第2章

URL与资源

我们可以把因特网当作一个巨大的正在扩张的城市，里面充满了各种可看的东西，可做的事情。你和其他居民，以及到这个正在蓬勃发展的社区旅游的游客都要为这个城市大量的景点和服务使用标准命名规范。博物馆、饭店和家庭住址要使用街道地址，消防局、老板的秘书，以及抱怨你太少打电话给她的母亲要使用电话号码。

所有的东西都有一个标准化的名字，以帮助人们寻找城市中的各种资源。书籍有ISBN 号，公交车有线路号，银行账户有账户编码，个人有社会保险号码。明天，你要到机场的 31 号出口去接你的生意伙伴。每天早上你都要乘坐红线火车，并在Kendall 广场站出站。

所有人都对这些名字的标准达成了一致，所以才能方便地共享这座城市的宝藏。你告诉出租车司机把你载到 McAllister 大街 246 号，他就知道你是什么意思了（即使他走的是一条很远的路）。

URL 就是因特网资源的标准化名称。URL 指向每一条电子信息，告诉你它们位于何处，以及如何与之进行交互。

本章，我们将介绍以下内容：

- URL 语法，以及各种 URL 组件的含义及其所做的工作；
- 很多 Web 客户端都支持的 URL 快捷方式，包括相对 URL 和自动扩展 URL；
- URL 编码和字符规则；
- 支持各种因特网信息系统的常见 URL 方案；
- URL 的未来，包括 URN——这种框架可以在对象从一处搬移到另一处时，保持稳定的访问名称。

23

2.1 浏览因特网资源

URL 是浏览器寻找信息时所需的资源位置。通过 URL，人类和应用程序才能找到、使用并共享因特网上大量的数据资源。URL 是人们对 HTTP 和其他协议的常用访问点：一个人将浏览器指向一个 URL，浏览器就会在幕后发送适当的协议报文来获取人们所期望的资源。

URI 是一类更通用的资源标识符，URL 实际上是它的一个子集。URI 是一个通用的概念，由两个主要的子集 URL 和 URN 构成，URL 是通过描述资源的位置来标识资源的，而 URN（本章稍后会介绍）则是通过名字来识别资源的，与它们当前所处位置无关。

HTTP 规范将更通用的概念 URI 作为其资源标识符，但实际上，HTTP 应用程序处理的只是 URI 的 URL 子集。本书有时会不加区分地使用 URI 和 URL，但我们讲的基本上都是 URL。

比如说，你想要获取 URL http://www.joes-hardware.com/seasonal/index-fall.html。那么 URL 分以下三部分。

- URL 的第一部分（http）是 URL 方案（scheme）。方案可以告知 Web 客户端怎样访问资源。在这个例子中，URL 说明要使用 HTTP 协议。
- URL 的第二部分（www.joes-hardware.com）指的是服务器的位置。这部分告知 Web 客户端资源位于何处。
- URL 的第三部分（/seasonal/index-fall.html）是资源路径。路径说明了请求的是服务器上哪个特定的本地资源。

对此的说明请参见图 2-1。

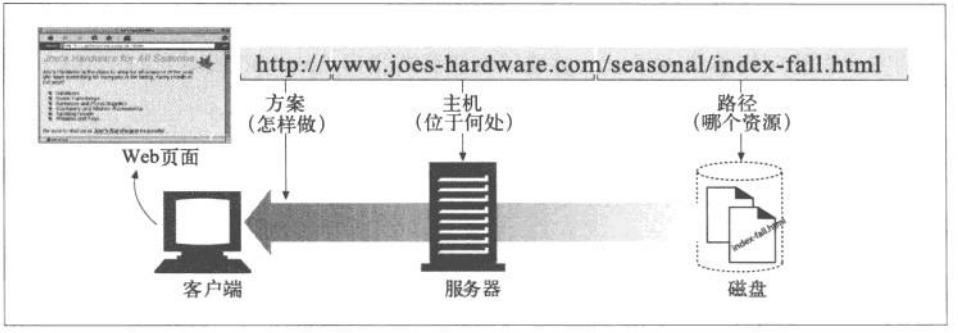


图 2-1　URL 是怎样与浏览器、客户端、服务器以及服务器文件系统中的位置进行关联的

URL 可以通过 HTTP 之外的其他协议来访问资源。它们可以指向因特网上的任意资源，比如个人的 E-mail 账户：

```
mailto:president@whitehouse.gov
```


或者其他协议，比如通过文件传输协议（File Transfer Protocol，FTP）才能获取的各种文件：

```
ftp://ftp.lots-o-books.com/pub/complete-price-list.xls
```

或者从流视频服务器上下载电影：

```
rtsp://www.joes-hardware.com:554/interview/cto_video
```

URL 提供了一种统一的资源命名方式。大多数 URL 都有同样的："方案 :// 服务器位置 / 路径"结构。因此，对网络上的每个资源以及获取这些资源的每种方式来说，命名资源的方法都只有一种，这样不管是谁都可以用名字来找到这个资源了。但是，事情并不是一开始就是这样的。

URL出现之前的黑暗岁月

在 Web 和 URL 出现之前，人们要靠分类杂乱的应用程序来访问分布在网络中的数据。大多数人都不会幸运地拥有所有合适的应用程序，或者不能够理解，也没有足够的耐心来使用这些程序。

