Chapter 2

Accessing Web Resources using URL Connections

Advanced Topics in Java

Khalid Azim Mughal
khalid@ii.uib.no
http://www.ii.uib.no/~khalid/atij/

Version date: 2003-10-01

Overview

- HTTP-support for Client-Side Applications through the following classes:
  - `URL`
  - `URLConnection`
  - `HttpURLConnection`
- Usefulness of `URLEncoder`/ `URLDecoder` classes.

URL Connections

- *Client-side* support for accessing and retrieving web resources.
- Encapsulate much of the low-level (TCP/IP stack) complexity involved in accessing web resources.
- Support for URL connections is provided in the `java.net` package by the following important classes:
  - `URL`
  - `URLConnection`
  - `HttpURLConnection`

Universal Resource Identifier: URI

- A URI is a superset of URL and URN. It is an identifier that identifies a resource. The resource may or may not exist. Neither does it imply how we can retrieve the resource.
  `ATIJ/lecture-notes-kam/atij-application-protocols`

Universal Resource Locator: URL

- A URL specifies a unique address/location for a resource on the Web.
- Common form:

  <protocol>`://`*<hostname>[*`:`*<TCP port number>]*`/`*<pathname>[*`?`*<query>][*`#`*<reference>]*

  `http://www.ii.uib.no:80/~khalid/pgjc2e/`

  `mailto:khalid@ii.uib.no?Subject=Urgent%20Message`

  `http://www.w3.org/TR/REC-html32#intro` <--- Tag to indicate particular part of a document.

Universal Resource Name: URN

- A URN is a unique identifier that identifies a resource, irrespective of its location and mode of retrieval.
  `ISBN: 0-201-72828-1`

The URL class

- Represents a URL (Uniform Resource Locator), i.e. a unique address/location to access a web resource.

  <protocol>`://`*<hostname>*[`:`*<port>*]`/`*<pathname>*[`?`*<query>*][`#`*<reference>*]

- A web resource can be:
  - a file
  - a directory
  - a query to a database or to a search engine

Note that an `URL` *instance need not represent a valid resource, but it must contain the following components: protocol, hostname and pathname.*

URL Constructors

- All constructors throw a `java.net.MalformedURLException` if the *protocol* is missing or unknown.
- If the port is not specified, the default port for the protocol is assumed.
- When constructing a `URL`, an appropriate stream protocol handler (`URLStreamHandler`) is automatically loaded.

Constructor	Example
`URL(String urlStr)` `throws MalformedURLException`	`URL url3 = new URL( "http://www.bond.edu.au" + "/it/subjects/subs-pg.htm#inft718" );`
`URL(String protocol, String hostname, String filename)` `throws MalformedURLException`	`URL url4 = new URL( "ftp", "www.javaworld.com", "javaforums/ubbthreads.txt" );`

Constructor	Example
`URL(String protocol, String hostname, int portNumber, String filename) throws MalformedURLException`	`URL url9 = new URL( "http", "java.sun.com", 80, "/j2se/1.4.2/docs/api/index.html" );`

`URL(URL context, String spec) throws MalformedURLException`	`URL url5 = new URL( "http://www.ii.uib.no" );`
	`URL url6 = new URL( url5, "undervisning" );`
	`//Final URL: "http://www.ii.uib.no/undervisning"`
	
	`URL url10 = new URL( null, // Same as first constructor.`
	`"http://java.sun.com" +`
	`"/j2se/1.4.2/docs/api/index.html" );`

Misc. URL Methods

- Get the different components of the URL instance (*See* `URLParser.java`).

`String getProtocol()`
`String getHost()`
`String getPort()`
`String getFile()`
`String getPath()`
`String getQuery()`
`String getRef()`
	If no port is present, -1 is returned by the `getPort()` method.
	If no file name or path is present, empty string is returned.
	The string returned by the `getFile()` method has the query, if any, but the `getPath()` method excludes the query.
	If no query or reference is present, `null` is returned.

- Compare URL instances.

`boolean equals(Object obj)`	The `equal()` method can block as it requires name resolution.
`boolean sameFile(URL url)`	The `sameFile()` method excludes the reference component.

- Convert a URL to a string.

`String toString()`
`String toExternal()`
	Both methods give identical results.

Retrieving a Resource via an `URL`

- Open an input stream to retrieve the resource identified by the `URL` instance.

`InputStream openStream()`	Establishes a connection with the server and returns an input stream to retrieve the source. *See* `FetchResourceViaURL.java`.

- Retrieve the contents of resource identified by the `URL` instance.

`Object getContent()`
`       throws IOException`
	The method is equivalent to `openConnection().getContent()`.
	See `FetchResourceViaMethodgetContent.java`.
	See also class `URLConnection`.

- Return an `URLConnection` instance which can be used to retrieve the contents of resource identified by the `URL` instance.

`URLConnection openConnection()`	The method does *not* establish any connection to retrieve the resource. *See* class `URLConnection`.

- The URL class *only* provides an input stream to retrieve the contents of the resource.
  - Other information about the request sent or the response received is not accessible.

The `URLConnection` Class

- A `URLConnection` represents a communications link between the application and a URL.
- A `URLConnection` allows access to all pertinent information about the requests it sends and the responses it receives.
  - Allows interaction with the resource and makes querying of requests and responses possible.
- The class is abstract, and a concrete `URLConnection` is obtained via an `URL` instance.

```
URL url = new URL( urlStr );
URLConnection connection = url.openConnection();
// No connection established so far.
```

Misc. `URLConnection` Methods

- Customizing setup parameters for the connection.

