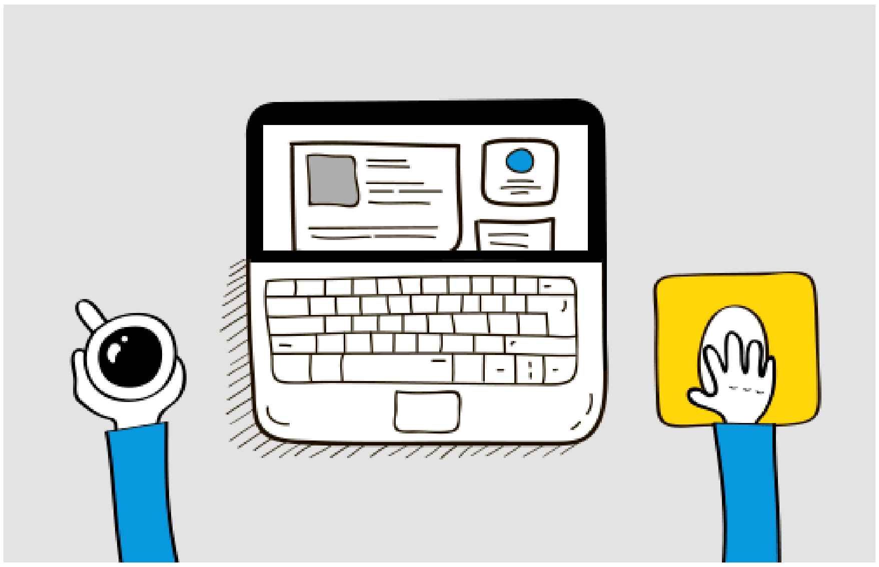
Java 11: A New Way to handle HTTP & WebSockets in Java!

by Adrian D. Finlay

Posted on August 10, 2018 at 1:06 PM UTC [-4]



Source: Giphy

Once upon a time, using the Java SE (Standard Edition) APIs to do common HTTP operations such as REST API calls might have been described as unnatural and cumbersome. Java 11 officially changes this. Coming with Java 11, the incubated HTTP APIs from Java 9 are now officially incorporated into the Java SE API. The *JDK Enhancement Proposal (JEP)* 321 was the JEP behind this effort. Since being integrated into Java 11, the API has seen a few changes. As of Java 11, **the API is now fully asynchronous.** This article will attempt to show you basic use of the new API by performing a REST API call. We will be using openJDK 11.

The APIs use `java.util.concurrent.CompleteableFuture<T>` to provide asynchronous, non-blocking request/response behavior allowing for dependent actions. The API used new-style OOP method chaining, like a builder which returns an object that may be affected by the method call.

In addition to the standard documentation, you can learn how to practically apply the `CompleteableFuture<T>` API below.

20 Examples of Using Java's CompletableFuture - DZone Java

*This post revisits Java 8's CompletionStage API and specifically its implementation in the standard Java library…*dzone.com

The new HTTP APIs can be found in `java.net.HTTP.*`.

The new APIs provide native support for HTTP 1.1/2, WebSocket. The core classes & interface providing the core functionality include:

- The HttpClient class, `java.net.http.HttpClient`
- The HttpRequest class, `java.net.http.HttpRequest`
- The HttpResponse<T> interface, `java.net.http.HttpResponse`
- The WebSocket interface, `java.net.http.WebSocket`

The following are the types in the API:

Classes

- java.net.http.HttpClient
- java.net.http.HttpHeaders
- java.net.http.HttpRequest
- java.net.http.HttpRequest.BodyPublishers
- java.net.http.HttpRequest.BodyHandlers
- java.net.http.HttpRequest.BodySubscribers

Interfaces

- java.net.http.HttpClient.Builder
- java.net.http.HttpClient.BodyPublisher
- java.net.http.HttpRequest.BodyPublisher
- java.net.http.HttpResponse<T>
- java.net.http.HttpResponse.BodyHandler<T>
- java.net.http.HttpResponse.BodySubscriber<T>
- java.net.http.HttpResponse.PushPromiseHandler<T>
- java.net.http.HttpResponse.ResponseInfo
- java.net.http.WebSocket

