互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

第1章 网上冲浪

1

万维网变得越来越普遍的一个重要原因就是它的易用性，普通人不需要经过多少培训就可以使用它来完成一些很有价值的工作。但是在这表象之后，Web 也是一个用于分布式计算的功能强大的平台。

那些让普通人易于使用互联网的原则，也同样适用于某些自动化的软件 agent“用户”。比如，一些软件被设计用来在不同的银行账户间自动地进行货币交易（或者用来完成现实世界的其他任务），为了完成相应的任务，它们所采用的基本技术和人类所使用的技术是相同的。

就本书而言，有三项技术支撑着当今的互联网，它们分别是：URL 命名约定、HTTP 协议和 HTML 文档格式。URL 和 HTTP 是比较简单的，但是如果想要将它们应用到分布式编程中，和大多数的 web 开发人员相比，你就必须更加了解它们的细节。本书的前几章就专门用来帮助大家学习这些内容。

HTML 的内容就有点复杂了。在 web API 的世界里，有许许多多的数据格式在试图取代 HTML 的地位。对这些数据格式的研究将从第 5 章开始，并占据本书的多个章节的篇幅。暂时，我将重点放在 URL 和 HTTP 上面，仅仅使用 HTML 作为一个例子。

我计划从一个和万维网相关的简单故事开始，以此来解释隐藏在万维网设计背后的原则以及驱动它成功的缘由。虽然你对 Web 很熟悉，但是对于那些使得 Web 运转起来的技术和概念你可能并没听说过，所以这个故事是很简单的，以便于你能够理解它。如果你对类似于“将超媒体作为应用状态的引擎 (hypermedia as the engine of application state)”的技术不是很清楚，我希望你能通过这个简单而又具体的例子来得到一些收获。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

那我们现在开始吧。

2

场景1：广告牌

一天，Alice 正在城里散步，她看到了一个广告牌（见图 1-1）。

图1-1 广告牌

（顺便说一下，这个虚构的广告牌上面宣传的是我为这本书设计的一个真实的网站，你可以自己试着访问一下。）

这个广告牌是在 20 世纪 90 年代中期建立的，那时候 Alice 已经可以记事了，她至今都还记得这个广告牌刚开始展示这个 URL 时候公众的反应。起初，人们都拿这些看起来很怪异的字符串开玩笑，那时的人们并不清楚“http://”和“youtypeitwepostit.com”的真正含义。但是 20 年后的今天，几乎每个人都知道如何使用这个 URL：将 URL 输入到 web 浏览器的地址栏，然后按下回车键。

Alice 也是这样做的：她拿出了自己的手机，将“*http://www.youtypeitwepostit.com/*”输进了浏览器地址栏。我们的故事的第一个场景就到此结束了，我们保留一个悬念：URL 的另一端是什么呢？

资源和表述

非常抱歉中断了这个故事，但是我需要介绍一些基本术语。Alice 的 web 浏览器会试图向特定的 web 服务器的 URL:*http://www.youtypeitwepostit.com/* 发送一条 HTTP 请求。一个 web 服务器可以管理许多不同的URL，并且每个URL 被授权访问服务器上的不同数据。

我们所说的 URL 是一些事物的 URL，比如一个产品、一个用户或主页等。这些用 URL

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

命名的事物的专业术语是资源 (resource)。

这个 URL:*http://www.youtypeitwepostit.com/* 标识的资源应该就是那个广告牌上宣传的网站的主页。但是也许只有将这个故事继续下去，等到 Alice 的浏览器真正发送了 HTTP 请求以后，我们才能确定这些猜想。

web 浏览器为一个资源发送了 HTTP 请求以后，服务器会发送一个文档作为响应（通常是一个 HTML 文档，但是有时候是二进制图片或者其他东西）。不论服务器发送了什么文档，我们都将这个文档称为资源的表述 (representation of the resource)。

3

每个 URL 标识一个资源。客户端向某个 URL 发送了一条 HTTP 请求以后，它就会收到对应资源的表述。客户端从来都不会直接看到资源，看到的都是资源的表述。

我会在第 3 章讲解更多关于资源和表述的内容。现在我只是想要使用资源和表述这两个术语来开始讨论可寻址性原则。

可寻址性

每个 URL 代表一个也仅代表一个资源。如果一个网站上面有两个从概念上讲并不相同的事物，那么我们就应该将它们作为两个资源并为它们分配不同的 URL。当网站违背了这个规则的时候，我们便会因为在使用时无所适从而心情沮丧。有一些餐厅的网站在这一点上做得就很差劲。有时候，整个网站堆砌在一个 Flash 界面里，并没有提供那些指向我们就其本身可以讨论的资源的独立 URL，比如菜单以及用来显示餐厅地址的地图等。

可寻址性原则就是说每个资源应该有一个属于自己的 URL。如果你的程序里面有些东西非常重要，它就应该有一个唯一的名字，一个 URL，这样你和你的用户就可以非常清楚地、毫无歧义地引用它了。

场景2：主页

回到我们的故事中，当 Alice 将广告牌上的 URL 输入到她的浏览器地址栏的时候，她就通过因特网向 web 服务器上面的 *http://www.youtypeitwepostit.com/* 发送了一个 HTTP 请求：

```
GET / HTTP/1.1
Host: www.youtypeitwepostit.com
```

web 服务器处理这个请求（Alice 和她的 web 浏览器都不需要知道是如何处理的）然后发送响应结果：

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

```
HTTP/1.1 200 OK
Content-type: text/html

<!DOCTYPE html>
<html>
    <head>
        <title>Home</title>
    </head>
    <body>
        <div>
          <h1>You type it, we post it!</h1>
          <p>Exciting! Amazing!</p>

          <p class="links">
            <a href="/messages">Get started</a>
            <a href="/about">About this site</a>
          </p>
        </div>
    </body>
</html>
```

在响应信息的头部的 200 是状态码，也被叫作响应码。这是服务器告诉客户端发生了什么事情的快捷方式。实际上还有很多其他的 HTTP 状态码，我会在附录 A 中对它们都进行一一介绍，但是最常见的状态码是我们前面见到的状态码 200。200（OK）表示这个请求被接受并正确无误地处理了。

Alice 的 web 浏览器将响应数据作为 HTML 文档进行了解析，并以图形化的形式展示了出来（见图 1-2）。

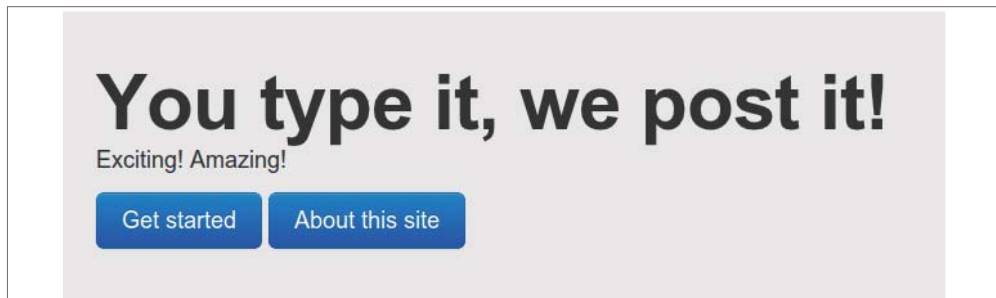

图1-2 You Type It… 主页

现在 Alice 可以浏览这个网页了，她终于明白那个广告牌在说什么了。这个广告牌是在给一个类似于 Twitter 的微博网站做宣传，尽管这个网站实际上并不像广告牌说的那样令人兴奋，但是作为一个例子也已经足够了。

Alice 和 web 服务器的第一次即时互动揭示了 Web 的一系列更重要的特性。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

短会话（Short Session）

故事发展到了这里，Alice 的 web 浏览器正在显示那个网站的主页。从她的角度来看，她已经“登录到”了这个页面，这也就是她在虚拟的网络空间的当前位置。但是就服务器考虑而言，Alice 哪也不存在。服务器也已经忘记了她的存在。

