BY MATTHEW GILLIARD • 2021-03-30

TWITTER FACEBOOK LINKEDIN

5 Maneiras de Fazer uma Chamada HTTP em Java

Fazer chamadas HTTP é uma funcionalidade fundamental da programação moderna, e geralmente é uma das primeiras coisas que você vai querer fazer ao aprender uma linguagem de programação nova. Para programadores Java, existem diversas maneiras de se fazer isso - bibliotecas core na JDK e bibliotecas de terceiros. Este artigo vai te introduzir

aos clientes HTTP Java que eu geralmente uso. Se você usa outros, ótimo! Me conte mais sobre eles. Veja o que vou cobrir neste artigo:

Core Java:

- HttpURLConnection
- HttpClient

Bibliotecas Populares:

- ApacheHttpClient
- OkHttp
- Retrofit

Eu vou usar a API de Imagem Astronômica do Dia das APIs da NASA para os código de exemplo, e o código está todo no GitHub em um projeto baseado em Java 11.

APIs Core do Java para se fazer chamadas http no Java

Desde o Java 1.1 já havia um cliente HTTP nas bibliotecas core com o JDK. Com o Java 11, um novo cliente foi adicionado. Um destes pode ser uma boa escolha se você for sensível em relação a adicionar dependências extras em seu projeto.

Java 1.1 HttpURLConnection

Antes de tudo, nós vamos deixar em maiúsculas siglas em nomes de classes ou não? Decida-se. De qualquer maneira, feche seus olhos e se imagine em 1997. O Titanic estava sendo um sucesso de bilheteria e inspirando milhares de memes, as Spice Girls lançaram um álbum líder de vendas, mas a maior notícia do ano com certeza foi o HttpURLConnection, que foi adicionado ao Java 1.1. É assim que você deveria usar para se fazer uma chamada `GET` :

```java
// Create a neat value object to hold the URL
URL url = new URL("https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY");

// Open a connection(?) on the URL(??) and cast the response(???)
HttpURLConnection connection = (HttpURLConnection) url.openConnection();

// Now it's "open", we can set the request method, headers etc.
connection.setRequestProperty("accept", "application/json");

// This line makes the request
InputStream responseStream = connection.getInputStream();

// Manually converting the response body InputStream to APOD using Jackson
ObjectMapper mapper = new ObjectMapper();
APOD apod = mapper.readValue(responseStream, APOD.class);

// Finally we have the response
System.out.println(apod.title);
```

[código completo no GitHub]

Isso parece ser bem verboso, e eu acho a ordem das coisas um pouco confusa (por que configuramos os headers *depois* de abrir uma URL?). Se você precisar fazer chamadas mais complexas incluindo conteúdo no body de uma chamada `POST`, ou timeouts customizados etc, tudo é possível, mas nunca achei essa API intuitiva.

Quando você deve usar `HttpURLConnection`, então? Se você estiver suportando clientes que usam versões mais velhas do Java **e** você não pode adicionar uma dependência, isso pode funcionar para você. Eu suspeito que esse seja o caso apenas para uma pequena minoria de desenvolvedores, mas você pode acabar vendo isso em códigos mais velhos – para casos mais modernos, continue lendo.

HttpClient do Java 11

Mais de 20 anos depois da `HttpURLConnection`, tivemos Pantera Negra nos cinemas, e um novo Cliente HTTP foi adicionado ao Java 11: `java.net.http.HttpClient`. Este cliente tem uma API muito mais lógica, e suporta HTTP/2 e Websockets. Ele também tem a opção de fazer chamadas síncronas e assíncronas usando a API `CompletableFuture`.

99 das vezes em 100, quando eu faço uma chamada HTTP, eu quero ler a resposta do corpo no próprio código. Bibliotecas que tornam este processo difícil não vão me animar. O HttpClient aceita um `BodyHandler` que pode converter uma resposta HTTP em uma classe de sua escolha. Existem alguns handlers embutidos: `String`, `byte[]` para dados

binários, `Stream<String>` que se divide em linhas, e algumas outras. Você também pode definir a sua própria, que pode ser útil já que não existe um `BodyHandler` embutido para tratar JSON. Eu escrevi um (aqui) baseado no Jackson seguindo um exemplo do Java Docs. Ele retorna um `Supplier` para a classe APOD, e assim podemos chamar `.get()` quando precisarmos do resultado.