- java.net.http.WebSocket.Builder
- java.net.http.WebSocket.Listener

Enumeration

- java.net.http.HttpClient.Redirect
- java.net.http.HttpClient.Version

Exception

- java.net.http.HttpTimeoutException
- java.net.http.WebSocketHandshakeException

Pre-Java 11 vs. Java 11: Handling REST API Calls:

Pre-Java 11

```
/*
Copyright (C) 2018 Adrian D. Finlay. All rights reserved.

Licensed under the MIT License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    https://opensource.org/licenses/MIT

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER INCLUDING AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
==========================================================================
**/

```

```java
package com.adriandavid.http.prejava11;


import java.io.File;
import java.net.URI;
import java.util.Scanner;
import java.time.Duration;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.Files;
import java.io.InputStream;
import java.io.FileOutputStream;
import java.nio.charset.Charset;
import java.net.Authenticator;
import java.net.ProxySelector;
import java.net.http.WebSocket;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.HttpURLConnection;
import java.nio.file.StandardOpenOption;
import java.nio.file.StandardCopyOption;

/* HTTP GET Request, Response (Pre-Java 11 API)  */
public class RESTDemo {

        private String[] args;
        private final String API_ENDPOINT = "https://onyxfx-api.herokuapp.com/nbaBasicStatBean?";
        private final String API_ENDPOINT2 = "http://api.giphy.com/v1/gifs/";
    private final String API_ENDPOINT2_ID = "yoJC2COHSxjIqadyZW";
    private final String API_ENDPOINT2_KEY = "ZtFzb5dH6w9aYjoffJQ0RqlAsS5s0xwR";


        public RESTDemo (String[] args) {
         this.args = args;
    };

    public void call() throws Exception {
        //HTTP GET REQUEST
        var HTTP_CLIENT= (HttpURLConnection)
                                URI.create(
                                    new StringBuilder(API_ENDPOINT)
                                    .append("firstName=").append(args[0])
                                    .append("&surname=").append(args[1])
                                    .append("&season=").append(args[2])
                                    .toString())
                                .toURL()
                                .openConnection();
        HTTP_CLIENT.setRequestMethod("GET");

        var HTTP_CLIENT2 = (HttpURLConnection)
                            URI.create(  //Set the appropriate endpoint
                                new StringBuilder(API_ENDPOINT2)
                                .append(API_ENDPOINT2_ID)
                                .append("?api_key=").append(API_ENDPOINT2_KEY)
                                .append("&data")
                                .append("&type")
                                .append("&images")
                                .toString() )
                            .toURL()
                            .openConnection();
        HTTP_CLIENT2.setRequestMethod("GET");


        //HTTP RESPONSE
        var HTTP_RESPONSE = HTTP_CLIENT.getInputStream();
        var scn = new Scanner(HTTP_RESPONSE);
        var json_sb = new StringBuilder();
        while (scn.hasNext()) {
            json_sb.append(scn.next());
        }
        var JSON = json_sb.toString();

        if (HTTP_CLIENT2.getContentType().contains("json")) {
            var stream_in = (InputStream)(HTTP_CLIENT2.getContent());
            var stream_out = new FileOutputStream (new File("response1.json"));
            stream_in.transferTo(stream_out);
            stream_in.close();
            stream_out.close();
        }
        else
            return;  // suffice for now.

        // HTTP STATUS
        var statusCode = HTTP_CLIENT.getResponseCode();
        var statusCode2 = HTTP_CLIENT2.getResponseCode();

        // HANDLE RESPONSE
        if (statusCode == 200 || statusCode == 201)
            System.out.println("Success! -- Pre-Java 11 REST API Call\n" +
                args[1] + "," + args[0] + "[" + args[2] +"]\n" + JSON);
        else
            System.out.println("Failure! -- Pre-Java 11 REST API Call");

        System.out.println("--------------------------------");

        if (statusCode2 == 200 || statusCode2 == 201) {
            System.out.println("Success! -- Pre-Java 11 REST API Call\n" + "Let's download the file!" )
            }
            else
                System.out.println("Failure! -- Pre-Java 11 REST API Call");

          System.out.println("--------------------------------");
      };
  };
```

