
JavaOne™

Future<HttpClient>: A Sneak Peek into a New HttpClient API

Michael McMahon
Danny Coward

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, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Program Agenda

- Background: legacy API
- New API requirements (for JDK8)
- Details of new API with code examples
- Open issues
- Q&A

Http Interactions

Http Request

POST /poem.jsp HTTP/1.1

Headers

Host: www.myrhme.com

Content-Type: text/plain

Body

Mary had a little lamb

She thought it very silly

She threw it up into the air

And caught it by ...

Http Interactions

Http Request

POST /poem.jsp HTTP/1.1

Headers

Host: www.myrhme.com

Content-Type: text/plain

Body

Mary had a little lamb

She thought it very silly

She threw it up into the air

And caught it by ...

Http Interactions

Http Request

POST /poem.jsp HTTP/1.1

Headers

Host: www.myrhme.com

Content-Type: text/plain

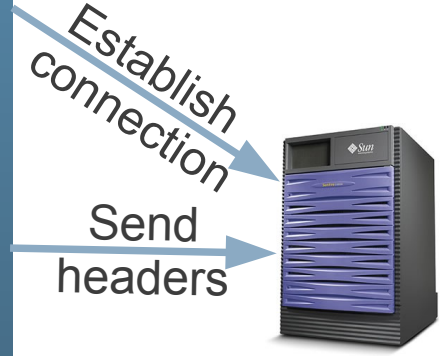
Body

Mary had a little lamb

She thought it very silly

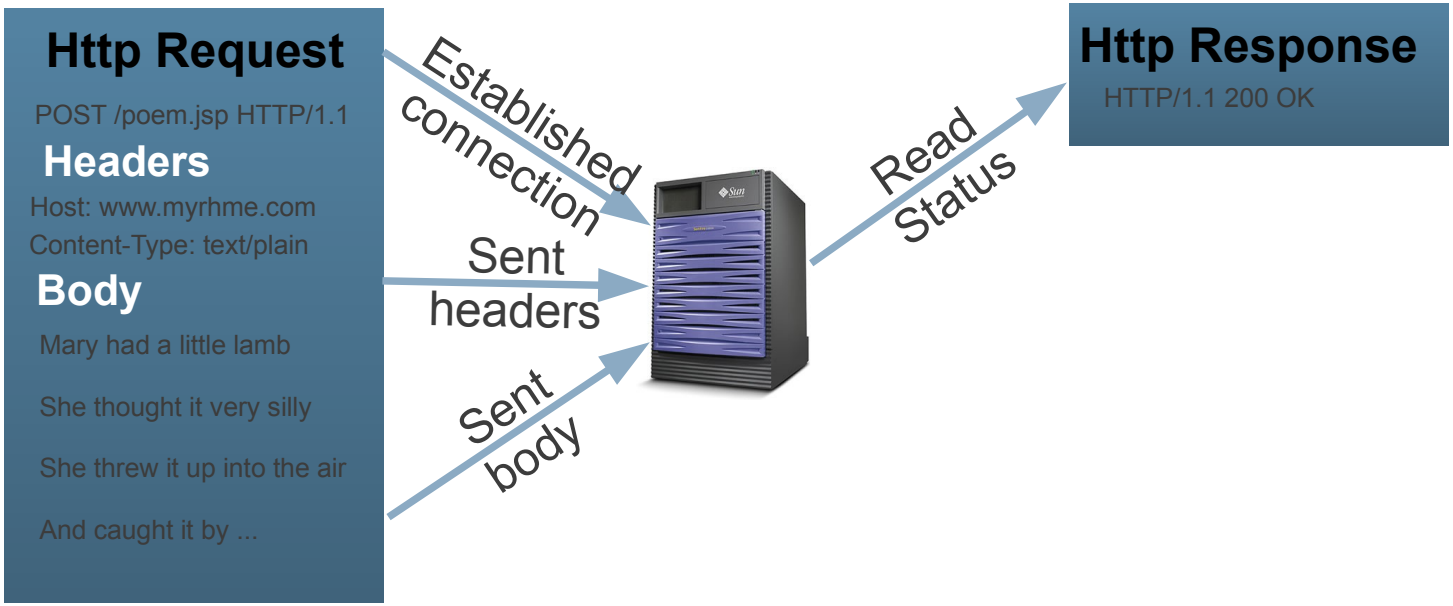
She threw it up into the air

And caught it by ...