在 URL 出现之前，要想和朋友共享 complete-catalog.xls 文件，就得说这样一些话："用 FTP 连接到 ftp.joes-hardware.com 上。用匿名登录，然后输入你的用户名作为密码。变换到 pub 目录。转换为二进制模式。现在，可以将名为 complete-catalog.xls 的文件下载到本地文件系统，并在那里浏览这个文件了。"

现在，像网景的 Navigator 和微软的 Internet Explorer 这样的浏览器都将很多这样的功能捆绑成一个便捷包。通过 URL，这些应用程序就可以通过一个接口，以统一的方式去访问许多资源了。只要说"将浏览器指向 ftp://ftp.lots-o-books.com/pub/complete-catalog.xls"就可以取代上面那些复杂的指令了。

URL 为应用程序提供了一种访问资源的手段。实际上，很多用户可能都不知道他们的浏览器在获取所请求资源时所使用的协议和访问方法。

有了 Web 浏览器，就不再需要用新闻阅读器来阅读因特网新闻，或者用 FTP 客户端来访问 FTP 服务器上的文件了，而且也无需用电子邮件程序来收发 E-mail 报文了。URL 告知浏览器如何对资源进行访问和处理，这有助于简化复杂的网络世界[1]。应用程序可以使用 URL 来简化信息的访问过程。

URL 为用户及他们的浏览器提供了找到信息所需的所有条件。URL 定义了用户所需
25 的特定资源，它位于何处以及如何获取它。

2.2 URL的语法

URL 提供了一种定位因特网上任意资源的手段，但这些资源是可以通过各种不同的方案（比如 HTTP、FTP、SMTP）来访问的，因此 URL 语法会随方案的不同而有所不同。

这是不是意味着每种不同的 URL 方案都会有完全不同的语法呢？实际上，不是的。大部分 URL 都遵循通用的 URL 语法，而且不同 URL 方案的风格和语法都有不少重叠。

注 1：浏览器通常会用其他应用程序来处理特殊的资源。比如，Internet Explorer 就装载了一个 E-mail 应用程序来处理那些表示 E-mail 资源的 URL。

大多数 URL 方案的 URL 语法都建立在这个由 9 部分构成的通用格式上：

```
<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>
```

几乎没有哪个 URL 中包含了所有这些组件。URL 最重要的 3 个部分是方案（scheme）、主机（host）和路径（path）。表 2-1 对各种组件进行了总结。

表2-1 通用URL组件

组 件	描 述	默 认 值
方案	访问服务器以获取资源时要使用哪种协议	无
用户	某些方案访问资源时需要的用户名	匿名
密码	用户名后面可能要包含的密码，中间由冒号（:）分隔	<E-mail 地址 >
主机	资源宿主服务器的主机名或点分 IP 地址	无
端口	资源宿主服务器正在监听的端口号。很多方案都有默认端口号（HTTP 的默认端口号为 80）	每个方案特有
路径	服务器上资源的本地名，由一个斜杠（/）将其与前面的 URL 组件分隔开来。路径组件的语法是与服务器和方案有关的（本章稍后会讲到 URL 路径可以分为若干个段，每段都可以有其特有的组件。）	无
参数	某些方案会用这个组件来指定输入参数。参数为名 / 值对。URL 中可以包含多个参数字段，它们相互之间以及与路径的其余部分之间用分号（;）分隔	无
查询	某些方案会用这个组件传递参数以激活应用程序（比如数据库、公告板、搜索引擎以及其他因特网网关）。查询组件的内容没有通用格式。用字符“?”将其与 URL 的其余部分分隔开来	无
片段	一小片或一部分资源的名字。引用对象时，不会将 frag 字段传送给服务器，这个字段是在客户端内部使用的。通过字符“#”将其与 URL 的其余部分分隔开来	无

比如，我们来看看 URL：http://www.joes-hardware.com:80/index.html，其方案是 http，主机为 www.joes-hardware.com，端口是 80，路径为 /index.html。

2.2.1 方案——使用什么协议

方案实际上是规定如何访问指定资源的主要标识符，它会告诉负责解析 URL 的应用程序应该使用什么协议。在我们这个简单的 HTTP URL 中所使用的方案就是 http。

方案组件必须以一个字母符号开始，由第一个“:”符号将其与 URL 的其余部分分隔开来。方案名是大小写无关的，因此 URL“http://www.joes-hardware.com”和“HTTP://www.joes-hardware.com”是等价的。

2.2.2 主机与端口

要想在因特网上找到资源，应用程序要知道是哪台机器装载了资源，以及在那台机器的什么地方可以找到能对目标资源进行访问的服务器。URL 的*主机*和*端口*组件提供了这两组信息。

主机组件标识了因特网上能够访问资源的宿主机器。可以用上述主机名（www.joes-hardware.com），或者 IP 地址来表示主机名。比如，下面两个 URL 就指向同一个资源——第一个 URL 是通过主机名，第二个是通过 IP 地址指向服务器的：

```
http://www.joes-hardware.com:80/index.html
http://161.58.228.45:80/index.html
```

