

中华人民共和国国家标准

GB/T 17969.8—2024/ISO/IEC 9834-8:2014

代替 GB/T 17969.8—2010

信息技术 对象标识符登记机构操作规程 第 8 部分：通用唯一标识符（UUIDs） 的生成及其在对象标识符中的使用

Information technology—Procedures for the operation of object identifier
registration authorities—Part 8: Generation of universally unique
identifiers(UUIDs) and their use in object identifiers

(ISO/IEC 9834-8:2014, IDT)

2024-05-28 发布

2024-05-28 实施

国家市场监督管理总局
国家标准化管理委员会 发布

目次

前言 III

引言 IV

1 范围 1

2 规范性引用文件 1

3 术语和定义 2

4 缩略语 3

5 记法 3

6 UUID 结构和表示 4

7 使用 UUID 作为联合 UUID 弧的主体整数值和 Unicode 标签 5

8 使用 UUID 来生成 URN 6

9 UUID 的比较和排序规则 6

10 验证..... 6

11 保留位..... 6

12 UUID 字段用途和传输字节次序..... 7

13 设置基于时间的 UUID 字段 9

14 设置基于名称的 UUID 字段 9

15 设置基于随机数的 UUID 字段 10

附录 A（资料性） 有效生成基于时间的 UUID 的算法 11

附录 B（资料性） 基于名称的 UUID 的特性 13

附录 C（资料性） 在系统中随机数的生成 14

附录 D（资料性） 实例实现 15

参考文献 29

前 言

本文件按照 GB/T 1.1—2020《标准化工作导则 第 1 部分：标准化文件的结构和起草规则》的规定起草。

本文件是 GB/T 17969 的第 8 部分。GB/T 17969 已经发布了以下部分：

- 信息技术 开放系统互连 OSI 登记机构的操作规程 第 1 部分：一般规程和国际对象标识符树的顶级弧(GB/T 17969.1—2015)；
- 信息技术 开放系统互连 OSI 登记机构的操作规程 第 3 部分：ISO 和 ITU-T 联合管理的顶级弧下的客体标识符弧的登记(GB/T 17969.3—2008)；
- 信息技术 开放系统互连 OSI 登记机构的操作规程 第 5 部分：VT 控制客体定义的登记表(GB/T 17969.5—2000)；
- 信息技术 开放系统互连 OSI 登记机构的操作规程 第 6 部分：应用进程和应用实体(GB/T 17969.6—2000)；
- 信息技术 对象标识符登记机构操作规程 第 8 部分：通用唯一标识符(UUIDs)的生成及其在对象标识符中的使用(GB/T 17969.8—2024)。

本文件代替 GB/T 17969.8—2010《信息技术 开放系统互连 OSI 登记机构操作规程 第 8 部分：通用唯一标识符(UUID)的生成和登记及其用作 ASN.1 客体标识符部件》，与 GB/T 17969.8—2010 相比，除结构调整和编辑性改动外，主要技术变化如下：

- a) 更改原第 7 章标题“使用 UUID 来形成 OID”为“使用 UUID 作为联合 UUID 弧的主体整数值和 Unicode 标签”，并修改第 7 章相应内容；
- b) 删除了原第 16 章“UUID 的登记及其用作 OID 部件”。

本文件等同采用 ISO/IEC 9834-8:2014《信息技术 对象标识符登记机构操作规程 第 8 部分：通用唯一标识符(UUIDs)的生成及其在对象标识符中的使用》。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别专利的责任。

本文件由全国信息技术标准化技术委员会(SAC/TC 28)提出并归口。

本文件起草单位：中国电子技术标准化研究院、深圳赛西信息技术有限公司、中国科学院计算技术研究所、联想(北京)有限公司、中移(杭州)信息技术有限公司、西安航天自动化股份有限公司、浙江晶日科技股份有限公司、天津鲲鹏信息技术有限公司、上海天臣微纳米科技股份有限公司、重庆邮电大学、重庆邮电大学工业互联网研究院、江苏中天互联科技有限公司。

本文件主要起草人：蔡廷晓、王婷、郭雄、杨宏、刘敏、苏静茹、卓兰、孙旭、张弛、韩世豪、孙胜、陈飞、陈驭政、贾景润、范营营、沈杰、李应龙、周立雄、孟振亚、黄庆卿、施超、陶怡、陈娟、李润泽、董速、时宗胜。

本文件及其所代替文件的历次版本发布情况：

- 2010 年首次发布为 GB/T 17969.8—2010；
- 本次为第一次修订。

引 言

GB/T 17969 旨在规定对象标识符登记机构的操作规程,基于 ISO/IEC 9834 编制,拟由 5 个部分构成。

- 信息技术 开放系统互连 OSI 登记机构的操作规程 第 1 部分:一般规程和国际对象标识符树的顶级弧(GB/T 17969.1—2015)。目的是规定国际对象标识符树及其顶级弧。
- 信息技术 开放系统互连 OSI 登记机构的操作规程 第 3 部分:ISO 和 ITU-T 联合管理的顶级弧下的客体标识符弧的登记(GB/T 17969.3—2008)。目的是规定该对象标识符弧的登记规程,包括修订、添加项等。
- 信息技术 开放系统互连 OSI 登记机构的操作规程 第 5 部分:VT 控制客体定义的登记表(GB/T 17969.5—2000)。目的是规定登记表各项内容。
- 信息技术 开放系统互连 OSI 登记机构的操作规程 第 6 部分:应用进程和应用实体(GB/T 17969.6—2000)。目的是规定应用进程和应用实体的登记规程。
- 信息技术 对象标识符登记机构操作规程 第 8 部分:通用唯一标识符(UUIDs)的生成及其在对象标识符中的使用(GB/T 17969.8—2024)。目的是规定通用唯一标识符的生成规程及其在对象标识符中的使用。

信息技术 对象标识符登记机构操作规程
第 8 部分：通用唯一标识符(UUIDs)
的生成及其在对象标识符中的使用

1 范围

本文件规定了生成 128 位标识符的格式和生成规则，这些标识符保证是全球唯一的，或大概率是全球唯一的。

遵守本文件以每 100 ns 生成一个新的通用唯一标识符(UUID)既适用于暂时使用，也可作为永久标识符。

本文件源自 UUID 及其生成的早期非标准规范，并且在技术上等同于这些早期规范。

本文件还规定和允许使用 UUID 作为联合 UUID 弧之下的各弧对应的主值(定义了 Unicode 标签)。这使用户能生成和使用此类弧而无需任何登记规程。

本文件还规定和允许使用 UUID 来形成统一资源名称(URN)。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本(包括所有的修改单)适用于本文件。

ISO/IEC 8802-3 信息技术 系统间远程通信和信息交换 局域网和城域网 特定要求 第 3 部分：带碰撞检测的载波侦听多址访问(CSMA/CD)的访问方法和物理层规范[Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications]

注：GB/T 15629.3—2014 信息技术 系统间远程通信和信息交换 局域网和城域网 特定要求 第 3 部分：带碰撞检测的载波侦听多址访问(CSMA/CD)的访问方法和物理层规范(ISO/IEC 8802-3:2000, MOD)

ITU-T X.680(2008)|ISO/IEC 8824-1:2008 信息技术 抽象语法记法一(ASN.1)：基本记法规范[Information technology—Abstract Syntax Notation One (ASN.1): Specification of basic notation]

注：GB/T 16262.1—2006 信息技术 抽象语法记法一(ASN.1) 第 1 部分：基本记法规范(ISO/IEC 8824-1:2002, IDT)

ITU-T X.660|ISO/IEC 9834-1 信息技术 对象标识符登记机构操作规程：一般规程和国际对象标识符树的顶级弧(Information technology—Procedures for the operation of object identifier registration authorities: General procedures and top arcs of the international object identifier tree)

注：GB/T 17969.1—2015 信息技术 开放系统互连 OSI 登记机构的操作规程 第 1 部分：一般规程和国际对象标识符树的顶级弧(ISO/IEC 9834-1:2008, NEQ)

FIPS PUB 180-3 联邦信息处理标准，安全散列标准(SHS)[Federal Information Processing Standards Publication, Secure Hash Standard (SHS)]

