提升用户体验的UUID设计策略

南城 南城大前端 2024年04月19日 19:14 广东

> 本文翻译自 The UX of UUIDs，作者：Andreas Thomas， 略有删改。

唯一标识符在从用户身份验证到资源管理的所有应用程序中起着至关重要的作用。虽然使用标准UUID将满足您的所有安全问题，但我们可以为用户改进很多。

这篇文章讨论了唯一标识符（UUID）在应用程序中的重要性，以及如何通过一些改进来增强用户体验。

确保唯一性

唯一标识符对于区分系统中的各个实体至关重要。它们提供了一种可靠的方式，确保每个项目、用户或数据片段都有一个独特的身份。通过维持唯一性，应用程序可以有效地管理和组织信息，使操作更加高效，并促进数据的完整性。

任何安全生成的128位UUID对我们来说都足够了，有很多库可以生成UUID，或者你可以使用你选择的编程语言的标准库。在这篇博客中，我将使用Typescript示例，但基本思想适用于任何语言。

```
const id = crypto.randomUUID()
// '5727a4a4-9bba-41ae-b7fe-e69cf60bb0ab'
```

我们也可以止步于此，但我们做的更好，通过一些小的改进来增强用户体验：

- 使它们易于复制
- 添加前缀
- 更高效的编码
- 改变长度

复制UUID很麻烦

尝试通过双击它来复制以下UUID：

`c6b10dd3-1dcf-416c-8ed8-ae561807fcaf`

如果你幸运的话，你能复制整个UUID，但对大多数人来说，他们只得到了一个部分，提高唯一标识符可用性的一种方法是使它们易于复制。

这可以通过从UUID中移除连字符来实现，允许用户简单地双击标识符来复制。通过这个小改变可以大大改善用户在处理标识符时的体验。

在代码中移除连字符很简单，以下是如何在js/ts中做到这一点：

```
const id = crypto.randomUUID().replace(/-/g, "")
// fe4723eab07f408384a2c0f051696083
```

现在再试试复制它，感觉好多了！

添加前缀

我们可以通过添加一个有意义的前缀来帮助用户区分不同环境或系统中的资源。例如，Stripe使用像 `sk_live_` 这样的前缀来表示生产环境的密钥，或者使用 `cus_` 来表示客户标识符。通过增加这样的前缀，我们可以确保清晰度并减少混淆的可能性，特别是在多个环境共存的复杂系统中。

```
const id = `hello_${crypto.randomUUID().replace(/-/g, "")}`
// hello_1559debea64142f3b2d29f8b0f126041
```

命名前缀就像命名变量一样是一门艺术。你想要有描述性，但尽可能短。我稍后会分享我们的命名策略。

使用base58编码

除了使用十六进制表示标识符，我们还可以考虑更高效的编码方式，比如 `base58` 。Base58编码使用更大的字符集并避免使用模糊的字符，例如大写字母I和小写字母l，从而在不牺牲可读性的情况下生成更短的标识符字符串。

例如一个8个字符长的base58字符串，可以存储大约30,000倍于8个字符十六进制字符串的状态。在16个字符时，base58字符串可以存储889,054,070种组合。

你很可能仍然选择使用你语言的标准库来做到这一点，但你也可以使用像 `nanoid` 这样的库，它适用于大多数语言。

```
import { customAlphabet } from "nanoid";

export const nanoid = customAlphabet("123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrs

const id = `prefix_${nanoid(22)}`
// prefix_KSPKGySWPqJWWWa37RqGaX
```

我们在这里生成了一个22个字符长的ID，它可以编码大约是UUID的100倍的数量，同时比UUID短10个字符。总数越多，说明可能重复的情况就越少。

-	字符	长度	总数
UUID	16	32	2^122 = 5.3e+36
Base58	58	22	58^22 = 6.2e+38

改变长度

并非所有标识符都需要有高水平的碰撞抵抗力。在某些情况下，较短的标识符就足够了，这取决于应用程序的具体要求。通过减少标识符的长度，我们可以生成更短的ID，同时仍然保持可接受的唯一性水平。

减少ID的长度可能是好的，但你需要小心，并确保你的系统能够抵御ID冲突。幸运的是这一点可以在数据库层很容易做到，在我们的MySQL数据库中，我们主要将ID用作主键，数据库保护我们免受冲突。如果一个ID已经存在，我们只需生成一个新的并重试。如果我们的冲突率显著上升，我们可以简单地增加所有未来ID的长度。

长度	示例	总状态
nanoid(8)	re6ZkUUV	1.3e+14
nanoid(12)	pfpPYdZGbZvw	1.4e+21
nanoid(16)	sFDUZScHfZTfkLwk	1.6e+28
nanoid(24)	u7vzXJL9cGqUeabGPAZ5XUJ	2.1e+42
nanoid(32)	qkvPDeH6JyAsRhaZ3X4ZLDPSLFP7MnJz	2.7e+56

实际使用

最后我想分享一下我们在这里的实现以及我们如何在代码库中使用它。我们使用一个简单的函数，它接受一个类型化的前缀，然后为我们生成ID。这样我们可以确保我们总是对同类型的ID使用相同的前缀。当你的系统中有多种类型的ID时，这特别有用。

```
import { customAlphabet } from "nanoid";

export const nanoid = customAlphabet("123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrs

const prefixes = {
  key: "key",
  api: "api",
  policy: "pol",
  request: "req",
  workspace: "ws",
  keyAuth: "key_auth", // <-- 这是内部的，不需要短或漂亮
  vercelBinding: "vb",
  test: "test", // <-- 仅用于测试
} as const;

export function newId(prefix: keyof typeof prefixes): string {
  return [prefixes[prefix], nanoid(16)].join("_");
}
```

当我们在代码库中使用它时，我们可以确保总是对正确类型的ID使用正确的前缀。

```
import { newId } from "@unkey/id";

const id = newId("workspace")
// ws_dYuyGV3qMKvebjML

const id = newId("keyy")
// 无效，因为`keyy`不是有效的前缀名称
```

最后

本文讨论了在应用程序中使用唯一标识符（UUID）时通过以下方式提升用户体验。

1. **简化复制**：通过去除UUID中的连字符，可以让用户更容易地通过双击复制整个标识符。
2. **添加前缀**：使用有意义的前缀可以帮助用户区分不同环境或资源，例如Stripe使用特定的前

缀来区分生产环境密钥和客户标识符。

3. **高效编码**：考虑使用如base58这样的编码方式，它使用更大的字符集并避免使用易混淆的字符，从而生成更短且可读性强的标识符字符串。

4. **改变长度**：根据应用的具体需求，可以通过减少标识符的长度来生成更短的ID，同时保持足够的唯一性。

5. **数据库层的保护**：在数据库层面，可以使用主键约束来防止ID冲突，如果冲突率上升，可以通过增加未来ID的长度来解决。

通过实施这些改进，我们可以增强应用程序中唯一标识符的可用性和效率。这将为用户和开发者提供更好的体验，因为他们与系统中的各种实体进行交互和管理。无论是轻松复制标识符、区分不同环境，还是实现更短、更易读的标识符字符串，这些策略都可以促进一个更用户友好和健壮的识别系统。

看完本文如果觉得有用，记得点个赞支持，收藏起来说不定哪天就用上啦～

专注前端开发，分享前端相关技术干货，公众号：南城大前端（ID: nanchengfe）

关注公众号👇👇👇