端口组件标识了服务器正在监听的网络端口。对下层使用了 TCP 协议的 HTTP 来说，默认端口号为 80。

2.2.3 用户名和密码

更有趣的组件是*用户*和*密码*组件。很多服务器都要求输入用户名和密码才会允许用户访问数据。FTP 服务器就是这样一个常见的实例。这里有几个例子：

```
ftp://ftp.prep.ai.mit.edu/pub/gnu
ftp://anonymous@ftp.prep.ai.mit.edu/pub/gnu
ftp://anonymous:my_passwd@ftp.prep.ai.mit.edu/pub/gnu
http://joe:joespasswd@www.joes-hardware.com/sales_info.txt
```

第一个例子没有用户或密码组件，只有标准的方案、主机和路径。如果某应用程序使用的 URL 方案要求输入用户名和密码，比如 FTP，但用户没有提供，它通常会插入一个默认的用户名和密码。比如，如果向浏览器提供一个 FTP URL，但没有指定用户名和密码，它就会插入 anonymous（匿名用户）作为你的用户名，并发送一个默认的密码（Internet Explorer 会发送 IEUser，Netscape Navigator 则会发送 mozilla）。

第二个例子显示了一个指定为 anonymous 的用户名。这个用户名与主机组件组合在一起，看起来就像 E-mail 地址一样。字符“@”将用户和密码组件与 URL 的其余部分分隔开来。

在第三个例子中，指定了用户名（anonymous）和密码（my_passwd），两者之间由字符“:”分隔。

2.2.4 路径

URL 的路径组件说明了资源位于服务器的什么地方。路径通常很像一个分级的文件系统路径。比如：

http://www.joes-hardware.com:80/seasonal/index-fall.html

这个 URL 中的路径为 /seasonal/index-fall.html，很像 UNIX 文件系统中的文件系统路径。路径是服务器定位资源时所需的信息。[2]可以用字符“/”将 HTTP URL 的路径组件划分成一些路径段（path segment）（还是与 UNIX 文件系统中的文件路径类似）。每个路径段都有自己的参数（param）组件。

2.2.5 参数

对很多方案来说，只有简单的主机名和到达对象的路径是不够的。除了服务器正在监听的端口，以及是否能够通过用户名和密码访问资源外，很多协议都还需要更多的信息才能工作。

负责解析 URL 的应用程序需要这些协议参数来访问资源。否则，另一端的服务器可能就不会为请求提供服务，或者更糟糕的是，提供错误的服务。比如，像 FTP 这样的协议，有两种传输模式，二进制和文本形式。你肯定不希望以文本形式来传送二进制图片，这样的话，二进制图片可能会变得一团糟。

为了向应用程序提供它们所需的输入参数，以便正确地与服务器进行交互，URL 中有一个参数组件。这个组件就是 URL 中的名值对列表，由字符“;”将其与 URL 的其余部分（以及各名值对）分隔开来。它们为应用程序提供了访问资源所需的所有附加信息。比如：

ftp://prep.ai.mit.edu/pub/gnu;type=d

在这个例子中，有一个参数 type=d，参数名为 type，值为 d。

如前所述，HTTP URL 的路径组件可以分成若干路径段。每段都可以有自己的参数。比如：

http://www.joes-hardware.com/hammers;sale=false/index.html;graphics=true

这个例子就有两个路径段，hammers 和 index.html。hammers 路径段有参数 sale，其值为 false。index.html 段有参数 graphics，其值为 true。

注 2：这是一种简化的说法。在 18.2 节我们会看到，路径并不总能为资源定位提供足够的信息。有时服务器还需要其他的信息。

2.2.6 查询字符串

很多资源，比如数据库服务，都是可以通过提问题或进行查询来缩小所请求资源类型范围的。

假设 Joe 的五金商店在数据库中维护着一个未售货物的清单，并可以对清单进行查询，以判断产品是否有货，那就可以用下列 URL 来查询 Web 数据库网关，看看编号为 12731 的条目是否有货：

```
http://www.joes-hardware.com/inventory-check.cgi?item=12731
```

这个 URL 的大部分都与我们见过的其他 URL 类似。只有问号（?）右边的内容是新出现的。这部分被称为*查询*（query）组件。URL 的查询组件和标识网关资源的 URL 路径组件一起被发送给网关资源。基本上可以将网关当作访问其他应用程序的访问点（第 8 章会对网关进行详细的讨论）。

图 2-2 中有一个作为 Joe 的五金商店清单查询应用程序的网关的服务器，在这个例子中向此服务器发送了一个查询组件。查询的目的是检查清单中是否有尺寸为 large、颜色为 blue 的条目 12731。

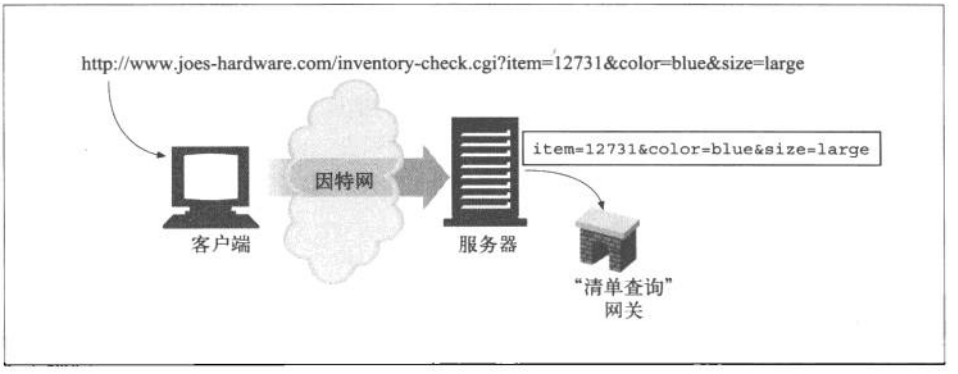


图 2-2 发送给网关应用程序的 URL 查询组件

在本章稍后会看到，除了有些不合规则的字符需要特别处理之外，对查询组件的格式没什么要求。按照常规，很多网关都希望查询字符串以一系列“名 / 值”对的形式出现，名值对之间用字符“&”分隔：