5

HTTP 会话只维持在一次请求过程中，客户端发送请求，服务器进行响应。这就意味着，Alice 可以将她的手机关一晚上，然后，当她的浏览器从内部缓存（cache）中恢复出这个网页以后，她依旧可以单击页面上的任意一个链接，网页应该还是可以继续工作的（和 SSH 会话相比，如果你将你的电脑关机，SSH 会话就终止了）。

Alice 甚至可以将她手机里面的网页一直打开着，在她在六个月后再次单击网页上面的链接时，web 服务器还是会迅速地做出响应，仿佛她只是等待了几秒钟。Web 服务器不需要为了 Alice 而通宵工作。当 Alice 不发送 HTTP 请求的时候，服务器并不清楚 Alice 的存在。

这项原则有时候就被称为无状态性（statelessness）。我想这是一个令人困惑的术语，因为系统里面的客户端和服务器端都要保存状态：它们只是保存不同类型的状态。“无状态性”术语是指服务器不关心客户端的状态（在后面的章节中，我将讲解不同类型的状态）。

自描述消息（self-descriptive message）

通过查看前面的 HTML，我们会清楚看到这个网站不仅仅有一个主页。主页的标签包含两个链接：其中一个是相对路径 URL“/about”（也就是 *http://www.youtypeitwepostit.com/about*），另一个是“/messages”（也就是 *http://www.youtypeitwepostit.com/messages*）。起初，Alice 只知道一个 URL——那个 URL 指向主页——但是现在她知道三个 URL 了。服务器正在慢慢将它的结构揭示给 Alice。

我们现在可以根据服务器揭示给 Alice 的信息给网站画一张地图了（见图 1-3）。

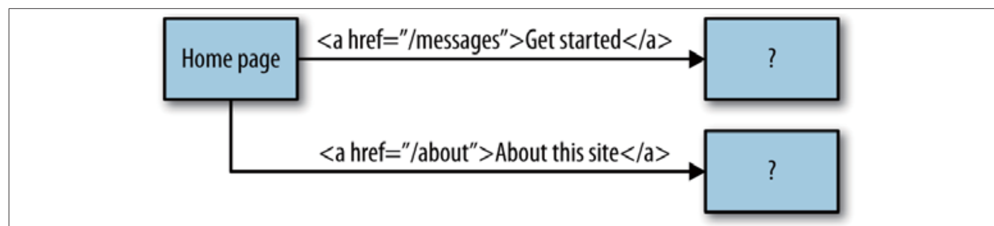

图1-3 网站地图

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

/messages 和 */about* 这两个链接的背后是什么呢？唯一确定的办法就是单击这些链接并找到真相。但是在这之前，Alice 可以查看一下 HTML 标记或浏览器呈现的可视化页面，进而根据这些信息来做一下推测：写有“About this site”文字的链接很可能是指向一个介绍这个网站的网页，而另一个写着“Get started”的链接很可能是能够让她真真正正地发布一条消息的页面。

6

当你请求一个网页的时候，你收到的 HTML 文档不仅仅可以提供给你所要求的即时信息，还会帮助你来决定下一步的操作。

场景3：链接

在浏览了主页以后，Alice 决定继续对这个网站做出进一步的尝试。她很自然地单击了那个写着“Get started”的链接。任何时候，你在浏览器上单击一个链接都表示你要求浏览器发送一个 HTTP 请求。

Alice 所单击的那个链接的代码如下：

```
<a href="/messages">Get started</a>
```

她的浏览器和上次一样向同样的服务器发送了一个 HTTP 请求：

```
GET /messages HTTP/1.1
Host: www.youtypeitwepostit.com
```

请求中的 GET 是一个 HTTP 方法（HTTP method），也就是大家所知道的的 HTTP 动词（HTTP verb）。HTTP 方法是客户端告诉服务器端它想要如何操作一个资源的方法。“GET”方法是最常见的 HTTP 方法。它的意思是“将这个资源的表述提供给我”。对于浏览器而言，GET 是默认值。当你点击一个链接，或者向地址栏输入一个 URL 的时候，你的浏览器就发送了一个 GET 的请求。

服务器处理这个特定的 GET 请求，发送了 */messages* 的表述给浏览器：

```
HTTP/1.1 200 OK
Content-type: text/html
...

<!DOCTYPE html>
<html>
    <head>
        <title>Messages</title>
    </head>
    <body>
      <div>
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

```
          <h1>Messages</h1>

          <p>
            Enter your message below:
          </p>

          <form action="http://youtypeitwepostit.com/messages" method="post">
              <input type="text" name="message" value="" required="true"
                     maxlength="6"/>
              <input type="submit" value="Post" />
          </form>

          <div>
            <p>
              Here are some other messages, too:
            </p>
            <ul>
              <li><a href="/messages/32740753167308867">Later</a></li>
              <li><a href="/messages/7534227794967592">Hello</a></li>
            </ul>
          </div>

          <p class="links">
            <a href="http://youtypeitwepostit.com/">Home</a>
          </p>

        </div>
      </body>
  </html>
```

7

像之前一样，Alice 的浏览器将 HTML 文本进行了图形渲染（见图 1-4）。

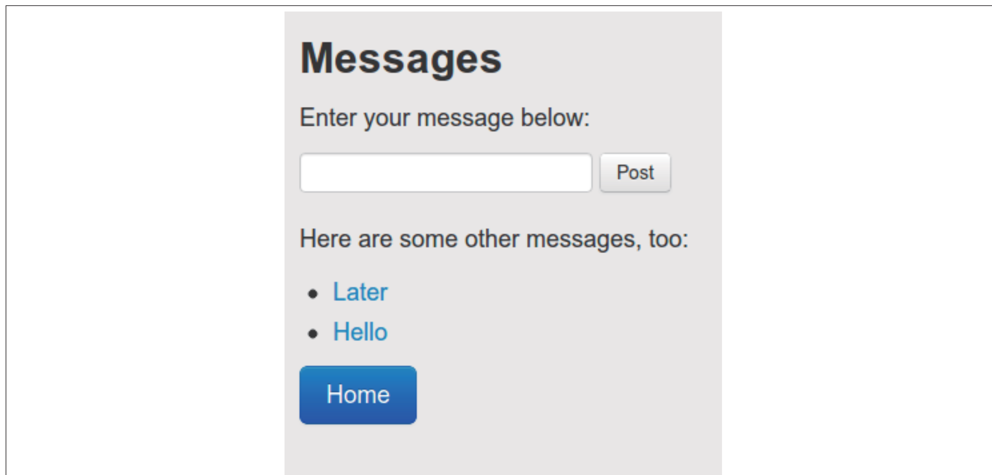

图1-4 You Type It… “Get started” 页面

互动出版网
China-Pub.com
专业图书网上第一专营店
www.china-pub.com
独家提供样章

Alice浏览这个页面以后，她会发现这个页面是一个消息列表，列表里面罗列了其他人已经发布到这个网站的消息，页面的上方还有一个引人注目的文本框和一个Post按钮。

现在我们获取到了这个服务器运行的更多信息，图1-5展示了到现在为止，Alice的浏览器所了解到的新的网站地图。

8

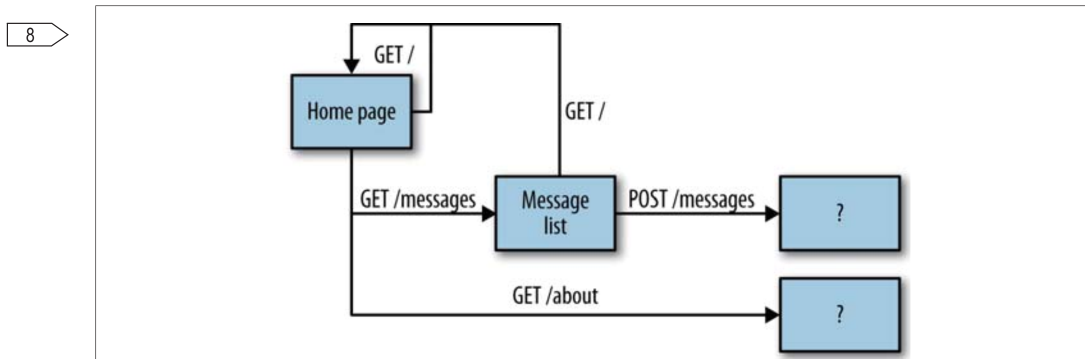

图1-5 关于You Type It…网站的浏览器的视图

标准方法

Alice的浏览器前两次发送的HTTP请求都是使用GET作为它们的HTTP方法，但是在这个最新的表述里面却有一部分特殊的HTML，它们的作用是当Alice单击Post按钮时触发一个HTTP POST请求：