Esta é uma chamada síncrona:

```
// create a client
var client = HttpClient.newHttpClient();

// create a request
var request = HttpRequest.newBuilder(
       URI.create("https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY"))
   .header("accept", "application/json")
   .build();

// use the client to send the request
var response = client.send(request, new JsonBodyHandler<>(APOD.class));

// the response:
System.out.println(response.body().get().title);
```

Para uma chamada assíncrona, o `client` e `request` são feitos da mesma maneira, então chamamos `.sendAsync()` ao invés de `.send` :

```
// use the client to send the request
var responseFuture = client.sendAsync(request, new JsonBodyHandler<>(APOD.c

// We can do other things here while the request is in-flight

// This blocks until the request is complete
var response = responseFuture.get();

// the response:
System.out.println(response.body().get().title);
```

[código completo no GitHub]

Bibliotecas de Terceiros para Clientes HTTP

Se os clientes nativos não são bons o suficiente para você, não se preocupe! Existem várias bibliotecas que você pode usar em seu projeto que vão dar conta do recado.

HttpClient do Apache

Os clientes HTTP da Apache Software Foundations existem há muitos anos. Eles são muito usados e são a base para várias bibliotecas de alto nível. A história é um pouco confusa. O antigo HttpClient padrão já não é mais desenvolvido, e a versão nova (também chamada de HttpClient) está dentro do projeto HttpComponents. A versão 5.0 foi lançada no início de 2020, adicionando suporte HTTP/2. A biblioteca também suporta chamadas síncronas e assíncronas.

No geral a API é bem baixo nível - você tem de implementar muitas coisas. O código a seguir chama a API da NASA. Ela não parece tão difícil de se usar, mas eu pulei vários tratamentos de erro que você deveria deixar no código de produção, e de novo, eu tive de adicionar Jackson no código para tratar a resposta JSON. Também seria bom você configurar uma plataforma de log para evitar avisos no stdout (nada demais, mas me deixa um pouco conturbado). De qualquer forma, aqui está nosso código:

```
1  ObjectMapper mapper = new ObjectMapper();
2
3  try (CloseableHttpClient client = HttpClients.createDefault()) {
4
5     HttpGet request = new HttpGet("https://api.nasa.gov/planetary/apod?api_
6
7     APOD response = client.execute(request, httpResponse ->
8         mapper.readValue(httpResponse.getEntity().getContent(), APOD.class)
9
10    System.out.println(response.title);
```

[código completo no GitHub]

O Apache fornece vários exemplos para chamadas síncronas e assíncronas.

OkHttp

OkHttp é um Cliente HTTP da Square com várias funcionalidades embutidas, como lidar automaticamente com GZIP, cacheamento de respostas e novas tentativas ou fallbacks para outros hosts em caso de erros na rede, assim como HTTP/2 e Websockets. A API é limpa, mas não tem nenhum tratamento de JSON embutido, então novamente, precisamos usar o Jackson:

```java
ObjectMapper mapper = new ObjectMapper();

OkHttpClient client = new OkHttpClient();

Request request = new Request.Builder()
   .url("https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY")
   .build(); // defaults to GET

Response response = client.newCall(request).execute();

APOD apod = mapper.readValue(response.body().byteStream(), APOD.class);

System.out.println(apod.title);
```

[código completo no GitHub]

O código está ok, mas o real poder do OkHttp fica claro quando você usa o Retrofit junto.

Retrofit

Retrofit é outra biblioteca da Square, construída por cima da OkHttp. Junto com todas as funcionalidades de baixo nível da OkHttp, ela adiciona uma maneira para se construir classes Java que abstraem os detalhes HTTP e apresentam uma ótima API amigável em Java.

Primeiro, precisamos criar uma interface que declara os métodos que queremos chamar contra a API APOD, com annotations definindo como esses correspondem à chamada HTTP:

```java
public interface APODClient {
   @GET("/planetary/apod")
   @Headers("accept: application/json")
   CompletableFuture<APOD> getApod(@Query("api_key") String apiKey);
}
```

O retorno do tipo `CompleatableFuture<APOD>` torna isso um cliente assíncrono. A Square fornece outros adaptadores ou você pode escrever o seu próprio. Ter uma interface como essa te ajuda a mockar os clientes para testes, o que eu gosto bastante.

