維基百科

通用唯一识别码

维基百科，自由的百科全书

通用唯一识别码（英語：**U**niversally **U**nique **Id**entifier，缩写：**UUID**）是用于计算机体系中以识别信息的一个128位标识符。

根据标准方法生成，不依赖中央机构的注册和分配，UUID具有唯一性，这与其他大多数编号方案不同。重复UUID码概率接近零，可以忽略不计。

目录

历史

1990年代， UUID 原本是用于 阿波罗电脑的 网络计算系统 （NCS），后被用于 开放软件基金会（OSF）的 分布式计算环境（DCE）。DCE UUID的初始设计基于NCS UUID，其设计受 Domain/OS 中定义和使用的（64位）唯一标识符的启发，这是一个也由 阿波罗电脑 设计的操作系统。后来，Microsoft Windows 平台采用 DCE 设计作为全局唯一标识符（GUID）。

2005年7月，RFC 4122 为 UUID 注册了一个 URN 命名空间，并制定了早期的规范。当 RFC 4122 作为 IETF 标准发布时，ITU 基于先前的标准和 RFC 4122 早期版本标准化了 UUID。

标准

UUID 由开放软件基金会（OSF）标准化，作为 分布式计算环境（DCE）的一部分。

UUID 被作为 ISO/IEC 11578:1996 "Information technology – Open Systems Interconnection – Remote Procedure Call（RPC）" 中的一部分，最近在 ITU-T Rec. X.667 | ISO / IEC 9834-8：2005 规范中。

互联网工程任务组（Internet Engineering Task Force，IETF）公布的标准 RFC 4122，技术上等同于 ITU-T Rec. X.667 | ISO/IEC 9834-8。

定义

UUID是由一個16進位下的32位數所構成，故UUID理論上的總數為$16^{32}=2^{128}$，約等於3.4×10^{38}。也就是說若每纳秒（ns）產生1萬億個UUID，要花100億年才會將所有UUID用完。

UUID的標準型式包含32個16進位數字，以連字號分為五段，形式為 8-4-4-4-12 的32個字元。範例：

```
550e8400-e29b-41d4-a716-446655440000
```

UUID亦可刻意重複以表示同類。例如說微軟的COM中，所有元件皆必須實作出IUnknown介面，方法是產生一個代表IUnknown的UUID。無論是程序試圖存取元件中的IUnknown介面，或是實作IUnknown介面的元件，只要IUnknown一被使用，皆會被參考至同一個ID：`00000000-0000-0000-C000-000000000046`。

格式

在其规范的文本表示中，UUID 的 16 个 8 位字节表示为 32 个十六进制（基数16）数字，显示在由连字符分隔 '-' 的五个组中，"8-4-4-4-12" 总共 36 个字符（32 个字母数字字符和 4 个连字符）。

例如：

```
123e4567-e89b-12d3-a456-426655440000
xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx
```

四位数字 `M`表示 UUID 版本，数字 `N`的一至三个最高有效位表示 UUID 变体。在例子中，*M* 是 1_{10} 而且 *N* 是 a（$10xx_2$），這意味着此 UUID 是「变体1」、「版本1」UUID；即基于时间的 DCE/RFC 4122 UUID。

规范的 `8-4-4-4-12` 格式字符串基于 UUID 的16个字节的“记录布局”：

UUID 记录结构

名称	长度 （字节）	长度（16进制数字码长）	说明
time_low	4	8	整数：低位 32 bits 时间
time_mid	2	4	整数：中间位 16 bits 时间
time_hi_and_version	2	4	最高有效位中的 4 bits“版本”，后面是高 12 bits 的时间
clock_seq_hi_and_res clock_seq_low	2	4	最高有效位为 1-3 bits“变体”，后跟13-15 bits 时钟序列
node	6	12	48 bits 节点 ID

这些字段对应于「版本1」和「版本2」基于时间的 UUID 中的字段，但是相同的 "8-4-4-4-12" 表示用于所有UUID，即使对于构造不同的UUID也是如此。

RFC 4122 第 3 节要求以小写形式生成字符，同时对输入不区分大小写，尽管一些常用的实现违反了此规则。

Microsoft GUID有时用周围的大括号表示：

```
{123e4567-e89b-12d3-a456-426655440000}
```

不应将此格式与“ Windows注册表格式” 混淆，后者指的是花括号内的格式。

RFC 4122 为UUID 定义了统一资源名称（URN）命名空间。作为URN呈现的UUID如下：

```
urn:uuid:123e4567-e89b-12d3-a456-426655440000
```