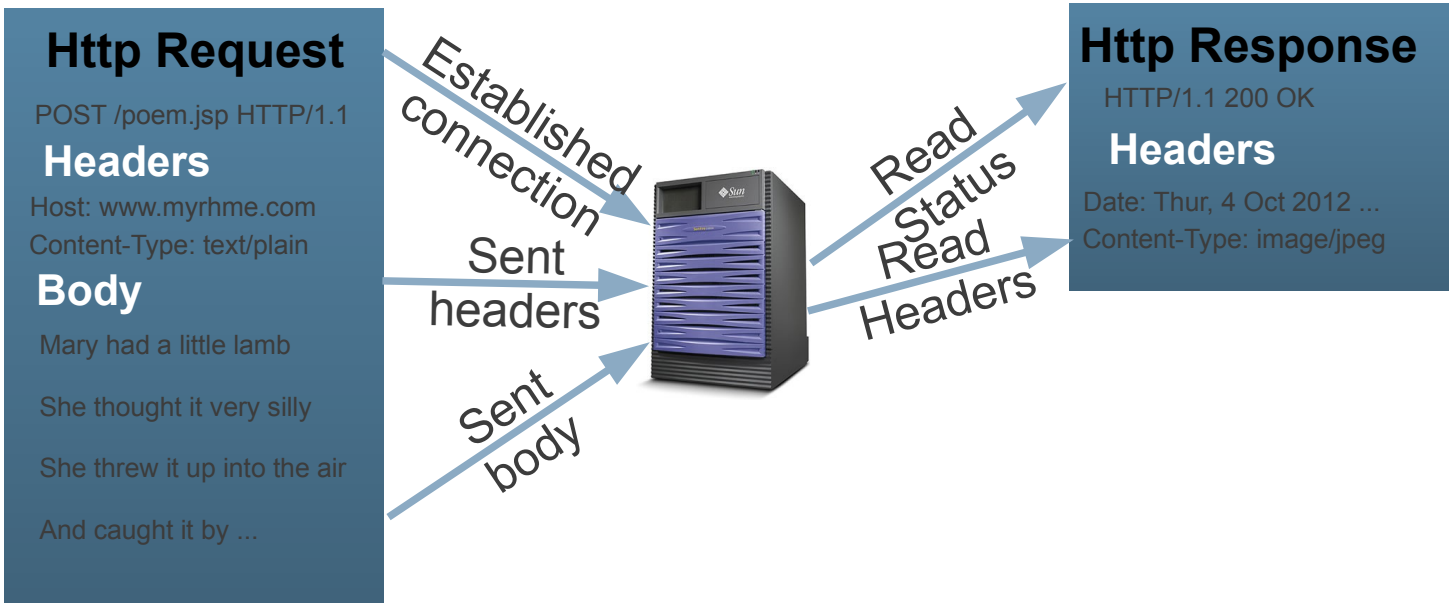


Http Interactions

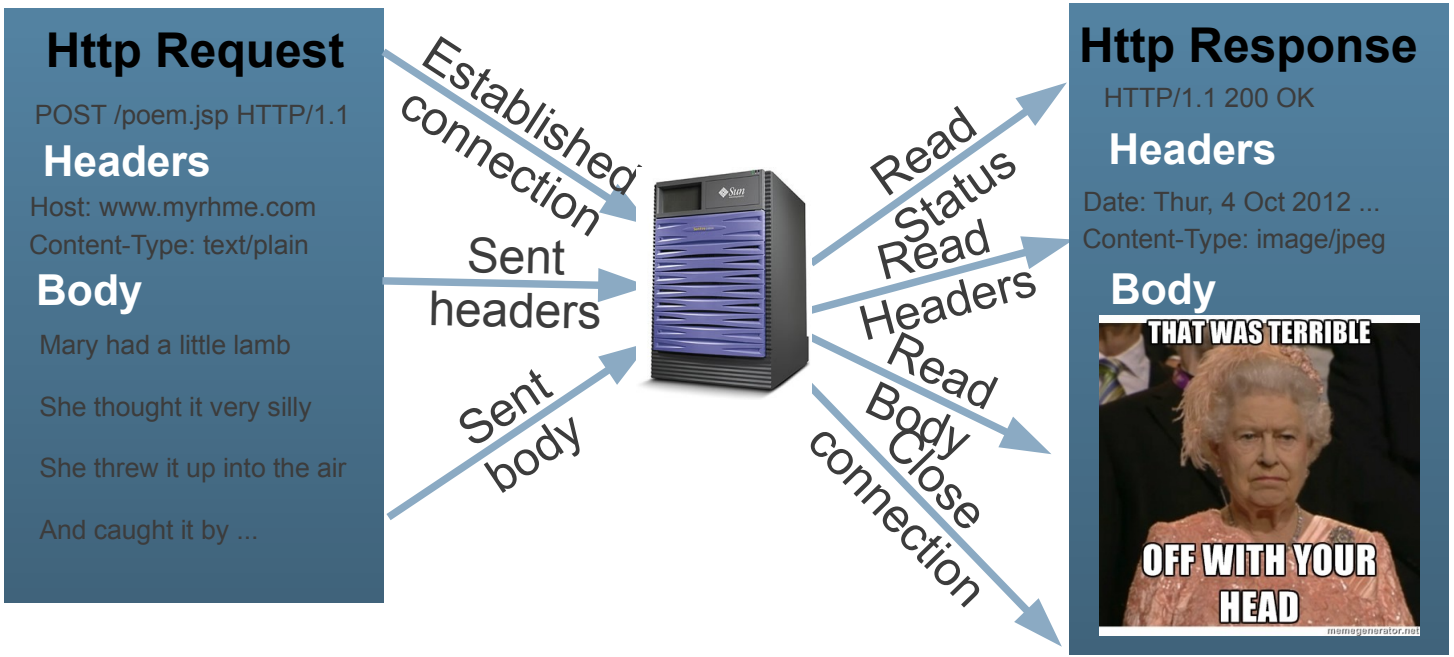
Http Interactions



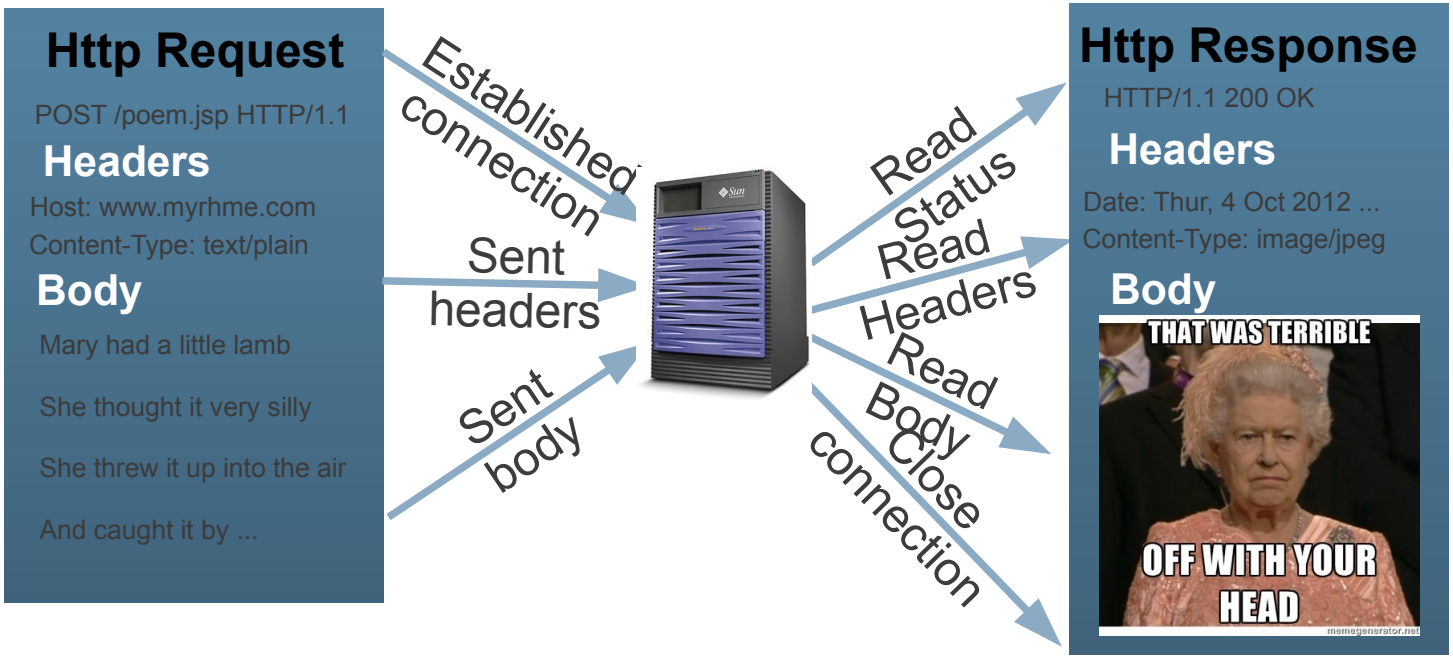
Http Interactions



Http Interactions



Http Interactions



Java API today: **java.net.HttpURLConnection**

HttpURLConnection example

```
static void get(URL url) throws IOException {
        InputStream responseBody = null; int http_status;
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        try {
            connection.addRequestProperty("Content-Type", "text/html");
            connection.setDoOutput(true);
            connection.connect();
            OutputStream os = connection.getOutputStream();
            os.write("hello world ?".getBytes());
            os.write("world ?".getBytes());
            os.close();
            if (connection.getResponseCode() != 200)
                // handle redirects, server errors...
            responseBody = connection.getInputStream();
            String conType = connection.getHeaderField("Content-Type");
        } catch (IOException e) {// handle other horrible things }
        try {
            processResponseBody(responseBody);
        } catch (IOException e) {
            // a number of connection or read failures
        } finally {
            connection.disconnect(); // honor system
        }
```

HttpURLConnection example

```
static void get(URL url) throws IOException {
        InputStream responseBody = null; int http_status;
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        try {
            connection.addRequestProperty("Content-Type", "text/html");
            connection.setDoOutput(true);
            connection.connect();
            OutputStream os = connection.getOutputStream();
            os.write("hello ".getBytes());
            os.write("world ?".getBytes());
            os.close();
            if (connection.getResponseCode() != 200)
                // handle redirects, server errors...
            responseBody = connection.getInputStream();
            String conType = connection.getHeaderField("Content-Type");
        } catch (IOException e) {// handle other horrible things }
        try {
            processResponseBody(responseBody);
        } catch (IOException e) {
            // a number of connection / read failures
        } finally {
            connection.disconnect(); // honor system
        }
```

API spread across URLConnection and HttpURLConnection

HttpURLConnection example

```
static void get(URL url) throws IOException {
        InputStream responseBody = null; int http_status;
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        try {
            connection.addRequestProperty("Content-Type", "text/html");
            connection.setDoOutput(true);
            connection.connect();
            OutputStream os = connection.getOutputStream();
            os.write("hello world ?".getBytes());
            os.write("world ?".getBytes());
            os.close();
            if (connection.getResponseCode() != 200)
                // handle redirects, server errors...
            responseBody = connection.getInputStream();
            String conType = connection.getHeaderField("Content-Type");
        } catch (IOException e) {// handle other horrible things }
        try {
            processResponseBody(responseBody);
        } catch (IOException e) {
            // a number of connection or read failures
        } finally {
            connection.disconnect(); // honor system
        }
```


HttpURLConnection example

Connection state not modeled in API (part 1)

```
static void get(URL url) throws IOException {
        InputStream responseBody = null; int http_status;
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        try {
            connection.connect();
            connection.addRequestProperty("Content-Type", "text/html");
            connection.setDoOutput(true);
            OutputStream os = connection.getOutputStream();
            os.write("hello ".getBytes());
            os.write("world ?".getBytes());
            os.close();
            if (connection.getResponseCode() != 200)
                // handle redirects, server errors...
            responseBody = connection.getInputStream();
            String conType = connection.getHeaderField("Content-Type");
        } catch (IOException e) {// handle other horrible things }
        try {
            processResponseBody(responseBody);
        } catch (IOException e) {
            // a number of connection / read failures
        } finally {
            connection.disconnect(); // honor system
        }
```

HttpURLConnection example

```
static void get(URL url) throws IOException {
        InputStream responseBody = null; int http_status;
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        try {
            connection.addRequestProperty("Content-Type", "text/html");
            connection.setDoOutput(true);
            connection.connect();
            OutputStream os = connection.getOutputStream();
            os.write("hello world ?".getBytes());
            os.write("world ?".getBytes());
            os.close();
            if (connection.getResponseCode() != 200)
                // handle redirects, server errors...
            responseBody = connection.getInputStream();
            String conType = connection.getHeaderField("Content-Type");
        } catch (IOException e) {// handle other horrible things }
        try {
            processResponseBody(responseBody);
        } catch (IOException e) {
            // a number of connection or read failures
        } finally {
            connection.disconnect(); // honor system
        }
```

