The New HTTP Client API in Java 11

developer.oracle.com

Sergey Kuksenko
Java Platform Group, Oracle
October, 2018

Live for the Code

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

About me

- Java/JVM Performance Engineer at Oracle, @since 2010
- Java/JVM Performance Engineer, @since 2005
- Java/JVM Engineer, @since 1996

Intro

GET something

```
try {
    URL url = new URL("http://some.url.you.want.to.get.org");
    HttpURLConnection con = (HttpURLConnection) url.openConnection();
    con.setRequestMethod("GET");
    con.setRequestProperty("User-Agent", "Java 1.1");
    if(con.getResponseCode() == 200) {
        System.out.println(readInputStream(con.getInputStream()));
    } else {
        System.out.println("Something wrong there!");
    }
} catch (IOException e) {
    System.out.println("Something wrong here!");
}
```

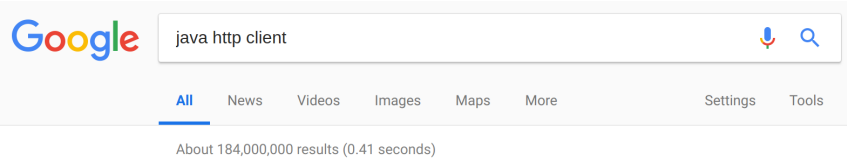
write some utils

```
public static String readInputStream(InputStream is) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(is));
    String inputLine;
    StringBuffer streamContent = new StringBuffer();
    while ((inputLine = br.readLine()) != null) {
        streamContent.append(inputLine);
    }
    br.close();
    return streamContent.toString();
}
```

POST something

```
try {
    URL url = new URL("http://some.url.you.want.to.post.org");
    HttpURLConnection con = (HttpURLConnection) url.openConnection();
    con.setRequestMethod("POST");
    con.setRequestProperty("User-Agent", "Java 1.1");
    con.setDoOutput(true);
    DataOutputStream dos = new DataOutputStream(con.getOutputStream());
    dos.writeBytes("post=something");
    dos.flush();
    dos.close();
    if (con.getResponseCode() == 200) {
        System.out.println(readInputStream(con.getInputStream()));
    } else {
        System.out.println("Something wrong there!");
    }
} catch (IOException e) {
    System.out.println("Something wrong here!");
}
```

What to do?



- Async Http Client?
- Feign?
- OkHttp?
- ...???

- Spring RestTemplate?
- Jetty HTTP Client?
- Netty HTTP Client?

New Java Http Client

GET something

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("http://some.url.you.want.to.get.org"))
        .GET()
        .header("User-Agent", "Java 11")
        .build();
try {
    HttpResponse<String> response = client.send(request,
                                      HttpResponse.BodyHandlers.ofString());
    if (response.statusCode() == 200) {
        System.out.println(response.body());
    } else {
        System.out.println("Something wrong there!");
    }
} catch (IOException e) {
    System.out.println("Something wrong here!");
}
```

GET something

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("http://some.url.you.want.to.get.org"))
        .GET()
        .header("User-Agent", "Java 11")
        .build();
try {
    HttpResponse<String> response = client.send(request,
            responseInfo -> responseInfo.statusCode() == 200 ?
                HttpResponse.BodySubscribers.ofString(UTF_8) :
                HttpResponse.BodySubscribers.replacing("Something wrong there!")
    );
} catch (IOException e) {
    System.out.println("Something wrong here!");
}
```

GET something

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("http://some.url.you.want.to.get.org"))
        .GET()
        .header("User-Agent", "Java 11")
        .build();
CompletableFuture<HttpResponse<String>> responseFuture = client.sendAsync(request,
        responseInfo -> responseInfo.statusCode() == 200 ?
                HttpResponse.BodySubscribers.ofString(UTF_8) :
                HttpResponse.BodySubscribers.replacing("Something wrong there!")
);
responseFuture
        .thenApply(HttpResponse::body)
        .exceptionally(ex -> "Something wrong here!")
        .thenAccept(System.out::println);
```

GET something

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("http://some.url.you.want.to.get.org"))
        .GET()
        .header("User-Agent", "Java 11")
        .build();
client.sendAsync(request,
        responseInfo -> responseInfo.statusCode() == 200 ?
                HttpResponse.BodySubscribers.ofString(UTF_8) :
                HttpResponse.BodySubscribers.replacing("Something wrong there!"))
        .thenApply(HttpResponse::body)
        .exceptionally(ex -> "Something wrong here!")
        .thenAccept(System.out::println);
```