Depois de declarar a interface, pedimos ao Retrofit para criar uma implementação para fazermos chamadas em uma URL base. Isso também é útil para testes de integração para podermos trocar a URL base. Para criar o cliente, o código se parece com isso:

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.nasa.gov")
    .addConverterFactory(JacksonConverterFactory.create())
    .build();

APODClient apodClient = retrofit.create(APODClient.class);

CompletableFuture<APOD> response = apodClient.getApod("DEMO_KEY");

// do other stuff here while the request is in-flight

APOD apod = response.get();

System.out.println(apod.title);
```

[código completo no GitHub]

Autenticação da API

Se existem vários métodos em nossa interface, e todas precisam de uma chave de API, é possível configurar isso tudo adicionando um `HttpInterceptor` à base do `OkHttpClient`. O cliente customizado pode ser adicionado ao `Retrofit.Builder`. O código para criar o cliente é:

```
private OkHttpClient clientWithApiKey(String apiKey) {
    return new OkHttpClient.Builder()
            .addInterceptor(chain -> {
                Request originalRequest = chain.request();
                HttpUrl newUrl = originalRequest.url().newBuilder()
                    .addQueryParameter("api_key", apiKey).build();
                Request request = originalRequest.newBuilder().url(newUrl).b
                return chain.proceed(request);
            }).build();
}
```

[código completo no GitHub]

E gosto deste tipo de API Java, exceto para todos casos mais simples. Construir classes para representar APIs remotas sempre é uma boa abstração que vem bem com injeção de dependência, e deixar que o Retrofit faça isso para você baseado em um cliente OkHttp personalizado é ótimo.

Outros Clientes HTTP para Java

Desde que postei este artigo no Twitter, fiquei impressionado com as discussões sobre quais clientes HTTP eles usam. Se nenhum dos acima é bem o que você quer, veja essas sugestões:

- REST Assured – um cliente HTTP desenhado para testar seus serviços REST. Oferece uma interface fluente para fazer chamadas e métodos úteis para fazer validações das respostas.
- cvurl – um wrapper para o HttpClient do Java 11 que simplifica algum dos casos que você pode encontrar ao fazer chamadas complexas.
- Feign – Similar ao Retrofit, Feign pode construir classes de interfaces com annotations. É altamente flexível com várias opções para se fazer e ler chamadas, métricas, retentativas e mais.
- Spring RestTemplate (Síncrono) e WebClient (assíncrono) – se você está acostumado com o Spring para todo o resto em seu projeto, pode ser uma boa ideia continuar com esse ecossistema. Baeldung tem um artigo comparando-os.
- MicProfile Rest Client – outro cliente do tipo "construa uma classe de uma interface com annotations", este é bem interessante porque você pode aproveitar a mesma interface para criar um servidor web também, garantindo que o cliente e servidor batem. Se você estiver construindo um serviço *e* um cliente para o serviço, ele pode ser perfeito para você.

Resumo

Existem várias opções de Cliente HTTP em Java – para casos simples, eu recomendaria o `java.net.http.HttpClient` nativo. Para casos mais complexos, eu iria com o Retrofit ou Feign. Bom hacking, mal posso esperar para ver o que você construiu!

> Este artigo foi traduzido do original "5 ways to make HTTP requests in Java".

RATE THIS POST ★★★★★

AUTHORS  Matthew Gilliard

REVIEWERS  Luís Leão

Search

POSTS BY STACK

JAVA .NET PHP RUBY PYTHON SWIFT ARDUINO JAVASCRIPT

POSTS BY PRODUCT

EMAIL SMS VOICE MMS VIDEO CONVERSATIONS IOT TASK ROUTER VERIFY FLEX SIP TWILIO CLIENT STUDIO

CATEGORIES

Code, Tutorials and Hacks

Customer Highlights

Developers Drawing The Owl

News

Stories From The Road

The Owl's Nest: Inside Twilio

LANGUAGES

JAPANESE GERMAN SPANISH PORTUGUESE FRENCH