HttpURLConnection example

```
static void get(URL url) throws IOException {
        InputStream responseBody = null; int http_status;
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        try {
            connection.addRequestProperty("Content-Type", "text/html");
            connection.setDoOutput(true);
            connection.connect();
            OutputStream os = connection.getOutputStream();
            os.write("hello ".getBytes());
            if (connection.getResponseCode() != 200)
                // handle redirects, server errors...
            os.write("world ?".getBytes());
            os.close();

            responseBody = connection.getInputStream();
            String conType = connection.getHeaderField("Content-Type");
        } catch (IOException e) {// handle other horrible things }
        try {
            processResponseBody(responseBody);
        } catch (IOException e) {
            // a number of connection / read failures
        } finally {
            connection.disconnect(); // honor system
        }
```

Connection state not modeled in API (part 2)

HttpURLConnection example

```
static void get(URL url) throws IOException {
        InputStream responseBody = null; int http_status;
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        try {
            connection.addRequestProperty("Content-Type", "text/html");
            connection.setDoOutput(true);
            connection.connect();
            OutputStream os = connection.getOutputStream();
            os.write("hello world ?".getBytes());
            os.write("world ?".getBytes());
            os.close();
            if (connection.getResponseCode() != 200)
                // handle redirects, server errors...
            responseBody = connection.getInputStream();
            String conType = connection.getHeaderField("Content-Type");
        } catch (IOException e) {// handle other horrible things }
        try {
            processResponseBody(responseBody);
        } catch (IOException e) {
            // a number of connection or read failures
        } finally {
            connection.disconnect(); // honor system
        }
```

HttpURLConnection example

```
static void get(URL url) throws IOException {
        InputStream responseBody = null; int http_status;
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        try {
            connection.addRequestProperty("Content-Type", "text/html");
            connection.setDoOutput(true);
            connection.connect();
            OutputStream os = connection.getOutputStream();
            os.write("hello ".getBytes());
            os.write("world ?".getBytes());
            os.close();
            if (connection.getResponseCode() != 200)
                // handle redirects, server errors...
            responseBody = connection.getInputStream();
            String conType = connection.getHeaderField("Content-Type");
        } catch (IOException e) {// handle other horrible things }
        try {
            processResponseBody(responseBody);
        } catch (IOException e) {
            // a number of connection / read failures
        } finally {
            connection.disconnect(); // honor system
        }
```

Only blocking mode for read and write

HttpURLConnection drawbacks

- Connection API does not model request/response lifecycle
- Awkward class hierarchy under URLConnection
- Ok for common cases, but inflexible
- Synchronous/blocking mode only
- Limited to JDK implementation

Other Http Clients

- Apache HTTPClient
- Android HTTPClient
- Grizzly Http Client API
- OracleHTTP Client
- Jetty Http Client
- and others....

New Http client library

- Work in progress
  - Stable API: package java.net.httpclient
  - Working code and tests
- Planned for JDK 8
- Project page
  - http://java.net/projects/http-client

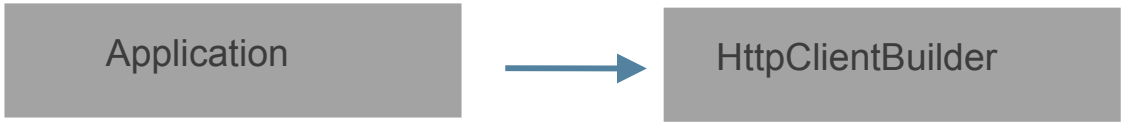
New Http client features

- Equal and exceed the functionality of HttpURLConnection
- Better model of the Http request/response interactions
- Easy to learn, easy to use
- Both asynchronous and blocking read/write modes
- Request and response filtering
- Pluggable providers
- Http Upgrade mechanism

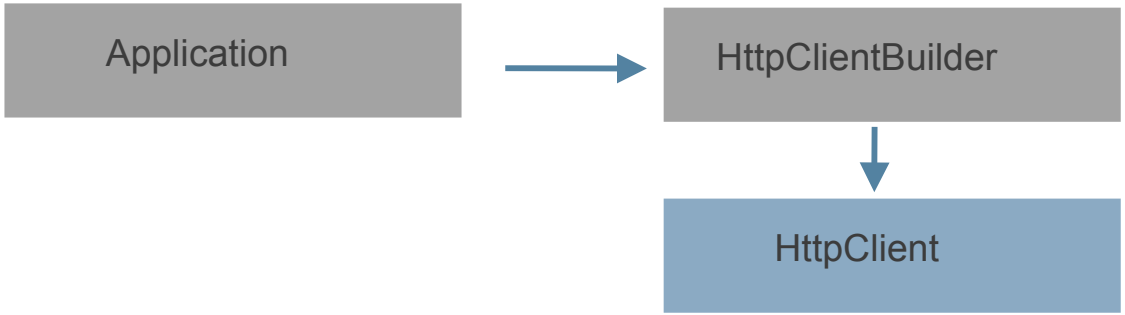
New Http client api: object creation flow

Application

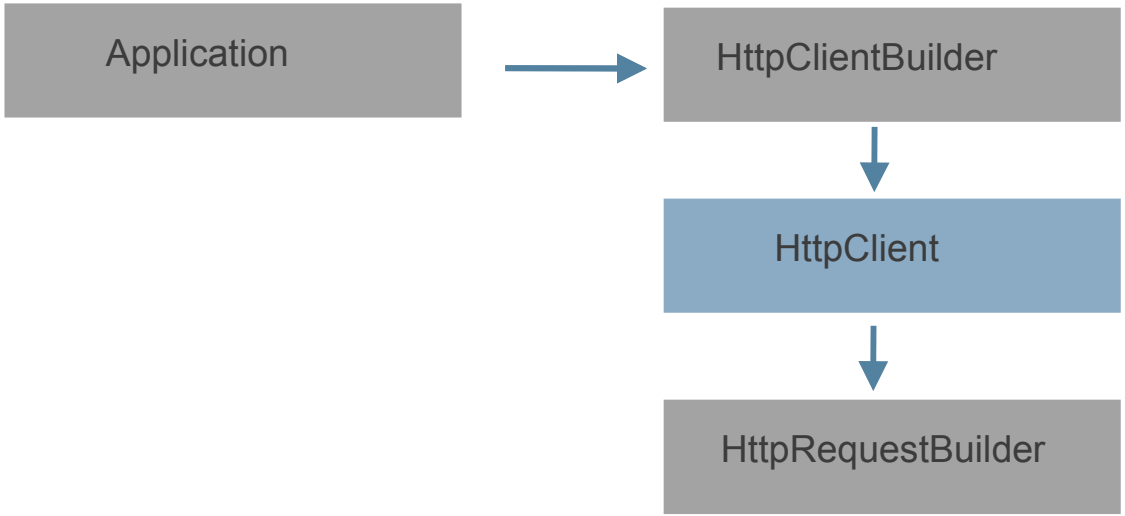
New Http client api: object creation flow



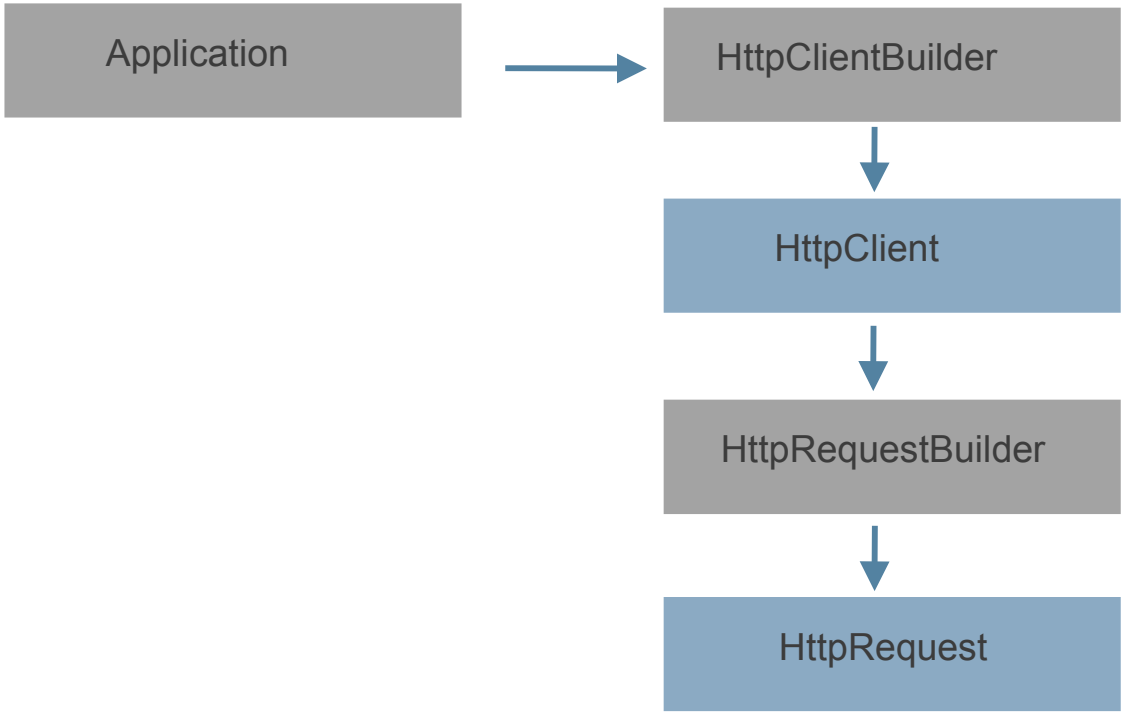
New Http client api: object creation flow