```
<form action="http://youtypeitwepostit.com/messages" method="post">
    <input type="text" name="message" value="" required="true"
           maxlength="6"/>
    <input type="submit" />
</form>
```

HTTP标准（RFC 2616）定义了客户端可以应用到一个资源上的8种方法。在本书中，我将重点放在其中的5个方法上面，它们分别是GET、HEAD、POST、PUT和DELETE。在第3章中，我会对这些方法以及一个扩展方法PATCH进行更加细致的讲解。现在，最重要的事情就是记住这里有一些标准方法。

提出新的HTTP方法也不是不可能的（PATCH方法就是后来提出的），但是要付出很大的代价。在这个方面，它并不像一门编程语言：在编程语言的世界里，你可以将你的方法命名为任何东西。在我为这个例子而搭建这个简单的微博网站的时候，我并没有定义类似于GETHOMEPAGE（获取主页）和HELLOPLEASESHOWMETHEMESSAGELIST THANKSBYE（你好，请向我显示消息列表，谢谢，再见）这样的新HTTP方法。我使

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

用 GET 方法来同时实现了“显示主页”和“显示消息列表”的目的，因为在这两个例子中，GET 方法（“将资源的表述提供给我”）最符合 HTTP 接口以及我想要的。我并不是通过定义新的方法来区分主页和消息列表页，而是把这两个文档看作不同的资源：每个资源都有它自己的 URL，每个资源都可以通过 GET 进行访问。

场景4：表单和重定向

9

回到我们的故事，Alice 对这个微博网站的表单很有兴趣，她在文本框中输入了“Test”，然后单击了 Post 按钮。

现在，Alice 的浏览器又发送了一个 HTTP 请求：

```
POST /messages HTTP/1.1
Host: www.youtypeitwepostit.com
Content-type: application/x-www-form-urlencoded

message=Test&submit=Post
```

服务器返回的信息如下：

```
HTTP/1.1 303 See Other
Content-type: text/html

Location: http://www.youtypeitwepostit.com/messages/5266722824890167
```

Alice 的浏览器之前发送的两个 GET 请求，服务器返回了 HTTP 状态码 200（“OK”），并提供了一个可用于浏览器显示的 HTML 文档。但是这一次，服务器并没有返回 HTML 文档，取而代之的是在 Location 报头里面提供了另一个 URL 链接以及在响应信息头部提供状态码 303（“See Other”），而不是 200 (“OK”)。

状态码 303 是告诉 Alice 的浏览器要自动向 Location 报头提供的那个 URL 发起第四个 HTTP 请求。这个过程不需要获得 Alice 的许可，浏览器仅仅执行了下面的操作：

```
GET /messages/5266722824890167 HTTP/1.1
```

这一次，浏览器返回了 200（“OK”）状态码以及一个 HTML 文档：

```
HTTP/1.1 200 OK
Content-type: text/html

<!DOCTYPE html>
<html>
    <head>
        <title>Message</title>
    </head>
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

```
        <body>
            <div>
                <h2>Message</h2>
                <dl>
                    <dt>ID</dt><dd>2181852539069950</dd>
                    <dt>DATE</dt><dd>2014-03-28T21:51:08Z</dd>
                    <dt>MSG</dt><dd>Test</dd>
                </dl>
                <p class="links">
                    <a href="http://www.youtypeitwepostit.com/">Home</a>
                </p>
            </div>
        </body>
    </html>
```

Alice的浏览器以可视化的方式显示了这个文档(见图1-6),然后就继续等待Alice的输入。

图1-6 You Type It… 所提交的消息

我可以肯定你之前一定遇到过 HTTP 重定向，HTTP 协议就包含了很多类似于 HTTP 重定向的细微的功能特性，其中有些可能你都没有接触过。我们有很多方法来让服务器告诉客户端如何采用不同的方式来处理一个响应，我们也有很多方法来让客户端将条件或者额外的功能特性添加到一个请求中。API 设计的很大一部分工作内容就是恰当地使用这些功能特性。第 11 章介绍了 HTTP 中对于 web API 非常重要的一些功能特性，并且附录 A 和附录 B 提供了关于这一主题的补充信息。

通过浏览图形化页面，Alice 看到她提交的消息（“Test”）现在已经是 YouTypeItWePostIt.com 网站上面一个正式的帖子了。我们的故事到此就结束了。Alice 已经完成了她的试用这个微博网站的目标，但是在这四次简单的交互中，还是有许多需要学习的内容。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

应用状态（Application State）

图 1-7 是一个状态图，它从浏览器的角度展示了 Alice 的完整的探索之旅。

11

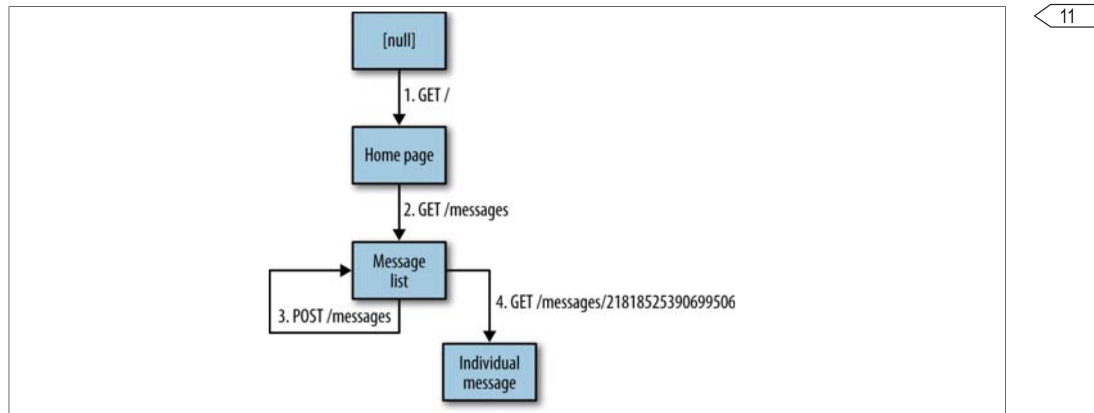

图1-7 Alic的探险: 客户端的角度

当 Alice 刚启动她手机上的浏览器的时候，浏览器没有加载任何页面，界面上还是一片空白。然后，Alice 输入了一个 URL 并且通过 GET 请求将浏览器带到了网站的主页。Alice 单击了一个链接，第二个 GET 请求将浏览器带到了消息列表页面。她提交了一个表单，这触发了第三个请求（一个 POST 请求）。对应的响应是一个 HTTP 重定向，这个重定向是由浏览器自动做出的。最后，Alice 的浏览器停留在了一个显示有 Alice 发布的消息的页面。

这张图里面的每个状态都对应到 Alice 的浏览器窗口打开的一个特定的页面（或者没有页面）。在 REST 的世界里，我们将这些信息（比如你停留在哪个页面？）称为*应用状态*（application state）。

