21 HTTP的高级篇 - HTTPClient（Java）

HttpClient API

HttpClient API是在2018年9月发布的Java 11中引入的。但是，它早在Java 9的前一年就已经可以作为预览功能使用。因为API需要一段时间的打磨才能变得成熟和完善。所以，从Java 11开始，HttpClient API是Java标准库的一部分。这意味着你不需要再向应用程序添加任何外部依赖关系即可使用此API。 HttpClient API替代了在Java标准库中存在了很长时间的HttpURLConnection API。稍后，看我心情要不要说一下为什么需要替换此API。与HttpURLConnection一样，新的HttpClient API支持HTTP 2和WebSocket通信。而且它还支持HTTP的早期版本。 HttpClient API的另一个重要功能是，它提供了同步阻塞和异步非阻塞的方法来执行HTTP请求。HttpClient API的设计目标是在常见情况下易于使用，但是在复杂情况下也具有足够的功能。

HttpClient

HttpClient API提供了三个重要的类型。所有这些类型都存在于java.net.http包中。首先，有HttpClient类本身。其中包含两个重要的方法：

- send->send方法执行对服务器的同步和阻塞调用。
- sendAsync->sendAsync方法执行异步的非阻塞调用。

你不能直接实例化HttpClient类。有一个newBuilder方法为你提供了一个构建器类。HttpClient API中的大多数类型都使用了这种构建器模式。

HttpRequest

发送方法的参数之一是HttpRequest。 HttpRequest包含你希望获得的所有信息，例如请求所针对的URI，可能需要与请求一起发送的HTTP headers，以及指示它是GET，PUT，POST还是其他HTTP的方法。与HttpClient相同，你不能直接构造HttpRequest。但是可以通过构建器来做（HttpRequest.Builder）。Builder总是返回不可变的对象。所以，一旦创建，无论是HttpRequest还是HttpClient 都无法更改了。有请求那就必然也会有响应。

HttpResponse

我们来看一下HttpResponse类型。除了URI，headers和statusCode，通常HttpResponse中最重要的部分是正文。这就是HTTP服务器返回的有效负载。了解了这三种类型，你就可以开始使用API来执行HTTP请求了。我们来看一个最简单的示例。

Hello World 小程序

从创建HttpClient实例开始。我们知道你可以使用构建器模式来执行此操作。但是，HttpClient上还有一个名为newHttpClient的静态方法，该方法将返回应用了所有默认设置HttpClient实例。我们的这个例子就以使用它开始。然后，我们需要创建一个请求。在这里，我们将构建器模式与HttpRequest.newBuilder方法一起使用。我们可以将URI传递给newBuilder方法，我们使用csdn的网址。默认情况下，HttpRequest构建器将向服务器构造一个GET请求。现在，我们只需调用build并返回一个HttpRequest即可。到目前为止，还没有执行实际的HttpRequest。我们只创建了Client和一个请求。现在我们需要发送一个请求。我们可以调用client.send方法并传递刚刚创建的HttpRequest。 send方法还有第二个参数，我们传入一个所谓的BodyHandler（不用理会，只是一个过客）。client.send方法返回一个HttpResponse对象，该对象包装成字符串并提供了有关很多响应的元数据。是不是很简单。代码如下

```
HttpClient httpClient = HttpClient.newHttpClient();
HttpRequest httpRequest = HttpRequest.newBuilder(URI.create("https://csdn.net")).buil
HttpResponse<String> response = httpClient.send(httpRequest, HttpResponse.BodyHandler
```

复制

为什么HttpURLConnection被打入冷宫

我现在心情还不错，来给你们说一下为什么HttpURLConnection被打入冷宫。

这是一个悲伤的故事，那是一个寒冷的冬天。。。（回归正题）这其中有多个原因为什么需要替换HttpURLConnection，首先这是一个非常古老的API。 Java的第一个版本于20多年前1996年发布。HttpURLConnection被添加到Java的JDK 1.1中，该版本于1997年发布。这也刚刚好是HTTP 1.1的被设计出来的时间。在对HTTP请求和响应以及典型的交互模式进行建模方面，事情并没有现在那么清晰。现在看来，HttpURLConnection及其相关类中有很多过度抽象（谁告诉我抽象是好事来着，你出来，我保证不打你）。这些抽象使映射HttpURLConnection方法中发生的情况和实际HTTP发生的情况变得相对困难。