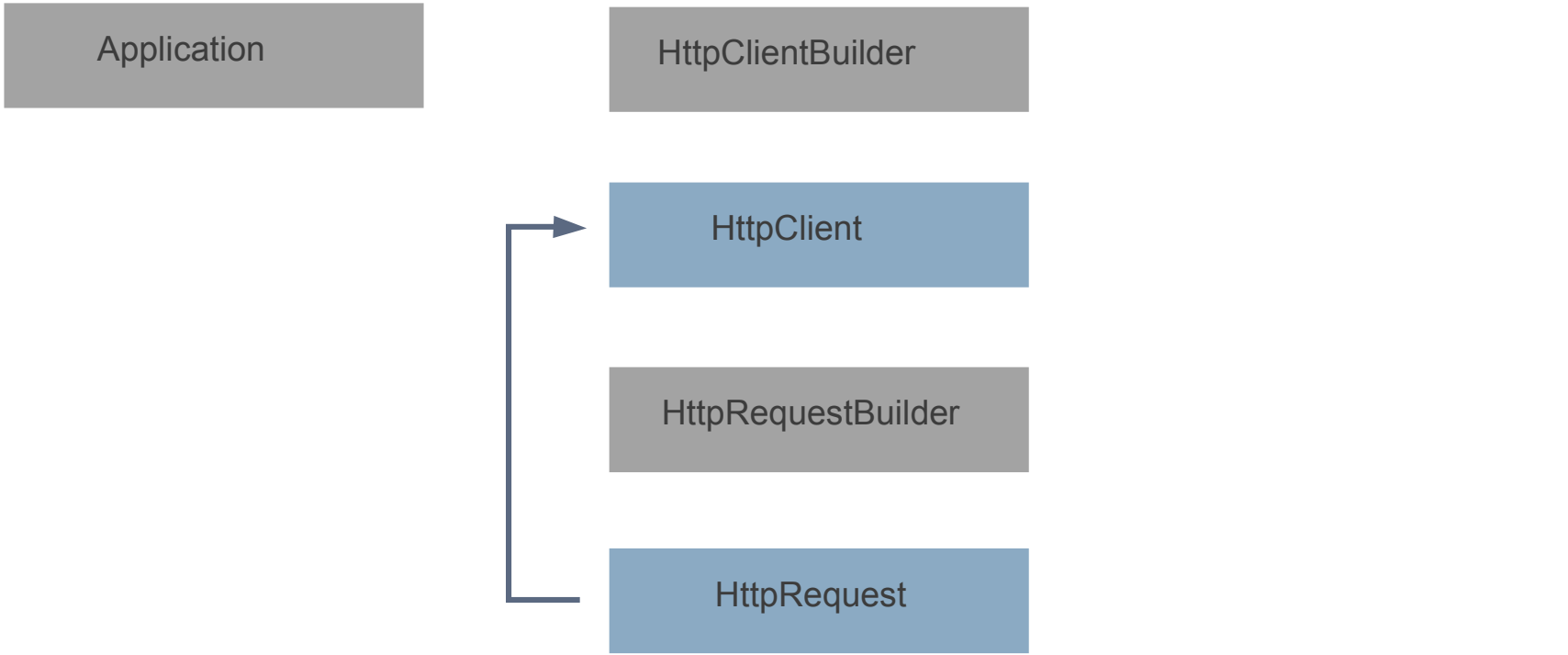
New Http client api: object creation flow



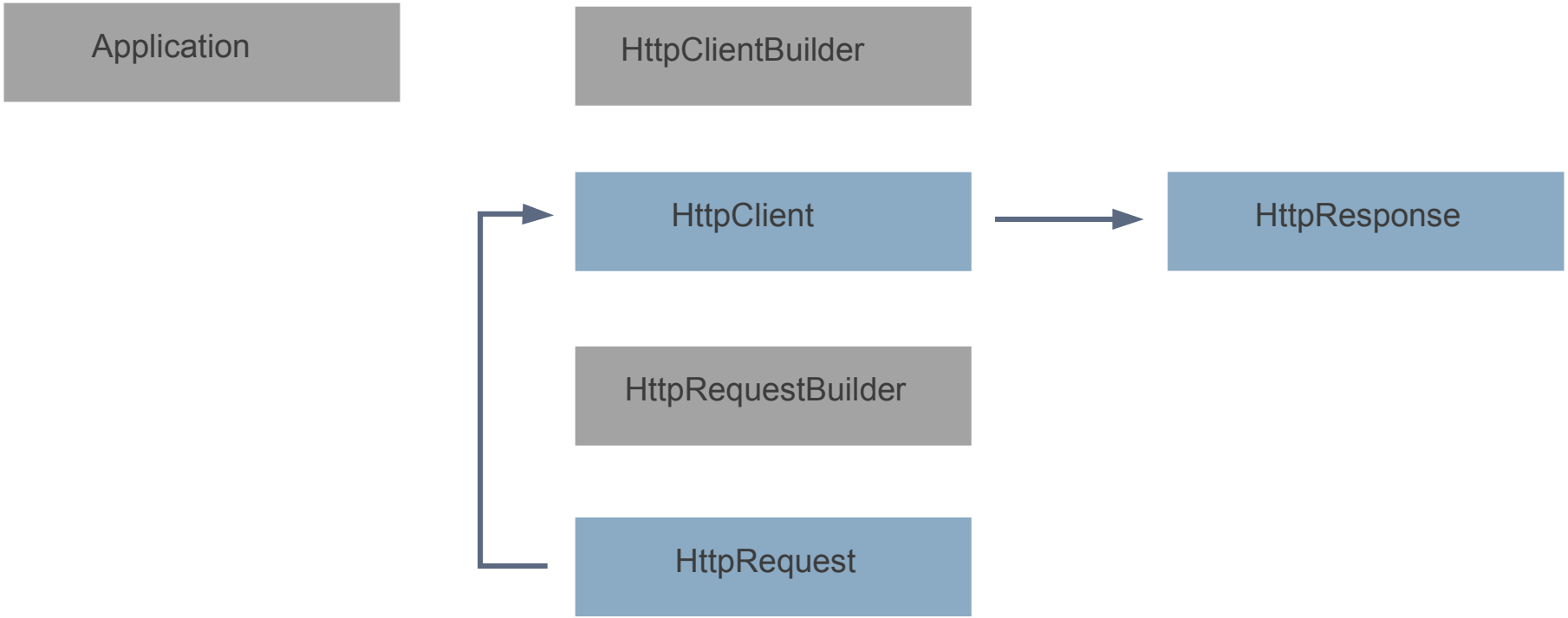
New Http client api: object creation flow



New Http client api: object interaction



New Http client api: object interaction



HttpURLConnection example

```
static void get(URL url) throws IOException {
        InputStream responseBody = null; int http_status;
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        try {
            connection.addRequestProperty("Content-Type", "text/html");
            connection.setDoOutput(true);
            connection.connect();
            OutputStream os = connection.getOutputStream();
            os.write("hello world ?".getBytes());
            os.write("world ?".getBytes());
            os.close();
            if (connection.getResponseCode() != 200)
                // handle redirects, server errors...
            responseBody = connection.getInputStream();
            String conType = connection.getHeaderField("Content-Type");
        } catch (IOException e) {// handle other horrible things }
        try {
            processResponseBody(responseBody);
        } catch (IOException e) {
            // a number of connection or read failures
        } finally {
            connection.disconnect(); // honor system
        }
```

New Client API Example

```
import java.net.httpclient.*;

HttpClient client = HttpClientBuilder
                        .createClientBuilder()
                        .build();
HttpRequestBuilder requestBuilder = client.createRequestBuilder(uri);
requestBuilder.addHeader("Content-Type", "text/html");
HttpRequest request = requestBuilder
                        .setBody("hello world ?".getBytes())
                        .build();
HttpResponse response = client.getResponse(request);
if (response.getResponseCode() != 200)
        // redirects, server errors

InputStream responseBody = response.getBodyAsInputStream();
try {
    processResponseBody(responseBody); // have to wait....
} catch (IOException e) {
    // response body processing errors
}
```

New Client API Example

```
import java.net.httpclient.*;

HttpClient client = HttpClientBuilder
                        .createClientBuilder()
                        .build();
HttpRequestBuilder requestBuilder = client.createRequestBuilder(uri);
requestBuilder.addHeader("Content-Type", "text/html");
HttpRequest request = requestBuilder
                        .setBody("hello world ?".getBytes()`
                        .build();
HttpResponse response = client.getResponse(request);
if (response.getResponseCode() != 200)
        // redirects, server errors

InputStream responseBody = response.getBodyAsInputStream();
try {
    processResponseBody(responseBody); // have to wait....
} catch (IOException e) {
    // response body processing errors
}
```