当你上网的时候，你从一个应用状态转换到另一个应用状态，这个过程都对应于你单击的一个链接或者提交的一个表单。不是所有的状态之间的转换都是可用的。Alice 不能在主页直接提交一个 POST 请求，因为主页没有提供让浏览器生成 POST 请求的表单。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

资源状态（resource state）

图 1-8 是从 web 服务器的角度展示 Alice 的探索之旅的状态图。

12

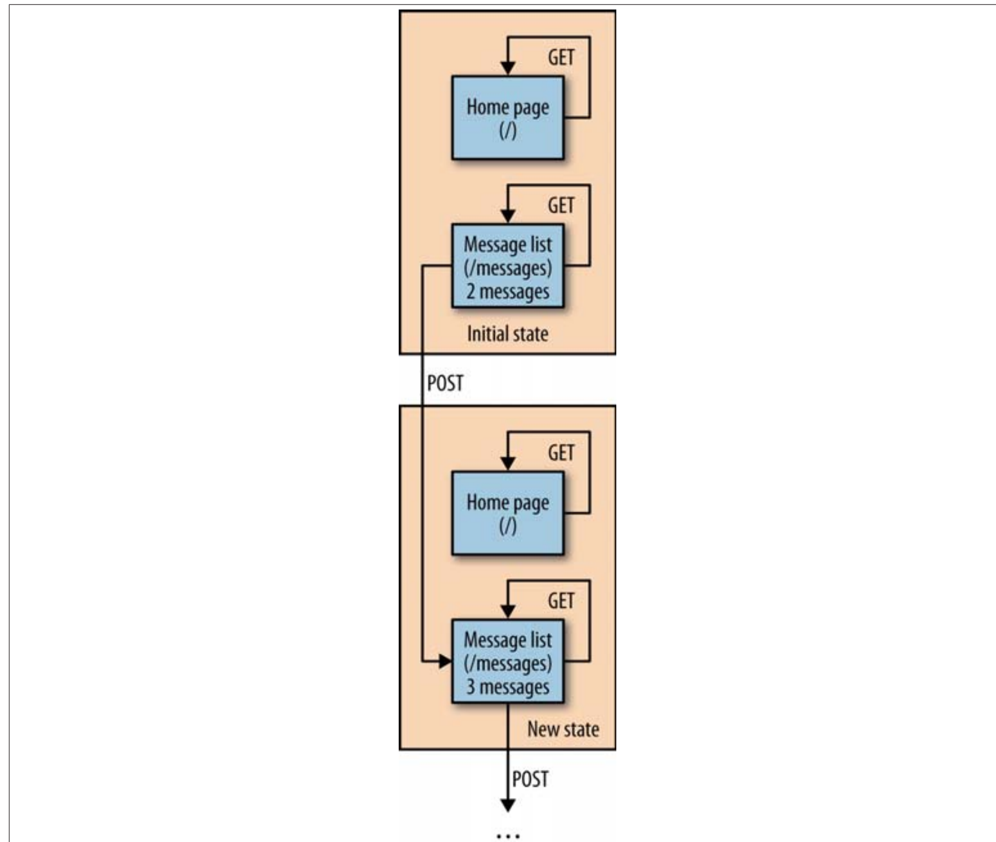

图1-8 Alice的探险：服务器的角度

服务器管理了两个资源：主页（通过“/”提供服务）和消息列表（通过“/messages”提供服务）。（服务器同时也将每条单独的消息作为资源进行管理，我为了简化状态从图中省略掉了这些资源。）简单来说，这些资源的状态就叫作*资源状态*。

故事开始的时候，消息列表中有两条消息：“Hello”和“Later”。浏览器向主页发送 GET 请求不会改变资源状态，因为主页是一个永远不会变化的静态文档。浏览器发送 GET 请求到消息列表页也同样不会改变资源的状态。

但是，当 Alice 向消息列表发送了一个 POST 请求的时候，这个请求将服务器切换到了一个新的状态。现在消息列表包含了三条消息：“Hello”、“Later”以及“Test”。现在已经没有办法回到原来的状态了，但是这个新的状态和原来的状态还是很相似的。和以前

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

一样，向主页和消息列表发送 GET 请求不会引起任何变化，但是向消息列表发送一个新的 POST 请求会使得增加第四条消息到列表里面。

由于 HTTP 会话非常短，所以服务器不知道客户端的应用状态的任何信息。客户端也不能直接控制资源状态——所有的资料都保存在服务器端。 然而，网站却正常运行着。网站是通过 REST- 表述性状态移交（representational state transfer）工作的。 13

应用状态保存在客户端，但是服务器端可以通过发送表述（representation）——HTML 文档来操纵它。在这种情况下，提交的这个表单描述了可能的状态转换（state transition）。资源状态是保存在服务器端的，但是客户端可以通过向服务器发送表述（representation）——提交一个 HTML 表单来操作它，在这种情况下，这个提交的表单描述了客户端所期望的新的状态。

连通性（connectedness）

在这个故事中，Alice 向 YouTypeItWePostIt.com 发送了 4 个 HTTP 请求，并且获得了 3 个 HTML 文档作为返回结果。尽管 Alice 没有一一单击这些文档的所有链接，但是我们可以使用那些链接来从客户端的角度构建一幅粗略的网站地图。

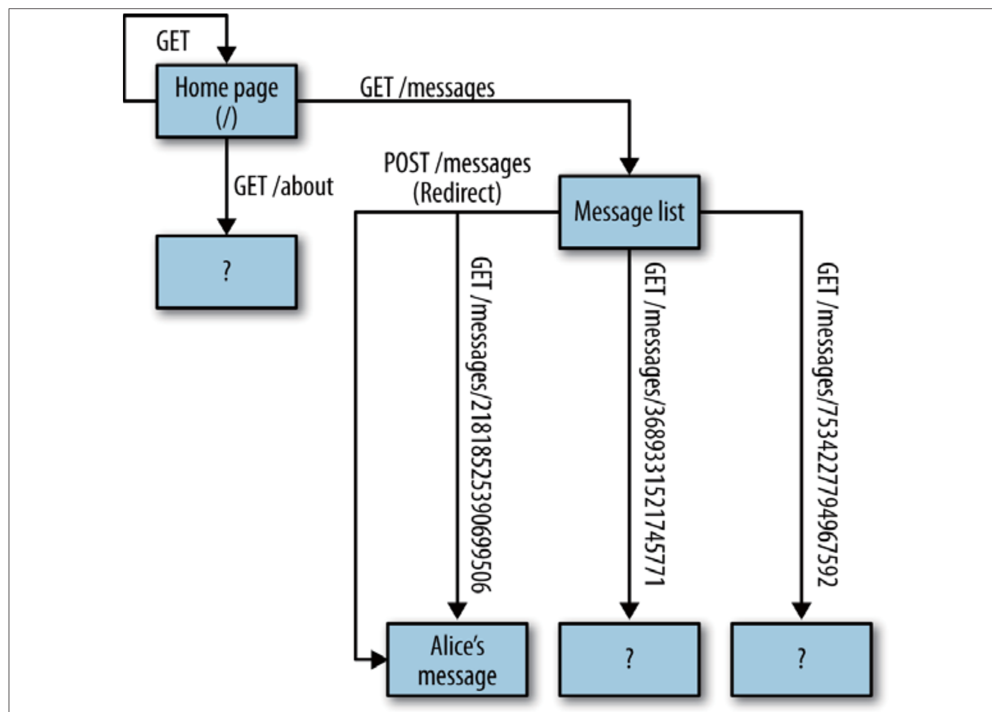

图1-9 客户端所看到的事物

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

这是一个由 HTML 页面组成的网。这个网间的连线是 HTML<a> 标签和 <form> 标签，它们分别描述了 Alice 可能会发起的各个 GET 或者 POST 的 HTTP 请求。我将此称为连通性原则：每个网页会告诉你如何获取相邻的网页。

