任何重载信息。尽管 MAC 地址以层次结构分配，层次结构仅用于委托，从而分散地址分配，对于有助于找到它的任何特性（如网卡接入网络的物理地点）并没有确保的关系。正如在寻找一辆铁路车辆而只知道它的唯一标识符，解析承载它的特定物理设备的 MAC 地址是很难的，除非人们对于从哪儿找起已经有了好主意。

正在努力寻找如何将软件许可证捆绑到特定计算机的人们，有时会提议将许可证关联到计算机的以太网 MAC 地址，因为该地址是全球唯一的。除了一些计算机没有以太网接口和其他有不止一个接口的问题，此方法的麻烦是，如果计算机上的以太网接口卡失效而需要更换，即使系统的位置、软件、它的主人都不变，新卡将有不同的 MAC 地址。此外，如果失效卡后来修复好并重新安装到另一个系统中，那个系统现在将拥有以前关联到第一个系统的 MAC 地址。MAC 地址因此仅被正确地看做是一个特定硬件组件的唯一名字，而不是它所嵌入的系统的名字。

决定是什么组成了一个由可更换部件构建的系统的独特标识，最终是一个制约命名方案设计人员任意选择的公约。这个选择类似于建立木制船只标识的问题。如果在 300 年的过程中，船上每块木头都已被替换，它还是同一条船吗？显然，船舶登记处说“是”。他们没有把船舶的名字与任何一个组成部分相关联；该名字而是与整个船只相关联。回答这个标识问题，能够澄清 2.2.5 节中讨论的 COMPARE 操作的 3 个含义中的哪个是最合适于一

个特定设计。

3.1.5 预期用户与用户友好的名字

有些命名方案是为了由人使用。在这样的名字空间中的名字通常是用户选择的和带有帮助记忆的值的用户友好字符串如“economics report”、“shopping list”或“Joe.Smith”，并广泛地用做文件名和电子邮箱名字。在解决用户友好的名字中的模糊性（即非唯一性）可能是可以接受的，因为在交互系统中，使用名字的人会被要求解决模糊性。

其他命名方案主要针对由机器使用。在这些方案中，名字不必有帮助记忆的值，所以它们通常是整数，往往设计成适合装入寄存器以及允许快速、明确的解析的固定宽度。内存地址和磁盘扇区地址就是例子。有时术语“标识符”用于一个不是为了让人理解的名字，但这种用法绝不是普遍的。拟由机器使用的名字通常是机械化选择的。

当一个名字预期是用户友好的，在它是一个独特、容易解析的标识符的需要与反映其他如易记或者同样成为现有地点或个人名字等非技术性的值的需要之间出现一种紧张局势。这种紧张局势可能会通过维持除了用户友好的名字以外的第二个面向机器的标识符来解决，因此大公司的计费系统通常有一个账户名和一个账号。第二个标识符可以是独特的，因此解决了模糊性并避免有关账户的名字重载问题。例如，人的名字通常被重载以带有家族历史元数据（如姓氏、以母亲的姓氏给定的中间名，或者附加的“小”或“三”），它们往往不是唯一的。要求人名选定始终唯一的建议，遭遇文化和个人身份的反对。为了避免这些问题，保持个人记录的大多数系统，分配不同的唯一标识符给人们，既包括用户友好的名字，也包括在它们元数据中的唯一标识符。

在用户友好名字选择过程中的困惑的另一个例子，被发现在大写和小写字母的使用上。上溯到 20 世纪 60 年代中期，计算机系统只使用大写字母，打印的计算机输出似乎是全是大写的。

有一些终端和打印机有小写字母，但人们不得不写一个依赖于设备的应用程序来使用该特性，正如今天不得不写一个依赖于设备的应用程序以使用虚拟现实头盔（helmet）。1965 年，Multics 分时共享系统的设计者将小写字母引入到文件系统的名字。这是第一次有人尝试它，他们错了。UNIX 文件系统的设计者复制了这一错误。进一步，许多现代文件系统复制了 UNIX 设计，以避免改变一个广泛使用的接口。该错误是名字“Court docket 5”和“Court Docket 5”可以绑定到不同文件。所导致的对于“最小惊讶原则”的违反，能够引起严重的混乱，因为计算机硬性强制一个大多数人习惯于在纸面上看到的区别。实施这种区别的系统称为大小写区分的。

在名字中允许大小写字母的更加用户友好的方式是，允许用户指定一个偏好的大小写字母组合用于名字的存储和显示，但是进行名字比较时，强制所有字母字符转换成相同的大小写状态。这样，当另一个人录入名字时，大小写不必与显示的形式精确匹配。以这种方式操作的系统称为大小写保留的。无论是因特网域名系统（详见 4.4 节）和 Macintosh 文件系统提供这种更加用户友好的命名接口。减少大小写混乱的一个不太令人满意的方法是大小写强制转换，所有名字都被强制转换并以一种大小写状态存储。在一个大小写强制转换系统中，约束名字的出现是以一种可能会干扰好的人体工程学的方式。