Motivation

Motivation

- 1996 - `URLConnection`
- 1996 - HTTP/1.0 (RFC 1945)
- 1997 - `HttpURLConnection`
- 1997 - HTTP/1.1 (RFC 2068)
- 1999 - HTTP/1.1 (RFC 2616)
- 2015 - HTTP/2 (RFC 7540)

Motivation

`HttpURLConnection/URLConnection`:

- outdated API
- API designed before HTTP/1.1
- hard to use, many undocumented behaviors
- blocking mode only

HTTP Client history

- May 2014 - JEP 110: HTTP/2 Client
- Sep 2017 - JDK 9 shipped (Incubator)
- Mar 2018 - JDK 10, updated API (Incubator)
- Sep 2018 - JDK 11, JEP 321: HTTP Client (Standard)

JEP 110 Goals:

- Support HTTP/2
- Asynchronous API
- Usage of new Java features
- ...

HTTP/2

- Binary format
- Single connection. Multiplexing.
- Header compression
- Server Push
- ...

HTTP Client

HTTP Client

- Supports HTTP/1.1 and HTTP/2
- Prefers HTTP/2, by default
  - Upgrade clear text request
  - Negotiate h2 in the APLN for HTTP over TLS

The Java HTTP Client

The Java HTTP Client

- Incubated in Java 9, JEP 110
- Standardized in Java 11, JEP 321
  - Module `java.net.http`
  - Package `java.net.http`
  - Class `java.net.http.HttpClient`
- Part of the Java SE Platorm
  - Classes and types are visible just like any others Java SE
  - Modular applicaton code should `require java.net.http`

java.net.http

- HttpClient
- HttpRequest
- HttpResponse<T>

HttpClient

```
HttpClient client = HttpClient.newBuilder()
        .authenticator(Authenticator)
        .connectTimeout(Duration)
        .cookieHandler(CookieHandler)
        .executor(Executor)
        .followRedirects(HttpClient.Redirect)
        .proxy(ProxySelector)
        .sslContext(SSLContext)
        .sllParameters(SSLParameters)
        .version(HttpClient.Version) // HTTP_1_1,  HTTP_2
        .build();
```

HttpRequest

```
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("http://openjdk.java.net"))
        .GET()
        .timeout(Duration.ofSeconds(42))
        .version(HttpClient.Version.HTTP_2)
        .build();
```

HttpRequest.Builder

- Build request header

```
header(String name, String value)
headers(String... headers)
setHeader(String name, String value)
```

HttpRequest.Builder

- Build request header

```
header(String name, String value)
headers(String... headers)
setHeader(String name, String value)
```

- Set request method

```
GET()
DELETE()
POST(HttpRequest.BodyPublisher bodyPublisher)
PUT(HttpRequest.BodyPublisher bodyPublisher)
method(String method, HttpRequest.BodyPublisher bodyPublisher)
```

HttpRequest

Once built an HttpRequest is immutable,
and can be sent multple times.

HttpRequest.BodyPublishers

Factory methods for common body types

```
noBody()
ofByteArray(byte[] buf)
ofByteArray(byte[] buf, int offset, int length)
ofByteArrays(Iterable<byte[]> iter)
ofFile(Path path)
ofInputStream(Supplier<? extends InputStream> streamSupplier)
ofString(String body)
ofString(String s, Charset charset)
```

HttpResponse

```
interface HttpResponse<T> {

        T            body();

        int          statusCode();

        HttpHeaders  headers();

        HttpRequest  request();

        URI          uri();

        HttpClient.Version  version();

        Optional<SSLSession> sslSession();

}
```

HttpClient

- Synchronous request(blocking)

```
HttpResponse<T>

    send(HttpRequest, HttpResponse.BodyHandler<T>);
```