IETF RFC 1321 MD5 消息摘要算法(The MD5 Message-Digest Algorithm)

3 术语和定义

下列术语和定义适用于本文件。

3.1 ASN.1 记法

本文件使用了在 ITU-T X.680|ISO/IEC 8824-1 中定义的下列术语：

- a) 协调世界时(UTC) Coordinated Universal Time(UTC)；
- b) 对象标识符类型 object identifier type；
- c) OID 国际化资源标识符类型 OID internationalized resource identifier type。

3.2 登记机构

本文件使用了在 ITU-T X.660|ISO/IEC 9834-1 中定义的下列术语：

- a) 国际对象标识符树 International Object Identifier tree；
- b) IRI/URI 值 IRI/URI value；
- c) OID 国际化资源标识符 OID internationalized resource identifier；
- d) 对象标识符 object identifier；
- e) 登记 registration；
- f) 登记机构 registration authority；
- g) 登记规程 registration procedures；
- h) 次标识符 secondary identifier；
- i) Unicode 字符 Unicode character；
- j) Unicode 标签 Unicode label。

3.3 网络术语

本文件使用了在 ISO/IEC 8802-3 中定义的下列术语：

MAC 地址 MAC address。

3.4 附加定义

3.4.1

密码质量级随机数 cryptographic-quality random-number

由某一机制生成的随机数或伪随机数，它确保重复生成的值均足够分散以满足在密码工作中使用（并在此类工作中使用）。

3.4.2

联合 UUID 弧 Joint UUID arc

由 ASN.1 对象标识符值 {joint-iso-itu-t(2) uuid(25)} 和 ASN.1 OID 国际化资源标识符值“/UUID”标识的国际对象标识符树节点下的弧。

3.4.3

基于名称的版本 name-based version

针对名称空间名称以及在该名称空间中的一个名称使用密码散列技术后生成的某个 UUID。

3.4.4

名称空间 name space

生成对象名称的系统，该系统确保在该名称空间内生成无歧义性名称标识。

注：名称空间的例子有网络域名系统、URN、OID、目录可辨别名（见参考文献[1]）和编程语言中的保留字。

3.4.5

基于随机数的版本 random-number-based version

使用随机数或伪随机数来生成的某个 UUID。

3.4.6

标准 UUID 保留字段 standard UUID variant

通过本文件规定的可能格式的 UUID 的保留字段。

注：在历史上，已经存在其他 UUID 格式的规范对保留字段的规定不同于本文件，根据所有这些保留字段格式生成的 UUID 都是不同的。

3.4.7

基于时间的版本 time-based version

通过标识系统的 MAC 地址以及基于当前协调世界时的时钟值所获得唯一性的某个 UUID。

3.4.8

通用唯一标识符 universally unique identifier, UUID

按照本文件生成的 128 位值，或按照某些历史规范生成且确保了系统间的以及时间上的唯一值（也见 3.4.6）。

4 缩略语

下列缩略语适用于本文件。

- ASN.1 抽象语法记法一（Abstract Syntax Notation One）
- GUID 全球唯一标识符（Globally Unique Identifier）
- MAC 媒体访问控制（Media Access Control）
- MD5 消息摘要算法 5（Message Digest algorithm 5）
- OID 对象标识符（Object Identifier）
- OID-IRI OID 国际化资源标识符（OID internationalized resource identifier）
- SHA-1 安全散列算法 1（Secure Hash Algorithm 1）
- URL 统一资源定位符（Uniform Resource Locator）
- URN 统一资源名称（Uniform Resource Name）
- UTC 协调世界时（Coordinated Universal Time）
- UUID 通用唯一标识符（Universally Unique Identifier）

5 记法

- 5.1 本文件使用第一个和最后一个的术语规定了 UUID 的八位字节序列。第一个八位字节也称作“八位字节 15”，最后一个八位字节称作“八位字节 0”。
- 5.2 UUID 内的位还被编号为“位 127”到“位 0”，其位 127 作为八位字节 15 的最高有效位，位 0 作为八位字节 0 的最低有效位。
- 5.3 本文件中使用时和表时，最高有效八位字节（和最高有效位）被显示在页面的左边。这与八位字节的传输次序相对应，其最左边的八位字节首先发送。
- 5.4 本文件中所使用的许多值被表达为某一给定长度（设为 N）的无符号整数值。N 位无符号整数值的位被编号为“位(N-1)”到“位 0”，其中位(N-1)作为最高有效位，位 0 作为最低有效位。
- 5.5 这些记法仅用于本文件。其在计算机存储器中的表示并未标准化，而是取决于系统体系结构。

6 UUID 结构和表示

6.1 UUID 字段结构

6.1.1 UUID 被规定为 6 个字段的有序序列。UUID 依据这些 UUID 字段的拼接而定。UUID 各字段命名如下：

- a) “TimeLow”字段；
- b) “TimeMid”字段；
- c) “VersionAndTimeHigh”字段；
- d) “VariontAndClockSegHigh”字段；
- e) “ClockSeqLow”字段；
- f) “Node”字段。

6.1.2 UUID 各字段按照以上列出的有效次序进行定义，其中“TimeLow”为最高有效字段（“TimeLow”的位 31 是 UUID 的位 127），“Node”为最低有效字段（“Node”的位 0 是 UUID 的位 0）。

6.1.3 这些 UUID 字段的内容依据版本字段、保留字段、时间字段、时钟序列字段以及节点字段的无符号整数值来规定（每个都具有固定的位长度）。这些值的设置在第 12 章中规定，它们与上述 UUID 字段的映射在 12.1 中规定。

注：正如某些 UUID 字段（例如，TimeLow、TimeMid 和 TimeHigh）的部分名称所示，来自特定无符号整数值（例如，从时间值的位 59 至位 0）的 UUID（位 127 至位 0），其位序列次序和在该无符号整数值中的位序列次序不相同。这是由于历史原因造成的。

6.2 二进制表示

6.2.1 UUID 应按二进制表示为 16 个八位字节，通过将每个字段按无符号整数固定长度编码成一个或多个八位字节，并将其拼接而成的。每个字段使用的八位字节数应是：

- a) “TimeLow”字段：4 个八位字节；
- b) “TimeMid”字段：2 个八位字节；
- c) “VersionAndTimeHigh”字段：2 个八位字节；
- d) “VariontAndClockSegHigh”字段：1 个八位字节；
- e) “ClockSeqLow”字段：1 个八位字节；
- f) “Node”字段：6 个八位字节。

注：UUID 字段的这种次序在计算机系统中是一种常用表示，并且以十六进制文本表示（见 6.4）。

6.2.2 每个 UUID 字段的无符号整数编码中的最高有效位应是其第一个八位字节（八位字节 N，即最高有效八位字节）的最高有效位，无符号整数编码中的最低有效位应是其最后一个八位字节（八位字节 0，即最低有效八位字节）的最低有效位。

6.2.3 UUID 各字段应按其高低顺序拼接而成（见 6.1.2），首先是最高有效字段，最后是最低有效字段。

6.3 单个整数值的表示

UUID 能表示为单个整数值。为了获得 UUID 的单个整数值，应把其二进制表示的 16 个八位字节处理为无符号整数编码，即把该整数编码的最高有效位作为其 16 个八位字节的第一个八位字节（八位字节 15）的最高有效位（位 7），把该整数编码的最低有效位作为其 16 个八位字节的最后一个八位字节（八位字节 0）的最低有效位（位 0）。

注：当 UUID 形成第 7 章中规定的联合 UUID 弧的主整数值时，才使用单个整数值。

6.4 十六进制表示

对于十六进制格式,二进制格式的八位字节应通过十六进制数字串表示,每个二进制格式的八位字节使用两个十六进制数字,第一个为八位字节 15 的四个高阶位,第二个为八位字节 15 的四个低阶位,依次类推,最后一个为八位字节 0 的低阶位(见 6.5)。每对十六进制表示的相邻字段之间应插入一个连字符减号(编码 45)(见 ISO/IEC 10646)，“VariantAndClockSeqHigh”字段和“ClockSeqLow”字段之间除外(见第 8 章中的示例)。

