Java 11 HttpClient API

The Java 11 HttpClient API, located in the java.net.http package, is a standardized and modern way to send HTTP requests and receive responses in Java applications. It was previously an incubating feature in Java 9 and 10 and became a standard part of the Java SE platform in Java 11.

Java 11 java.net.http.HttpClient is a modern HTTP client bundled in the JDK (module java.net.http).  
It replaces the old HttpURLConnection and gives built-in support for:

* HTTP/1.1 and HTTP/2
* synchronous and asynchronous requests (CompletableFuture)
* request/response body handlers (strings, byte arrays, files, streams)
* WebSocket client
* configurable timeouts, redirects, proxy, authentication (via Authenticator)
* TLS/SSL configuration (Transport Layer Security /  Secure Sockets Layer )

**When & where to use it**

Use HttpClient when:

* You need plain REST calls (GET/POST/PUT/DELETE) from Java code.
* You want HTTP/2 support and multiplexing for better performance.
* You want async non-blocking calls using CompletableFuture.
* You prefer built-in JDK solution (no extra dependency).  
  Where not ideal:
* Advanced cookie management or fancy connection pooling — external libs (OkHttp/Apache) may be easier.
* Very advanced features like multipart streaming with progress or special retry logic — still doable but may need extra code.

Code examples:

HttpAsyncExample:

package HTTPAPI;

import java.net.URI;

import java.net.http.HttpClient;

import java.net.http.HttpRequest;

import java.net.http.HttpResponse;

import java.util.concurrent.CompletableFuture;

public class HttpAsyncExample {

public static void main(String[] args) {

HttpClient client = HttpClient.newHttpClient();

HttpRequest request = HttpRequest.newBuilder()

.uri(URI.create("https://jsonplaceholder.typicode.com/posts/2"))

.build();

CompletableFuture<HttpResponse<String>> future =

client.sendAsync(request, HttpResponse.BodyHandlers.ofString());

future.thenApply(HttpResponse::body)

.thenAccept(body -> System.out.println("Async Response: " + body))

.join(); // wait for completion

}

}

Output:

Async Response: {

"userId": 1,

"id": 2,

"title": "qui est esse",

"body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"

}

HttpFileDownloadExample

package HTTPAPI;

import java.net.URI;

import java.net.http.HttpClient;

import java.net.http.HttpRequest;

import java.net.http.HttpResponse;

import java.nio.file.Path;

public class HttpFileDownloadExample {

public static void main(String[] args) throws Exception {

HttpClient client = HttpClient.newHttpClient();

HttpRequest request = HttpRequest.newBuilder()

.uri(URI.create("https://jsonplaceholder.typicode.com/posts"))

.build();

Path file = Path.of("output.json");

HttpResponse<Path> response = client.send(request, HttpResponse.BodyHandlers.ofFile(file));

System.out.println("Downloaded file: " + response.body().toAbsolutePath());

}

}

Output:

Downloaded file: D:\Eclipse Data\Prasanth\Java8\Java11Features\output.json

HttpGetExample:

package HTTPAPI;

import java.net.URI;

import java.net.http.HttpClient;

import java.net.http.HttpRequest;

import java.net.http.HttpResponse;

public class HttpGetExample {

public static void main(String[] args) throws Exception {

HttpClient client = HttpClient.newHttpClient();

HttpRequest request = HttpRequest.newBuilder()

.uri(URI.create("https://jsonplaceholder.typicode.com/posts/1"))

.GET()

.build();

HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());

System.out.println("Status Code: " + response.statusCode());

System.out.println("Response Body: " + response.body());

}

}

Output:

Status Code: 200

Response Body: {

"userId": 1,

"id": 1,

"title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",

"body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"

}

HttpPostExample:( Helper Class (Reusable))

package HTTPAPI;

import java.net.URI;

import java.net.http.HttpClient;

import java.net.http.HttpRequest;

import java.net.http.HttpResponse;

import java.time.Duration;

import java.util.concurrent.CompletableFuture;

public class HttpHelper {

private final HttpClient client;

public HttpHelper() {

this.client = HttpClient.newBuilder()

.version(HttpClient.Version.HTTP\_2)

.connectTimeout(Duration.ofSeconds(10))

.build();

}

public CompletableFuture<String> getJsonAsync(String url) {

HttpRequest request = HttpRequest.newBuilder()

.uri(URI.create(url))

.header("Accept", "application/json")

.GET()

.build();

return client.sendAsync(request, HttpResponse.BodyHandlers.ofString())

.thenApply(HttpResponse::body);

}

}

**package** HTTPAPI;

**import** java.net.URI;

**import** java.net.http.HttpClient;

**import** java.net.http.HttpRequest;

**import** java.net.http.HttpResponse;

**public** **class** HttpPostExample {

**public** **static** **void** main(String[] args) **throws** Exception {

HttpClient client = HttpClient.*newHttpClient*();

String json = "{\"title\":\"foo\",\"body\":\"bar\",\"userId\":1}";

HttpRequest request = HttpRequest.*newBuilder*()

.uri(URI.*create*("https://jsonplaceholder.typicode.com/posts"))

.header("Content-Type", "application/json")

.POST(HttpRequest.BodyPublishers.*ofString*(json))

.build();

HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.*ofString*());

System.***out***.println("Status Code: " + response.statusCode());

System.***out***.println("Response Body: " + response.body());

}

}

Output:

Status Code: 201

Response Body: {

"title": "foo",

"body": "bar",

"userId": 1,

"id": 101

}

**Key Points to Remember**

* HttpClient should be reused (thread-safe).
* sendAsync → returns CompletableFuture, non-blocking.
* .join() → blocks until async result is ready.
* Use .header("Accept","application/json") to tell server you want JSON.
* Exception handling can be added using .exceptionally(ex -> {...}).