14

网络作为一个整体按照连通性原则运转，这个原则更为人所熟知的叫法是“将超媒体作为应用状态引擎（hypermedia as the engine of application state)”，有时候简写为 HATEOAS 。我更倾向于“连通性”或者“超媒体约束”，因为“将超媒体作为应用状态引擎”听起来比较吓人。但是现在，你应该没有理由害怕了。你现在已经知道什么是应用状态——简单说就是客户端当前处在哪个网页上面。超媒体是对类似于 HTML 链接、表单等的事物抽象出来的通用术语，服务器端可以通过这种技术来向客户端说明下一步的操作。

我们所说的超媒体是应用状态的引擎，其实就是说我们都是通过填写表单以及访问各种链接来浏览 Web 的。

与众不同的Web

Alice 的故事看起来并不怎么吸引人，因为万维网在过去的 20 年间已经成为最主流的互联网应用。但是在 20 世纪 90 年代，这是一个非常令人激动的故事。如果你将万维网和其他早期的竞争者相比较的话，你会发现它们的不同之处。

Gopher 协议（在 RFC 1436 中定义）看起来很像 HTTP，但是它缺少可寻址性。它没有一个简洁的方式来标识 Gopherspace 里面的一个特定文档。最起码在万维网为 Gopherspace 考虑和发布 URL 标准（首次定义在 RFC 1738）之前并不存在这种寻址的能力。这个后来发布的 URL 标准提供了和 *http://* 类似的 *gopher://* 的 URL 方案。

FTP，这个在 web 出现之前用于文件传输的非常流行的协议（RFC959 中定义）也缺乏寻址性。在 RFC1738 定义 *ftp://* 这样的 URL 方案之前，根本没有一个机器可读的方案可以指定某个 FTP 服务器上面的一个文件。你不得不长篇大论地来解释这个文件到底在哪里。为了定位服务器上的一个文件就要花费很大的精力，这是极大的浪费。

FTP 也是一个长会话的协议。一个普通的用户可以登录到 FTP 服务器上并无限期地占用服务器的一个 TCP 连接。与之对照的是，就算是一个“持久性”的 HTTP 连接最多也不会占用一个 TCP 连接超过 30 秒。

20 世纪 90 年代见证了太多的用于检索不同类型的文档和数据库的互联网协议，像 Archie、Veronica、Jughead、WAIS 和 Prospero。但是最终证明，我们不需要所有的这些协议。我们仅仅需要的是能够向不同类型的网站发送 GET 请求。所有的那些协议都渐

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

渐消亡了或者被网站取而代之。它们那些复杂的特定协议的规则都合并为统一的 HTTP GET。

一旦 Web 接管了这些，就更难以证明创建新的应用协议的合理性了。为什么要在你可以搭建一个每个人都可以使用的网站的时候，开发一种只有计算机专业人员才能理解的工具呢？所有成功的后 Web 协议（post-Web protocols）都是在做一些 Web 做不了的事情：P2P 协议比如 BitTorrent，实时协议比如 SSH。对于大部分的目的，HTTP 已经足够使用了。

Web 的这种空前的灵活性都来源于 REST 原则。在 20 世纪 90 年代，我们已经发现 Web 比其他的竞争者工作得更好。在 2000 年，Roy T. Fielding 的博士论文[注1] 解释了其中的奥秘，并将其精简为“REST”这个术语。

15

Web API 落后于Web

Fielding 的文章也花了很大篇幅解释 21 世纪初的许多 Web API 的问题。我前面介绍的那个简单网站比当今大部分网站部署的 Web API 或者自称的 REST API 要精致得多。如果你曾经设计过 Web API，或者为这些 API 写过客户端程序，你肯定遇到过一些下面的问题。

- Web API 经常有大量的阅读文档来告诉你如何为不同的资源构造 URL。其实这就像花很大篇幅来告诉你如何在 FTP 服务器上找到一个指定的文件一样。如果网站是这样做的，没有人会愿意使用这个网站。
  和每次告诉你要往浏览器输入什么 URL 不同，Web 通常都是在 <a> 和 <form> 这样的超媒体控件中嵌入 URL，你可以通过单击链接或者提交表单来激活它。
  在 REST 的世界中，将有关构造 URL 的信息放到单独的阅读文档中违背了连通性和自描述信息的原则。
- 许多网站都有帮助文档，但是你上次阅读这些文档是什么时候的事情了？除非有很严重的问题（比如你想要提交一些信息，但是所有的尝试都以失败告终），更简单的方法是随便点击网站的一些链接，并通过浏览这些服务器发送给你的这些互相连接的、能够自我描述的 HTML 文档来确定网站是如何运作的。
  而如今的 API 呈现资源的方式更像是一个巨型的选项菜单，而不是一张相互连通的网。这使得很难了解资源之间的相互影响。
- 要集成一个新的 API 不可避免地需要编写新的定制化软件，或者安装别人编写的一次性的代码库。但是当你要访问一个新的网站的时候，你并不需要为此而编写任何的定制化的软件。你看到广告牌上的一个 URL，你直接将这个 URL 输入到你的浏览器就可以了，对于世界上所有的网站而言，你都可以使用同一个浏览器

注 1　Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. 博士论文，加利福尼亚大学欧文分校，2000.

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

来访问它们。

16

我们不可能做到让一个 API 客户端能理解世界上所有的 API，但是当今很多的客户端包含着许多实际上应该进行重构成为通用库的代码。这只有在 API 提供了自描述的表述的前提下才有可能实现。

- 当 Web API 发生了变化以后，定制化的 API 客户端就不能正常使用了，并且需要维护者为此进行一些代码修复。但是当网站经过重新设计改版以后，用户可能会抱怨一段时间，然后慢慢适应新的版式。他们的浏览器在此期间不会停止工作。在 REST 的世界中，网站的改版是封装在服务器提供的自描述的 HTML 文档中的，所以，一个能理解旧版的文档的客户端也是能够理解新的版式的。

这些就是本书中试图去解决的一些问题。好消息是，在以前，这样的情况更加严重和恶劣。在以前，用不安全的方式使用安全的 HTTP 方法设计 REST API 或者将应用状态和资源状态混合使用的情况比比皆是。现在这些情况已经比较少见到了。现在的设计已经好多了，但是还可以更好。

语义挑战

现在该讲一下负面的消息了。正如文章前面的故事——Alice 浏览网站的故事，Alice 访问网站的过程非常顺利，这要归功于一个运行速度很慢但是又非常昂贵的硬件：Alice 本人。每次浏览器显示了一个网页，Alice，这个人类就会去浏览这个渲染过的页面，然后决定下一步的操作。Web 就是在人类不断地决定单击哪个链接以及填写哪个表单的过程中运行着。

Web API 的目标是在没有人类参与的前提下完成相应的工作。但是我们该如何编写程序让计算机来决定单击哪个链接呢？计算机可以解析 HTML 标记 `<a href="/messages">Get started</a>`，但是它并不能理解“Get started”这个词组。如果提供的自描述信息不能被软件理解，我们又何苦设计这种提供自描述信息的 API 呢？

Web API 设计的最大的挑战就是：消除“理解文档的结构”和“理解文档的含义”之间的语义鸿沟（semantic gap）。简单来讲，我将其称为是：*语义挑战*（semantic challenge）。现在，这方面的进展非常小，我们也不可能完全解决它。好消息是到现在为止，正是由于这方面的研究进展很少和有限，所以做出一些成绩还是比较容易的。我们现在要做的是开始一起工作，而不是重复对方的工作。

因为我谈到了 Web 的技术以及如何将这些技术应用到 API 设计，所以我会在后面的几章里对语义挑战方面的内容进行介绍。在第 8 章，我们会使用一些必要的工具来直接处理语义挑战的问题。

互动出版网 China-Pub.com
专业图书网上第一专营店
www.china-pub.com 独家提供样章

第2章 一个简单的API

在第1章中，我展示了一个非常简单的微博网站，它的网址是 *http:// www.youtypeitwepostit.com/*。与此同时，我也为该网站设计了一个可编程的API，你可以亲临 *http://www.youtypeitwepostit.com/api/* 来访问它。