- Asynchronous request

```
CompletableFuture<HttpResponse<T>>

    sendAsync(HttpRequest,
              HttpResponse.BodyHandler<T>);
```

HttpResponse.BodyHandlers

Factory methods for common body types

```
BodyHandler<Void> discarding();

BodyHandler<U> replacing(U value);

BodyHandler<String> ofString();
BodyHandler<String> ofString(Charset charset);

BodyHandler<Stream<String>> ofLines();

BodyHandler<InputStream> ofInputStream();

BodyHandler<byte[]> ofByteArray();
```

HttpResponse.BodyHandlers

Factory methods for common body types (cont.)

```
BodyHandler<T> buffering(BodyHandler<T> downstreamHandler, int bufferSize);


BodyHandler<Path> ofFile(Path file, OpenOption... openOptions);
BodyHandler<Path> ofFile(Path file);
BodyHandler<Path> ofFileDownload(Path directory, OpenOption... openOptions);
```

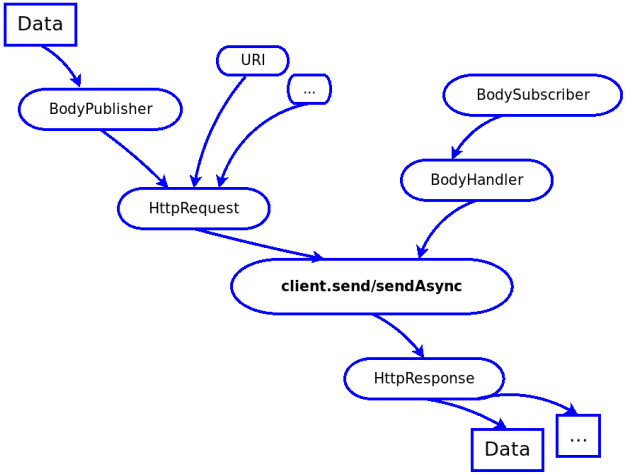
HTTP/2 server push

```
public CompletableFuture<HttpResponse<T>>

sendAsync(HttpRequest request,
          BodyHandler<T> responseBodyHandler,
          PushPromiseHandler<T> pushPromiseHandler);
```

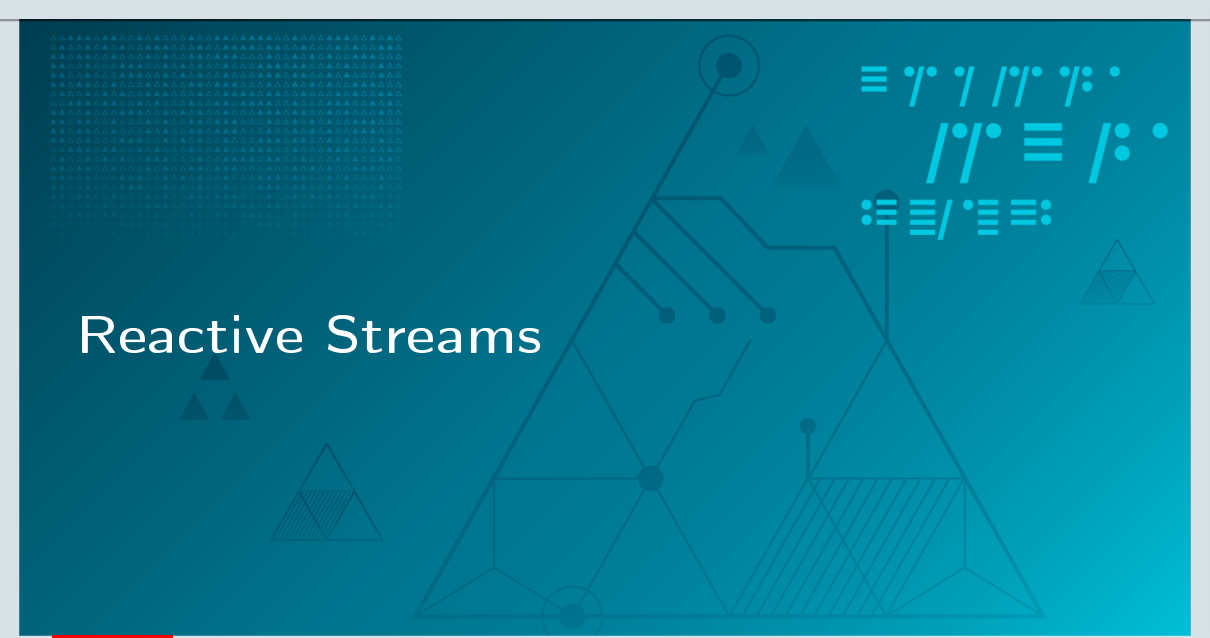
HttpResponse.PushPromiseHandler

```
public interface PushPromiseHandler<T> {

    public void applyPushPromise(
        HttpRequest initiatingRequest,
        HttpRequest pushPromiseRequest,
        Function<BodyHandler<T>,CompletableFuture<HttpResponse<T>>> acceptor
    );


    public static <T> PushPromiseHandler<T>
    of(Function<HttpRequest,BodyHandler<T>> pushPromiseHandler,
       ConcurrentMap<HttpRequest,CompletableFuture<HttpResponse<T>>> pushPromisesMap);

}
```