Configure client scope properties

Configure request scope properties

Various ways to fill the request body

Various ways to execute the request

Various ways to obtain the response body

New Http client api: overview of main entities

Type	Description	Mutable
HttpClientBuilder	Entry point, used to build/configure HttpClients	YES
HttpClient	Used to create and send htttp requests	NO
HttpRequestBuilder	Builds/configures HttpRequests	YES
HttpRequest	Encapsulates all request state	NO
HttpResponse	Encapsulates all response state	NO
HttpHeaders	Manages headers sent or received	YES

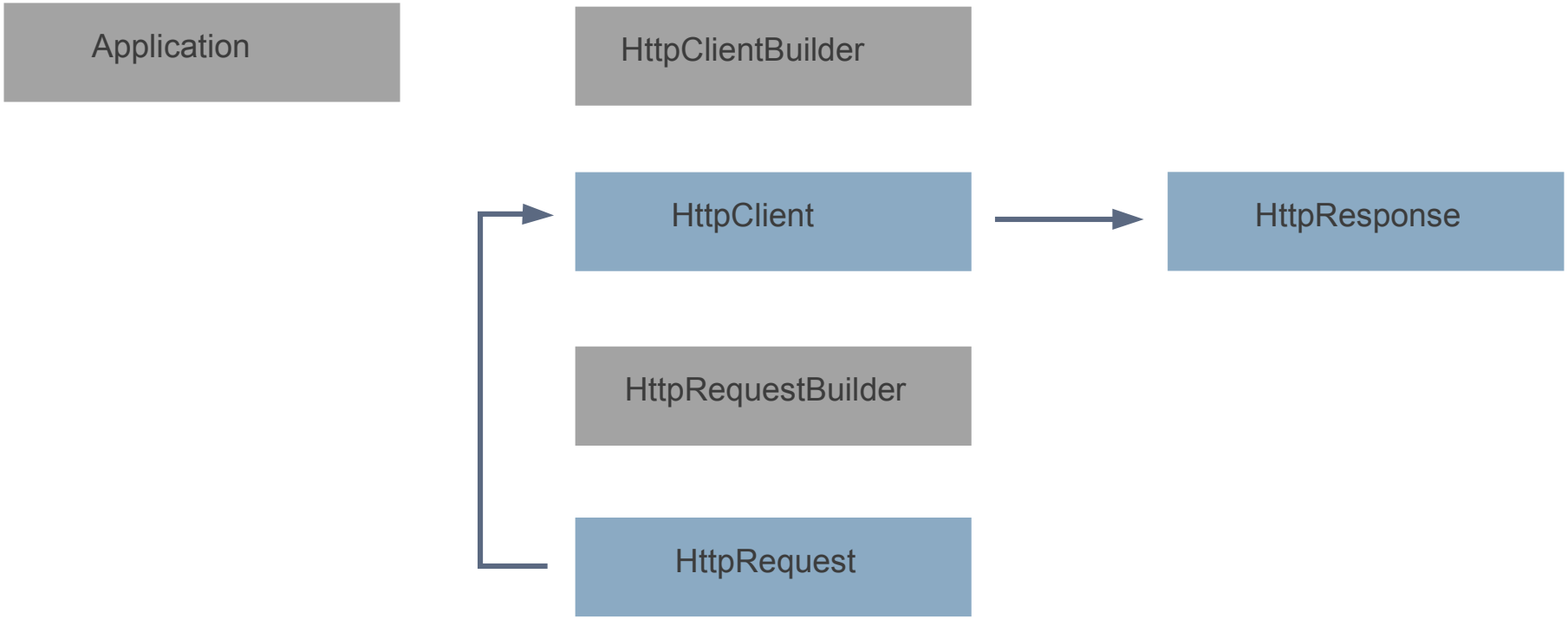
New Http client api: secondary control entities

Type	Description
HttpResponseHeadersHandler	Provides async notification of response headers
HttpResponseBodyHandler	Provides async notification of response body parts
HttpResponseErrorHandler	Provides async notification of request/response error
HttpMethod	Method enum (GET, POST. etc)
Filter	Can be chained together to pre/post process requests and responses

New Http client api: low level control entities

Type	Description
HttpConnectionCache	Interface for pooling/caching of TCP connections
SimpleHttpConnectionCache	Simple implementation of *HttpConnectionCache*
HttpUpgradeHandler	Mechanism for implementing higher level protocols that use Http as initial handshake (eg websockets)

New Http client api: object interaction



Creating an HttpClientBuilder

```
// Factory methods

class HttpClientBuilder {
    static HttpClientBuilder createClientBuilder();
    static HttpClientBuilder createClientBuilder(HttpClient);
}
```

Helper objects for an HttpClient instance

```
// all setXXX methods return "HttpClientBuilder", in Fluent style

public class HttpClientBuilder {
    ...
    setProxy(InetSocketAddress) {}
    InetSocketAddress getProxy() {}
    setSSLContext(SSLContext) {}
    SSLContext getSSLContext() {}
    setConnectionCache(HttpConnectionCache) {}
    HttpConnectionCache getConnectionCache() {}
    setCookieManager(CookieManager) {}
    CookieManager getCookieManager() {}
    setFilters(List<Class<? Extends Filter>) {}
    addFilter(Class<? Extends Filter>) {}
    List<Class<? Extends Filter>> getFilters() {}
    setExecutorService(ExecutorService) {}
    ExecutorService getExecutorService() {}
    ...
}
```

HttpClient instance configuration

```
// all setXXX methods return "HttpClientBuilder", in the Fluent style

public class HttpClientBuilder {
    ...
    setAllowPipelineMode(boolean){}
    setDefaultFollowsRedirects(boolean) {}
    setDefaultRequestBodyBufferLimit(long) {}
    setDefaultResponseBodyBufferLimit(long) {}
    setDefaultTimeout(long) {}
    setSocketSendBufferSize(int) {}
    setSocketReceiveBufferSize(int) {}
    setTcpNoDelay(boolean) {}
    usingProvider(String) {}
    ...
}
```

Creating the HttpClient instance

```
// creation of HttpClient instances

public class HttpClientBuilder {
     ...
     public HttpClient build(){}
     ...
}

// additionally, all helper objects and configuration
// properties are available on the HttpClient
// instance after it has been built
```

HttpClientBuilder: example

```
HttpClient client = HttpClientBuilder.createClientBuilder()
    .setProxy(new InetSocketAddress("proxy", 8080))
    .setExecutorService(Executors.newFixedThreadPool(20))
    .setConnectionCache(new SimpleConnectionCache(20, 10000))
    .setSSLContext(SSLContext.getDefault())
    // set resource parameters
    .setDefaultRequestBodyBufferLimit(64 * 1024)
    .setDefaultTimeout(10 * 1000)
    // add a logging filter
    .addFilter(MyLogFilter.class)
    // build the HttpClient
    .build();
```

HttpClient: request sending

```
public interface HttpClient {
      :
     InetSocketAddress getProxy();
     List<Class<? Extends Filter> getFilters();
     String getProviderName();
     long getDefaultTimeout();
     SSLContext getSSLContext();
     Future<HttpResponse> sendRequest(HttpRequest);
     OutputStream sendHeaders(HttpRequest, long);
}
```

HttpRequestBuilder

- Setting of request properties
- Overriding of default client configuration
- Setting/providing request body
- Setting callbacks for asynchronous mode
- Building (immutable) HttpRequest objects

HttpClient: request creation

```
public interface HttpClient {
    ...
    HttpRequestBuilder createRequestBuilder(URI);
    ...
}
```

HttpRequestBuilder: setting request properties

```
// Fluent style throughout

public interface HttpRequestBuilder {
    setMethod(HttpMethodBase);
    setRequestURI(URI);
    setHeaders(HttpHeaders)
    addHeader(String, String);
}
```

HttpRequestBuilder: override client configuration

```
// Fluent throughout

public interface HttpRequestBuilder {
    setTimeout(long);
    setRequestBodyBufferLimit(long);
    setUpgradeHandler(HttpUpgradeHandler)
}
```

HttpRequestBuilder: setting request body

```
public interface HttpRequestBuilder {
    setBody(byte[]);
    setBody(byte[], int offset, int length);
    setBody(ByteBuffer[] buffers);
    setBody(Iterator<ByteBuffer>);
    setBody(Iterable<ByteBuffer>);
}
```

HttpRequestBuilder: response callback interfaces

```
public interface HttpResponseHeadersHandler {
    void onHeaders(HttpResponse response);
}

public interface HttpResponseErrorHandler {
    void onError(HttpRequest request, Throwable error);
}

public interface HttpResponseBodyHandler {
    void onBodyPart(HttpResponse response, ByteBuffer buf, boolean last);
}
```