这个理想中的API具备了一些似曾相识的特性，而正是这些特性使得互联网简单易用。作为一名开发者，仅仅凭借在广告牌上看到的URL，你便可以理解如何使用它。

让我们将这个想象的场景继续延伸下去，同时来看看API是如何工作的。首先，假设你使用你的可编程客户端程序向广告牌上的URL发起了一次GET请求——这等效于将URL手动敲入web浏览器地址栏。至此你的客户端程序将开始接管通信，并对响应中的可用选项进行检查。它随后将可能访问响应中的链接（不一定是HTML链接）、填充表单（不一定是HTML表单）并最终完成你分配给它的任务。

本书并不会带领大家完全达成上述的理想目标。有些问题是我无法在一本书内解决的，比如：围绕因为标准缺失而产生的相关问题、工具支持方面现有水平的局限性问题以及计算机无法像人类般聪明的残酷事实。但是我们会一起向着这个目标进行深入地探索——其深度将远远超乎你的想象。

我已经说过，我们已经有了一个实实在在的微博API，它位于 *http://www.youtypeitwepostit.com/api/*。如果你是个具有探索精神的家伙，请尝试着编写一些代码来使用该API做些什么。看看在只知道这个URL的情况下，你能够对该API理解到什么程度。其实你在浏览各种网站时曾经完成过相同的事：你开始的时候所知道的全部信息仅仅是主页的URL，而你却可以由此逐渐完全理解这个网站。而如今只通过一个API，你又可以走多远呢？

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

如果你不愿意冒险又或者在为 web API 编写客户端程序方面没有太多经验的话（如果你在很久的未来才来阅读本书，我很可能已不再托管这个网站），我们将一起来完成这项工作。而第一步就是要获取该 API 首页的表述。

HTTP GET：安全的投注

18

如果你得到了一个以 *http://* 或 *https://* 开头的 URL，但是你并不知道该 URL 的另一头是什么，你首先可以做的就是向它发起一个 HTTP GET 请求。在 REST 的世界里，你除了知道指向资源的 URL 以外其他一无所知。你需要进一步去发现你的可选项，这意味着你需要从该资源处获取一个表述。这便是 HTTP GET 的作用所在。

你可以通过使用某种编程语言编写代码来发起 GET 请求，但是当我们只是对某个陌生 API 进行初步探测时，采用像 Wget 这样的命令行工具通常会更容易些。在下面的示例中，我使用了 -S 和 –O 两个参数选项，前者将打印出来自服务器的完整的 HTTP 响应内容，而后者将以打印出响应文档内容的方式替换原来将响应内容保存为文件的方式。[注1]

```
$ wget -S -O - http://www.youtypeitwepostit.com/api/
```

上面的命令将向服务器发起一个如下的请求：

```
GET /api/ HTTP/1.1
Host: www.youtypeitwepostit.com
```

HTTP 规范告诉我们，GET 请求是为了获取一个表述而作的一次请求。该种类型的请求主观上并没有去改变服务器上资源状态的意图。这就是说如果你得到了一个 URL 而没有更多信息可供参考时，你总是可以向它发起一个 GET 请求从而得到一个资源表述作为回报。你的 GET 请求将不会造成如删除所有数据这样的破坏性效果，因此我们说 GET 是一个安全的方法。

当然我们也允许服务器在处理 GET 请求时改变一些附属性的事物，例如递增计数器或将请求日志记录到文件中，但是这并非是 GET 请求原本的用途。没有人会仅仅为了递增计数器而去使用 HTTP 请求。

而在现实生活中，我们并不能保证 HTTP GET 请求是安全的。一些较早的设计会强制你使用 HTTP GET 请求来删除数据，但是这种糟糕的特性在新近的设计中相当少见。大部分的 API 设计者现在都能理解：客户端之所以频繁向 URL 发起 GET 请求只是为了看看

注 1 –O 参数在 wget 帮助文档中的描述是将响应内容写入指定的文件，这与作者的描述看似不符。但是当没有指定输出文件名时，wget 会将内容输出到默认输出流，即命令行控制台。若不指定 –O，wget 会根据请求的资源名称生成默认的文件，并将响应内容写入该文件，所以这与作者的描述的效果是一致的。——译者注

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

该 URL 背后的内容是什么。在设计时给 GET 请求赋予重大的副作用是不合理的。

如何读取HTTP响应

为了响应我发起的 GET 请求，服务器发送了如下所示的数据块：

```
HTTP/1.1 200 OK
ETag: "f60e0978bc9c458989815b18ddad6d75"
Last-Modified: Thu, 10 Jan 2013 01:45:22 GMT
Content-Type: application/vnd.collection+json

{ "collection":
  {
    "version" : "1.0",
    "href" : "http://www.youtypeitwepostit.com/api/",
    "items" : [
    { "href" : "http://www.youtypeitwepostit.com/api/
messages/21818525390699506",
          "data": [
           { "name": "text", "value": "Test." },
           { "name": "date_posted", "value": "2013-04-22T05:33:58.930Z" }
          ],
          "links": []
        },

    { "href" : "http://www.youtypeitwepostit.com/api/messages/3689331521745771",
      "data": [
       { "name": "text", "value": "Hello." },
       { "name": "date_posted", "value": "2013-04-20T12:55:59.685Z" }
      ],
      "links": []
    },

    { "href" : "http://www.youtypeitwepostit.com/api/messages/7534227794967592",
      "data": [
       { "name": "text", "value": "Pizza?" },
       { "name": "date_posted", "value": "2013-04-18T03:22:27.485Z" }
      ],
      "links": []
    }
    ]
  },
  "template": {
    "data": [
       {"prompt": "Text of message", "name": "text", "value":""}
    ]
  }
}
```

19

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

通过以上内容我们能获悉到哪些信息呢？首先，每一个 HTTP 响应可以被分成 3 个部分：

状态码，有时我们也称它为响应码

这部分是由三位数字组成的，它简要说明了请求目前的进展。响应码是 API 客户端从响应中最先看到的信息，它奠定了响应剩余部分的基调。以上的示例中，我们看到的状态码是 200（OK），这是客户端所期盼的——这意味着一切进展顺利。

在附录 A 中，我对所有的 HTTP 响应码进行了说明，同时还进行了一些有价值的扩展。

实体消息体（entity-body），有时我们也称它为消息体（body）

这部分是一个采用某种数据格式书写成的文档，并且我们预期该文档是可以被客户端所理解的。如果你将 GET 请求理解成为获取表述而发起的一次请求，那么你可以将实体消息体理解为你最终得到的表述（严格来说，整个 HTTP 响应都是“表述”，但是重要的信息通常都记录在实体消息体中）。

20

在本例中，实体消息体是响应中最后较大那部分文档，其中充斥着很多花括号。

响应报头

响应报头的发送顺序排在状态码和实体消息体之间，通常是一系列用于描述实体消息体和 HTTP 响应的键值对。在附录 B 中，我将对所有的标准 HTTP 报头进行说明，并做了一些有助于理解的延伸。

最重要的 HTTP 报头是 Content-Type，它向 HTTP 客户端说明了如何去理解实体消息体。因为它非常重要，所以它的值都具有特定的名称。我们将 Content-Type 报头的值称为实体消息体的*媒体类型*（media type）（它同样也可以被称为 MIME 类型或 *content type*，有时“media type”是可以带有连接符的：*media-type*）。

就平时人们通过浏览器就能看到的 Web 而言，最常见的媒体类型是 `text/html`（针对 HTML）以及图片类型，例如 image/jpeg。而在此例中的媒体类型，你或许闻所未闻：`application/ vnd.collection+json`。