该API太旧了（就是说你老，你能咋地），它不包含泛型，枚举和lambda，因此在现代Java中使用时感觉很笨拙（过时）。虽然从Java 11，HttpURLConnection已被HttpClient取代，我还是希望你能了解一下HttpURLConnection API。你还是可能会在旧代码中遇到它。只有通过查看旧的API，你才会欣赏到HttpClient给你带来的改进。

```
try{
    URL url = new URL("https://csdn.net");
    HttpURLConnection connection = (HttpURLConnection) url.openConnection();
    connection.setRequest("GET")
    connection.setRequestProperty("User-Agent", "Java 1.1");
    if(connection.getResponseCode() == 200) {
        System.out.println(readInputStream(connection.getInputStream()));
    } else {
        System.out.println("Something wrong there!");
    }
} catch(IOException ex) {
    System.out.println("Something wrong there!");
}
```

复制

说明一下，这并不是使用HttpURLConnection的最佳实践，因为它不能解决所有使用API的用例。我只是举一个例子。以此来指出一些与API不带优雅的地方。比如，这个第二行的类型转换，第三行没有Enum。因为在设计此API时还没有枚举这个概念。所以，你可以在此处轻松输入格式错误的字符串。然后可以向连接请求输入流，但这就是原始输入流。因此，我们需要编写一个辅助方法，在本例中为readInputStream，以获取原始输入流并将其转换为有用的东西。那是相当底层的操作。所以，不应该在新代码中再使用HttpURLConnection。

但是，如果你还没有使用Java 11并且没有访问HttpClient API的权限怎么办？你仍然不应该使用HttpURLConnection。在这种情况下，最好查看用于执行HTTP请求的第三方库。包括Apache HttpComponents项目，该项目提供HttpClient API，还有Square的OkHttp，这是Java的另一个开源HttpClient，还有更高级的库，例如JAX-RS REST Client。该REST客户端不仅执行HTTP请求，而且应用了REST原理，并且可以自动将JSON响应映射到Java对象。无论如何，现在都不应该使用HttpURLConnection API。如果你使用的是Java 11，请使用我们现在正在谈论的HttpClient API，如果不是，请使用这些第三方组件之一。

HttpClient的配置

我们不可能在一篇文章中，把一个API完全讲透，但是我们还是尽量的把一些关键点讲出来。现在来更深入地研究一下HttpClient API，包括诸如处理headers，接受cookie以及执行具有请求主体的HTTP请求的功能，这些请求与到目前为止所看到的HTTP GET请求不同。但是在继续使用这些功能之前，让我们更深入地了解一下HttpClient本身的配置选项。我们之前使用了新的HttpClient方法，该方法为我们提供了所有默认设置的HttpClient。作为HelloWorld来说，还不错。但是通常你要自己调整一些配置选项。所以，使用构建器API创建HttpClient时，有几个选项可以影响使用此HttpClient完成的请求的行为。这些配置选项中的大多数不能在请求级别覆盖。如果针对不同类型的请求则需要不同的配置，也就是说需要创建多个适当配置的HttpClient实例。我们将重点关注与安全性无关的配置。

HTTP Version版本

第一个配置是HTTP版本。使用HttpClient.newBuilder创建构建器后，可以使用version方法配置将使用的HTTP版本。版本枚举本身嵌套在HttpClient类的内部，一共有两个选项

- HttpClient.Version.HTTP*11*
- HttpClient.Version.HTTP_2

HTTP 2是默认选项，如果HttpClient配置为HTTP 2，但是服务器不支持HTTP 2的话，它将自动回退到HTTP 1.1。 HTTP / 2流量看起来与HTTP / 1.1流量完全不同。您也可以根据个人要求配置版本。