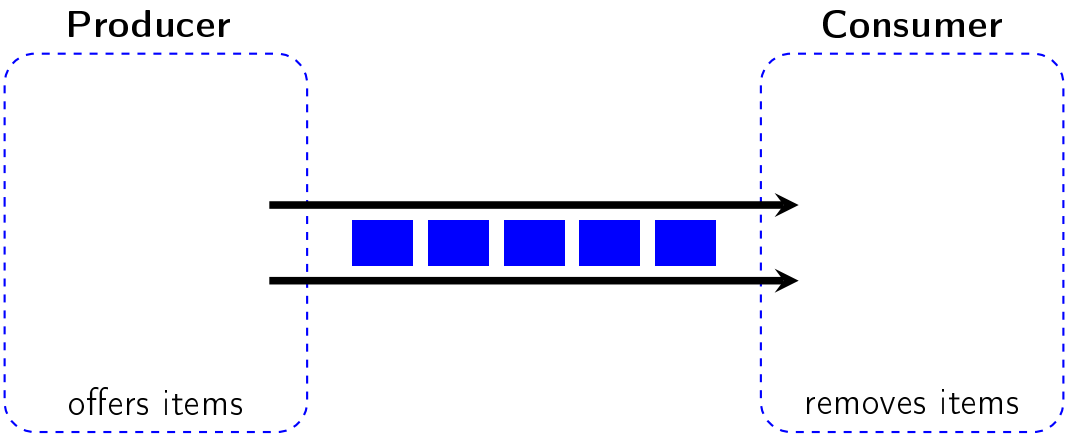
All together



Reactive Streams



Classic Producer-Consumer



Reactive streams

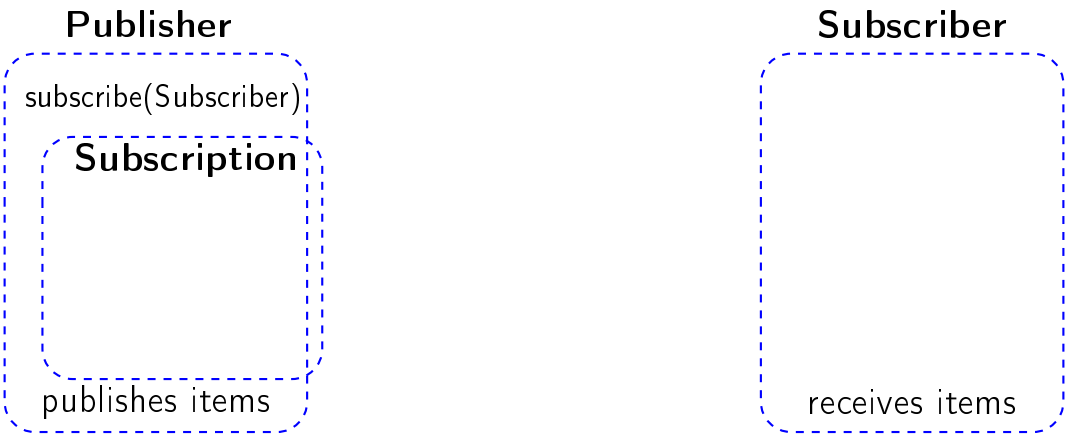
- `java.util.concurrent.Flow`, `@since Java9`
  - `j.u.c.Flow.Publisher<T>`
  - `j.u.c.Flow.Flow.Subscriber<T>`
  - `j.u.c.Flow.Flow.Flow.Subscription`
- One ~~ring~~ interface to rule them all