JSON

如果你是一个 web 开发人员，你或许已经认出该实体消息体其实是一个 JSON 文档。如果你没有，以下是一段为你准备的关于 JSON 的快速介绍。

JSON，参见 RFC 4627 中的表述，它是一种使用普通文本来表示简单数据结构的标准。它使用双引号来描述字符串：

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

```
"this is a string"
```

它使用方括号来描述列表：

```
[1, 2, 3]
```

它使用花括号来描述对象（键值对的集合）：{ "key" : "value" }

JSON 数据看上去非常像 JavaScript 或 Python 的代码。JSON 标准将约束建立在普通文本之上，它认为一个像 It was the best of times 这样的光秃秃的字符串是不能被接受的，即便任何人都能看到并理解它。要想成为合法的 JSON 数据，字符串必须用双引号括起来："It was the best of times。"

Collection+JSON

21

这么说来，该实体消息体文档就是 JSON 格式了，对吗？请别着急！你可以将这个文档交由 JSON 解析器来顺利地进行解析，但是这并不是 web 服务器希望你所做的。以下是服务器对你说的：

```
Content-Type: application/vnd.collection+json
```

这便与 JSON RFC 所描述的不符了，JSON 文档应该以 application/json 的类型提供，像这样：

```
Content-Type: application/json
```

那么 `application/vnd.collection+json` 又是何方神圣？很明显，这种格式也是基于 JSON 的，因为它看起来很像 JSON 并且它的媒体类型名中也含有"json"的字样。那它到底是什么呢？

如果你尝试在 web 中搜索"`application/vnd.collection+json`"，你将会发现它是一种注册为 Collection+JSON[注 2] 的媒体类型。当你向 *http:// www.youtypeitwepostit.com/api/* 发起一个 GET 请求时，你不会得到任何的 JSON 文档——你得到的其实是一个 Collection+JSON 文档。

在第 6 章中，我将会详细地讨论 Collection+JSON，而眼下先做个简短的介绍。Collection+JSON 是一个用于在 Web 上发布资源的可搜索列表的标准。JSON 将约束建立在普通文本之上，而 Collection+JSON 则将约束建立在 JSON 之上。服务器不仅能提供像 `application/vnd.collection +json` 这样的任意 JSON 文档，它也可以只提供 JSON 对象：

注 2　Collection+JSON 是一种在该页中定义的个人标准 (*http://amundsen.com/media-types/collection*)。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

```
{}
```

但是又不仅仅是一个对象，这个对象还必须拥有一个称为 collection 的属性，该属性可以映射到另一个对象：

```
{"collection": {}}
```

而该“collection”对象应该具有一个称为 `items` 的属性，该属性映射到了一个列表：

```
{"collection": {"items": []}}
```

在“items”列表属性中的项目也必须是对象：

```
{"collection": {"items": [{}, {}, {}]}}
```

如此反复，约束相承。最终你将会得到一个如你所见的高度格式化的文档。它的开始部分如下：

22

```
{ "collection":
  {
    "version" : "1.0",
     "href" : "http://www.youtypeitwepostit.com/api/",
     "items" : [

     { "href" : "http://www.youtypeitwepostit.com/api/messages/21818525390699506",
       "data": [
       { "name": "text", "value": "Test." },
       { "name": "date_posted", "value": "2013-04-22T05:33:58.930Z" }
      ],
      "links": []
    },
...
}
```

通过总览这个文档，所有这些约束的用途变得逐渐清晰起来。Collection+JSON 是提供列表的一种方式，我指的列表并非是标准 JSON 也可以提供的那种列表数据结构，而是一种描述 HTTP 资源的列表。

`collection` 对象拥有一个 `href` 属性，而它的值是一个 JSON 字符串。但是它不仅仅是一个字符串——它是我向其发起 GET 请求的 URL 地址：

```
{ "collection":
  {
    "href" : "http://www.youtypeitwepostit.com/api/"
  }
}
```

Collection+JSON 标准将该字符串定义为“用于获取一个文档表述的地址”（换句话说，

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

它是 collection 资源的 URL）。collection 的 items 列表中的每个对象都有其自己的 `href` 属性，并且每个值都是一个包含了一个 URL 的字符串，比如 *http://www.youtypeitwepostit.com/api/messages/ 21818525390699506*（换句话说，列表中的每一项都代表了一个拥有 URL 的 HTTP 资源）。

一个没有遵守以上规则的文档将不能算是一个 Collection+JSON 文档：它只能算是某种 JSON。通过遵守 Collection+JSON 的约束，你将获得使用资源和 URL 这些概念的能力。在 JSON 中只能使用像字符串和列表这样的简单元素，而以上这些概念在 JSON 中是没有定义的。

向API写入数据

我该如何使用 API 来向微博发布一条消息呢？以下是 Collection+JSON 规范里所描述的：

> 如果想要在 collection 中创建一个新的 item 项，客户端首先要使用**模板**对象来组装一个有效的 `item` 表述，然后使用 HTTP POST 来向服务器发送该表述以获得处理。

以上所述并不能算是一个按部就班的描述，但是它指明了答案的方向。Collection+JSON 的工作方式与 HTML 相似，在 HTML 中，服务器向你提供了各种表单（就比如 Collection +JSON 中的模板），你可以填充这些表单来创建文档，然后使用 POST 请求来向服务器发送该文档。

23

再次说明，第 6 章将会具体讨论 Collection+JSON，而随后我只会快速地介绍下相关的内容。我们再来查看下我早前向大家展示的那个大对象。它的 `template` 属性便是在 Collection+JSON 规范中提及的“**模板**对象”。

```
{
 ...
 "template": {
   "data": [
      {"prompt": "Text of message", "name": "text", "value":""}
   ]
}
```

我们来填写这个模板，我将 `value` 对应值中的空白字符串替换成了我想要发布的内容：

```
{ "template":
  {
    "data": [
     {"prompt": "Text of the message", "name": "text", "value": "Squid!"}
    ]
  }
}
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

随后我将这个填充完毕的模板作为 HTTP POST 请求的一部分进行发送：

```
POST /api/ HTTP/1.1
Host: www.youtypeitwepostit.com
Content-Type: application/vnd.collection+json

{ "template":
 {
  "data": [
   {"prompt": "Text of the message", "name": "text", "value": "Squid!"}
  ]
 }
}
```

（请注意，我请求中的 `Content-Type` 是 `application/vnd.collection+json`。该模板在填充后仍是一个有效的 Collection+JSON 文档。）

服务器做出了应答：

```
HTTP/1.1 201 Created
Location: http://www.youtypeitwepostit.com/api/47210977342911065
```

此处的 201 响应码（`Created`）相比 200（OK）稍显特殊；它表示一切进展顺利并且在本次响应中已针对我当次请求创建了一个新的资源，而 Location 报头给出了新生资源的 URL。

24

在第 1 章中，Alice 曾使用 web 界面在微博网站上发过帖子。而我现在已经成功使用该网站的 web API 完成了相同的事情。

HTTP POST:资源是如何生成的

为了向 collection 添加一个新的 item，你向 collection 的 URL 发送了一个 POST 请求。这里所涉及到的不仅仅是 Collection+JSON 是如何工作的内容，这里还涉及到了 HTTP 中有关 POST 的一个基本事实。RFC 2616，即 HTTP 规范中是这样描述 POST 的：

> 我们将 POST 设计为一个具备如下功能的统一方法：
>
> - 对现有资源的注解。
> - 向布告栏、新闻组、邮件列表或类似的文章组发布消息。
> - 向数据处理流程提供例如表单提交结果的数据块。
> - 通过追加操作来扩充数据库。

其中提到的第二个重点功能，“向布告栏、新闻组、邮件列表或类似的文章组发表消息”便准确地覆盖到了微博。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

我所发出的 POST 请求看上去非常像一个 HTTP 响应，因为它具有 Content-Type 报头和实体消息体。虽然我在早前所展示的 GET 请求中没有提供任何 HTTP 报头，但事实上任何 HTTP 请求都可以具有报头，甚至有好几个报头（比如 Accept）对于 GET 请求来说是非常重要的。后续我将会在这些特别重要的 HTTP 报头出现的时候进行重点讨论，但是请务必去附录 B 查阅标准 HTTP 报头的完整列表。

让我们继续往下走。再次回顾，以下是我通过 POST 请求获取的响应 ：