6.5 十六进制表示的形式语法

6.5.1 UUID 十六进制表示语法的形式定义是用 ITU-T X.680(2008)|ISO/IEC 8824-1:2008 第 5 章中定义的扩充 BNF 记法来规定的,但词项之间应没有空格除外。

6.5.2 在 BNF 规范中使用了“hexdigit”词项,并且定义如下:

词项的名称——hexdigit
“hexdigit”应由下列符号之一组成:
A B C D E F a b c d e f 0 1 2 3 4 5 6 7 8 9

6.5.3 十六进制表示的 UUID 应按以下格式产生:

UUID::=
 TimeLow
 "-" TimeMid
 "-" VersionAndTimeHigh
 "-" VariantAndClockSeqHigh ClockSeqLow
 "-" Node
TimeLow::=
 HexOctet HexOctet HexOctet HexOctet
TimeMid::=
 HexOctet HexOctet
VersionAndTimeHigh::=
 HexOctet HexOctet
VariantAndClockSeqHigh::=
 HexOctet
ClockSeqLow::=
 HexOctet
Node::=
 HexOctet HexOctet HexOctet HexOctet HexOctet HexOctet
HexOctet::=
 hexdigit hexdigit

6.5.4 软件生成十六进制表示的 UUID 时不应使用大写字母。

注:建议将所有人类可读格式中使用的十六进制表示限制为小写字母。然而,处理该表示法的软件要求能接受在 6.5.2 中规定的大小写字母。

7 使用 UUID 作为联合 UUID 弧的主体整数值和 Unicode 标签

7.1 使用 6.3 中规定的 UUID 单个整数值,UUID 能用作联合 UUID 弧的主体整数值。

7.2 在 6.4 中定义的十六进制表示的 UUID 也能用作弧的非整数 Unicode 标签。

示例：下面是用 UUID 形成 IRI/URI 值的例子：oid:/UUID/f81d4fae-7dec-11d0-a765-00a0c91e6bf6

8 使用 UUID 来生成 URN

使用 UUID 所生成的 URN(见 IETF RFC 2141)应使用字符串“urn:uuid:”标识,其后为 6.4 中定义的十六进制表示的 UUID。

示例：下面是用字符串表示的 UUID 生成 URN 的例子：urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6

注：一种可替代的 URN 格式(见参考文献[7])是可用的,但此格式不推荐使用 UUID 生成 URN。这个可替代格式采用了 6.3 中规定的单个整数值表示的 UUID,上述例子用此格式表示为：urn:oid: 2.25.329800735698586629295641978511506172918

9 UUID 的比较和排序规则

9.1 比较一对 UUID 时,按照有效次序(见 6.1.2)对 UUID 的每个字段(见 6.1)的值进行比较。当且仅当所有对应的字段值都相等时,两个 UUID 才相等。

注 1：比较两个 UUID 的算法等同于 6.3 中规定的比较单个整数表示值的算法。

注 2：这个比较使用了在 6.1.1 中规定的具体字段,而不是用 6.1.3 中列出和在第 12 章中规定的值(时间字段、时钟序列字段、保留字段、版本字段和节点字段)。

9.2 某个 UUID 被认为大于另一个 UUID 的条件是其在两者不同的最高有效字段中值更大。

9.3 十六进制表示的 UUID(见 6.4)按词典排序时,大的 UUID 应在小的 UUID 后面。

10 验证

除了能对保留位是否设置正确进行判断,以及在基于时间的 UUID 中是否设置了还没有到的时间值(因为不可能被赋值)之外,因为所有可取的值都可能出现,其他情况下均没有判断 UUID 在实际意义上是否有效的验证机制。

11 保留位

11.1 保留位是八位字节 7 中的 3 个最高有效位(位 7、位 6 和位 5),该八位字节是“VariantAndClockSeqHigh”字段的最高有效八位字节。

11.2 符合本文件的所有 UUID,其保留位应满足八位字节 7 的位 7 被设置为 1,八位字节 7 的位 6 被设置为 0。八位字节 7 的位 5 是时钟序列字段的最高有效位,并且应按照 12.4 来设置。

注：这里列出的位 5 作为保留位,是因为其值可区分历史格式。严格来说,它并不是本文件的保留字段的一部分,本文件仅使用了保留字段的两个位。

11.3 表 1 列出了保留位其他值的使用,以供参考。

表 1 保留位的使用

位 7	位 6	位 5	描述
0	—	—	保留,用于为网络计算系统提供后向兼容
1	0	—	在本文件中规定的保留位

表 1 保留位的使用（续）

位 7	位 6	位 5	描 述
1	1	0	保留,用于为微软公司提供后向兼容
1	1	1	保留,供本文件未来使用

12 UUID 字段用途和传输字节次序

12.1 概述

12.1.1 表 2 给出了用二进制表示的各种不同 UUID 字段的位置并总结了这些字段的用途。

表 2 UUID 字段的位置和使用

字 段	UUID 的八位字节号	描 述
“TimeLow”	15~12	时间值的低阶位(32 位)
“TimeMid”	11~10	时间值的中间位(16 位)
“VersionAndTimeHigh”	9~8	版本(4 位),后面跟着时间值的高阶位(12 位)
“VariontAndClockSeqHigh”	7	保留位(2 位),后面跟着时钟序列的高阶位(6 位)
“ClockSeqLow”	6	时钟序列的低阶位(8 位)
“Node”	5~0	节点(见 12.5)(48 位)

12.1.2 在图 1 中示出了以二进制表示的 UUID 字段的位置。

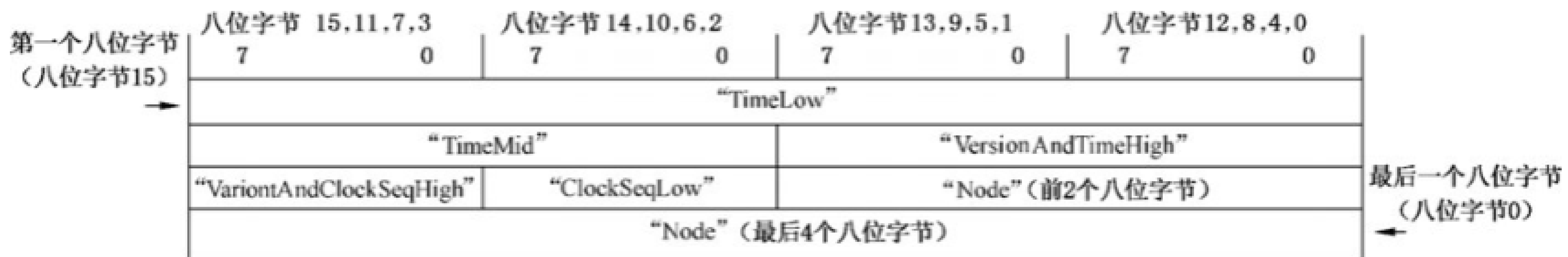


图 1 以二进制表示的 UUID 字段的位置

12.1.3 建议通信机制中传输 UUID 字段用二进制表示。二进制表示的 16 个八位字节作为一组连续的 16 个位集合来发送,传输次序按从第 1 个八位字节(八位字节 15)到最后一个八位字节(八位字节 0)。附录 D 提供了以 C 程序设计语言表示的用来生成 UUID 的完整程序实例。

- 注 1: 八位字节内的位次序由通信机制规范来确定。
- 注 2: 建议按上述规定次序使用 16 个连续八位字节传输 UUID,但协议规范可选择其他传输 UUID 的方法,包括分片传输或仅传输 UUID 的某些部分(例如,时间值的变动部分)。

12.2 版本字段

12.2.1 生成 UUID 的 3 个可选方式(基于时间的、基于名称的和基于随机数的)通过“VersionAndTimeHigh”字段的高 4 位(UUID 中的八位字节 9 的位 7~位 4)来标识和区分。使用这些不同的机制所生成的 UUID 称作“不同的 UUID 版本”。

注：将其描述为“不同的 UUID 版本”稍存在误导，但由于历史原因，该名称才被使用。UUID 格式不存在传统“版本号”的概念，新的版本可被定义为本文件的修订本。将来需要的任何新 UUID 格式可通过保留位的不同值来标识。