Priority 优先级

第二个配置是优先级。由于仅在HTTP 2协议中指定了优先级，所以这个配置选项仅影响HTTP 2的请求。优先级设置采用1-256范围内的整数，包括这种情况下的边界值。较高的数字表示较高的优先级。

Redirection 重定向

另一个设置与重定向策略有关。默认情况下，HttpClient配置为从不重定向。这意味着，当对要重定向到另一个URI的服务器执行请求时，HttpClient将不会遵循此重定向。你还可以将HttpClient配置为在服务器响应重定向状态代码时始终遵循重定向。最后，有一个正常的重定向策略，它与始终重定向相同，只是从安全资源重定向到非安全资源的情况除外。从安全位置重定向到非安全位置通常是安全问题。这就是为什么建议使用常规重定向策略而不是使用Always策略的原因。

- HttpClient.Redirect.NEVER

- HttpClient.Redirect.ALWAYS
- HttpClient.Redirect.NORMAL

Connection Timeout超时

它需要一个java.time.Duration，并且这是HttpClient等待建立与HTTP服务器连接的时间。如果未配置connectTimeout，则默认设置为无限期等待，这肯定不会是你想要的。connectTimeout与建立与服务器的TCP连接有关。如果花费的时间比配置的connectTimeout长，则将引发异常。

Custom Executor自定义执行器

最后，还有一个配置选项，用于设置供HttpClient实例使用的自定义执行程序。 HttpClient使用执行器来执行异步处理。默认情况下，在构建新的HttpClient时，它还会为此HttpClient实例化一个新的私有线程池。在某些情况下，你可能希望在不同的HttpClient之间共享一个执行程序。你可以通过自己创建或获取执行程序并将该执行程序传递给HttpClient构建器上的executor方法来实现。比如

```
Executor exec = Executors.newCachedThreadPool();
HttpClient.newBuilder().executor(exec);
```

复制

综合的例子

```
HttpClient client = HttpClient.newBuilder()
                .connectTimeout(Duration.ofSeconds(5))
                .followRedirects(HttpClient.Redirect.NORMAL)
                .build()
```

复制

请求的有效负载

我们一直都在使用简单的GET请求作为案例，这些请求不会将负载传输到HTTP服务器。我们现在来看一下如何创建包含有效负载的请求。此有效负载可以是纯文本，可以是JSON，也可以是任何任意二进制有效负载。通常，HTTP POST的请求主体中包含有效负载。

除了HttpRequest.Builder上的GET方法外，还有一个POST方法。与GET方法一样，POST方法也带有一个参数，即所谓的BodyPublisher。此BodyPublisher负责产生与POST请求一起发送的有效负载。从这个意义上讲，BodyPublisher与我们之前看到的用于处理响应有效负载的

BodyHandler类似。 它告诉HttpClient如何在给定Java对象的情况下构造HTTP请求的主体。你可以在BodyPublisher的类上找到现有的预定义BodyPublisher。比如，HttpRequest.Builder上还有PUT方法，这将使BodyPublisher提供的主体有效负载创建一个HTTP PUT请求。

- POST(BodyPublisher publisher)
- PUT(BodyPublisher publisher)

除了POST和PUT之外还有其他HTTP方法，例如PATCH。但是，并非每个HTTP方法在HttpRequest.Builder上都有其自己的方法。如果要创建除GET，POST或PUT之外的请求，则必须在HttpRequest.Builder上使用method方法。

- method(String method, BodyPublisher publisher)

方法采用两个参数，其中第一个是表示要请求执行的HTTP方法的字符串，以及一个BodyPublisher。

Headers and Cookies

你已经了解了如何使用主体创建HTTP请求。客户端向服务器执行HTTP请求时，它必须定义要使用的HTTP方法（比如GET）和要获取的资源比如 /index.html。但是，HTTP请求还有更多的要素。它还包括headers。

headers是简单的键/值对，其中可能包含有关请求的其他元数据。一个示例是主机头。比如

- Host：www.csdn.net

如你看到的，它显示为Host： 值是www.csdn.net。主机标头是HTTP 1.1和更高版本的必需标头之一，它由HttpClient根据创建请求时传递的URI由HttpClient自动为我们管理。除了这些强制性和自动管理的标头之外，有时你还希望向请求中添加其他标头。例如，您可能想添加一个accept标头。