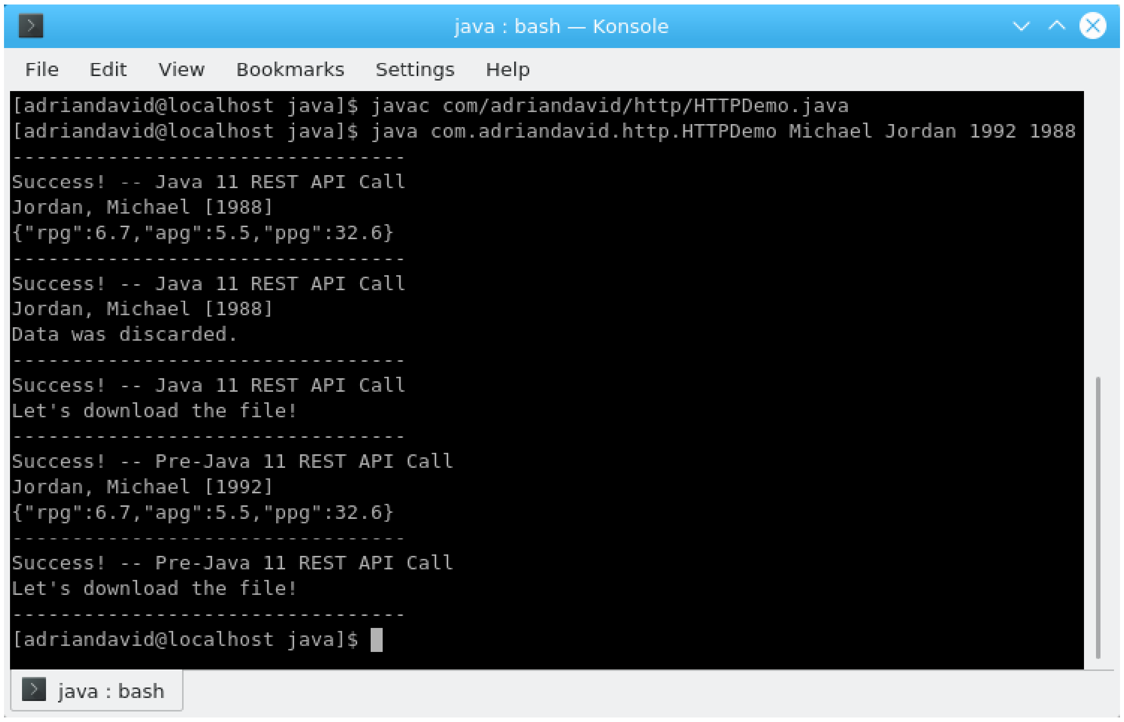
RESTDemo11.java hosted with ❤ by GitHub view raw