12.2.2 表 3 列出了目前定义的“UUID 版本”，使用了“VersionAndTimeHigh”字段的前 4 个位（UUID 中的八位字节 9 的位 7～位 4），并为每个位组合分配一个整数“版本”值。

注：为了保持与 UUID 的历史定义兼容，未使用版本值 2。版本值 0 和 6 到 15 保留，供将来使用。

表 3 当前定义的 UUID 版本

位 7	位 6	位 5	位 4	版本值	描 述
0	0	0	1	1	在本文件中规定的基于时间的版本（见第 13 章）
0	0	1	0	2	保留，供带有嵌入式 POSIX UUID 的 DCE 安全版本用
0	0	1	1	3	在本文件中规定的使用 MD5 散列的基于名称的版本（见第 14 章）
0	1	0	0	4	在本文件中规定的基于随机数的版本（见第 15 章）
0	1	0	1	5	在本文件中规定的使用 SHA-1 散列的基于名称的版本（见第 14 章）

12.3 时间字段

12.3.1 时间字段应是一个 60 位的值。

注：名称“时间”既适用于基于时间的 UUID 版本（版本 1），但也适用于 UUID 其他版本（版本 3 和 4）中的相应值的内容。

12.3.2 对于基于时间的 UUID 版本来说，时间应是协调世界时（UTC）按 100ns 间隔的计数，该协调世界时（UTC）是从 1582 年 10 月 15 日午夜〔格列高里历（Gregorian）改为赫里斯蒂安历（Christian）的日期〕开始的。

注 1：在国际 de l’Heure 办公署（国际时间办公署）建立之前，每分包含了精确的 60s。此后，根据需要增加（或潜在减少）每年的秒数时便会引入闰秒。

注 2：便携式系统可能会在确定协调世界时时遇到问题，因为它们通常被锁定到本地时区的时间。只要持续使用本地时区的时间，或变更时钟序列值（见 12.4），则其生成的 UUID 将仍然是唯一的。

注 3：对于无法访问广播时间信号的系统，如果夏令时发生变化或时钟序列值发生变化（见 12.4）的周期内不生成任何 UUID，则能使用记录本地时间的系统时钟并添加时差。

12.3.3 对于基于名称的 UUID 版本，应是由第 14 章规定的全球唯一名称来构成的一个 60 位的值。

注：全球唯一名称的示例包括 OID、URN 和目录可辨别名称（见参考文献〔1〕）。

12.3.4 对于基于随机数的 UUID 版本，这应是根据第 15 章规定的随机数或伪随机数生成的一个 60 位的值。

12.4 时钟序列字段

12.4.1 对于基于时间的 UUID 版本，时钟序列用来帮助避免时间值向后设置或者节点值变化时可能出现的重复。

注：名称“时钟序列”适用于基于时间的 UUID 版本，但也可用于基于名称的和基于随机数的 UUID 版本中对应值的内容。

12.4.2 如果时间值被向后设置，或可能已被向后设置（例如，当系统已关机断电），则 UUID 生成器不能确定是否已生成了比当前所设置的时间值更大的 UUID。在这种情况下，时钟序列值应予以更改。

注：如果知道时钟序列的先前值，则递增该值即可；否则，宜设置其为密码质量级随机数或伪随机数。

12.4.3 类似地，如果节点值发生变化（例如，由于在机器之间移动了网卡），时钟序列值应予以更改。

12.4.4 时钟序列应在初始阶段(即,在产生 UUID 系统的生命周期内仅一次)被初始化为非节点值导出的随机数。

注:这是为了使系统间的相关性减到最小,最大限度防止 MAC 地址在系统间快速移动或切换。

12.4.5 对于基于名称的 UUID 版本,时钟序列应是根据第 14 章中规定的名称构成的一个 14 位的值。

12.4.6 对于基于随机数的 UUID 版本,时钟序列应是根据第 15 章中规定的随机或伪随机生成的一个 14 位值。

12.5 节点字段

12.5.1 对于基于时间的 UUID 版本,节点值应由 MAC 地址(见 ISO/IEC 8802-3)组成,通常是某一网络接口的主机地址。

12.5.2 对于具有多 MAC 地址的系统,能使用除组播地址外任何可用地址。UUID 的八位字节 5(“节点”的第 1 个八位字节)应被设置为 MAC 地址的第一个八位字节,该 MAC 地址是由符合 ISO/IEC 8802-3 的系统来发送的。

注 1:该八位字节包含全球/本地位和单播/组播位。要求单播/组播位设置为单播,以避免与根据 12.5.3 规定所生成的地址相冲突。

注 2:能从 MAC 地址登记机构获得一批 MAC 地址(见参考文献[5])。

12.5.3 对于没有 MAC 地址的系统,可以使用密码质量级随机数或伪随机数(参见附录 C)。在这种地址中,应设置为组播。

注:这是为了确保生成的地址不与根据 12.5.2 规定的从网卡获取的地址相冲突。

12.5.4 对于基于名称的 UUID,节点值应是根据第 14 章中规定的通过正则化和散列化从全球唯一名称中构建的一个 48 位值。

12.5.5 对于基于随机数的 UUID,节点值应是根据第 15 章中规定的随机或伪随机生成的一个 48 位的值。

13 设置基于时间的 UUID 字段

基于时间的 UUID 字段应设置如下。

- 按照 12.3 和 12.4 的规定,确定 UUID 中的 UTC 时间和时钟序列值。
- 就此算法而言,将时间视为 60 位无符号整数,将时钟序列视为 14 位无符号整型。按顺序对每个值中的位进行编号,其中 0 表示最低有效位。
- 以相同的有效次序,将“TimeLow”字段设置为时间的最低有效 32 位(位 31 至位 0)。
- 以相同的有效次序,将“TimeMid”字段设置为时间的位 47 到位 32。
- 以相同的有效次序,将“VersionAndTimeHigh”字段的 12 个最低有效位(位 11 至位 0)设置为时间的位 59 至位 48。
- 将“VersionAndTimeHigh”字段的 4 个最高有效位(位 15 至位 12)设置为 12.2 中规定的 4 位版本号。
- 以相同的有效次序,将“ClockSeqLow”字段设置为时钟序列的 8 个最低有效位(位 7 至位 0)。
- 以相同的有效次序,将“VariantAndClockSeqHigh”字段的 6 个最低有效位(位 5 至位 0)设置为时钟序列的 6 个最高有效位(位 13 至位 8)。
- 分别将“VariantAndClockSeqHigh”字段的 2 个最高有效位(位 7 和位 6)设置为“1”和“0”。
- 以和地址相同的有效次序,将节点字段设置为 48 位 MAC 地址。

14 设置基于名称的 UUID 字段

本章规定了基于名称的 UUID 的产生规程。14.1 规定了所有散列函数的一般规程(也见 ISO/

IEC 10118-3)。14.2 规定了 MD5 的用途,14.3 规定了 SHA-1 的用途。基于名称的 UUID 宜具有的特性参见附录 B。

注: MD5 的使用局限于要求与现有 UUID 后向兼容性的情况,因为 SHA-1 提供了碰撞概率更小的散列算法,即不同散列材料中产生相同散列值的概率更低(参见 C.4)。

14.1 基于名称的 UUID 字段应设置如下。

——分配一个 UUID 作为“名称空间标识符”,该名称空间内的名称生成了的全部 UUID。

注: D.9 推荐 UUID 用于 4 个常用的名称空间。

——将名称转换成正则的八位字节序列(由其名称空间的标准或约定所定义的)。

——名称空间标识符与名称串联拼接后,使用 14.2 或 14.3 规定的散列函数来计算并得到 16 个八位字节散列值。按照 IETF RFC 1321(对于 MD5)和 FIPS PUB 180-3(对于 SHA-1)的规定,该散列值的八位字节的编号为 0~15。