`void setIfModifiedSince(long time)`	Only fetches data that has been modified since the specified time (in seconds, from midnight, GMT, 1970-01-01).
`void setUseCaches(boolean permit)`	If `permit` is `true` (default), the connection can cache documents.
`void setDoInput(boolean status)`	If `status` is `true` (default), then the connection can be used to receive a response.
`void setDoOutput(boolean status)`	If `status` is `true`, then the connection can be used to send a request. The default status is `false`.
`void setAllowUserInteraction( boolean allow)`	If `allow` is `true`, then the user can be password authenticated.

- Customizing general request header fields

`void setRequestProperty( String key, String value)`	The key/value pair must be permissible according to the protocol.

The `set`*-methods above have corresponding* `get`*-methods.*

- Establishing a connection to the remote resource.

`void connect() throws IOException`	Establishes connection and retrieves response header fields. The call is ignored if the connection is already established.

- Querying response header information.

`String getHeaderFieldKey(int n)`	Returns header field key at index `n` (n>=0), or `null` for invalid n.
`String getHeaderField(int n)`	Returns header field value at index `n` (n>=0), or `null` for invalid n.
`String getHeaderField( String field)`	Returns the value of the `field`.
`Map getHeaderFields()`	Returns an unmodifiable `Map` of header field name - value entries.

`String getContentLength()`
`String getContentType()`
`String getContentEncoding()`
`String getDate()`
`String getExpiration()`
`String getLastModified()`
	Return the value of a specific response header field.

- Obtaining the input and output streams of the connection.

```
InputStream getInputStream()
          throws IOException
OutputStream getOutputStream()
          throws IOException
```

- Obtaining the contents of the requested resource.

`Object getContent() throws IOException`	A suitable content handler is chosen depending on the content type.

Retrieving a Resource via an `URLConnection`

- *See* `FetchResourceViaURLConnection.java`.

1. Create an `URL` instance with the address of the resource.
```
      url = new URL( urlStr );
```
2. Obtain an `URLConnection` from the `URL` instance.
```
      URLConnection connection = url.openConnection();
```
3. Customize any request fields.
```
      connection.setRequestProperty("User-Agent",
                         "Mozilla/4.0 (compatible; JavaApp)");
      connection.setRequestProperty("Referer",
                         "http://www.ii.uib.no/");
      connection.setUseCaches(false);
```
4. Establish a connection to the remote resource, which also sends the request.
   - A response will be issued by the server.
```
      connection.connect();
```

5. Query the response header information.

```
    System.out.println("Content-Type:     "
                       + connection.getContentType());
    System.out.println("Content-Length:   "
                       + connection.getContentLength());
    System.out.println("Content-Encoding: "
                       + connection.getContentEncoding());
    System.out.println("Date:             "
                       + connection.getDate());
    System.out.println("Expiration-Date:  "
                       + connection.getExpiration());
    System.out.println("Last-modified:    "
                       + connection.getLastModified());
```

– Alternatively, header fields can also be looked up using a map.
  Following code prints all the header fields:

```
  Map allFields = connection.getHeaderFields();
  System.out.println("No. of field headers: " + allFields.size());
  System.out.println(allFields);
```

6. Obtain an input stream to access the resource content.

```
      InputStream input = connection.getInputStream();
      reader = new BufferedReader(
                  new InputStreamReader(input));
      System.out.println("Reading the contents ...");
      for(;;) {
        String line = reader.readLine();
        if (line == null) break;
        System.out.println(line);
      }
```

– Alternatively, we use the `getContent()` method.
  See `FetchResourceViaMethodgetContent.java`.

The **HttpURLConnection** Class

- The HttpURLConnection class is a subclass of the URLConnection class.
- It provides *HTTP-specific* functionality for dealing with HTTP requests and responses.
- The class defines constants for the HTTP response codes that can occur is a response status line.

```
HttpURLConnection.HTTP_OK              // HTTP Status-Code 200: OK
HttpURLConnection.HTTP_NOT_FOUND       // HTTP Status-Code 404: Not Found
HttpURLConnection.HTTP_NOT_IMPLEMENTED // HTTP Status-Code 501: Not Implemented
```

- As the class does not have a public constructor, a `HttpURLConnection` is often obtained as follows:

```
URL url = new URL( urlStr );                          // Create a URL.
URLConnection connection = url.openConnection();      // Get an URLConnection.
if (connection instanceof HttpURLConnection) {       // Is it a HttpURLConnection?
   HttpURLConnection httpConnection = (HttpURLConnection) connection;
   // Can access http-functionality of the connection.
}
```

  – If the *protocol* of the `URL` is HTTP then the `URLConnection` returned is a `HttpURLConnection`.

Misc. **HttpURLConnection** Methods

- In addition to inheriting methods from the `URLConnection` class, the `HttpURLConnection` overrides some methods from the superclass and also defines some HTTP-specific methods of its own.

`void setRequestMethod( String method) throws ProtocolException`	Sets the request method to use for the connection. The request method is be subject to the protocol restrictions. Default method is GET.
`String getRequestMethod()`	Returns the request method that will be used.
`void connect()`	Inherited from the superclass `URLConnection`. It establishes a connection and sends the request, with the server subsequently issuing the response.
`int getResponseCode() throws IOException`	Returns the response code in the status line.
`String getResponseMessage() throws IOException`	Returns the status message from the status line.
`void disconnect()`	Future requests are unlikely on this connection.

- The procedure for retrieving a resource using a `HttpURLConnection` is very similar to that of using a `URLConnection`, with the added functionality of accessing HTTP features.

  See `FetchResourceViaHttpURLConnection.java`.