29

```
http://www.joes-hardware.com/inventory-check.cgi?item=12731&color=blue
```

在这个例子中，查询组件有两个名 / 值对：item=12731 和 color=blue。

2.2.7 片段

有些资源类型，比如 HTML，除了资源级之外，还可以做进一步的划分。比如，对一个带有章节的大型文本文档来说，资源的 URL 会指向整个文本文档，但理想的情况是，能够指定资源中的那些章节。

为了引用部分资源或资源的一个片段，URL 支持使用片段（frag）组件来表示一个资源内部的片段。比如，URL 可以指向 HTML 文档中一个特定的图片或小节。

片段挂在 URL 的右手边，最前面有一个字符“#”。比如：

```
http://www.joes-hardware.com/tools.html#drills
```

在这个例子中，片段 drills 引用了 Joe 的五金商店 Web 服务器上页面 /tools.html 中的一个部分。这部分的名字叫做 drills。

HTTP 服务器通常只处理整个对象，[3] 而不是对象的片段，客户端不能将片段传送给服务器（参见图 2-3）。浏览器从服务器获得了整个资源之后，会根据片段来显示你感兴趣的那部分资源。

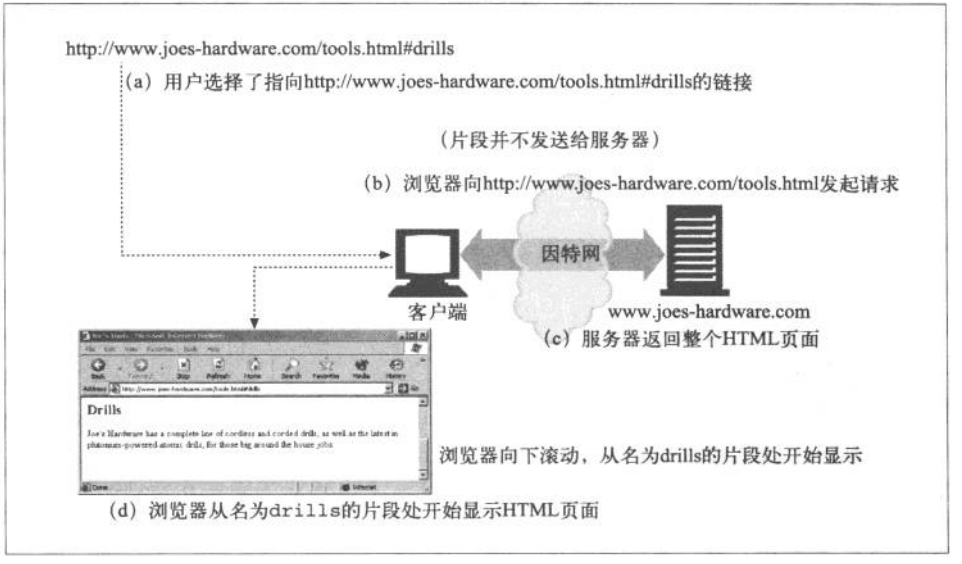


图 2-3 服务器处理的是整个对象，因此 URL 片段仅由客户端使用

注 3：在 15.9 节会看到 HTTP Agent 代理可能会请求某个字节范围内的对象，但在 URL 片段的上下文中，服务器会发送整个对象，由 Agent 代理将片段标识符应用于资源。

2.3 URL快捷方式

Web客户端可以理解并使用几种URL快捷方式。相对URL是在某资源内部指定一个资源的便捷缩略方式。很多浏览器还支持URL的“自动扩展”，也就是用户输入URL的一个关键（可记忆的）部分，然后由浏览器将其余部分填充起来。2.3.2节对此进行了解释。

2.3.1 相对URL

URL有两种方式：*绝对的*和*相对的*。到目前为止，我们只见过绝对URL。绝对
30 URL中包含有访问资源所需的全部信息。

另一方面，相对URL是不完整的。要从相对URL中获取访问资源所需的全部信息，就必须相对于另一个，被称为其*基础*（base）的URL进行解析。

相对URL是URL的一种便捷缩略记法。如果你手工写过HTML的话，可能就会发现相对URL是多么便捷了。例2-1是一个嵌入了相对URL的HTML文档实例。

例2-1 带有相对URL的HTML代码片段

```
<HTML>
<HEAD><TITLE>Joe's Tools</TITLE></HEAD>
<BODY>
<H1> Tools Page </H1>
<H2> Hammers <H2>
<P> Joe's Hardware Online has the largest selection of <A HREF="./
hammers.html">hammers
</A> on earth.
</BODY>
</HTML>
```

