概述

当我们在构建、测试、发布一套新的HTTP API时，包括我在内的大多数人都不知道他们所构建的每一个组件的复杂性和细微差别。

即使你对每一个组件都有深刻的理解，也可能会有太多的信息在你的脑海中出现。

以至于我们不可能一下把所有的信息进行梳理，形成成体系的API测试策略，下面我们就HTTP API测试将其checklist进行细化。

主要从以下四个方面进行:

- HTTP
- API设计
- 内容
- 安全
- 客户端
- 其他

HTTP

HTTP RFC(Request For Comments)相关文档规定了HTTP交互机制及参数选项，因此你需要了解相关RFC文档才能做好HTTP API的测试：

HTTP1.0: https://tools.ietf.org/html/rfc1945 HTTP1.1: https://tools.ietf.org/html/rfc7232 HTTP2: https://tools.ietf.org/html/rfc7540

在进行HTTP API测试时，以下协议选项或机制是应该考虑覆盖的：

1. HTTP方法的安全性和幂等性。Http协议规定了不同方法的安全特性和幂等特性，作为服务提供者的服务器必需为客户端提供这些特性。安全性，仅指该方法的多次调用不会产生副作用，不涉及传统意义上的"安全"，这里的副作用是指资源状态。即，安全的方法不会修改资源状态，尽管多次调用的返回值可能不一样(被其他非安全方法修改过)。幂等性，是指该方法多次调用返回的效果(形式)一致，客户端可以重复调用并且期望同样的结果。

HTTP方法的安全性和幂等性见下表：

方法	安全性	幂等性
GET	是	是
HEAD	是	是
OPTIONS	是	是
DELETE	否	是

PUT	否	是
POST	否	否

2. 认证，验证准确地实现HTTP身份验证，API应该提供一个401状态代码。

3. 状态码201 Created，使用“201 Created”标识响应代码来表示请求已成功处理，并导致创建新资源。201响应可以包括位置标头中的新资源URI。

4. 状态码202 Accepted，使用“202 Accepted”的响应代码来表示请求是有效的，并且将被处理，但是还没有完成。通常情况下，这是在服务器端后台处理队列的情况下使用的。

5. 状态码4xx与5xx状态代码之间有一个重要的区别:4xx和5xx的状态码:4xx代码是用来表示客户端错误的，而5xx代码表示服务器端错误。

6. 状态码410 Gone响应代码是一种未得到充分利用的响应代码，该代码通知客户端在该URL中使用的资源，但不再是。可以在您的API中使用它来表示已删除、归档或过期的项。

7. 状态码100-Continue——如果API客户端准备发送一个大型实体的请求，比如POST、PUT或补丁，他们可以在HTTP头中发送“Expect:100-Continue”，然后在发送实体实体之前等待“100继续”响应。这允许API服务器在浪费带宽返回错误响应(例如401或403)之前验证请求的有效性。支持这种功能不是很常见，但是它可以提高API的响应能力并在某些场景中减少带宽。

8. Connection Keep-Alive，为多个API请求维护与API服务器的连接可能是一个很大的性能改进。如果配置正确，几乎每个web服务器都应该支持保持生命连接。

9. HTTP Compression，HTTP压缩既可以用于响应主体(接受编码:gzip)，也可以用于请求主体(内容编码:gzip)来提高HTTP API的网络性能。

10. HTTP Caching, 在API响应中提供一个cache-control报头。如果它们不能缓存，“cache-control:no-cache”将确保代理和浏览器能够理解这一点。如果它们是可缓存的，则需要考虑各种因素，比如缓存是否可以由代理共享，或者资源是否"fresh"。

11. Cache Validation, 如果有可缓存的API，那么应该在响应中提供最后修改或ETag头，然后支持If-修饰性请求，因为有条件的请求。这将允许客户机检查它们的缓存副本是否仍然有效，并在不需要时阻止完整的资源下载。如果正确实现，可以使条件请求比通常的请求更有效，并节省一些服务器端负载。

12. Conditional Modifications, ETag头也可以用于支持资源的条件修改。通过在获取上提供一个ETag头，稍后的POST、补丁或删除请求可以提供一个if-match头来检查它们是否在更新或删除它们上次看到的相同状态的资源。

13. Absolute Redirects, 对http/1.1的重定向(例如:……201、301、302、303、307响应代码)应该包含位置响应头中的一个绝对URI。许多客户机在位置上支持相对uri，但是如果希望API与许多客户机广泛兼容，那么应该在任何重定向中使用绝对uri。

14. Link Response Header，在RESTful API中，即使响应的内容类型没有提供链接的自然方式(例如，PDF或图像表示)，通常也需要提供指向其他资源的链接。RFC5988指定了在响应头中提供链接的方法。

15. Canonical URLs, 对于具有多个URL的资源，RFC6596定义了提供规范URL链接的一致方法。

16. Chunked Transfer Encoding, 如果有大量的内容响应，则转换编码:分块是一种很好的方式来对客户端进行

响应。它将减少服务器和中间服务器的内存使用需求(特别是实现HTTP压缩)，并提供更快的第一个字节响应。

17. Error Handling in Chunked Transfer Encoding, 在执行并实现分块传输编码之前，要弄清楚如何处理在请求中出现的错误。一旦开始处理响应，就无法更改HTTP状态代码。通常，需要定义一种表示内容类型内的错误的方法。