- Accept: text/html

accept标头告诉服务器我们想要的首选响应类型，此处表明我们想取回HTML文档。诸如accept之类的标题是HTTP规范的一部分，但它们是可选的。因此，如果你想添加这样的标头来执行请求，则必须为此做一些工作。也可以向HTTP请求中添加任意的，未指定的标头。在API中的写法就是这样

```
HttpRequest.newBuilder(URI.create("https://csdn.net"))
            .header("Accept", "text/html")
            .build()
```

复制

如果你要有多个标题，那当然也可以。只需重复使用header方法，直到将所需的所有标头添加到请求中即可。甚至可以使用headers方法而不是header方法一次性添加多个标题。标头始终需要偶数个参数，因为每个标头名称都必须带有一个值。如果您添加相同的标头，使用相同的标头名称，并多次使用不同的值，那么所有这些值都会出现在标头中，因为HTTP定义了标头可以具有多个值。如果你要绝对确定标头没有多个值，则也可以使用setHeader方法，该方法也可以使用标头的名称和标头的值，但不必将值添加到标头中替换当前值。

当你构建请求并通过HttpClient发送请求时，HttpClient将负责以正确的方式将标头和值添加到HTTP请求。对于HTTP 1.1和HTTP 2，它的执行方式完全不同。 HTTP 1.1是纯文本协议，并且标头将添加到此纯文本中请求。 HTTP 2是一个二进制协议，标头将以二进制格式编码，甚至经过专门压缩。不过你不用担心这些，对于你来说，API是相同的，复杂性全都隐藏在HttpClient的实现中。

还有另一种与HTTP紧密联系的机制，HTTP本身是无状态请求响应协议。我们从服务器请求一些东西，服务器提供响应，然后客户端和服务器彼此相忘于江湖。但是，在许多情况下，你希望保持有关服务器和客户端之间交互的某些状态（我就是忘不掉我的前女友，这可咋办）。

Cookie是一种主要用于浏览器的方式。 Cookies包含服务器定义的状态，但是状态由客户端保留，然后在必要时再发送回服务器。有趣的是，这种机制是基于标头构建的。服务器还可以在响应中包含标头。并且，当服务器包含Set-Cookie标头时，客户端应将此标头解释为要保留给下一个请求的状态，这也是该机制在浏览器中的工作方式。当浏览器看到Set-Cookie标头，并且在标头的主体中包含一些键-值对时，它将把这些名称/值对存储在与请求域相关联的所谓持久性cookie中。然后，每当对同一个域提出新请求时，浏览器将包含一个Cookie标

头。 Cookie头的值是先前存储的名称/值对。这样，浏览器和HTTP服务器可以在不同的无状态HTTP请求之间创建持久状态的错觉。当然，只有当服务器和客户端都知道Set-Cookie和Cookie标头时，整个设置才有效。你可以尝试使用HttpClient自己实现此目的，因为它全都与Set-Cookie和Cookie标头有关。因为关于Cookie的行为方式复杂性要很高，所以你其实并不想自己管理。好消息是HttpClient为我们提供了一个用于配置cookie的处理。你可以使用setCookieHandler来配置HttpClient使用它。如果你希望HttpClient使用cookie，那么一种快速的入门方法是使用CookieManager类。 CookieManager是CookieHandler的JDK内部的具体实现。在此示例中，我们创建一个新的管理器，第一个参数是Cookie持久存储在其中的CookieStore。


```
CookieManager cm = new CookieManager(null, CookiePolicy.ACCEPT_ALL)
HttpClient client = HttpClient.newBuilder().cookieHandler(cm).build()
client.send(HttpRequest.newBuilder(URI.create("https://csdn.net")).build(), HttpRespc
System.out.println(cm.getCookieStore().getURIs())
System.out.println(cm.getCookieStore().getCookies())
```

复制

还是那句话，不可能把所有的内容在一小节上全讲解，如果有人感兴趣的话，给我在文下留言，如果留言多的话，我会再整理一篇。

上一页 下一页