例2-1是资源：

http://www.joes-hardware.com/tools.html

的HTML文档。

在这个HTML文档中有一个包含了URL ./hammers.html的超链接。这个URL看起来是不完整的，但实际上是个合法的相对URL。可以相对于它所在文档的URL对其进行解释；在这个例子中，就是相对于Joe的五金商店Web服务器的资源/tools.
31 html。

使用缩略形式的相对URL语法，HTML的编写者就可以省略URL中的方案、主机和其他一些组件了。这些组件可以从它们所属资源的*基础*URL中推导出来。其他资源的URL也可以用这种缩略形式来表示。

在例 2-1 中，基础 URL 为：

http://www.joes-hardware.com/tools.html

用这个 URL 作为基础，可以推导出缺失的信息。我们知道资源名为 ./hammers.html，但并不知道方案或主机名是什么。通过这个基础 URL，可以推导出方案为 http，主机为 www.joes-hardware.com。图 2-4 对此进行了说明。

图 2-4 使用基础 URL

相对 URL 只是 URL 的片段或一小部分。处理 URL 的应用程序（比如浏览器）要能够在相对和绝对 URL 之间进行转换。

还需要注意的是，相对 URL 为保持一组资源（比如一些 HTML 页面）的便携性提供了一种便捷方式。如果使用的是相对 URL，就可以在搬移一组文档的同时，仍然保持链接的有效性，因为相对 URL 都是相对于新基础进行解释的。这样就可以实现在其他服务器上提供镜像内容之类的功能了。

1. 基础URL

转换处理的第一步就是找到基础 URL。基础 URL 是作为相对 URL 的参考点使用的。可以来自以下几个不同的地方。

- *在资源中显式提供*
  有些资源会显式地指定基础 URL。比如，HTML 文档中可能会包含一个定义了基础 URL 的 HTML 标记 <BASE>，通过它来转换那个 HTML 文档中的所有相对 URL。

- *封装资源的基础 URL*
  如果在一个没有显式指定基础 URL 的资源中发现了一个相对 URL，如例 2-1 所示，可以将它所属资源的 URL 作为基础（如例中所示）。

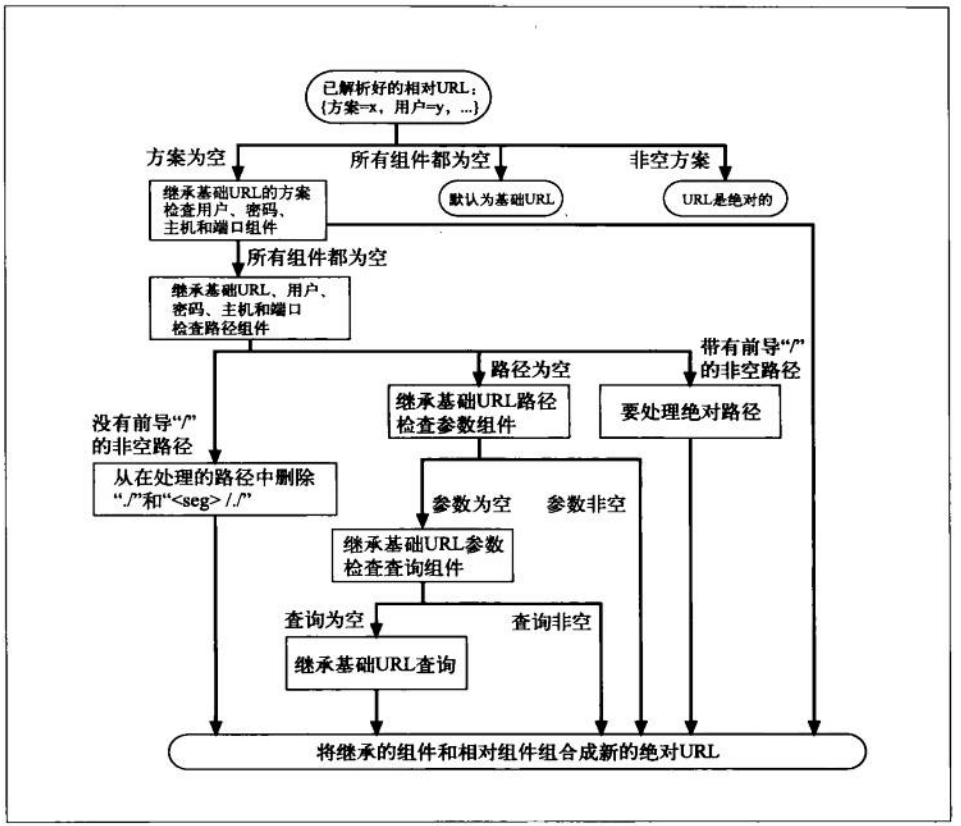
- *没有基础 URL*
  在某些情况下，没有基础 URL。这通常意味着你有一个相对 URL；但有时可能只是一个不完整或损坏了的 URL。

32

2. 解析相对引用

前面我们介绍了 URL 的基本组件和语法。要将相对 URL 转换为一个绝对 URL，下一步要做的就是将相对 URL 和基础 URL 划分成组件段。

实际上，这样只是在解析 URL，但这种做法会将其划分成一个个组件，因此通常会称作分解（decomposing）URL。只要将基础和相对 URL 划分成了组件，就可以应用图 2-5 中的算法来完成转换了。