18. X-HTTP-Method-Override, 一些HTTP客户端除了GET和POST以外什么都不支持;可以通过POST来隧道其他HTTP方法，并使用实际的标准x-HTTP-method-重写头来记录"真正的"HTTP方法。

19. URL Length，如果API支持复杂的或任意的过滤选项作为GET参数，请记住，客户端和服务器都可以在超过2000个字符的URL上存在兼容性问题。

API设计

学习和了解良好的API设计原则，有利于你深入对API的测试和验证，以确保API具备更好的可用性、安全性。

1. Statelessness， 始终保持应用服务器的状态，就可以轻松且毫无痛苦地扩展和维护了。

2. Content Negotiation, 如果想要支持资源的多个表示，您可以使用Content Negotiation（内容协商）(例如。接受标头)，或者不同的url的不同的url(例如。格式=json)，也可以将内容协商资源重定向到特定的格式。

3. URI Templates, URI模板是一种定义良好的机制，用于向客户端提供URL组合功能，或者将URL访问模式记录到终端用户。

4. Design for Intent，不要仅仅通过API公开内部业务对象。设计API具有语义意义，并匹配用户所使用示例。

5. Versioning, 从理论上讲，如果预先设计了一个非常棒的API，那么可能永远不需要在API中创建不兼容。对于我们中的实用主义者来说，在API url中放置版本控制(例如。a/v1/path)，这样就有了一个机制来控制和升级API了。

6. Authorization, 通过认证来控制哪些API是可以公开访问的，哪些API是必须认证后才能访问的，从而对API的访问权限进行控制管理

7. Bulk Operations, 如果能够减少发出更少的请求与服务器进行交互，那么批量操作的API是一个很不错的设计。

8. Pagination, 分页在API中有两个主要目的：一是减少了向客户机发送的不需要的数据量，并且减少了应用服务器上不必要的计算量； 二是更多不同的模式用于进行分页的收集资源;

9. Unicode， 统一的字符编码

10. Error Logging, 确保有日志机制记录API的错误，并将用户输入导致的错误与应用程序的错误分开记录

内容

1. Content Types， 要把内容类型进行详细的阐述可能需要一整本书。这里主要指出其重要性，在开发过程中，我们应该尽可能的复用标准或是经过实践检验的内容格式，例如Atom, Collection+JSON, JSON HAL,

or XHTML。

2. HATEOAS，作为应用程序状态引擎的超媒体是一个REST约束。简单地描述，这意味着你的内容应该通过链接和表单告诉客户它下一步要做什么。

3. Date/time，当在API中提供日期/时间值时，使用包含时区信息的格式要统一。

安全

1. SSL-考虑是否应该在HTTP和HTTPS下提供API，或者只使用HTTPS。HTTPS是一个越来越受欢迎的选择。

2. CSRF跨站点请求伪造，如果您的API接受您的交互用户使用的相同的身份验证配置，那么您可能很容易受到CSRF攻击。所以要有机制防CSRF攻击。

3. Throttling，确保一个API用户不能通过编写一个非常愚蠢的API客户端来降低您的系统性能。如果API用户超过了应该为他们提供的API请求限制，那么就给他们一个503的响应，并带有一个retry-header。

4. Subtle Denial of Service, 防DDoS攻击

客户端

为了确保我们提供的API能被用户有效的使用，我们应该为用户定制一些基本的规则，以防止API被用户滥用：

1. Connection Keep-Alive, 一些HTTP客户端库要求您做一些额外的工作来启用持久连接。持久的连接可能会对您的API的性能产生重大影响。

2. 401 before Authorization, 一些HTTP客户端库的另一个奇怪之处在于，通常需要一个“401未授权”的响应，然后才会用授权标头发出请求。这可以为您的API请求增加很多时间，特别是在移动网络中，高延迟将成为一个纠结的问题。

3. Expect: 100-continue, 至少有一个API客户端默认使用“Expect:100continue”;如果它没有接收到“100continue”响应，它将在3秒超时之后继续请求。如果您不支持“100continue”，最好在客户端禁用该能力，否则会导致服务性能的下降。

其他

1. Documentation, 编写API文档确实很无聊，但是手工编写的文档通常是最好的文档。一定要包含一些可运行的代码或curl命令行，以帮助用户尽可能快地上手。

2. Design with a Customer, 不要闷着头设计API，要尽可能多与你的用户交流、交互。

3. FeedBack，确保为API用户提供了一种方法，可以对API进行反馈。这可以通过你的支持渠道，也可以是一个托管论坛，也可以是一个邮件列表。尽量让用户不受摩擦的影响。

4. Automated Testing，API应该是为构建自动化测试所做的最简单的事情。毕竟，它是为自动化而制造的。利用它!

扫一扫关注微信公众号：