——将“TimeLow”字段的八位字节 3~0 设置为该散列值的八位字节 3~0。

——将“TimeMid”字段的八位字节 1 和 0 设置为该散列值的八位字节 5 和 4。

——将“VersionAndTimeHigh”字段的八位字节 1 和 0 设置为散列值的八位字节 7 和 6。

——针对所用的散列函数,据 12.2 表 3 中的 4 位版本号来覆盖“VersionAndTimeHigh”字段的 4 个最高有效位(位 15~位 12)。

——将“VariantAndClockSeqHigh”字段设置为该散列值的八位字节 8。

——分别用“1”和“0”来覆盖“VariantAndClockSeqHigh”字段的 2 个最高有效位(位 7 和位 6)。

——将“ClockSeqLow”字段设置为该散列值的八位字节 9。

——将“Node”字段的八位字节 5~0 设置为该散列值的八位字节 15~10。

14.2 本条规定了使用 MD5 作为散列函数的基于名称的 UUID,但是 MD5 不应用于新生成的 UUID(参见 C.4)。对于 MD5 散列函数,在 14.1 中引用的“散列值”是通过 IETF RFC 1321 所规定的作为八位字节 0 到 15 的 16 个八位字节值。

注: MD5 这一规范及相关版本只是为了向后兼容早期版本规格。

14.3 本条规定了使用 SHA-1 作为散列函数的基于名称的 UUID。对于 SHA-1 散列函数,14.1 引用的“散列值”应是从由 FIPS PUB 180-3 所规定的 160 位报文摘要值中所获得的 20 个八位字节值中的八位字节 0~15。20 个八位字节值的八位字节 16~19 应丢弃。通过将 160 位值的最高有效位设置为 20 个八位字节值中的第 1 个八位字节(八位字节 0)的最高有效位,以及最低有效位设置为 20 个八位字节值中的最后一个八位字节(八位字节 19)的最低有效位,20 个八位字节值应从 FIPS PUB 180-3 的 160 位消息摘要值获得。

15 设置基于随机数的 UUID 字段

15.1 基于随机数的 UUID 字段应设置如下。

——分别将“VariantAndClockSeqHigh”字段的 2 个最高有效位(位 7 和位 6)设置为“1”和“0”。

——将“VersionAndTimeHigh”字段的 4 个最高有效位(位 15 到位 12)设置为 12.2 规定的 4 位版本号。

——将 UUID 的所有其他位设置为随机(或伪随机)生成的值。

注: 伪随机数可能多次产生相同的值。强烈推荐使用密码质量级随机数,以便降低重复值的概率。

15.2 附录 C 提供了关于在系统中如何生成随机数的指南。

附 录 A

(资料性)

有效生成基于时间的 UUID 的算法

本附录描述了一种能用于在计算机系统中重复生成基于时间的 UUID 的算法。

A.1 基本算法

A.1.1 下列算法是简单的、正确的,但是低效率的。

- 获得系统范围的全局锁。
- 从系统范围共享的稳定存储器(例如,文件)中,读取 UUID 生成程序的状态:用来生成最近一个 UUID 的时间、时钟序列和节点的值。
- 获得自 1582 年 10 月 15 日 00:00:00.00 以来的按 100 ns 为间隔的时间计数,此时间用 60 位表示而作为时间值。
- 获得当前节点值。
- 如果状态是不可用的(例如,不存在或被损坏),或者保存的节点值不同于当前节点值,则生成随机的时钟序列值。
- 如果状态是可用的,但保存的时间值迟于当前时间值,则递增时钟序列值。
- 将状态(当前时间、时钟序列和节点值)保存回稳定存储器中。
- 释放全局锁。
- 按照第 13 章中的步骤,由当前时间、时钟序列和节点值构建标准格式的 UUID。

A.1.2 如果不需要频繁地生成 UUID,上述算法可能就足够了。然而,如果对性能有更高要求,该基本算法的问题包括:

- 每次从稳定存储中读取状态是低效率的;
- 系统时钟的精度可能不是 100 ns;
- 每次将状态写入稳定存储器是低效率的;
- 共享跨越进程边界的状态可能是低效率的。

A.1.3 上述各问题能通过对读写状态和读取时钟的功能的局部改进、以模块化的方式解决。这将在下列各条中依次论述。

A.2 读稳定存储器

A.2.1 如果该状态被读入系统范围的共享易失性存储器(并且每当稳定存储器更新时就更新),则该状态只需要在引导时从稳定存储器读取一次。

A.2.2 如果某一实现没有任何稳定存储器可用,那么它能一直反馈值不可用。这是最不理想的实现,因为这将增加创建新的时钟序列数据的频度,从而增加重复的可能性。

A.2.3 如果节点值绝不可能改变(例如,网卡与系统不可分),或者如果任何变化会使时钟序列重新初始化为随机值,则可直接返回当前节点值,而不需要将其保存在稳定存储器中。

A.3 系统时钟精度

A.3.1 时间值由系统时间生成,其精度可低于要求的时间精度。

A.3.2 如果 UUID 不需要频繁地被生成,则时间能简化为系统时间乘以每系统时间片内的 100 ns 的计数值。

A.3.3 如果系统在单个时间间隔内请求太多 UUID,而超过了生成程序的正常范围,则 UUID 服务宜

返回一个差错或停止 UUID 生成程序、直到系统时钟赶上。

A.3.4 高精度时间值能通过记录用同一系统时间值生成了多少个 UUID,并借此构造时间值的低阶位来模拟。计数范围将在 0 和每系统时间片内 100 ns 计数值之间。

注：如果处理器经常请求生成过量 UUID,则能为系统分配额外的 MAC 地址,这将使得每个时间值内提供更多可用的 UUID,从而实现更高速度的分配。

A.4 写稳定存储器

在每次生成 UUID 时,该状态不总是需要被写入稳定存储器。在稳定存储器内的时间值能定期设置为大于 UUID 中所使用的时间值。只要生成的 UUID 的时间值小于该值,并且时钟序列和节点的值保持不变,就只需要更新共享的易失存储器内拷贝。此外,如果在稳定存储器内的时间值将来小于系统重新启动所需的典型时间,这样即使系统崩溃都不会引发时钟序列的重置。

A.5 共享跨越进程的状态

如果每次生成 UUID 时访问共享的开销太大,则能实现系统范围生成程序,该程序支持单次调用即分配一批时间值,这样每个进程的生成程序可以直接从中分配,直到用完为止。

附录 B
(资料性)
基于名称的 UUID 的特性

B.1 基于名称的 UUID 意指自某名称空间导出的名称(该名称是唯一的)产生的 UUID。名称和名称空间的概念宜按广义地解释,而不局限于字面名字。从名称空间中分配名称并确保其唯一性的机制或约定不属于本文件内容。

注:为了避免递归问题,基于名称的 UUID 不宜从以基于名称的 UUID 结尾的 OID 中来生成。

B.2 按照第 14 章以合适地选择的名称空间而生成基于名称的 UUID 的特性如下:

- 同一名称空间中的同一名称在不同时间生成的 UUID 将是相等的;
- 同一名称空间中的两个不同的名称生成的 UUID,其很大概率是不同的;
- 两个不同的名称空间中的相同名称中而生成的 UUID,其很大概率是不同的;
- 如果两个基于名称的 UUID 是相等的,则这两 UUID 很大概率是从同一名称空间中的同一名称来生成的。

附 录 C

(资料性)

在系统中随机数的生成

C.1 如果系统不具备生成密码质量级随机数的能力,则在大多数系统中通常有相当多随机数来源能供生成。这些来源因系统而异,但通常包括:

- 已使用存储器的百分比;
- 以字节表示的主存储器的容量;
- 以字节表示的空闲主存储器的总量;
- 以字节表示的分页或交换文件的大小;
- 分页或对换文件的空闲字节数;
- 以字节表示的用户虚拟地址空间的大小;
- 以字节表示的可用用户地址空间的大小;
- 以字节表示的启动盘驱动器的大小;
- 以字节表示的启动驱动器上空闲磁盘空间的大小;
- 当前时间;
- 系统当次启动以来的时间;
- 在各种系统目录中文件的大小;
- 在各种系统目录中文件的创建,最后读取和修改的时间;
- 各种系统资源(堆,等等)的利用率;
- 当前鼠标光标位置;
- 当前插入记号位置;
- 当前正在运行进程、线程数量;
- 桌面视窗和活动视窗的句柄或 ID;
- 调用程序的堆栈指针值;
- 调用程序的进程和线程标识符;
- 各种处理器体系结构特定的性能计数器(已执行的指令数,高速缓存未命中数,页表缓存命中数或者是未命中数等)。