Reactive streams

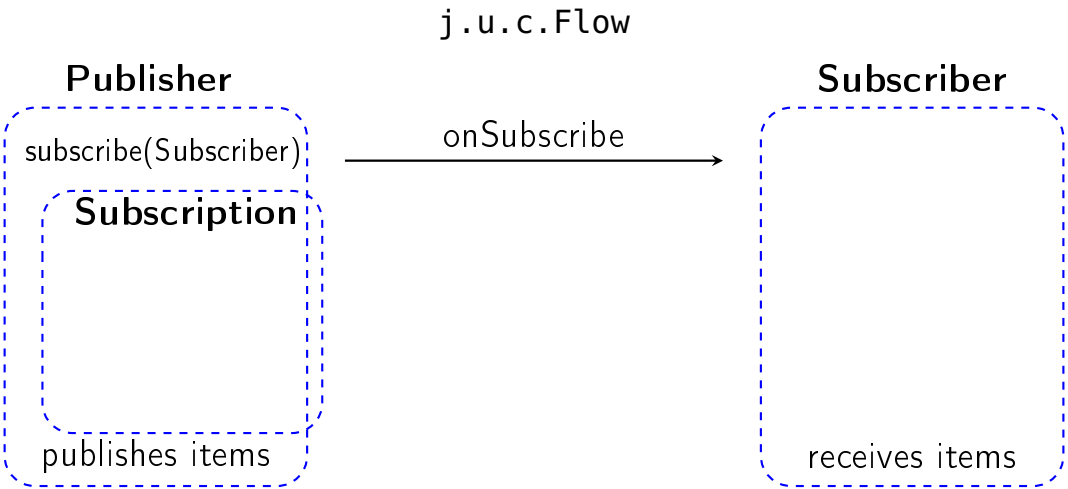
`j.u.c.Flow`

Reactive streams

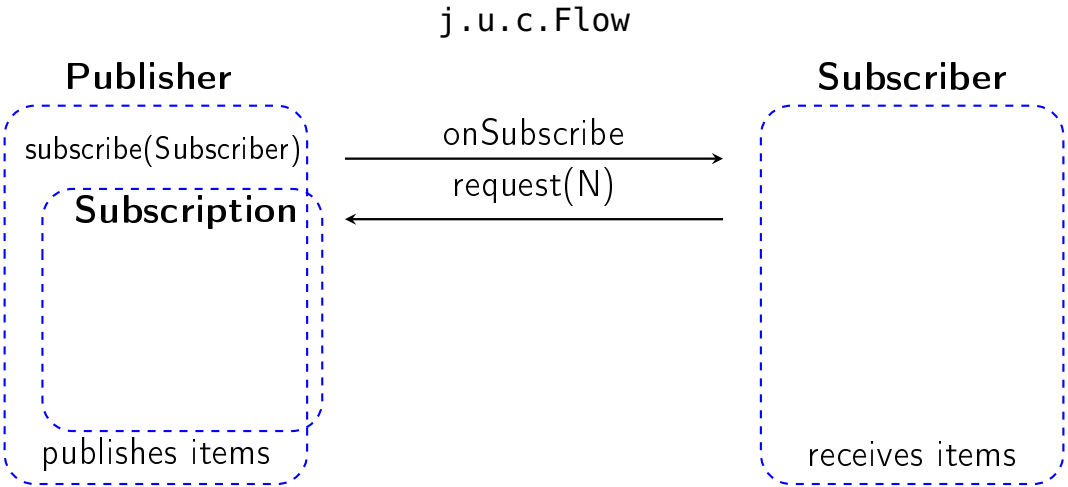
`j.u.c.Flow`



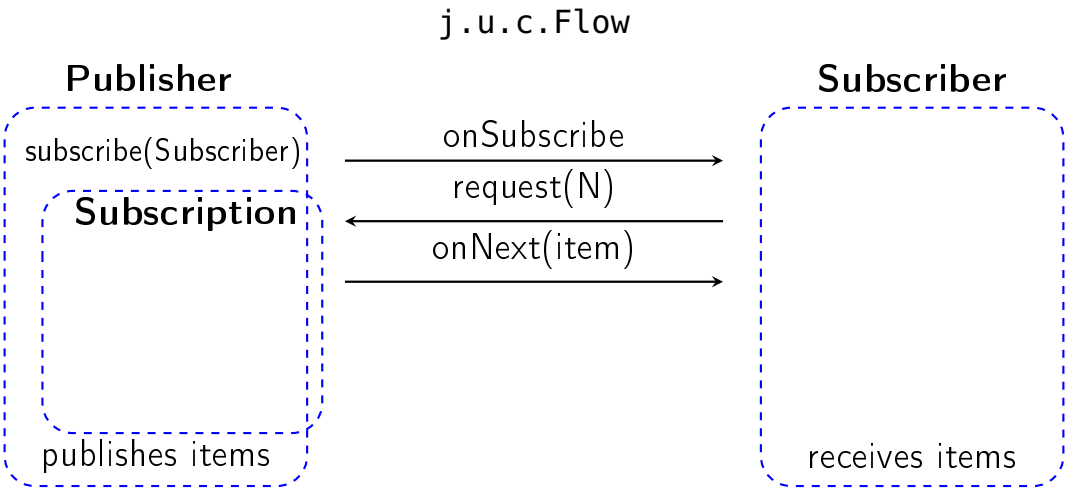
Reactive streams



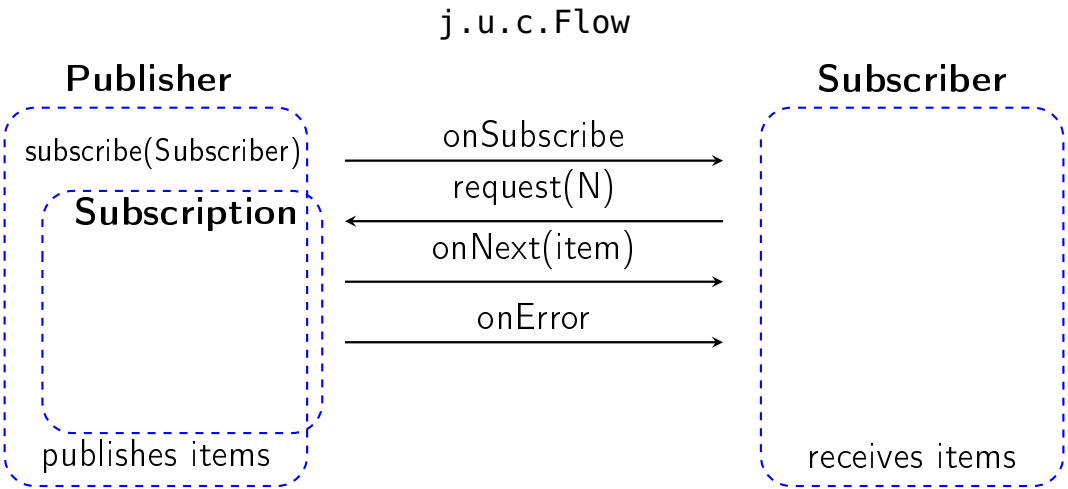
Reactive streams



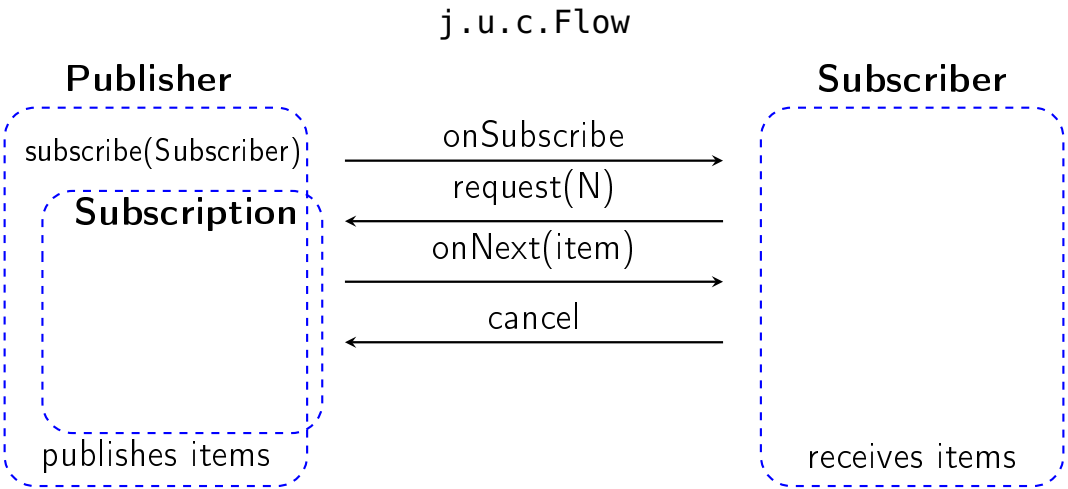
Reactive streams



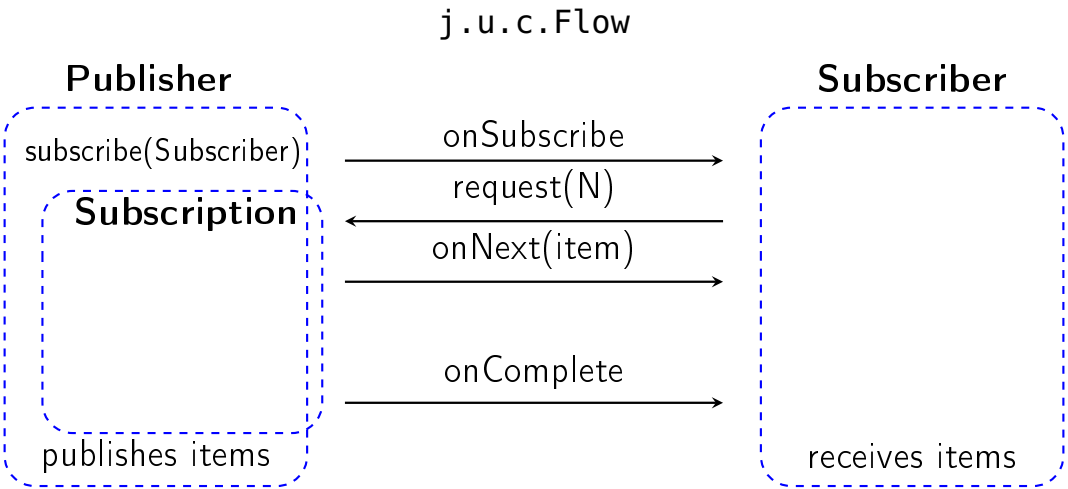
Reactive streams



Reactive streams



Reactive streams



Data as reactive streams

- `HttpClient` sends data to the server
- `HttpClient` receives data from the server

Data as reactive streams

- ~~`HttpClient` sends data to the server~~
- `HttpClient` publishes data to the server
- ~~`HttpClient` receives data from the server~~
- `HttpClient` subscribes to data from the server

HttpRequest.BodyPublisher

```
public interface BodyPublisher extends Flow.Publisher<ByteBuffer> {

    long contentLength();

}
```

HttpResponse.BodyHandler

```
@FunctionalInterface
public interface BodyHandler<T> {

    public BodySubscriber<T> apply(ResponseInfo responseInfo);

}

public interface ResponseInfo {

    public int statusCode();

    public HttpHeaders headers();

    public HttpClient.Version version();

}
```

HttpResponse.BodySubscriber

```
public interface BodySubscriber<T> extends Flow.Subscriber<List<ByteBuffer>> {

    public CompletionStage<T> getBody();

}
```