33 图 2-5 将相对 URL 转换成绝对 URL

这个算法将一个相对 URL 转换成了其绝对模式，之后就可以用它来引用资源了。这个算法最初是在 RFC 1808 中制定的，后来被合并到了 RFC 2396 中。

可以对例 2-1 中的 ./hammers.html 实例使用图 2-5 中描述的算法。

(1) 路径为 ./hammers.html，基础 URL 为 http://www.joes-hardware.com/tools.html。
(2) 方案为空，沿着图表的左半边向下处理，继承基础 URL 方案（HTTP）。
(3) 至少一个组件非空，一直处理到底端，继承主机和端口组件。
(4) 将来自相对 URL（路径：./hammers.html）的组件与我们继承来的组件（方案：http，主机：www.joes-hardware.com，端口：80）合并起来，得到新的绝对 URL：http ://www.joes-hardware.com/hammers.html。

2.3.2 自动扩展URL

有些浏览器会在用户提交 URL 之后，或者在用户输入的时候尝试着自动扩展 URL。这就为用户提供了一条捷径：用户不需要输入完整的 URL，因为浏览器会自动扩展。

这些“自动扩展”特性有以下两种方式。

- *主机名扩展*
  在主机名扩展中，只要有些小提示，浏览器通常就可以在没有帮助的情况下，将你输入的主机名扩展为完整的主机名。
  比如，如果在地址栏中输入 yahoo，浏览器就会自动在主机名中插入 www. 和 .com，构建出 www.yahoo.com。如果找不到与 yahoo 匹配的站点，有些浏览器会在放弃之前尝试几种扩展形式。浏览器通过这些简单的技巧来节省你的时间，减少找不到的可能。
  但是，这些主机名扩展技巧可能会为其他一些 HTTP 应用程序带来问题，比如代理。第 6 章将详细讨论这些问题。

- *历史扩展*
  浏览器用来节省用户输入 URL 时间的另一种技巧是，将以前用户访问过的 URL 历史存储起来。当你输入 URL 时，它们就可以将你输入的 URL 与历史记录中 URL 的前缀进行匹配，并提供一些完整的选项供你选择。因此，如果你输入了一个以前访问过的 URL 的开始部分，比如 http://www.joes-，浏览器就可能会建议使用 http://www.joes-hardware.com。然后你就可以选择这个地址，不用输入完整的 URL 了。

34

注意，与代理共同使用时，URL 自动扩展的行为可能会有所不同。6.5.6 节将对此进行进一步讨论。

2.4 各种令人头疼的字符

URL 是*可移植的*（portable）。它要统一地命名因特网上所有的资源，这也就意味着要通过各种不同的协议来传送这些资源。这些协议在传输数据时都会使用不同的机制，所以，设计 URL，使其可以通过任意因特网协议安全地传输是很重要的。

安全传输意味着 URL 的传输不能丢失信息。有些协议，比如传输电子邮件的简单邮件传输协议（Simple Mail Transfer Protocol，SMTP），所使用的传输方法就会剥去一些特定的字符。[4] 为了避开这些问题，URL 只能使用一些相对较小的、通用的安全字母表中的字符。

除了希望 URL 可以被所有因特网协议进行传送之外，设计者们还希望 URL 是*可读的*。因此，即使不可见、不可打印的字符能够穿过邮件程序，从而成为可移植的，也不能在 URL 中使用。[5]

URL 还得是*完整的*，这就使问题变得更加复杂了。URL 的设计者们认识到有时人们可能会希望 URL 中包含除通用的安全字母表之外的二进制数据或字符。因此，需要有一种转义机制，能够将不安全的字符编码为安全字符，再进行传输。

本节总结了 URL 的通用字母表和编码规则。

2.4.1 URL字符集

默认的计算机系统字符集通常都倾向于以英语为中心。从历史上来看，很多计算机应用程序使用的都是 US-ASCII 字符集。US-ASCII 使用 7 位二进制码来表示英文打字机提供的大多数按键和少数用于文本格式和硬件通知的不可打印控制字符。

由于 US-ASCII 的历史悠久，所以其可移植性很好。但是，虽然美国用户使用起来很便捷，它却并不支持在各种欧洲语言或全世界数十亿人使用的数百种非罗马语言
35 中很常见的变体字符。

而且，有些 URL 中还会包含任意的二进制数据。认识到对完整性的需求之后，URL 的设计者就将*转义序列*集成了进去。通过转义序列，就可以用 US-ASCII 字符集的有限子集对任意字符值或数据进行编码了，这样就实现了可移植性和完整性。

2.4.2 编码机制

为了避开安全字符集表示法带来的限制，人们设计了一种编码机制，用来在 URL 中表示各种不安全的字符。这种编码机制就是通过一种“转义”表示法来表示不安全

注 4：这是由报文的 7 位二进制码编码造成的。如果源端是以 8 位或更多位编码的，就会有部分信息被剥离。
注 5：不可打印字符中包括空格符（注意，RFC 2396 建议应用程序忽略空格符）。

字符的，这种转义表示法包含一个百分号（%），后面跟着两个表示字符 ASCII 码的十六进制数。

表 2-2 中列出了几个例子。

表2-2 一些编码字符示例

字 符	ASCII码	示例URL
~	126(0x7E)	http://www.joes-hardware.com/%7Ejoe
空格	32(0x20)	http://www.joes-hardware.com/more%20tools.html
%	37(0x25)	http://www.joes-hardware.com/100%25satisfaction.html