C.2 另外,诸如当前的计算机名称和操作系统名称,虽然严格而论并非随机的,但仍将有助于区分本机和其他系统。

C.3 使用上述数据生成节点值的确切算法因系统而异,因为这些数据和获取数据的函数通常因系统而异。然而,一种通用的方法是将上述尽可能多的数据源累积到同一个数据区中,并使用诸如 SHA-1 消息摘要算法,再从散列值中提取 6 个八位字节,且按之前所述设置组播位。

C.4 还能使用其他散列函数,如 MD5 和 ISO/IEC 10118 中指定的散列函数。唯一的要求是,其运算结果是适度随机的,即满足输入是均匀分布、输出也是均匀分布,并且输入中的单个位变化能导致输出一半的位都会改变(然而,不推荐使用 MD5 散列函数生成新的 UUID,因为最近的研究表明其输出值不是均匀分布的)。

附 录 D
(资料性)
实例实现

D.1 所提供的文件

本实现由 6 个文件组成,这 6 个文件为 copyrt.h,uuid.h,uuid.c,sysdep.h,sysdep.c 和 utest.c。其中 uuid.h 和 uuid.c 是按第 13 章、第 14 章和第 15 章中所描述的与系统无关的 UUID 生成算法的实现,同时包括了在附录 A 中所描述的所有优化算法(除 A.5 共享跨越进程的状态外)。本实现假定支持 64 位整数,这使代码更清晰。

注:已在 Linux(Red Hat 4.0)上用 GCC(2.7.2)和 Windows NT 4.0 上用 VC++5.0 测试了该代码。

D.2 copyrt.h 文件

下列所有源文件宜被视为包含以下版权声明:

```
/*
 * * Copyright (c) 1990- 1993, 1996 Open Software Foundation, Inc.
 * * Copyright (c) 1989 by Hewlett-Packard Company, Palo Alto, Ca. &
 * * Digital Equipment Corporation, Maynard, Mass.
 * * Copyright (c) 1998 Microsoft.
 * * To anyone who acknowledges that this file is provided "AS IS"
 * * without any express or implied warranty: permission to use, copy,
 * * modify, and distribute this file for any purpose is hereby
 * * granted without fee, provided that the above copyright notices and
 * * this notice appears in all source code copies, and that none of
 * * the names of Open Software Foundation, Inc., Hewlett-Packard
 * * Company, or Digital Equipment Corporation be used in advertising
 * * or publicity pertaining to distribution of the software without
 * * specific, written prior permission. Neither Open Software
 * * Foundation, Inc., Hewlett-Packard Company, Microsoft, nor Digital
 * * Equipment Corporation makes any representations about the
 * * suitability
 * * of this software for any purpose.
 */
```

D.3 uuid.h 文件

```
#include "copyrt.h"
#undef uuid_t
typedef struct {
    unsigned32  time_low;
    unsigned16  time_mid;
    unsigned16  time_hi_and_version;
```

```

    unsigned8    clock_seq_hi_and_reserved;
    unsigned8    clock_seq_low;
    byte         node[6];
} uuid_t;

/* uuid_create -- generate a UUID */
int uuid_create(uuid_t * uuid);

/* uuid_create_from_name -- create a UUID using a "name"
   from a "name space" */
void uuid_create_from_name(
    uuid_t * uuid,          /* resulting UUID */
    uuid_t nsid,           /* UUID of the namespace */
    void * name,           /* the name from which to generate a UUID */
    int namelen            /* the length of the name */
);

/* uuid_compare -- Compare two UUIDs "lexically" and return
   -1      u1 is lexically before u2
   0      u1 is equal to u2
   1      u1 is lexically after u2
   Note that lexical ordering is not temporal ordering!
*/
int uuid_compare(uuid_t * u1, uuid_t * u2);

```

D.4 uuid.c 文件

```

#include "copyrt.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "sysdep.h"
#ifdef _WINDOWS_
    #include <arpa/inet.h>
#endif
#include "uuid.h"

/* various forward declarations */
static int read_state(unsigned16 * clockseq, uuid_time_t * timestamp,
    uuid_node_t * node);
static void write_state(unsigned16 clockseq, uuid_time_t timestamp,
    uuid_node_t node);

```

```

static void format_uuid_v1(uuid_t * uuid, unsigned16 clockseq,
    uuid_time_t timestamp, uuid_node_t node);
static void format_uuid_v3(uuid_t * uuid, unsigned char hash[16]);
static void get_current_time(uuid_time_t * timestamp);
static unsigned16 true_random(void);

/* uuid_create -- generator a UUID */
int uuid_create(uuid_t * uuid)
{
    uuid_time_t timestamp, last_time;
    unsigned16 clockseq;
    uuid_node_t node;
    uuid_node_t last_node;
    int f;

    /* acquire system-wide lock so we're alone */
    LOCK;

    /* get time, node identifier, saved state from non-volatile storage */
    get_current_time(&timestamp);
    get_ieee_node_identifier(&node);
    f = read_state(&clockseq, &last_time, &last_node);

    /* if no NV state, or if clock went backwards, or node identifier
       changed (e.g., new network card) change clockseq */
    if (! f || memcmp(&node, &last_node, sizeof node))
        clockseq = true_random();
    else if (timestamp < last_time)
        clockseq++;

    /* save the state for next time */
    write_state(clockseq, timestamp, node);

    UNLOCK;

    /* stuff fields into the UUID */
    format_uuid_v1(uuid, clockseq, timestamp, node);
    return 1;
}

/* format_uuid_v1 -- make a UUID from the timestamp, clockseq,
    and node identifier */
void format_uuid_v1(uuid_t * uuid, unsigned16 clock_seq,

```

```

        uuid_time_t timestamp, uuid_node_t node)
{
    /* Construct a version 1 uuid with the information we've gathered
       plus a few constants. */
    uuid->time_low = (unsigned long)(timestamp & 0xFFFFFFFF);
    uuid->time_mid = (unsigned short)((timestamp >> 32) & 0xFFFF);
    uuid->time_hi_and_version =
        (unsigned short)((timestamp >> 48) & 0x0FFF);
    uuid->time_hi_and_version |= (1 << 12);
    uuid->clock_seq_low = clock_seq & 0xFF;
    uuid->clock_seq_hi_and_reserved = (clock_seq & 0x3F00) >> 8;
    uuid->clock_seq_hi_and_reserved |= 0x80;
    memcpy(&uuid->node, &node, sizeof uuid->node);
}

/* data type for UUID generator persistent state */
typedef struct {
    uuid_time_t ts;                /* saved timestamp */
    uuid_node_t node;              /* saved node identifier */
    unsigned16 cs;                 /* saved Clock Sequence */
} uuid_state;

static uuid_state st;

/* read_state -- read UUID generator state from non-volatile store */
int read_state(unsigned16 * clockseq, uuid_time_t * timestamp,
               uuid_node_t * node)
{
    static int initd = 0;
    FILE * fp;

    /* only need to read state once per boot */
    if (! initd) {
        fp = fopen("state", "rb");
        if (fp == NULL)
            return 0;
        fread(&st, sizeof st, 1, fp);
        fclose(fp);
        initd = 1;
    }
    * clockseq = st.cs;
    * timestamp = st.ts;
    * node = st.node;
}

```



```

    return 1;
}

/* write_state -- save UUID generator state back to non-volatile
   storage */
void write_state(unsigned16 clockseq, uuid_time_t timestamp,
                 uuid_node_t node)
{
    static int inited = 0;
    static uuid_time_t next_save;
    FILE * fp;

    if (! inited) {
        next_save = timestamp;
        inited = 1;
    }

    /* always save state to volatile shared state */
    st.cs = clockseq;
    st.ts = timestamp;
    st.node = node;
    if (timestamp >= next_save) {
        fp = fopen("state", "wb");
        fwrite(&st, sizeof st, 1, fp);
        fclose(fp);
        /* schedule next save for 10 seconds from now */
        next_save = timestamp + (10 * 10 * 1000 * 1000);
    }
}

/* get_current_time -- get time as 60-bit 100ns ticks since UUID epoch.
   Compensate for the fact that real clock resolution is
   less than 100ns. */
void get_current_time(uuid_time_t * timestamp)
{
    static int inited = 0;
    static uuid_time_t time_last;
    static unsigned16 uuids_this_tick;
    uuid_time_t time_now;

    if (! inited) {
        get_system_time(&time_now);
        uuids_this_tick = UUIDS_PER_TICK;
    }