编码

UUID 的二进制编码因系统而异。UUID的变体1完全以大端序（big-endian）二进制存储与传输 UUID 。

例如，`00112233-4455-6677-8899-aabbccddeeff` 编码为字节 `00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff`。

其他系统，特别是 Microsoft 在其 COM/OLE 库中对 UUID 的字符串表示，使用混合端格式，其中 UUID 的前三组是小端序/小尾序（little-endian），后两组是 大端序/大尾序（big-endian）。

例如，`00112233-4455-6677-8899-aabbccddeeff` 编码为字节 `33 22 11 00 55 44 77 66 88 99 aa bb cc dd ee ff`。

变体

UUID的**变体**（variant）字段，占1或2或3比特。RFC 4122定义了4种变体：

- 变体 0 (最显著位为0，形如 $0xxx_2$) 用于向后兼容已经过时的1988年开发的 Apollo 网络计算系统（NCS）1.5 UUID 格式。 前6字节是48比特时间戳（从1980年1月1日UTC开始的4微秒

的滴答数），随后2个字节保留，再1个字节是地址族（address family，使用了0..13个情形），最后7个字节是主机ID。这类似于UUID的版本1.[1]

- 变体 1 (形如$10xx_2$)，定义在RFC 4122/DCE 1.1 UUIDs, 或"Leach–Salz" UUID。它是按照大端序作为二进制存储与传输。
- 变体 2 (形如$110x_2$)，RFC称“保留，微软公司向后兼容”。早期的Microsoft Windows平台的GUID使用该格式。它是按照小端序作为二进制存储与传输。
- 形如$111x_2$ 保留未使用。

目前，Microsoft `guidgen` 工具软件产生变体1的结果。微软文档[2]与RFC 4122称GUID与UUID是同义词。

版本

对于「变体（variants）1」和「变体2」，标准中定义了五个版本（versions），并且在特定用例中每个版本可能比其他版本更合适。

版本由 M 字符串中指示。

版本1 - UUID 是根据时间和 节点ID（通常是MAC地址）生成；

版本2 - UUID是根据标识符（通常是组或用户ID）、时间和节点ID生成；

版本3、版本5 - 确定性UUID 通过散列（hashing）命名空间（namespace）标识符和名称生成；

版本4 - UUID 使用随机性或伪随机性生成。

Nil UUID

Nil UUID是一个特例，值为 `00000000-0000-0000-0000-000000000000` ；也就是说，所有位都设置为 0。

版本1（日期时间和MAC地址）

版本1的UUID，是根据 60-bit 的时间戳 和 节点（生成UUID的计算机）的 48-bit MAC地址 而生成的。

时间戳的是这样计算的。自1582年10月15日午夜起协调世界时（UTC）以来的100纳秒间隔数，公历首次采用的日期。RFC 4122声明时间值在公元3400左右滚动，取决于所使用的算法，这意味着 60-bit 时间戳是有符号数量。但是，某些软件（如libuuid库）将时间戳视为无符号，将转存时间设置为 5236 AD。

13-bit 或 14-bit「无统一」（uniquifying）时钟序列扩展了时间戳，以便处理处理器时钟不能足够快地前进的情况，或者每个节点有多个处理器和 UUID 生成器的情况。对于每个「版本1」UUID 对应于空间（节点）和时间（间隔和时钟序列）中的单个点，两个正确生成的「版本1」UUID 无意中相同的可能性实际上为零。由于时间和时钟序列总共74位，每个节点 id 可以生成 2^{74}（1.8×10^{22} 或 18 sextillion）個「版本1」UUID，每个节点 id 的最大平均速率为每秒 1630 亿。

与其他 UUID 版本相比，基于来自网卡的 MAC 地址的「版本1」和「版本2」UUID，部分依赖于由中央注册机构发布的标识符，即 MAC 地址的组织唯一标识符（OUI）部分，由 IEEE 发布给网络设备制造商。基于网卡 MAC地址的「版本1」和「版本2」UUID 的唯一性还取决于网卡制造商正确地为其卡分配唯一的 MAC地址，这与其他制造过程一样容易出错。

使用节点的网络 MAC 地址作为节点 ID ，意味着可以将「版本1」UUID 追溯回创建它的计算机。文档有时可以跟踪到通过文字处理软件嵌入到它们中的 UUID 创建或编辑它们的计算机。在找到 Melissa 病毒的创建者时使用了这个隐私漏洞。