Java 11

```java
/*

==============================================================================
**/

package com.adriandavid.http.java11;


import java.io.File;
import java.net.URI;
import java.util.Scanner;
import java.time.Duration;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.Files;
import java.io.InputStream;
import java.io.FileOutputStream;
import java.nio.charset.Charset;
import java.net.Authenticator;
import java.net.ProxySelector;
import java.net.http.WebSocket;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.HttpURLConnection;
import java.nio.file.StandardOpenOption;
import java.nio.file.StandardCopyOption;

/* HTTP GET Request, Response (Java 11 APIs) */
public class RESTDemo {
    private final String API_ENDPOINT = "https://onyxfx-api.herokuapp.com/nbaBasicStatBean?";
    private final String API_ENDPOINT2 = "http://api.giphy.com/v1/gifs/";
    private final String API_ENDPOINT2_ID = "yoJC2COHSxjIqadyZW";
    private final String API_ENDPOINT2_KEY = "ZtFzb5dH6w9aYjoffJQ0RqlAsS5s0xwR";
    private final HttpResponse.BodyHandler<String> asString = HttpResponse.BodyHandlers.ofS
    private final HttpResponse.BodyHandler<Void> asDiscarded = HttpResponse.BodyHandlers.d
    private final HttpResponse.BodyHandler<InputStream> asInputStream= HttpResponse.BodyHan
    private final HttpClient HTTP_CLIENT = HttpClient.newBuilder().version(HttpClient.Version.HTT
                            .followRedirects(HttpClient.Redirect.NORMAL).proxy(ProxySelec
    private String[] args;

    public RESTDemo (String[] args) {
        this.args = args;
    };

    public void call() throws Exception {

        if (args.length < 4) {
            System.out.println("An invalid amount of arguments was supplied.");
            return;
        }

        System.out.println("---------------------------------");

        // HTTP GET REQUEST
        var HTTP_REQUEST = HttpRequest.newBuilder()
            .uri(URI.create( //Set the appropriate endpoint
                    new StringBuilder(API_ENDPOINT)
                    .append("firstName=").append(args[0])
                    .append("&surname=").append(args[1])
                    .append("&season=").append(args[2])
                    .toString() ) )
            .timeout(Duration.ofMinutes(1))
            .header("Content-Type", "application/json")
            .build();
        var HTTP_REQUEST2 = HttpRequest.newBuilder()
            .uri(URI.create( //Set the appropriate endpoint
                    new StringBuilder(API_ENDPOINT2)
                    .append(API_ENDPOINT2_ID)
                    .append("?api_key=").append(API_ENDPOINT2_KEY)
                    .append("&data")
                    .append("&type")
                    .append("&images")
                    .toString() ) )
            .timeout(Duration.ofMinutes(1))
            .header("Content-Type", "application/json")
            .build();


        // SEND HTTP GET REQUEST, RECIEVE OBJECT FOR HTTP GET RESPONSE
        var HTTP_RESPONSE = HTTP_CLIENT.send(HTTP_REQUEST, asString);
        var HTTP_RESPONSE2 = HTTP_CLIENT.send(HTTP_REQUEST, asDiscarded);
        var HTTP_RESPONSE3 = HTTP_CLIENT.send(HTTP_REQUEST2, asInputStream);

        // HTTP STATUS CODE
        var statusCode = HTTP_RESPONSE.statusCode();
        var statusCode2 = HTTP_RESPONSE2.statusCode();
        var statusCode3 = HTTP_RESPONSE3.statusCode();


        // HANDLE RESPONSE
        if (statusCode == 200 || statusCode == 201)
            System.out.println("Success! -- Java 11 REST API Call\n" +
                args[1] + "," + args[0] + "[" + args[3] +"]\n" + HTTP_RESPONSE.body()
        else
            System.out.println("Failure! -- Java 11 REST API Call");

        System.out.println("---------------------------------");

        if (statusCode2 == 200 || statusCode2 == 201)
            if (HTTP_RESPONSE2.body() == null)
            System.out.println("Success! -- Java 11 REST API Call\n" +
                args[1] + "," + args[0] + "[" + args[3] +"]\n" + "Data was discarded.");
        else
            System.out.println("Failure! -- Java 11 REST API Call");

        System.out.println("---------------------------------");

        if (statusCode3 == 200 || statusCode3 == 201) {
            System.out.println("Success! -- Java 11 REST API Call\n" + "Let's download the file! ");
            var HTTP_STREAM = HTTP_RESPONSE3.body();
```

```
                Files.copy(HTTP_STREAM, new File("response2.json").toPath(), StandardCopyOption.R
                HTTP_STREAM.close();
            }
            else
                System.out.println("Failure! -- Java 11 REST API Call");

            System.out.println("--------------------------------");
        };
    };
```

RESTDemo.java hosted with ❤ by **GitHub** **view raw**



```
[adriandavid@localhost java]$ javac com/adriandavid/http/HTTPDemo.java
[adriandavid@localhost java]$ java com.adriandavid.http.HTTPDemo Michael Jordan 1992 1988
--------------------------------
Success! -- Java 11 REST API Call
Jordan, Michael [1988]
{"rpg":6.7,"apg":5.5,"ppg":32.6}
--------------------------------
Success! -- Java 11 REST API Call
Jordan, Michael [1988]
Data was discarded.
--------------------------------
Success! -- Java 11 REST API Call
Let's download the file!
--------------------------------
Success! -- Pre-Java 11 REST API Call
Jordan, Michael [1992]
{"rpg":6.7,"apg":5.5,"ppg":32.6}
--------------------------------
Success! -- Pre-Java 11 REST API Call
Let's download the file!
--------------------------------
[adriandavid@localhost java]$
```

My Impression

The new API is, semantically, everything you would expect of a Java API in 2018. For a start, it is verbose. Secondly, it is modular. Thirdly, it follows the new OOP style of method chaining (or using builders) to construct objects. Fourthly, it feels more natively HTTP.