```

```

    initd = 1;
}

for ( ; ; ) {
    get_system_time(&time_now);

    /* if clock reading changed since last UUID generated, */
    if (time_last != time_now) {
        /* reset count of uuids gen'd with this clock reading */
        uuids_this_tick = 0;
        time_last = time_now;
        break;
    }
    if (uuids_this_tick < UUIDS_PER_TICK) {
        uuids_this_tick++;
        break;
    }
    /* going too fast for our clock; spin */
}
/* add the count of uuids to low order bits of the clock reading */
*timestamp = time_now + uuids_this_tick;
}

/* true_random -- generate a crypto-quality random number.
 * * This sample doesn't do that. * * */
static unsigned16 true_random(void)
{
    static int initd = 0;
    uuid_time_t time_now;

    if (! initd) {
        get_system_time(&time_now);
        time_now = time_now / UUIDS_PER_TICK;
        srand((unsigned int)(((time_now >> 32) ^ time_now) & 0xffffffff));
        initd = 1;
    }

    return rand();
}

/* uuid_create_from_name -- create a "name" from a "name
space" */
void uuid_create_from_name(uuid_t * uuid, uuid_t nsid, void * name,

```

```

                                int namelen)
{
    MD5_CTX c;

    unsigned char hash[16];
    uuid_t net_nsid;

    /* put name space identifier in network byte order so it hashes the
       same no matter what endian machine we're on */
    net_nsid = nsid;
                                htonl(net_nsid.time_low);
                                htons(net_nsid.time_mid);
                                htons(net_nsid.time_hi_and_version);

    MD5Init(&c);
    MD5Update(&c, &net_nsid, sizeof net_nsid);
    MD5Update(&c, name, namelen);
    MD5Final(hash, &c);

    /* the hash is in network byte order at this point */
    format_uuid_v3(uuid, hash);
}

/* format_uuid_v3 -- make a UUID from a (pseudo)random 128-bit number */
void format_uuid_v3(uuid_t * uuid, unsigned char hash[16])
{
    /* convert UUID to local byte order */
    memcpy(uuid, hash, sizeof * uuid);
                                htonl(uuid->time_low);
                                htons(uuid->time_mid);
                                htons(uuid->time_hi_and_version);

    /* put in the variant and version bits */
    uuid->time_hi_and_version &= 0x0FFF;
    uuid->time_hi_and_version |= (3 << 12);
    uuid->clock_seq_hi_and_reserved &= 0x3F;
    uuid->clock_seq_hi_and_reserved |= 0x80;
}

/* uuid_compare -- Compare two UUIDs "lexically" and return */
#define CHECK(f1, f2) if (f1 != f2) return f1 < f2 ? -1 : 1;
int uuid_compare(uuid_t * u1, uuid_t * u2)

```

```

{
    int i;

    CHECK(u1->time_low, u2->time_low);
    CHECK(u1->time_mid, u2->time_mid);
    CHECK(u1->time_hi_and_version, u2->time_hi_and_version);
    CHECK(u1->clock_seq_hi_and_reserved, u2->clock_seq_hi_and_reserved);
    CHECK(u1->clock_seq_low, u2->clock_seq_low)
    for (i = 0; i < 6; i++) {
        if (u1->node[i] < u2->node[i])
            return -1;
        if (u1->node[i] > u2->node[i])
            return 1;
    }
    return 0;
}
# undef CHECK

```

D.5 sysdep.h 文件

```

#include "copyrt.h"
/* remove the following define if you aren't running WIN32 */
#define WININC 0

#ifdef WININC
#include <windows.h>
#else
#include <sys/time.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/times.h>
#endif

#include "global.h"
/* change to point to where MD5 .h's live; RFC 1321 has sample
   implementation */
#include "md5.h"

/* set the following to the number of 100ns ticks of the actual
   resolution of your system's clock */
#define UUIDS_PER_TICK 1024

/* Set the following to a calls to get and release a global lock */

```

```

#define LOCK
#define UNLOCK

typedef unsigned long unsigned32;
typedef unsigned short unsigned16;
typedef unsigned char unsigned8;
typedef unsigned char byte;

/* Set this to what your compiler uses for 64-bit data type */
#ifdef WININC
#define unsigned64_t unsigned_int64
#define I64(C) C
#else
#define unsigned64_t unsigned long long
#define I64(C) C##LL
#endif

typedef unsigned64_t uuid_time_t;
typedef struct {
    char nodeID[6];
} uuid_node_t;

void get_ieee_node_identifier(uuid_node_t * node);
void get_system_time(uuid_time_t * uuid_time);
void get_random_info(char seed[16]);

```

D.6 sysdep.c 文件

```

#include "copyrt.h"
#include <stdio.h>
#include <string.h>
#include "sysdep.h"

/* system dependent call to get MAC node identifier.
   This sample implementation generates a random node identifier. */
void get_ieee_node_identifier(uuid_node_t * node)
{
    static int inited = 0;
    static uuid_node_t saved_node;
    unsigned char seed[16];
    FILE * fp;

    if (! inited) {

```

```

    fp = fopen("nodeid", "rb");
    if (fp) {
        fread(&saved_node, sizeof saved_node, 1, fp);
        fclose(fp);
    }
    else {
        get_random_info(seed);
        seed[0] |= 0x80;
        memcpy(&saved_node, seed, sizeof saved_node);
        fp = fopen("nodeid", "wb");
        if (fp) {
            fwrite(&saved_node, sizeof saved_node, 1, fp);
            fclose(fp);
        }
    }

    initied = 1;
}

* node = saved_node;
}

/* system dependent call to get the current system time. Returned as
   100 ns ticks since UUID epoch, but resolution may be less than 100 ns. */
#ifdef _WINDOWS_

void get_system_time(uuid_time_t * uuid_time)
{
    ULARGE_INTEGER time;

    /* Windows NT keeps time in FILETIME format which is 100ns ticks since
       Jan 1, 1601. UUIDs use time in 100ns ticks since Oct 15, 1582.
       The difference is 17 Days in Oct + 30 (Nov) + 31 (Dec)
       + 18 years and 5 leap days. */
    GetSystemTimeAsFileTime((FILETIME *) &time);
    time.QuadPart +=
        (unsigned_int64) (1000 * 1000 * 10)           // seconds
        * (unsigned_int64) (60 * 60 * 24)             // days
        * (unsigned_int64) (17+30+31+365 * 18+5);      // # of days
    * uuid_time = time.QuadPart;
}

void get_random_info(char seed[16])

```

```

{
    MD5_CTX c;
    struct {
        MEMORYSTATUS m;
        SYSTEM_INFO s;
        FILETIME t;
        LARGE_INTEGER pc;
        DWORD tc;
        DWORD l;
        char hostname[MAX_COMPUTERNAME_LENGTH + 1];
    } r;

    MD5Init(&c);
    GlobalMemoryStatus(&r.m);
    GetSystemInfo(&r.s);
    GetSystemTimeAsFileTime(&r.t);
    QueryPerformanceCounter(&r.pc);
    r.tc = GetTickCount();
    r.l = MAX_COMPUTERNAME_LENGTH + 1;
    GetComputerName(r.hostname, &r.l);
    MD5Update(&c, &r, sizeof r);
    MD5Final(seed, &c);
}

# else

void get_system_time(uuid_time_t * uuid_time)
{
    struct timeval tp;

    gettimeofday(&tp, (struct timezone *)0);

    /* Offset between UUID formatted times and Unix formatted times.
       UUID UTC base time is October 15, 1582.
       Unix base time is January 1, 1970. */
    * uuid_time = (tp.tv_sec * 10000000) + (tp.tv_usec * 10)
        + I64(0x01B21DD213814000);
}

void get_random_info(char seed[16])
{
    MD5_CTX c;
    struct {

```

```

    struct timeval t;
    char hostname[257];
} r;

MD5Init(&c);
gettimeofday(&r.t, (struct timezone *)0);
gethostname(r.hostname, 256);
MD5Update(&c, &r, sizeof r);
MD5Final(seed, &c);
}

#endif