RFC 4122 确实允许「版本1」（或2）UUID 中的 MAC 地址被随机的 48 位节点 id 替换，因为该节点没有 MAC 地址，或者因为不希望暴露它。在这种情况下，RFC要求节点 id 的第一个八位字节的最低有效位应设置为 1，这对应于MAC地址中的多播位，并且设置它用于区分随机生成节点 id 的 UUID 和基于来自网卡的 MAC 地址的 UUID，网卡通常具有单播 MAC 地址。

版本2（日期时间和MAC地址，DCE安全版本）

RFC 4122 保留了“DCE security” UUID 的「版本2」；但它没有提供任何细节。因此，许多 UUID 实现省略了「版本2」。但是，「版本2」UUID 的规范由 DCE 1.1 身份验证和安全服务规范提供。

「版本2」 UUID 类似于「版本1」，除了时钟序列的最低有效8 bits 被“本地域（local domain）”号替换，并且时间戳的最低有效32 bits 由在指定本地域内有意义的整数标识符替换。在 POSIX 系统上，本地域号 0 和 1 分别用于用户 ID（UIDs）和组 ID（GIDs），其他本地域号用于站点定义。在非 POSIX 系统上，所有本地域号都是站点定义的。

在 UUID 中包含 40 bits 域/标识符（domain/identifier）的能力需要权衡。一方面，40 bits 允许每个节点 id 有大约1万亿个 域/标识符 值。另一方面，时钟值被截断为 28 个最高有效位，而「版本1」中为60 bits；「版本2」的 UUID 中的时钟每429.49秒進行一次标记（tick），略多于7分钟，而不是「版本1」中的每 100 纳秒；并且，「版本2」仅 6 bits 时钟序列，「版本1」中 14 bits；每7分钟时钟周期内，每个 节点/域/标识符（node/domain/identifier）只能生成64个唯一的UUID，而版本1的时钟序列值为16,384个。 因此，「版本2」可能不适合于以每个 节点/域/标识符 生成 UUID 的情况，其速率约 7秒以上1次。

版本3和版本5（基于命名空间名称）

「版本3」和「版本5」的 UUID 被生成雜湊（hashing）一个命名空间标识符和一个名称。「版本3」使用 MD5 作为散列算法，「版本5」使用 SHA1。

名称空间标识符本身就是一个 UUID。该规范提供了 UUID 用来表示命名空间为了统一资源定位符（URLs），完整域名、对象标识符和 X.500 轻型目录访问协议 ；但任何所需的UUID都可以用作命名空间指示符。

要确定与给定命名空间和名称对应的「版本3」UUID，命名空间的 UUID 将转换为字节串，与输入名称连接，然后用 MD5 进行散列，产生 128 bits。然后将六或七位替换为固定值，即 4-bit 版本（例如“版本3”的 0011），以及 2-bit 或 3-bit UUID「变体（variant）」（例如 10 指示RFC 4122 UUID，或 110 指示传统 Microsoft GUID）。由于预定了6或7 bits，因此只有121 或 122 bits 有助于 UUID 的唯一性。

「版本5」UUID 类似，但使用 SHA1 而不是 MD5。由于 SHA1 生成 160-bit 摘要，因此在替换版本和变体位之前，摘要将截断为 128-bits。

「版本3」和「版本5」UUID 具有相同名称空间和名称，将映射到同一 UUID ；除了暴力搜索之外，命名空间和名称都不能从 UUID 中确定。

RFC 4122 推荐「版本5」（SHA1）而不是「版本3」（MD5），并建议不要使用任一版本的 UUID 作为安全凭证。

版本4（随机）

随机生成「版本4」UUID。与其他 UUID 一样，4-bit 用于指示「版本4」，2-bit 或 3-bit 用于指示变体（variant）（10 或 110 分别用于 变体 1 和 2）。因此，对于变体1（即大多数 UUID），随机「版本4」UUID 将具有 6 个预定的变体和版本位，为随机生成的部分留下122位，「版本4」变体1 UUID 可能共计 2^{122} 或5.3×10^{36}（5.3 undecillion，大数名词）个。「版本4」变体2 UUID（传统GUID）的可能有一半，因为可用的随机位少一个，变量消耗 3 bits。