Verbosity

The bane and (to some) the valor of java has always been verbosity. In Java, unlike most duck typed languages, it is easier to the eye to induce types and behavior. Java API typically follows a natural approach to naming as one might expect in the real world—it is usually intuitive. In addition, java's typing semantics are explicit, adding to this verbosity. However, the typical verbosity was reduced in my examples through my frequent use of Java 10's **var**. Nevertheless, the API feels a bit verbose.

However, credit must be given to the API writers for allowing the implementation to be as configured or as least configured as desired. The new OOP style has allowed for that. For that matter, it should also be noted that there are many things that one can configure in the HTTP request response cycle. Therefore, a great deal of verbosity is expected.

Modularity

Following the long standing unix tradition, long beloved by the Java community and in line with Java 9 efforts, the API is quite modular. The API is careful split into pieces that offer little to no cruft—one gets what one wants and usually nothing more. In addition, there is little dependency on classes not immediately relevant to the task at hand.

One exception to this rule that could be argued is the creation of a WebSocket. The most straightforward (and API recommended) way to create a WebSocket is to use an instance of WebSocket.Builder. WebSocket.Builder instances are most straightforwardly created by using `java.net.http.HttpClient.newWebSocketBuilder()`. Once that call has been made, it is typically chained with a call to `java.net.WebSocket.Builder.buildAsynce(URI uri, WebSocket.LIstener)` to produce a CompleteableFuture<WebSocket> object associated with the WebSocket. The WebSocket may then be retrieved by calling get() on the CompleteableFuture object.

Method Chaining

What I like about designing an API with method chaining as a means to construct objects is that it allows for objects that are as least configured as you would like them to be or as highly configured as you would like them to be. This result can be achieved by way of overloading constructors, of course, but after the object has been created one has to call a setter() method to update or a property on the object. With builders, one may retrieve the modified object all in one call. It plays well into handling things like HTTP, where there are many properties to specify. It would be unwieldy to have to put in a 10 argument constructor list to modify an object. This style has been welcomed by the OOP community at large and is here to stay.

Native HTTP Feel

Lastly, the API feels more native to HTTP. Method names such as body() and headers() are more appropriate ways to name a method than getContent() and getHeaderField(), getHeaderKey(). The old APIs seemed to me to be to abstract and concerned with networking in a general sense as opposed to HTTP. It's also more intuitive in terms of specifying BodyHandlers, and so on. In the old API, this felt like an unnatural operation, or rather, specifying them felt like incorporating a foreign citizen; The design and use of the API can lead to some unexpected behavior. This is, however, just my impression.

Advantages over HTTPUrlConnection

The advantages that have personally caught my eye over the old API are:

1. Improved Support for HTTP, HTTP/2
2. Native HTTP Feel, HTTP is a first class citizen
3. Asynchronous, Non-Blocking Implementation
4. New API, works more naturally with modern language features

The Source Code

Click the git logo to obtain the full source code, including examples using WebSockets or click here.

Follow @afinlay5 42 Watch 0 Star 1 Fork 0 Download

Source:

Looney Tunes Ending

Leave a Comment:

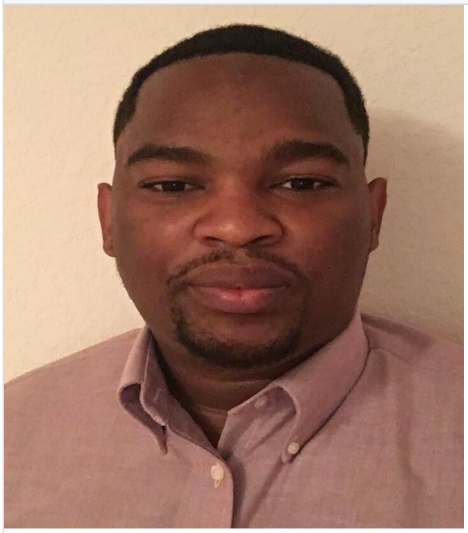
Submit

View Count: 8,368

Categories

Java
Java 11
Programming
Code
Technology
Software Development

Author



Adrian D. Finlay is a passionate engineer who revels at all things code. Adrian mainly tinkers with C++, Java, C#, Python, SQL, HTML5, CSS3, & Scala. Adrian also holds the Oracle Certified Associate in Java SE 8 Programming.

Alternative Publications

- Hackernoon/Medium: `https://bit.ly/JavaHttp2`

Download this article

Portable Document Format (PDF): `Java11HttpWS.pdf`