Making Custom Subscriber

e.g. I don't need a file, just MD5

```
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("http://some.url.you.want.to.md5.org"))
        .GET()
        .build();

HttpResponse<byte[]> response = client.send(request,
                        HttpResponse.BodyHandlers.ofByteArray());
if(response.statusCode() == 200) {
    MessageDigest digest = MessageDigest.getInstance("MD5");
    byte[] md5 = digest.digest(response.body());
    ...
```

e.g. MD5Subscriber

```
public class MD5Subscriber implements HttpResponse.BodySubscriber<byte[]> {

    public CompletionStage<byte[]> getBody();

    public void onSubscribe(Flow.Subscription subscription);

    public void onNext(List<ByteBuffer> item);

    public void onError(Throwable throwable);

    public void onComplete();

}
```

e.g. MD5Subscriber

```
public class MD5Subscriber implements HttpResponse.BodySubscriber<byte[]> {

    private final CompletableFuture<byte[]> future = new CompletableFuture<>();

    private MessageDigest digest;

    private Flow.Subscription subscription;

    public MD5Subscriber() {
        try {
            digest = MessageDigest.getInstance("MD5");
        } catch (NoSuchAlgorithmException e) {
            // do something here
        }
    }
```

e.g. MD5Subscriber

```
public class MD5Subscriber implements HttpResponse.BodySubscriber<byte[]> {

    private final CompletableFuture<byte[]> future = new CompletableFuture<>();

    private MessageDigest digest;

    private Flow.Subscription subscription;

    @Override
    public CompletionStage<byte[]> getBody() {
        return future;
    }
```

e.g. MD5Subscriber

```
public class MD5Subscriber implements HttpResponse.BodySubscriber<byte[]> {

    private final CompletableFuture<byte[]> future = new CompletableFuture<>();

    private MessageDigest digest;

    private Flow.Subscription subscription;

    @Override
    public void onSubscribe(Flow.Subscription subscription) {
        this.subscription = subscription;
        subscription.request(1);
    }
```

e.g. MD5Subscriber

```
public class MD5Subscriber implements HttpResponse.BodySubscriber<byte[]> {

    private final CompletableFuture<byte[]> future = new CompletableFuture<>();

    private MessageDigest digest;

    private Flow.Subscription subscription;

    @Override
    public void onNext(List<ByteBuffer> item) {
        item.forEach(digest::update);
        subscription.request(1);
    }
```

e.g. MD5Subscriber

```
public class MD5Subscriber implements HttpResponse.BodySubscriber<byte[]> {

    private final CompletableFuture<byte[]> future = new CompletableFuture<>();

    private MessageDigest digest;

    private Flow.Subscription subscription;

    @Override
    public void onError(Throwable throwable) {
        future.completeExceptionally(throwable);
    }
```

e.g. MD5Subscriber

```
public class MD5Subscriber implements HttpResponse.BodySubscriber<byte[]> {

    private final CompletableFuture<byte[]> future = new CompletableFuture<>();

    private MessageDigest digest;

    private Flow.Subscription subscription;

    @Override
    public void onComplete() {
        future.complete(digest.digest());
    }

}
```

e.g. I don't need a file, just MD5

```
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("http://some.url.you.want.to.md5.org"))
        .GET()
        .build();

HttpResponse<byte[]> response =
                        client.send(request, ri ->  new MD5Subscriber());
byte[] md5 = response.body();
...
```

Appendix

advanced client configuration

Client settings frame (HTTP/2)

Property (`jdk.httpclient`)	Client default value	HTTP/2 standard value
`hpack.maxheadertablesize`	16K	4K
`enablepush`	1	1
`maxstreams`	100	
`windowsize`	16M	64K-1
`maxframesize`	16K	16K

Property (jdk.httpclient)	HTTP	default value
bufsize	1.1, 2	16K
connectionWindowSize	2	32M
auth.retrylimit	1.1, 2	3
redirects.retrylimit	1.1, 2	5
keepalive.timeout	1.1	1200 secs
connectionPoolSize	1.1	0 (unbounded)

Q & A ?