Please visit website: http://cxyroad.com

GET和POST请求到底有什么区别?
==========

在计算机网络世界中，HTTP是开发者们会频繁接触的一个应用层通信协议，其中的Get和Post方法更是像吃饭、喝水一样不可或缺，不过很多同学却对它俩有些误解，或者说没有真正理解HTTP。

比如GET请求没有POST请求安全，因为Get请求的参数都暴露在URL上了，任何人都能看见；而POST请求的数据被封装了起来，看起来更安全。还有同学可能会认为GET只能用来获取数据，POST只能用来提交数据；

但实际上这些都是误解。HTTP请求方法的设计初衷并不是围绕“安全性”来设计的，而是为了表达不同类型的动作和意图。而且根据实际情况，用炒锅煮饭、电饭锅炒菜也都是未尝不可的，没有绝对的好坏之分。

这篇文章就来深入探讨GET和POST的真正区别，纠正这些常见的误解。

HTTP的历史背景
---

在遥远的1989年，有一位名叫Tim Berners-Lee的英国物理学家和计算机学家的双料大能，有一天他提出了一个构想：通过一个东西将世界各地的信息联系起来，让人们可以方便地从一个文档跳转到另一个文档，他称这个东西为“统一资源定位符”（URL），这种方法称为“超文本传输协议”（HyperText Transfer Protocol，也就是HTTP），符合HTTP传输协议的文档称为“超文本文件”（HTML文件）。

#### HTTP解决了什么问题?

在HTTP被发明之前，互联网上的信息交换常用的方式有FTP、Telnet、Gopher、Usenet等，这些方式的用户体验、便捷性、信息组织方式和查询效率都不是很好，缺乏统一和标准化的方法来访问网络资源。Tim Berners-Lee面临的根本问题是：如何在不同的计算机系统之间高效、简便地共享和传递信息?

HTTP就是为了解决这个问题的。它通过定义一套规则，规定了客户端和服务器之间请求和响应的标准格式，使得文档（或者资源）的发布和接收变得简单明

了，也就是HTTP协议。HTTP的出现极大地促进了信息的共享，使得互联网从一个简单的、静态的信息集合，转变成了一个动态互联、内容丰富的全球性网络。

然后在HTTP的发展过程中，逐步定义了Get、Post、Put、Delete、Head、Option等多种资源操作方法，用来满足各种各样的文档访问需求。本文重点其中的Get和Post方法。

GET与POST的对比解析
----------

有了对HTTP的基本认识，我们再来看下Get和Post到底有何不同。

### Get和Post是什么?

GET和POST是HTTP中的两个最基本的请求方法。在HTTP协议的早期版本中，GET是最初定义的方法，主要用于请求文档；然后POST方法被引入，以支持更复杂的交互模式。在1996年的HTTP/1.0（RFC 1945）中首次明确定义了GET和POST方法。

我们先来看一个标准的说法：

* GET 主要用于请求访问已经被URI（统一资源标识符，URL是它的一个子集）识别的资源。它的设计目的是从服务器上获取数据，适用于信息的检索。
* POST 主要用于向指定的资源提交数据，请求服务器进行处理（例如，提交表单或上传文件）。数据被包含在请求体中。它的设计目的是允许用户向服务器传输数据，适用于更新或创建资源。

我们再用一个比喻来加深理解：

使用GET请求就像是去学校图书馆借一本书，你告诉图书管理员你需要的书名（通过URL的查询字符串传递参数），图书管理员找到书后将其递给你。

使用POST请求则像是你写了一张点餐单（扫码点餐流行之前一般手写），然后把它放进了一个信封交给服务员，服务员就去协调厨房制作菜品了，当然服务员还会把做好的菜给你送过来。

### 基础区别

我们可以从使用形式上对他俩进行一个基础的对比。

| **项目** | **Get请求** | **Post请求** |
| --- | --- | --- |
| ***\

，浏览器右下角时不时弹出一些广告小窗口，大小运营商抢着劫持大家的404页面，就是因为HTTP本身没有什么安全机制，网络中间人可以任意更改传输的HTML文件。

直到HTTPS的普及，这一严重威胁网络安全的行为才被遏制住，因为传输的数据被加密了，网络中间人看不懂被加密的HTML数据，也就没办法进行修改了。

HTTPS虽然以HTTP打头，但其实是HTTP和SSL/TLS的组合称呼，并不是HTTP本身的安全增强。在HTTP发送和接收数据之前，SSL/TLS会对数据进行加解密处理。

SSL的全称是Secure Sockets Layer，从这个名字我们就可以看出它是在套接字（Sockets）上工作的，套接字就是IP+端口号+(TCP/UDP)，是计算机网络传输层通信的端点，可以把SSL看成是HTTP（应用层）之下、TCP/UDP（传输层）之上的一个夹层。TLS的全称是Transport Layer Security，它是SSL的标准化版本，因为SSL是Netscape公司搞出来的，但是大家都想用，所以又发展出TLS来满足更为广泛和通用的需求。