2.4.3 字符限制

在 URL 中，有几个字符被保留起来，有着特殊的含义。有些字符不在定义的 US-ASCII 可打印字符集中。还有些字符会与某些因特网网关和协议产生混淆，因此不赞成使用。

表 2-3 列出了一些字符，在将其用于保留用途之外的场合时，要在 URL 中对其进行编码。

表2-3 保留及受限的字符

字 符	保留/受限
%	保留作为编码字符的转义标志
/	保留作为路径组件中分隔路径段的定界符
.	保留在路径组件中使用
..	保留在路径组件中使用
#	保留作为分段定界符使用
?	保留作为查询字符串定界符使用
;	保留作为参数定界符使用
:	保留作为方案、用户 / 口令，以及主机 / 端口组件的定界符使用
$, +	保留
@ & =	在某些方案的上下文中有特殊的含义，保留
{ } | \ ^ ~ [ ] '	由于各种传输 Agent 代理，比如各种网关的不安全处理，使用受限
< > "	不安全：这些字符在 URL 范围之外通常是有意义的，比如在文档中对 URL 自身进行定界（比如 http://www.joes-hardware.com），所以应该对其进行编码
0x00-0x1F, 0x7F	受限，这些十六进制范围内的字符都在 US-ASCII 字符集的不可打印区间内
>0x7F	受限，十六进制值在此范围内的字符都不在 US-ASCII 字符集的 7 二进制位范围内

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。

表 2-4 总结了最常见的一些方案。回顾一下 2.2 节有助于理解表格中的语法部分。

表2-4 常见的方案格式

方 案	描 述
http	超文本传输协议方案，除了没有用户名和密码之外，与通用的 URL 格式相符。如果省略了端口，就默认为 80。 基本格式： http://<host>:<port>/<path>?<query>#<frag> 示例： http://www.joes-hardware.com/index.html http://www.joes-hardware.com:80/index.html
https	方案 https 与方案 http 是一对。唯一的区别在于方案 https 使用了网景的 SSL，SSL 为 HTTP 连接提供了端到端的加密机制。其语法与 HTTP 的语法相同，默认端口为 443。 基本格式： https://<host>:<port>/<path>?<query>#<frag> 示例： https://www.joes-hardware.com/secure.html
mailto	Mailto URL 指向的是 E-mail 地址。由于 E-mail 的行为与其他方案都有所不同（它并不指向任何可以直接访问的对象），所以 mailto URL 的格式与标准 URL 的格式也有所不同。因特网 E-mail 地址的语法记录在 RFC 822 中。 基本格式： mailto:<RFC-822-addr-spec> 示例： mailto:joe@joes-hardware.com
ftp	文件传输协议 URL 可以用来从 FTP 服务器上下载或向其上载文件，并获取 FTP 服务器上的目录结构内容的列表。 在 Web 和 URL 出现之前 FTP 就已经存在了。Web 应用程序将 FTP 作为一种数据访问方案使用。URL 语法遵循下列通用格式。 基本格式： ftp://<user>:<password>@<host>:<port>/<path>;<params> 示例： ftp://anonymous:joe%40joes-hardware.com@prep.ai.mit.edu:21/pub/gnu/
rtsp，rtspu	RTSP URL 是可以通过实时流传输协议（Real Time Streaming Protocol）解析的音 / 视频媒体资源的标识符。 方案 rtspu 中的 u 表示它是使用 UDP 协议来获取资源的。 基本格式： rtsp://<user>:<password>@<host>:<port>/<path> rtspu://<user>:<password>@<host>:<port>/<path> 示例： rtsp://www.joes-hardware.com:554/interview/cto_video

38

（续）

方　案	描　述
file	方案 file 表示一台指定主机（通过本地磁盘、网络文件系统或其他一些文件共享系统）上可直接访问的文件。各字段都遵循通用格式。如果省略了主机名，就默认为正在使用 URL 的本地主机。 基本格式： file://<host>/<path> 示例： file://OFFICE-FS/policies/casual-fridays.doc
news	根据 RFC 1036 的定义，方案 news 用来访问一些特定的文章或新闻组。它有一个很独特的性质：news URL 自身包含的信息不足以对资源进行定位。 news URL 中缺乏到何处获取资源的信息——没有提供主机名或机器名称。从用户那里获取此类信息是解释程序的工作。比如，在网景浏览器的“选项”（Options）菜单中，就可以指定自己的 NNTP（news）服务器。这样，浏览器有了 news URL 的时候就知道应该使用哪个服务器了。 新闻资源可以从多台服务器中获得。它们被称为位置无关的，因为对它们的访问不依赖于任何一个源服务器。 news URL 中保留了字符“@”，用来区分指向新闻组的 news URL 和指向特定新闻文章的 news URL。 基本格式： news:<newsgroup> news:<news-article-id> 示例： news:rec.arts.startrek
telnet	方案 telnet 用于访问交互式业务。它表示的并不是对象自身，而是可通过 telnet 协议访问的交互式应用程序（资源）。 基本格式： telnet://<user>:<password>@<host>:<port>/ 示例： telnet://slurp:webhound@joes-hardware.com:23/