HttpRequestBuilder: setting response callbacks

```
public interface HttpRequestBuilder {
    ...
    onHeaders(HttpResponseHeadersHandler handler);
    onError(HttpResponseErrorHandler handler);
    onBodyPart(HttpResponseBodyHandler handler);
    ...
}
```

HttpRequestBuilder example

```
HttpClient client …;

URI uri = new URI(“http://www.foo.com/”);

HttpRequest req = client.createRequestBuilder(uri)
     .setRequestMethod(HttpMethod.GET)
     .setBody(“post data”.getBytes())
     .onError(myErrorHandler)
     .addHeader(“Some-header”, “some value”)
     .setTimeout(30 * 1000)
     .build();

// Now send the request through the client object
```

HttpRequest api

```
// public methods of HttpRequest

public interface HttpRequest {
    HttpClient getClient();
    boolean followsRedirects();
    String getContentType();
    HttpHeaders getHeaders();
    HttpMethodBase getMethod();
    HttpRequest getParent();
    URI getRequestURI();
    HttpRequestBuilder copyRequest();
}
```

Modes of operation

- Sending HttpRequest and receiving HttpResponse
- Supplying request body
- Receiving response body

Modes of operation

- Sending HttpRequest and receiving HttpResponse
  - Blocking
  - Asynchronously via callbacks
  - Asynchronously via Future<HttpResponse>
- Supplying request body

- Receiving response body

Modes of operation

- Sending HttpRequest and receiving HttpResponse
  - Blocking
  - Asynchronously via callbacks
  - Asynchronously via Future<HttpResponse>
- Supplying request body
  - Blocking OutputStream (push)
  - All at once
  - Iterator<ByteBuffer> or Iterable<ByteBuffer> (pull)
- Receiving response body

Modes of operation

- Sending HttpRequest and receiving HttpResponse
  - Blocking
  - Asynchronously via callbacks
  - Asynchronously via Future<HttpResponse>
- Supplying request body
  - Blocking OutputStream (push)
  - All at once
  - Iterator<ByteBuffer> or Iterable<ByteBuffer> (pull)
- Receiving response body
  - All at once
  - Blocking stream/Iterator
  - Asynchronously via callback

Setting request bodies:
“all at once”: byte array

```
byte[] requestBody = new byte [32 * 1024];
getData(requestBody);

HttpRequest request = client.createRequestBuilder()...
                                .setBody(requestBody)
                                .build();

client.sendRequest(request);
```

Setting request bodies:
“all at once”: byte array

```
byte[] requestBody = new byte [32 * 1024];
getData(requestBody);

HttpRequest request = client.createRequestBuilder()...
                                .setBody(requestBody)
                                .build();

client.sendRequest(request);
```

Setting request bodies:
“pull mode” Iterator

```
Iterator<ByteBuffer> iterator = new Iterator<ByteBuffer> () {
    public ByteBuffer next() { /* return next buffer */ }
    public boolean hasNext() { /* hasNext ? */ }
    public void remove() {/* not supported */ }
};
```

```
HttpRequest request = client.createRequestBuilder()...
                                .setBody(iterator)
                                .build();

client.sendRequest(request);
```

Setting request bodies:
“pull mode” Iterator

```
Iterator<ByteBuffer> iterator = new Iterator<ByteBuffer> () {
    public ByteBuffer next() { /* return next buffer */ }
    public boolean hasNext() { /* hasNext ? */ }
    public void remove() {/* not supported */ }
};


HttpRequest request = client.createRequestBuilder()...
                            .setBody(iterator)
                            .build();

client.sendRequest(request);
```

Setting request bodies:
“pull mode” Iterator

```
Iterator<ByteBuffer> iterator = new Iterator<ByteBuffer> () {
    public ByteBuffer next() { /* return next buffer */ }
    public boolean hasNext() { /* hasNext ? */ }
    public void remove() {/* not supported */ }
};


HttpRequest request = client.createRequestBuilder()...
                            .setBody(iterator)
                            .build();

client.sendRequest(request);
```

Setting request bodies:
Standard Iterator types

```
final Path path = FileSystems.getDefault().getPath("/some/file");
FileChannel chan = FileChannel.open(path, READ);
ReadableByteChannelIterator iter = new ReadableByteChannelIterator(chan);


HttpRequest request = client.createRequestBuilder()...
                            .setBody(iter)
                            .build();

client.sendRequest(request);
```

Setting request bodies:
Iterable example

```
final Path path = FileSystems.getDefault().getPath("/some/file");

Iterable<ByteBuffer> iterable = new Iterable<ByteBuffer>() {
    public Iterator<ByteBuffer> iterator() {
        try {
            FileChannel chan = FileChannel.open(path, READ);
            return new ReadableByteChannelIterator(chan);
        } catch (IOException e) { throw new RuntimeException(e); }
    }
};

HttpRequest request = client.createRequestBuilder()...
                            .setBody(iterable)
                            .build();
client.sendRequest(request);
```

Setting request bodies:
Iterable example

```
final Path path = FileSystems.getDefault().getPath("/some/file");

Iterable<ByteBuffer> iterable = new Iterable<ByteBuffer>() {
    public Iterator<ByteBuffer> iterator() {
        try {
            FileChannel chan = FileChannel.open(path, READ);
            return new ReadableByteChannelIterator(chan);
        } catch (IOException e) { throw new RuntimeException(e); }
    }
};

HttpRequest request = client.createRequestBuilder()...
                            .setBody(iterable)
                            .build();
client.sendRequest(request);
```

Setting request bodies:
Iterable example

```
final Path path = FileSystems.getDefault().getPath("/some/file");

Iterable<ByteBuffer> iterable = new Iterable<ByteBuffer>() {
    public Iterator<ByteBuffer> iterator() {
        try {
            FileChannel chan = FileChannel.open(path, READ);
            return new ReadableByteChannelIterator(chan);
        } catch (IOException e) { throw new RuntimeException(e); }
    }
};

HttpRequest request = client.createRequestBuilder()...
                            .setBody(iterable)
                            .build();
client.sendRequest(request);
```

Blocking example: no request body

```
HttpRequest request = client.createRequestBuilder()...
                                .build();

// simple blocking case no request body

client.sendRequest(request); // optional
HttpResponse response = client.getResponse(request); // blocks here
```

Blocking example: with request body
Push mode

```
HttpRequest request = client.createRequestBuilder()...
                                .build();

OutputStream body = client.sendHeaders(request, -1);

byte[] buf;
while ((buf = getData()) != null) {
    body.write(buf);
}
body.close();

HttpResponse response = client.getResponse(request);
```

Blocking example: with request body
Push mode

```
HttpRequest request = client.createRequestBuilder()...
                                .build();

OutputStream body = client.sendHeaders(request, -1);

byte[] buf;
while ((buf = getData()) != null) {
    body.write(buf);
}
body.close();

HttpResponse response = client.getResponse(request);
```

Blocking example: with request body
Push mode

```
HttpRequest request = client.createRequestBuilder()...
                                .build();

OutputStream body = client.sendHeaders(request, -1);

byte[] buf;
while ((buf = getData()) != null) {
    body.write(buf);
}
body.close();

HttpResponse response = client.getResponse(request);
```

Asynchronous interface: callbacks

```
HttpRequest request = client.createRequestBuilder()
```


Asynchronous interface: callbacks

```
HttpRequest request = client.createRequestBuilder()

   .onHeaders(HttpResponse response -> {
        // handle the response headers
    })
```

Asynchronous interface: callbacks

```
HttpRequest request = client.createRequestBuilder()

   .onHeaders(HttpResponse response -> {
        // handle the response headers
    })
   .onError((HttpRequest request, Throwable error) -> {
        // handle the error relating to request
    })
```

Asynchronous interface: callbacks

```
HttpRequest request = client.createRequestBuilder()

   .onHeaders(HttpResponse response -> {
        // handle the response headers
    })
   .onError((HttpRequest request, Throwable error) -> {
        // handle the error relating to request
    })
   .onBodyPart((HttpResponse response, ByteBuffer buffer, boolean last) -> {
        // handle the data in buffer
    })
```

Asynchronous interface: callbacks

```
HttpRequest request = client.createRequestBuilder()

   .onHeaders(HttpResponse response -> {
        // handle the response headers
    })
   .onError((HttpRequest request, Throwable error) -> {
        // handle the error relating to request
    })
   .onBodyPart((HttpResponse response, ByteBuffer buffer, boolean last) -> {
        // handle the data in buffer
    })
   .build();

client.sendRequest(request);
```