有了HTTPS，无论是GET还是POST请求，数据在传输过程中都是加密的。即使是GET请求，虽然参数在URL中，但在网络传输过程中也是安全的，中间人也不能进行偷看。

有了HTTPS，我们还能确保数据在传输过程中未被篡改。如果数据被篡改，接收方的校验将会失败，从而知道数据中途被人篡改。

有了HTTPS，我们还能验证网站的身份，确保你访问的网站不是假冒网站。这是通过使用SSL/TLS证书来实现的，当你的浏览器访问一个HTTPS网站时，浏览器和操作系统会验证这个证书的真实性。如果证书验证通过，就说明这个网站是可信的。

安全性不仅仅取决于是否能看到数据，更重要的是数据的传输过程是否加密，数据的来源是否可信。对于数据的加密和校验方法，也很有意思，我会单独再开一篇文章进行讲解。

### 万能的Post

上文提到，Post的设计本意是用来向指定的资源提交数据的，提交表单或者上传文件这种。但是Post也没有限制我们不能用它来做资源的查询、删除等，我们完全可以在Post的Url中带上一些参数，让服务端做各种增删改查的操作，以

及接收Post请求返回的数据。从这一点上看，Post就是万能的。

因为Post的这种特性，很多私有的业务协议也会完全基于Post进行定义，在HTTP Body中定义自己的业务协议规范，服务端收到后按照协议规范进行解析，执行相关的业务处理。举个库存管理协议的例子：

```
<inventoryRequest>
  <operation>
    <!-- 操作类型定义 -->
    <type>Add</type> | <type>READ</type> | <type>UPDATE</type> |
<type>Remove</type>
  </operation>
  <data>
    <!-- 如果是Add操作，可能需要的属性 -->
    <product id="123">
      <name>Product A</name>
      <quantity>100</quantity>
    </product>

    <!-- 如果是查询操作，可能需要包含的查询条件 -->
    <query>
      <namePattern>Product.*</namePattern>
      <minQuantity>50</minQuantity>
    </query>
  </data>
</inventoryRequest>

```

服务端解析这份协议时，首先看操作类型operation，然后再根据类型解析data中的数据，最后按照协议中的操作类型进行相应的处理。

其实不仅很多的私有业务协议会完全基于HTTP Post进行构建，大名顶顶的gRPC也是基于HTTP的Post方法搞出来的。当然gRPC使用的HTTP和我们日常浏览网页使用的不太一样，gRPC使用的是HTTP/2，它具备长连接、多路复用、服务器推送、头部压缩等特性，这会带来更低的延迟、更少资源消耗和更高的吞吐量，可以满足gRPC的性能要求。同时gRPC的设计初衷是支持客户端和服务器之间的双向流式通信，而POST方法能够支持发送请求体（Body），这对于gRPC传输复杂的消息结构是必需的，再使用其它HTTP方法没有多少必要。

但是gRPC选择HTTP还有另一个考虑，大量的现有的网络基础设施（如代理、

负载均衡器、API网关等）已经支持或可以较为容易地适应HTTP，即使是HTTP/2。gRPC基于HTTP可以更容易地融入现有IT环境，利用已有的服务治理工具和中间件进行监控、路由、安全控制等。

回到大量私有业务协议使用HTTP也是类似的，HTTP操作简单，网络兼容性好，可以方便的集成到现有的系统环境中，而且Post可以完全的覆盖各种业务操作需求。

另外HTTP协议规范也没有明确禁止 GET 请求中使用 HTTP Body 来传输数据。从技术角度讲，GET 请求可以携带 Body。但是很多客户端库、服务器软件、中间件、浏览器以及开发工具可能并不支持或默认禁用 GET 请求中的 Body，所以大家没有用Get来搞事情。

---

最后，我们做个简单的总结：

GET和POST被设计用来满足不同的网络交互需求，GET用于数据检索，主要目的是获取资源；Post用于数据提交，主要目的是创建和更新资源。

虽然POST请求看起来“更安全”，但无论是GET还是POST，本质上并非围绕安全性设计，要安全的传输，应该始终使用HTTPS，过加密传输层，保护数据不被中间人窃取或篡改。

Post具备高度的灵活性，它不仅适用于提交数据，也能用于资源的查询、删除等操作。这种灵活性使得POST成为许多私有业务协议和高性能通信协议（如gRPC）的首选基础。尽管Get请求理论上也能携带Body，但实际应用受到很多限制。

萤火架构，提升技术认知！

原文链接: https://juejin.cn/post/7351682240747503653