```

D.7 utest.c 文件

```

#include "copyrt.h"
#include "sysdep.h"
#include <stdio.h>
#include "uuid.h"

uuid_t NameSpace_DNS = { /* 6ba7b810-9dad-11d1-80b4-00c04fd430c8 */
    0x6ba7b810,
    0x9dad,
    0x11d1,
    0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
};

/* puid -- print a UUID */
void puid(uuid_t u)
{
    int i;

    printf("%08x-%04x-%04x-%02x%02x-", u.time_low, u.time_mid,
        u.time_hi_and_version, u.clock_seq_hi_and_reserved,
        u.clock_seq_low);
    for (i = 0; i < 6; i++)
        printf("%02x", u.node[i]);
    printf("\n");
}

/* Simple driver for UUID generator */
void main(int argc, char * * argv)
{
    uuid_t u;

```



```

int f;

uuid_create(&u);
printf("uuid_create(): "); puid(u);

f = uuid_compare(&u, &u);
printf("uuid_compare(u,u): %d\n", f); /* should be 0 */
f = uuid_compare(&u, &NameSpace_DNS);
printf("uuid_compare(u, NameSpace_DNS): %d\n", f); /* s.b. 1 */
f = uuid_compare(&NameSpace_DNS, &u);
printf("uuid_compare(NameSpace_DNS, u): %d\n", f); /* s.b. -1 */
uuid_create_from_name(&u, NameSpace_DNS, "www.widgets.com", 15);
printf("uuid_create_from_name(): "); puid(u);
}

```

D.8 utest 的示例输出

```

uuid_create(): 7d444840-9dc0-11d1-b245-5ffdce74fad2
uuid_compare(u,u): 0
uuid_compare(u, NameSpace_DNS): 1
uuid_compare(NameSpace_DNS, u): -1
uuid_create_from_name(): e902893a-9d22-3c7e-a7b8-d6e313b71d9f

```

D.9 名称空间 ID 样例

本节列出了一些可能感兴趣的名称空间所对应的名称空间标识,作为用 C 语言和上述定义的字符串形式的初始化结构。

```

/* Name string is a fully-qualified domain name */
uuid_t NameSpace_DNS = { /* 6ba7b810-9dad-11d1-80b4-00c04fd430c8 */
0x6ba7b810,
0x9dad,
0x11d1,
0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
};

/* Name string is a URL */
uuid_t NameSpace_URL = { /* 6ba7b811-9dad-11d1-80b4-00c04fd430c8 */
0x6ba7b811,
0x9dad,
0x11d1,
0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
};

/* Name string is an OID */

```

```
uuid_t NameSpace_OID = { /* 6ba7b812-9dad-11d1-80b4-00c04fd430c8 */  
0x6ba7b812,  
0x9dad,  
0x11d1,  
0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8  
};
```

```
/* Name string is a Directory distinguished name (in DER or a text output format) */  
uuid_t NameSpace_X500 = { /* 6ba7b814-9dad-11d1-80b4-00c04fd430c8 */  
0x6ba7b814,  
0x9dad,  
0x11d1,  
0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8  
};
```

参 考 文 献

[1] Recommendation ITU-T X.500 (2012) | ISO/IEC 9594-1:2012 Information technology—Open Systems Interconnection—The Directory: Overview of concepts, models and services

[2] ISO/IEC 10118-3 Information technology—Security techniques—Hash functions—Part 3: Dedicated hash-functions

[3] ISO/IEC 10646 Information technology—Universal Coded Character Set (UCS)

[4] ISO/IEC 11578:1996 Information technology—Open Systems Interconnection—Remote Procedure Call (RPC)

[5] IEEE, Request Form for an Individual Address Block (also known as an Ethernet Address Block) of 4,096 MAC Addresses. <http://standards.ieee.org/develop/regauth/iab/>

[6] IETF RFC 2141 URN Syntax

[7] IETF RFC 3061 (2001) A URN Namespace of Object Identifiers

[8] IETF RFC 4122 (2005) A Universally Unique Identifier (UUID) URN Namespace

[9] Open Group CAE; DCE; Remote Procedure Call, Specification C309, ISBN 1-85912-041-5, August 1994

[10] Zahn, L., Dineen, T., Leach, P. (January 1990), Network Computing Architecture, ISBN 0-13-611674-4

中 华 人 民 共 和 国
国 家 标 准

信息技术 对象标识符登记机构操作规程
第 8 部分：通用唯一标识符(UUIDs)
的生成及其在对象标识符中的使用
GB/T 17969.8—2024/ISO/IEC 9834-8:2014

*

中国标准出版社出版发行
北京市朝阳区和平里西街甲 2 号(100029)
北京市西城区三里河北街 16 号(100045)

网址:www.spc.net.cn

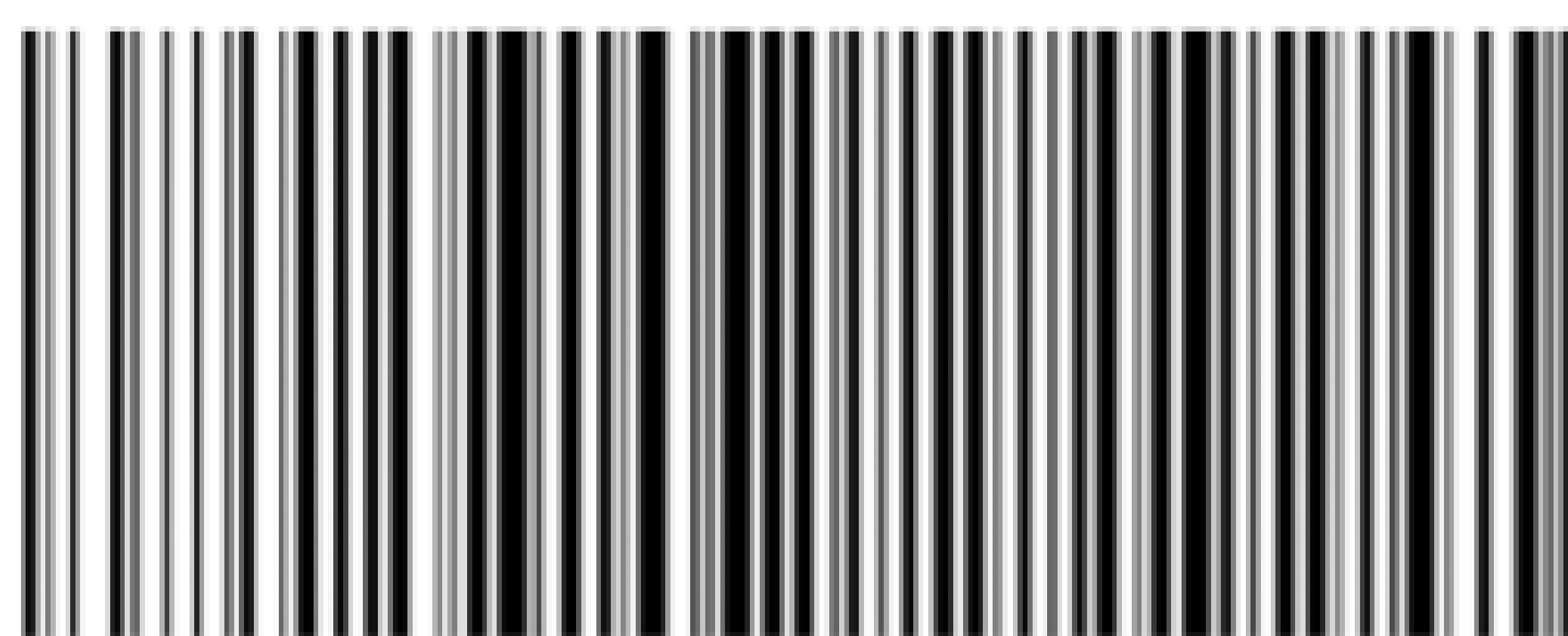
服务热线:400-168-0010

2024 年 5 月第一版

*

书号:155066·1-76463

版权专有 侵权必究



GB/T 17969.8-2024

www.bzxz.net

免费标准下载网