Asynchronous callbacks:
pre lambda equivalence

```
HttpRequest request = client.createRequestBuilder()
  .onHeaders(HttpResponse response -> {
       // handle the response headers
   })
```

might formerly have been done as

Asynchronous callbacks:
pre lambda equivalence

```
HttpRequest request = client.createRequestBuilder()
  .onHeaders(HttpResponse response -> {
       // handle the response headers
   })
```

might formerly have been done as

```
HttpRequest request = client.createRequestBuilder()
  .onHeaders(new HttpResponseHeadersHandler() {
     public void onHeaders(HttpResponse response) {
        // handle the response headers
     }
  })
```

Asynchronous callbacks:
conventional handler class

```
class Handler implements HttpResponseHeadersHandler, HttpResponseErrorHandler,
                         HttpResponseBodyHandler {
    public void onHeaders(HttpResponse resp) {  }
    public void onError(HttpRequest request, Throwable t) { }
    public void onBodyPart(HttpResponse resp, ByteBuffer buffer, boolean last) { }
}

Handler handler = new Handler();

HttpRequest request = client.createRequestBuilder()
   .onHeaders(handler)
   .onError(handler)
   .onBodyPart(handler)
   .build();
```

Asynchronous interface: Futures

```
HttpRequest request = client.createRequestBuilder()
   .build();
Future<HttpResponse> f = client.sendRequest(request);
```

Asynchronous interface: Futures

```
HttpRequest request = client.createRequestBuilder()
   .build();
Future<HttpResponse> f = client.sendRequest(request);

HttpResponse response = f.get();
```

HttpResponse api
main accessor methods

```
public interface HttpResponse {
    long getContentLength();
    String getContentType();
    HttpResponse getParent();
    HttpRequest getRequest();
    int getResponseCode();
    SSLSession getSSLSession();
    HttpHeaders getHeaders();
    HttpHeaders getTrailers();
}
```

HttpResponse api:
reading response body

```
public interface HttpResponse {
    long getContentLength();
    String getContentType();
    HttpResponse getParent();
    HttpRequest getRequest();
    int getResponseCode();
    SSLSession getSSLSession();
    HttpHeaders getHeaders();
    HttpHeaders getTrailers();
    ByteBuffer[] getBodyAsByteBuffers();
    int getBodyAsByteBuffers(ByteBuffer[] buffers, int maxLength);
}
```

Blocking reads. Returns body “all at once”

HttpResponse api:
reading response body

```
public interface HttpResponse {
    long getContentLength();
    String getContentType();
    HttpResponse getParent();
    HttpRequest getRequest();
    int getResponseCode();
    SSLSession getSSLSession();
    HttpHeaders getHeaders();
    HttpHeaders getTrailers();
    ByteBuffer[] getBodyAsByteBuffers();
    int getBodyAsByteBuffers(ByteBuffer[] buffers, int maxLength);
}
```

Blocking reads. Returns body “all at once”

Variants also defined for byte[]

HttpResponse api:
reading response body

```
public interface HttpResponse {
    long getContentLength();
    String getContentType();
    HttpResponse getParent();
    HttpRequest getRequest();
    int getResponseCode();
    SSLSession getSSLSession();
    HttpHeaders getHeaders();
    HttpHeaders getTrailers();
    ByteBuffer[] getBodyAsByteBuffers();
    int getBodyAsByteBuffers(ByteBuffer[] buffers, int maxLength);
    Iterator<ByteBuffer> getBodyAsByteBufferSource();
    InputStream getBodyAsInputStream();
}
```

Blocking reads. Returns body as stream.

Mixing asynchronous and blocking APIs

```
HttpRequest request = client.createRequestBuilder()

    .onHeaders(HttpResponse response -> {
         if (response.getResponseCode() == 200) {
             InputStream body = response.getBodyAsInputStream();
             ReaderThread reader = new ReaderThread(body).start();
         }
     })
    .onError((HttpRequest request, Throwable error) -> {
         // handle the error relating to request
     })
    .build();

  client.sendRequest(request);
```

Mixing asynchronous and blocking APIs

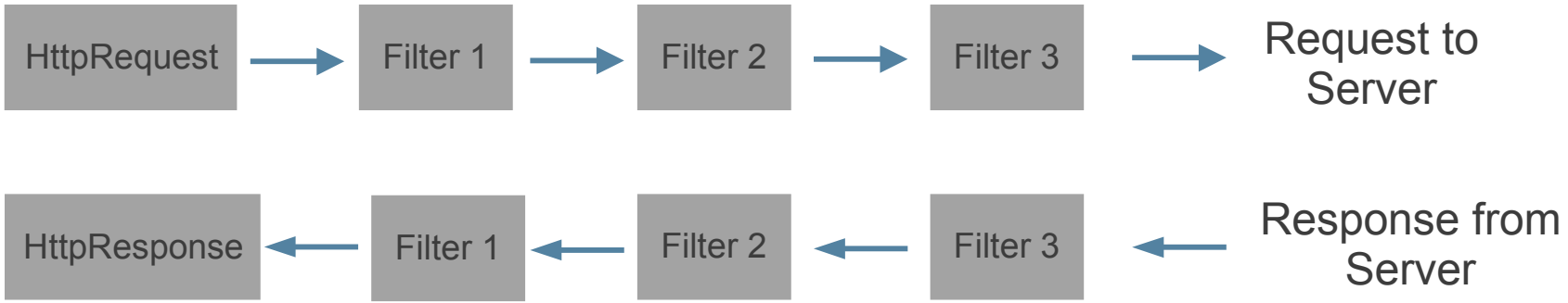
```
HttpRequest request = client.createRequestBuilder()

    .onHeaders(HttpResponse response -> {
         if (response.getResponseCode() == 200) {
             InputStream body = response.getBodyAsInputStream();
             ReaderThread reader = new ReaderThread(body).start();
         }
     })
    .onError((HttpRequest request, Throwable error) -> {
         // handle the error relating to request
     })
    .build();

  client.sendRequest(request);
```

Filter API

- Filters are organised in a chain/list per HttpClient
- Can modify/observe headers and body of requests/responses – also cause responses to be interrupted and/or re-issued.
- Separate Filter instance created for each request/response pair



Filter API

```
public interface Filter {
    void beforeRequest(HttpRequest request, HttpHeaders modifiableHeaders)
        default {}
    ByteBuffer[] onRequestBodyPart(ByteBuffer[] buffers, boolean complete)
        default {return buffers;}
    Result afterResponse(HttpResponse response)
        default {return Result.OK;}
    ByteBuffer[] onResponseBodyPart(ByteBuffer[] buffers, boolean complete)
        default {return buffers;}
}
```

Filter API

```
public interface Filter {
    void beforeRequest(HttpRequest request, HttpHeaders modifiableHeaders)
        default {}
    ByteBuffer[] onRequestBodyPart(ByteBuffer[] buffers, boolean complete)
        default {return buffers;}
    Result afterResponse(HttpResponse response)
        default {return Result.OK;}
    ByteBuffer[] onResponseBodyPart(ByteBuffer[] buffers, boolean complete)
        default {return buffers;}
}
```

Filter API

```
public interface Filter {
    void beforeRequest(HttpRequest request, HttpHeaders modifiableHeaders)
        default {}
    ByteBuffer[] onRequestBodyPart(ByteBuffer[] buffers, boolean complete)
        default {return buffers;}
    Result afterResponse(HttpResponse response)
        default {return Result.OK;}
    ByteBuffer[] onResponseBodyPart(ByteBuffer[] buffers, boolean complete)
        default {return buffers;}
}
```

Filter API

```
public interface Filter {
    void beforeRequest(HttpRequest request, HttpHeaders modifiableHeaders)
        default {}
    ByteBuffer[] onRequestBodyPart(ByteBuffer[] buffers, boolean complete)
        default {return buffers;}
    Result afterResponse(HttpResponse response)
        default {return Result.OK;}
    ByteBuffer[] onResponseBodyPart(ByteBuffer[] buffers, boolean complete)
        default {return buffers;}
}
```

Filter example:
request/response logger

```
public class LogFilter implements Filter {
    Logger logger = Logger.getLogger("com.foo.LogFilter");
    int byteCount = 0;

    void beforeRequest(HttpRequest request, HttpHeaders modifiableHeaders) {
        logger.info(request.getMethod());
        logger.info(request.getRequestURI());
    }
    Result afterResponse(HttpResponse response) {
        logger.info(response.getResponseCode());
        return Result.OK;
    }
    ByteBuffer[] onResponseBodyPart(ByteBuffer[] buffers, boolean complete) {
        for (int i=0; i<buffers.length)
            byteCount += buffers[i].remaining();
        if (complete)
            logger.info("Request body size: " + byteCount);
        return buffers; // no modification
    }
}
```

Filter example:
request/response logger

```
public class LogFilter implements Filter {
    Logger logger = Logger.getLogger("com.foo.LogFilter");
    int byteCount = 0;

    void beforeRequest(HttpRequest request, HttpHeaders modifiableHeaders) {
        logger.info(request.getMethod());
        logger.info(request.getRequestURI());
    }
    Result afterResponse(HttpResponse response) {
        logger.info(response.getResponseCode());
        return Result.OK;
    }
    ByteBuffer[] onResponseBodyPart(ByteBuffer[] buffers, boolean complete) {
        for (int i=0; i<buffers.length)
            byteCount += buffers[i].remaining();
        if (complete)
            logger.info("Request body size: " + byteCount);
        return buffers; // no modification
    }
}
```

