2.4.4 另外一点说明

你可能会感到奇怪，为什么使用一些不安全字符的时候并没有发生什么不好的事情。比如，你可以访问 http://www.joes-hardware.com/~joe 上的 Joe 主页，而无需对“~”字符进行编码。对某些传输协议来说，这并不是什么问题，但对应用程序开发人员来说，对非安全字符进行编码仍然是明智的。

应用程序要按照一定规范工作。客户端应用程序在向其他应用程序发送任意 URL 之前，最好把所有不安全或受限字符都进行转换[6]。只要对所有不安全字符都进行了编码，这个 URL 就是可在各应用程序之间共享的*规范形式*；也就无需操心其他应用程序会被字符的任何特殊含义所迷惑了。

最适合判断是否需要对字符进行编码的程序就是从用户处获取 URL 的源端应用程序。URL 的每个组件都会有自己的安全 / 不安全字符，哪些字符是安全 / 不安全的与方案有关，因此只有从用户那里接收 URL 的应用程序才能够判断需要对哪些字符进行编码。

当然，另一种极端的做法就是应用程序对所有字符都进行编码。尽管并不建议这么做，但也没有什么强硬且严格的规则规定不能对那些安全字符进行编码；但在实际应用中，有些应用程序可能会假定不对安全字符进行编码，这么做的话可能会产生一些奇怪的破坏性行为。

有时，有些人会恶意地对额外的字符进行编码，以绕过那些对 URL 进行模式匹配的应用程序——比如，Web 过滤程序。对安全的 URL 组件进行编码会使模式匹配程序无法识别出它们所要搜寻的模式。总之，解释 URL 的应用程序必须在处理 URL
37 之前对其进行解码。

有些 URL 组件要便于识别，并且必须由字母开头，比如 URL 的方案。更多关于不同 URL 组件中保留字符和不安全字符的使用指南请回顾 2.2 节[7]。

2.5 方案的世界

本节将介绍更多 Web 常用方案格式。附录 A 给出了一个相当完整的方案列表，及各种方案文档的参考文献。

注 6：这里我们特指的是客户端应用程序，而不是其他的 HTTP 中间点，比如代理。6.5.5 节探讨了代理和其他中间 HTTP 应用程序试图代表客户端修改（比如编码）URL 时可能产生的一些问题。

注 7：表 2-3 列出了各种 URL 组件的保留字符。总之，只应该对这些在传输过程中不安全的字符进行编码。