Module jdk.httpserver

Package com.sun.net.httpserver

`package com.sun.net.httpserver`

Provides a simple high-level Http server API, which can be used to build embedded HTTP servers. Both "http" and "https" are supported. The API provides a partial implementation of RFC 2616 (HTTP 1.1) and RFC 2818 (HTTP over TLS). Any HTTP functionality not provided by this API can be implemented by application code using the API.

The main components are:

- the `HttpExchange` class that describes a request and response pair,
- the `HttpHandler` interface to handle incoming requests, plus the `HttpHandlers` class that provides useful handler implementations,
- the `HttpContext` class that maps a URI path to a `HttpHandler`,
- the `HttpServer` class to listen for connections and dispatch requests to handlers,
- the `Filter` class that allows pre- and post- processing of requests.

The `SimpleFileServer` class offers a simple HTTP-only file server (intended for testing, development and debugging purposes only). A default implementation is provided via the `jwebserver` tool.

Programmers must implement the `HttpHandler` interface. This interface provides a callback which is invoked to handle incoming requests from clients. A HTTP request and its response is known as an exchange. HTTP exchanges are represented by the `HttpExchange` class. The `HttpServer` class is used to listen for incoming TCP connections and it dispatches requests on these connections to handlers which have been registered with the server.

A minimal Http server example is shown below:

```
    class MyHandler implements HttpHandler {
        public void handle(HttpExchange t) throws IOException {
            InputStream is = t.getRequestBody();
            read(is); // .. read the request body
            String response = "This is the response";
            t.sendResponseHeaders(200, response.length());
            OutputStream os = t.getResponseBody();
            os.write(response.getBytes());
            os.close();
        }
    }
    ...

    HttpServer server = HttpServer.create(new InetSocketAddress(8000), 0);
    server.createContext("/applications/myapp", new MyHandler());
    server.setExecutor(null); // creates a default executor
    server.start();
```

The example above creates a simple HttpServer which uses the calling application thread to invoke the handle() method for incoming http requests directed to port 8000, and to the path /applications/myapp/.

The `HttpExchange` class encapsulates everything an application needs to process incoming requests and to generate appropriate responses.

Registering a handler with a HttpServer creates a `HttpContext` object and `Filter` objects can be added to the returned context. Filters are used to perform automatic pre- and post-processing of exchanges before they are passed to the exchange handler.

For sensitive information, a `HttpsServer` can be used to process "https" requests secured by the SSL or TLS protocols. A HttpsServer must be provided with a `HttpsConfigurator` object, which contains an initialized `SSLContext`. HttpsConfigurator can be used to configure the cipher suites and other SSL operating parameters. A simple example SSLContext could be created as follows:

```
    char[] passphrase = "passphrase".toCharArray();
    KeyStore ks = KeyStore.getInstance("JKS");
    ks.load(new FileInputStream("testkeys"), passphrase);

    KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
    kmf.init(ks, passphrase);

    TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
    tmf.init(ks);

    SSLContext ssl = SSLContext.getInstance("TLS");
    ssl.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
```

In the example above, a keystore file called "testkeys", created with the keytool utility is used as a certificate store for client and server certificates. The following code shows how the SSLContext is then used in a HttpsConfigurator and how the SSLContext and HttpsConfigurator are linked to the HttpsServer.

```
     server.setHttpsConfigurator (new HttpsConfigurator(sslContext) {
         public void configure (HttpsParameters params) {

         // get the remote address if needed
         InetSocketAddress remote = params.getClientAddress();

         SSLContext c = getSSLContext();

         // get the default parameters
         SSLParameters sslparams = c.getDefaultSSLParameters();
         if (remote.equals (...) ) {
             // modify the default set for client x
         }

         params.setSSLParameters(sslparams);
         // statement above could throw IAE if any params invalid.
         // eg. if app has a UI and parameters supplied by a user.

         }
     });
```

Since:
1.6

Related Packages

Package	Description
com.sun.net.httpserver.spi	Provides a pluggable service provider interface, which allows the HTTP server implementation to be replaced with other implementations.

All Classes and Interfaces **Interfaces** **Classes** **Enum Classes**

Class	Description
Authenticator	Authenticator represents an implementation of an HTTP authentication mechanism.
Authenticator.Failure	Indicates an authentication failure.
Authenticator.Result	Base class for return type from `Authenticator.authenticate(HttpExchange)` method.
Authenticator.Retry	Indicates an authentication must be retried.
Authenticator.Success	Indicates an authentication has succeeded and the authenticated user principal can be acquired by calling `Authenticator.Success.getPrincipal()`.
BasicAuthenticator	BasicAuthenticator provides an implementation of HTTP Basic authentication.
Filter	A filter used to pre- and post-process incoming requests.
Filter.Chain	A chain of filters associated with a `HttpServer`.
Headers	HTTP request and response headers are represented by this class which implements the interface `Map`<`String`, `List` <`String`>>.
HttpContext	`HttpContext` represents a mapping between the root `URI` path of an application to a `HttpHandler` which is invoked to handle requests destined for that path on the associated `HttpServer` or `HttpsServer`.
HttpExchange	This class encapsulates a HTTP request received and a response to be generated in one exchange.
HttpHandler	A handler which is invoked to process HTTP exchanges.
HttpHandlers	Implementations of `HttpHandler` that implement various useful handlers, such as a static response handler, or a conditional handler that complements one handler with another.
HttpPrincipal	Represents a user authenticated by HTTP Basic or Digest authentication.
HttpsConfigurator	This class is used to configure the https parameters for each incoming https connection on a `HttpsServer`.
HttpServer	This class implements a simple HTTP server.
HttpsExchange	This class encapsulates a HTTPS request received and a response to be generated in one exchange and defines the extensions to `HttpExchange` that are specific to the HTTPS protocol.
HttpsParameters	Represents the set of parameters for each https connection negotiated with clients.
HttpsServer	This class is an extension of `HttpServer` which provides support for HTTPS.
Request	A view of the immutable request state of an HTTP exchange.
SimpleFileServer	A simple HTTP file server and its components (intended for testing, development and debugging purposes only).
SimpleFileServer.OutputLevel	Describes the log message output level produced by the server when processing exchanges.