Filter example:
request/response logger

```
public class LogFilter implements Filter {
    Logger logger = Logger.getLogger("com.foo.LogFilter");
    int byteCount = 0;

    void beforeRequest(HttpRequest request, HttpHeaders modifiableHeaders) {
        logger.info(request.getMethod());
        logger.info(request.getRequestURI());
    }
    Result afterResponse(HttpResponse response) {
        logger.info(response.getResponseCode());
        return Result.OK;
    }
    ByteBuffer[] onResponseBodyPart(ByteBuffer[] buffers, boolean complete) {
        for (int i=0; i<buffers.length)
            byteCount += buffers[i].remaining();
        if (complete)
            logger.info("Request body size: " + byteCount);
        return buffers; // no modification
    }
}
```

Filter example:
modification of body

```
public class EncryptionFilter implements Filter {
    private EncryptionState ...

    ByteBuffer[] onRequestBodyPart(ByteBuffer[] buffers, boolean complete) {
        ByteBuffer outBuffers = new ByteBuffer[buffers.length];
        for (int i=0; i<buffers.length; i++)
            outBuffers[i] = encrypt(buffers[i]);
        return outBuffers;
    }
    ByteBuffer[] onResponseBodyPart(ByteBuffer[] buffers, boolean complete) {
        ByteBuffer outBuffers = new ByteBuffer[buffers.length];
        for (int i=0; i<buffers.length; i++)
            outBuffers[i] = decrypt(buffers[i]);
        return outBuffers;
    }
}
```

Filter example:
modification of body

```
public class EncryptionFilter implements Filter {
    private EncryptionState ...

    ByteBuffer[] onRequestBodyPart(ByteBuffer[] buffers, boolean complete) {
        ByteBuffer outBuffers = new ByteBuffer[buffers.length];
        for (int i=0; i<buffers.length; i++)
            outBuffers[i] = encrypt(buffers[i]);
        return outBuffers;
    }
    ByteBuffer[] onResponseBodyPart(ByteBuffer[] buffers, boolean complete) {
        ByteBuffer outBuffers = new ByteBuffer[buffers.length];
        for (int i=0; i<buffers.length; i++)
            outBuffers[i] = decrypt(buffers[i]);
        return outBuffers;
    }
}
```

Filter example:
modification of body

```
public class PrependingFilter implements Filter {
    List<ByteBuffer> store = new LinkedList<>();

    ByteBuffer[] onRequestBodyPart(ByteBuffer[] buffers, boolean complete) {
        store.addAll(Arrays.asList(buffers));
        if (!complete)
            return null; // storing up all the buffers until end
        ByteBuffer specialBuffer = .. // new buffer containing header
        store.add(specialBuffer, 0); // prepend to list
        return store.toArray();
    }
}
```

Filter example:
modification of body

```
public class PrependingFilter implements Filter {
    List<ByteBuffer> store = new LinkedList<>();

    ByteBuffer[] onRequestBodyPart(ByteBuffer[] buffers, boolean complete) {
        store.addAll(Arrays.asList(buffers));
        if (!complete)
            return null; // storing up all the buffers until end
        ByteBuffer specialBuffer = .. // new buffer containing header
        store.add(specialBuffer, 0); // prepend to list
        return store.toArray();
    }
}
```

Filter example:
modification of body

```
public class PrependingFilter implements Filter {
    List<ByteBuffer> store = new LinkedList<>();

    ByteBuffer[] onRequestBodyPart(ByteBuffer[] buffers, boolean complete) {
        store.addAll(Arrays.asList(buffers));
        if (!complete)
            return null; // storing up all the buffers until end
        ByteBuffer specialBuffer = .. // new buffer containing header
        store.add(specialBuffer, 0); // prepend to list
        return store.toArray();
    }
}
```

Filter example: new request
implementing redirection

```
public class RedirectFilter implements Filter {

    Result afterResponse(HttpResponse response) {
        if (response.getResponseCode() == 301) {
            HttpRequestBuilder r = response.getRequest()
                                           .copyRequest();
            r.setRequestURI(response.getHeaders().getFirstValue("Location"));
            return Result.newRequest(r.build());
        }
        return Result.OK;
    }
}
```

HttpConnectionCache:
TCP connection pools/keep-alive

```
interface HttpConnectionCache {
    interface CachedConnection {
        void close();
        InetSocketAddress getAddress();
        long getAge();
        long getUseCount();
        boolean isClosed();
        boolean isProxy();
    }
    void returnToCache(CachedConnection connection);
    CachedConnection getConnection(InetSocketAddress destination);
}

public class SimpleHttpConnectionCache implements HttpConnectionCache {
    public SimpleHttpConnectionCache (long maxConnections, long keepAliveTime) {}
}
```

HttpConnectionCache
TCP connection pools/keep-alive

```
interface HttpConnectionCache {
    interface CachedConnection {
        void close();
        InetSocketAddress getAddress();
        long getAge();
        long getUseCount();
        boolean isClosed();
        boolean isProxy();
    }
    void returnToCache(CachedConnection connection);
    CachedConnection getConnection(InetSocketAddress destination);
}

public class SimpleHttpConnectionCache implements HttpConnectionCache {
    public SimpleHttpConnectionCache (long maxConnections, long keepAliveTime) {}
}
```

HttpConnectionCache:
TCP connection pools/keep-alive

```
interface HttpConnectionCache {
    interface CachedConnection {
        void close();
        InetSocketAddress getAddress();
        long getAge();
        long getUseCount();
        boolean isClosed();
        boolean isProxy();
    }
    void returnToCache(CachedConnection connection);
    CachedConnection getConnection(InetSocketAddress destination);
}

public class SimpleHttpConnectionCache implements HttpConnectionCache {
    public SimpleHttpConnectionCache (long maxConnections, long keepAliveTime) {}
}
```

HttpUpgradeHandler:
upgrade mechanism for protocols like websocket

```
interface HttpUpgradeHandler {
    void modifyRequest(HttpRequest request, HttpHeaders modifiableHeaders);

    boolean examineResponse(HttpResponse response);

    void setConnection(UpgradeConnection connection);
}

interface UpgradeConnection {
    void close();
    AsynchronousByteChannel getChannel();
    InputStream getInputStream();
    OutputStream getOutputStream();
    InetSocketAddress getPeer();
    boolean isConnected();
}
```

HttpUpgradeHandler:
upgrade mechanism for protocols like websocket

```
interface HttpUpgradeHandler {
    void modifyRequest(HttpRequest request, HttpHeaders modifiableHeaders);

    boolean examineResponse(HttpResponse response);

    void setConnection(UpgradeConnection connection);
}

interface UpgradeConnection {
    void close();
    AsynchronousByteChannel getChannel();
    InputStream getInputStream();
    OutputStream getOutputStream();
    InetSocketAddress getPeer();
    boolean isConnected();
}
```

HttpUpgradeHandler:
upgrade mechanism for protocols like websocket

```
interface HttpUpgradeHandler {
    void modifyRequest(HttpRequest request, HttpHeaders modifiableHeaders);

    boolean examineResponse(HttpResponse response);

    void setConnection(UpgradeConnection connection);
}

interface UpgradeConnection {
    void close();
    AsynchronousByteChannel getChannel();
    InputStream getInputStream();
    OutputStream getOutputStream();
    InetSocketAddress getPeer();
    boolean isConnected();
}
```

HttpUpgradeHandler:
upgrade mechanism for protocols like websocket

```
interface HttpUpgradeHandler {
    void modifyRequest(HttpRequest request, HttpHeaders modifiableHeaders);

    boolean examineResponse(HttpResponse response);

    void setConnection(UpgradeConnection connection);
}

interface UpgradeConnection {
    void close();
    AsynchronousByteChannel getChannel();
    InputStream getInputStream();
    OutputStream getOutputStream();
    InetSocketAddress getPeer();
    boolean isConnected();
}
```

Security permissions

- Permission checks will be similar to existing client for basic request/response interactions (with new HttpPermission class)
- SocketPermission required for proxy access
- New NetPermissions for setting/getting cache, filters, upgrade handlers etc.

Security permissions

```
// example HttpPermission policy grant

grant {
    permission java.net.HttpPermission "http://www.foo.com/subpath/", "GET,POST";
};
```

Finally {}

- Source and API can be downloaded from
  - http://java.net/projects/http-client/
  - Discussion on net-dev@openjdk.java.net
- Open issues
  - Authentication class not defined yet
  - Complete security permission work

Q & A

MAKE THE FUTURE JAVA

ORACLE


JavaOne™