一些伪随机数发生器缺少必要的熵来产生足够的伪随机数。例如，使用伪随机数生成器的 WinAPI GUID 生成器已被证明可生成遵循可预测模式的 UUID。 RFC 4122 建议“在各种主机上生成 UUID 的分布式应用程序必须愿意依赖所有主机上的随机数源。如果这不可行，则应使用名称空间变体。”

碰撞

当多次生成相同的 UUID 并将其分配给不同的指示对象时，就会发生冲突。对于使用来自网卡的唯一MAC地址的标准 版本1和2 的 UUID ，只有当实施与标准不同时，无论是无意还是故意，都可能发生冲突。

与 版本1 和 版本2 相比， UUID 使用随机生成的节点 ID，基于散列的 版本3 和 版本5 的 UUID ，以及随机 版本4 的 UUID ，即使没有实现问题也可能发生冲突，尽管可能性很小，通常可能是忽略。可以基于对生日问题的分析来精确地计算该概率。

例如，至少碰撞一次具有50%的概率，根据 版本4 需要生成的 UUID 的数量是 2.71×10^{18}，计算如下：

$$n \approx \frac{1}{2} + \sqrt{\frac{1}{4} + 2 \times \ln(2) \times 2^{122}} \approx 2.71 \times 10^{18}$$

这个数字相当于大约 85 年每秒产生 10 亿个 UUID ，每个 UUID 16 bytes/字节，包含这么多 UUID 的文件，大约 45 艾字节（EB），比目前存在的最大数据库大很多倍，它们都在数百PB的数量级。

最小数必须查找碰撞要的概率来生成 版本4 的 UUID 的 p 由下式近似计算：

$$\sqrt{2 \times 2^{122} \times \ln \frac{1}{1-p}}$$

因此，在 103 万亿 个 版本4 UUID 中找到重复的概率是 $\frac{1}{1000000000}$（十亿分之一）。

使用

重要用途包括 ext2/ext3/ext4 文件系统用户空间工具（e2fsprogs 使用 util-linux 提供的 libuuid），LUKS 加密分区，GNOME，KDE 和 Mac OS X，其中大部分源自 曹子德（Theodore Ts'o）的实现。

Solaris 中 UUID 的一种用途（使用Open Software Foundation实现）是识别正在运行的操作系统实例，以便在内核崩溃的情况下将故障转储数据与故障管理事件配对。

COM

Microsoft的组件对象模型（COM）中使用了几种GUID ：

IID - 接口标识符；（在系统上注册的那些存储在Windows注册表中）`[HKEY_CLASSES_ROOT\Interface]`

CLSID - 类标识符；（存储在）`[HKEY_CLASSES_ROOT\CLSID]`

LIBID - 类型库标识符；（存储于）`[HKEY_CLASSES_ROOT\TypeLib]`

CATID - 类别标识符；（它在一个类中的存在将其识别为属于某些类别类别）`[HKEY_CLASSES_ROOT\Component Categories]`

作为数据库主键

UUID 通常用作唯一键的数据库表。

Microsoft SQL Server 版本4 中的 NEWID 函数 Transact-SQL 返回标准随机 版本4 UUID，而 NEWSEQUENTIALID 函数返回类似于 UUID 的128 位标识符，这些 UUID 将按顺序提升，直到下次系统重新引导。

尽管名称如此，但 Oracle Database SYS_GUID 函数不会返回标准 GUID；相反，它根据主机标识符和进程或线程标识符返回一个16字节的 128 位 RAW 值，有点类似于 GUID。

PostgreSQL 包含一个 UUID 数据类型，并且可以通过使用模块中的函数生成大多数版本的 UUID。

MySQL 提供了一个 UUID 函数，它生成标准 版本1 UUID。

当 UUID 用作主键时，版本3、4和5 UUID 的随机性以及 版本1和2 UUID 内的字段的排序可能会产生数据库位置或性能问题。例如，2002年 Jimmy Nilsson 报告说，当用作主键的版本4 UUID 被修改为包含基于系统时间的非随机后缀时，Microsoft SQL Server 的性能显着提高。Nilsson 承认，这种所谓的“COMB”（组合时间-GUID）方法使UUID非标准并且更有可能被复制，但 Nilsson 仅在应用程序中要求唯一性。

参见

- 全局唯一标识符（GUID）

参考文献

1. uuid.c. [2020-10-09]. （原始内容存档于2021-02-24）.

2. Globally Unique Identifiers. Microsoft Developer Network. Microsoft. [2020-10-09]. （原始内容存档于2019-02-13）.