```
201 Created
Location: http://www.youtypeitwepostit.com/api/47210977342911065
```

当你收到一个 201 响应码时，后面的 Location 报头将会告诉你应该去何处找寻你刚才所创建的内容。RFC2616 详细说明了 201 响应码和 Location 报头的含义，但是为了清晰起见，Collection+JSON 规范同样也提及了这些内容。

如果我发送了如下的 GET 请求 ：

```
GET /api/47210977342911065 HTTP/1.1
Host: www.youtypeitwepostit.com
```

我将会看到熟悉的面孔 ：

```
HTTP/1.1 200 OK
Content-Type: application/vnd.collection+json
{ "collection":
  {
    "version" : "1.0",
    "href" : "http://www.youtypeitwepostit.com/api/47210977342911065",
    "items" : [

    { "href" : "http://www.youtypeitwepostit.com/api/messages/47210977342911065",
      "data": [
       { "name": "date_posted", "value": "2014-04-20T20:15:32.858Z" },
       { "name": "text", "value": "Squid!" }
      ],
      "links": []
    }
    ]
  }
}
```

25

我们通过一个完整的 application/vnd.collection +json 文档来表示这个单独的微博帖子。它是一个 collection，但是它的 items 列表只含有一个列表项。而这个经过填充的模板同样也是一个有效的 application/vnd.collection +json 文档，即使它根本没有使用 collection 属性。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

这是 Collection+JSON 的一个便利的特性。几乎文档中的所有内容都是可选的。这意味着你无须为了处理不同类型的文档而编写不同的解析器。Collection+JSON 使用相同的 JSON 格式来表示列表项、单独项、已填充的模板及搜索结果。

由约束带来解放

在 RESTful 设计中，一个违反直觉的经验就是：约束成为了一种解放手段。HTTP GET 方法的安全约束就是一个很好的例子。因为有了这个安全约束，当你还不知道可以对某个 URL 做些什么时，你总是可以先向它发送 GET 请求从而看看它的表述。就算最后发现这样做没有带来什么帮助，也不会产生任何糟糕的副作用，因为你所使用的是一个 GET 请求。这便是一个对解放的承诺，而唯一让这成为可能的原因是我们在服务器端制定了更加苛刻的约束。

如果服务器向你发送了一个普通文本文档，该文档中记录着 9，你将无法知晓它表示的是一个数字 9 还是一个字符串“9”。但是如果你得到的是一个 JSON 文档，那么你可以确信文档中所记录的 9 是一个数字。JSON 标准对文档的含义进行了约束，这样便为客户端和服务器进行有意义的交流提供了可能性。

在过去的若干年里，数以百计的公司都经历过以下的思考：

1. 我们需要一个 API。
2. 我们将使用 JSON 作为文档的格式。
3. 我们将使用 JSON 来发布我们的事物列表。

26

以上的三点想法都不错，但是不足以说明我们的 API 应该是什么样的。最终的结果就是数以百计的 API 表面上很相似（因为它们都是使用 JSON 来发布事物列表的！）但却无法完全兼容。对一个 API 的学习经验无法帮助客户端去理解下一个 API。

这意味着我们需要更多的约束，而 Collection+JSON 标准为我们提供了更多的约束。如果我选择使用自定义的 API 设计而非 Collection+JSON 的话，那么在我列表中的一个独立的项将可能看上去是以下这样的：

```
{
 "self_link": "http://www.youtypeitwepostit.com/api/messages/47210977342911065",
 "date": "2014-04-20T20:15:32.858Z",
 "text": "Squid!"
}
```

反之，如果我遵守了 Collection+JSON 的约束，那么一个独立的项看上去是这样的：

```
{ "href" : "http://www.youtypeitwepostit.com/api/messages/1xe5",
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

```
  "data": [
   { "name": "date_posted", "value": "2014-04-20T20:15:32.858Z" },
   { "name": "text", "value": "Squid!" }
  ],
  "links": []
}
```

自定义设计看上去确实更加简洁，但这并不非常重要——JSON 的压缩率原本就已经很好了。在换成 Collection+JSON 这种紧凑性稍逊的表述后，我却得到了大量更有用的特性：

- 我无须再向所有的用户说明 `href` 的值是一个 URL，而且我也无须对什么是 URL 进行说明。Collection+JSON 标准已经说明了一个 item 项的 `href` 包含了指向该 item 项的 URL。
- 我无须为了向用户说明 `text` 表示的是消息的文本内容而特地编写一个供人们阅读的独立文档。这些信息会直接展现在实际需要的地方——即出现在你在投递消息时所要填写的模板中。

  ```
  "template": {
    "data": [
      {"prompt": "Text of the message", "name": "text", "value": null}
    ]
  }
  ```

- 任何理解 `application/vnd.collection+json` 的代码库都可以自然而然地懂得如何使用我的 API。如果我选择了一种自定义的设计，那我将只能基于仅有的 JSON 解析器和 HTTP 代码库来编写全新的客户端代码，或者让我的用户们自行编写这些代码。

27

通过采纳 Collection+JSON 的约束，我从原本需要编写的一大堆的文档和代码中解放出来，同时我也将我的用户从学习一个接一个的自定义 API 中解放了出来。

应用语义所产生的语义鸿沟

当然，Collection+JSON 约束并不能约束所有的事情。Collection+JSON 并没有指定 collection 中的 items 应该是具有 `date_posted` 字段和 `text` 字段的微博帖子。这部分的工作是我完成的，因为我想为本书设计一个简单的微博实例。如果我选择以“烹饪书”来作为实例的话，我仍然可以使用 Collection+JSON，但是 items 的字段将可能换成是 `ingredients`（配料）和 `preparation_time`（准备时间）了。

我将这些设计中的额外信息称为*应用语义*（application semantic），因为它们在应用之间有着很大的差别。应用语义是引起我在第 1 章中所提及的语义鸿沟的原因。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

如果要设计一个真正的微博 API，我会选择远比 `text` 和 `date_posted` 更加复杂的应用语义。就该 API 自身来说，这没什么问题。但是时下有很多公司都在设计着各种微博 API，因此也冒出了很多的设计，可这些设计的应用语义是相互不兼容的，由此产生了很多不同的语义鸿沟。所有这些公司都在以不同的方式做着相同的事情，它们的用户需要编写不同的软件客户端来完成相同的任务。

了解到 Collection+JSON 无法处理语义兼容问题这一现实并不意味着我们不再需要使用 Collection+JSON，兼容性只是一个度的问题。从我们在 20 世纪 90 年代停止发明自定义的 Internet 协议并对 HTTP 进行标准化开始，我们已经向兼容性迈进了一大步。如果我们全都赞同以 JSON 文档提供服务，虽然从技术上来说这未必是个好主意，但是这样确实能缩窄我们的语义鸿沟，而标准化地采用 Collection+JSON 将会更进一步地改善这个问题。

如果微博 API 的发布者们可以走到一起并协议使用一组通用的应用语义，那么微博的语义鸿沟将荡然无存（这将可能涉及到 *profile*，我将会在第 8 章中讨论这块内容）。一旦我们共享更多的约束，我们将可以设计出更加兼容的接口，语义鸿沟也会变得更小，而我们的用户将受益更多。

28 也许你并不希望自家的 API 与对手的 API 进行互相协作，从而人为地扩大了语义鸿沟。但是在 API 差异化方面我们有比这种办法更好的方式。我在本书中的目标便是让你能专注于 API 中语义鸿沟藏身的部分，为填平鸿沟提供一些新的方式，因为从未有人尝试过这样做。