39

2.6　未来展望

URL 是一种强有力的工具。它可以用来命名所有现存对象，而且可以很方便地包含一些新格式。URL 还提供了一种可以在各种因特网协议间共享的统一命名机制。

但 URL 并不完美。它们表示的是实际的地址，而不是准确的名字。这就意味着 URL 会告诉你资源此时处于什么位置。它会为你提供特定端口上特定服务器的名字，告诉你在何处可以找到这个资源。这种方案的缺点在于如果资源被移走了，URL 也就不再有效了。那时，它就无法对对象进行定位了。

如果有了对象的准确名称，则不论其位于何处都可以找到这个对象，那该多完美啊。就像人一样，只要给定了资源的名称和其他一些情况，无论资源移到何处，你都能够追踪到它。

为了应对这个问题，因特网工程任务组（Internet Engineering Task Force，IETF）已经对一种名为统一资源名（uniform resource name，URN）的新标准做了一段时间的研究了。无论对象搬移到什么地方（在一个 Web 服务器内或是在不同的 Web 服务器间），URN 都能为对象提供一个稳定的名称。

永久统一资源定位符（persistent uniform resource locators，PURL）是用 URL 来实现 URN 功能的一个例子。其基本思想是在搜索资源的过程中引入另一个中间层，通过一个中间*资源定位符*（resource locator）服务器对资源的实际 URL 进行登记和跟踪。客户端可以向定位符请求一个永久 URL，定位符可以以一个资源作为响应，将客户端重定向到资源当前实际的 URL 上去（参见图 2-6）。更多有关 PURL 的信息，请访问 http://purl.oclc.org。

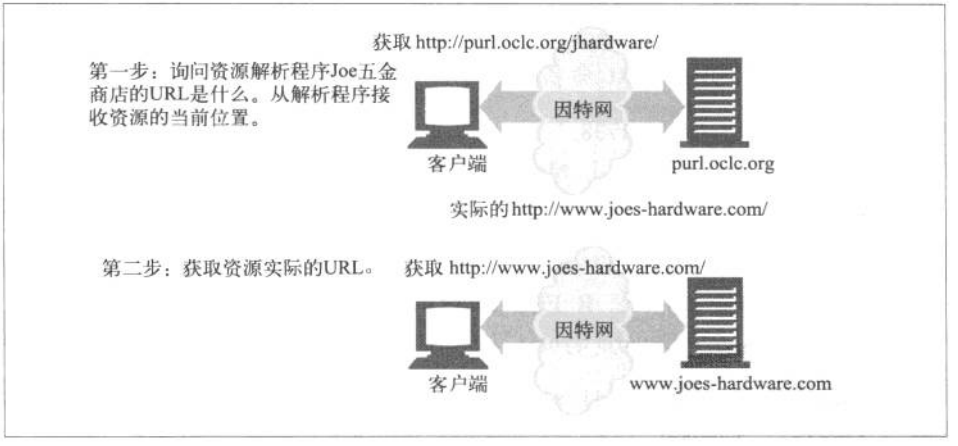


图 2-6　PURL 通过资源定位符服务器来命名资源的当前位置

如果不是现在，那是什么时候

URN 背后的思想已经提出一段时间了。实际上，如果去看看某些相关规范的发布日期，你可能会问，为什么它们现在都还没有投入使用。

从 URL 转换成 URN 是一项巨大的工程。标准化工作的进程很缓慢，而且通常都有很充分的理由。支持 URN 需要进行很多改动——标准主体的一致性，对各种 HTTP

应用程序的修改等。做这种改动需要进行大量的工作，而且很不幸（或者可能很幸运）的是 URL 还有很大的能量，还要等待更合适的时机才能进行这种转换。

在 Web 爆炸性增长的过程中，因特网用户——包括从计算机科学家到普通因特网用户的每一个人——都已经学会使用 URL 了。在备受笨拙语法（对新手来说）和顽固问题困扰的同时，人们已经学会了如何使用 URL，以及如何对付它们的一些缺陷。URL 有一些限制，但这并不是 Web 开发社区所面临的最紧迫的问题。

目前看来，在可预见的未来，因特网资源仍然会以 URL 来命名。它们无处不在，而且是 Web 的成功过程中一个非常重要的部分。其他命名方案想要取代 URL 还要一段时间。但是，URL 确实有其局限型，可能会出现新的标准（可能就是 URN），对这种标准进行部署会解决其中的某些问题。

2.7 更多信息

更多有关 URL 的信息，请参考以下资源。

- http://www.w3.org/Addressing/
  这是 W3C 有关 URI 和 URL 命名及寻址的 Web 页面。

- http://www.ietf.org/rfc/rfc1738
  RFC 1738，T. Berners-Lee、L. Masinter 和 M. McCahill 编写的“Uniform Resource
41 Locators（URL）”（“统一资源定位符”）。

- http://www.ietf.org/rfc/rfc2396.txt
  RFC 2396，T. Berners-Lee、R. Fielding 和 L. Masinter 编写的“Uniform Resource Identifiers（URL）：Generic Syntax”（“URL：通用语法”）。

- http://www.ietf.org/rfc/rfc2141.txt
  RFC 2141，R. Moats 编写的“URN Syntax”（“URN 语法”）。

- http://purl.oclc.org
  永久统一资源定位符的 Web 站点。

- http://www.ietf.org/rfc/rfc1808.txt
  RFC 1808，R. Fielding 编写的“Relative Uniform Resource Locators”（“相对统
42 一资源定位符”）。