Chapitre 8

Dans ce chapitre nous étudions les fonctionnalités Java permettant de travailler directement au niveau URL (*Universal Resource Locator*) et nous nous intéressons plus particulièrement au protocole HTTP (*Hyper Text Transfert Protocol*).

Un URL est une chaîne de caractères représentant une ressource accessible via Internet. Cette ressource peut être une adresse électronique, un fichier texte, un fichier *html*, ou un programme à exécuter à distance.

Différents types de ressources sont indiquées par leur schéma; chaque schéma différent étant implémenté par un protocole spécifique:

- un schéma *http* est implémenté par le protocole HTTP,
- un schéma *ftp* est implémenté par le protocole FTP,
- un schéma *mailto* est implémenté par le protocole SMTP,
- un schéma *telnet* est implémenté par le protocole TELNET.

La syntaxe générale d'un URL est composée du nom du schéma et du nom de la ressource:

```
<schema>:<ressource du schema>
```

La partie ressource du schéma dépend du protocole utilisé. Pour le protocole HTTP il s'agit du nom du serveur et du chemin d'accès au fichier *.*http* sur ce serveur.

Pour le protocole FTP, il est nécessaire de fournir le nom de l'utilisateur, son mot de passe, le nom du serveur FTP et le numéro du port (21 par défaut).

Exemples:

```
http://www.ireste.fr/fdl/vcl/index.html
ftp://pbakowsk:motdepasse@serveur:21
mailto:ythomas@ireste.fr
```

La classe java.net.URL

En langage Java, la classe *java.net.URL* contient les fonctions nécessaires au développement d'une application au niveau d'un URL. Pour créer un objet de la classe URL, nous disposons de plusieurs constructeurs. Tous les constructeur URL lèvent l'exception *MalformedURLException*.

```
public URL(String url) throws MalformedURLException
public URL(String baseURL, String url) throws MalformedURLException
public URL(String scheme, String baseURL, String url) throws MalformedURLException
```

Exemples:

```
public URL("http://www.ireste.fr/fdl") throws MalformedURLException
public URL("http://www.ireste.fr" ,"/fdl") throws MalformedURLException
```

```
public URL(“http”,“www.ireste.fr”, ”/fdl”) throws MalformedURLException
```

Lire les caractéristiques d’un URL

La classe URL offre plusieurs méthodes de type *get* permettant de connaître les caractéristiques d’un URL:

- *getProtocol()* - renvoie l’identificateur du protocole,
- *getHost()* - renvoie l’identificateur de la machine hôte,
- *getPort()* - renvoie le numéro du port,
- *getFile()* - renvoie le chemin d’accès au fichier,
- *getRef()* - renvoie la référence interne dans le fichier.

Exemple:

```
import java.net.*;
import java.io.*;


public class ParseURL {
   public static void main(String[] args) throws Exception {
      URL aURL = new URL("http://java.sun.com:80/docs/books/"
                         + "tutorial/index.html#DOWNLOADING");
      System.out.println("protocole = " + aURL.getProtocol());
      System.out.println("hote = " + aURL.getHost());
      System.out.println("chemin d’access = " + aURL.getFile());
      System.out.println("port = " + aURL.getPort());
      System.out.println("ref = " + aURL.getRef());
   }
}
```

Ci-dessous le résultat d’exécution de ce programme:

```
protocole = http
hote = java.sun.com
chemin d’access = /docs/books/tutorial/index.html
port = 80
ref = DOWNLOADING
```

Le programme suivant est une applet permettant de scruter l’implémentation de différents schéma sur un site serveur. A noter que le constructeur URL utilise trois arguments: schéma, hôte et chemin (*scheme*, *host*, *path*).

```
import java.applet.*;
import java.awt.*;
import java.net.*;


public class ProtocolTesterApplet extends Applet {
TextArea ta = new TextArea();
```

```
public void init() {
   this.setLayout(new BorderLayout(20,20));
   this.add("Center",ta);
}

public void start() {
String hote = "www.ireste.fr";
String chemin = "/fdl";
String[] schema = {"http", "https", "ftp", "mailto", "telnet", "file", "rmi","finger", "daytime","nfs"};
for(int i=0; i<schema.length;i++) {
try {
 URL u = new URL(schema[i],hote,chemin);
 ta.append(schema[i] + " est implemente\r\n");
 }
 catch(MalformedURLException e) {
  ta.append(schema[i] + " n'est pas implemente\r\n"); }
 }
}
}
```

Le résultat d'exécution affiché dans une fenêtre de type *TextArea*:

```
http est implemente
https est implemente
ftp est implemente
mailto est implemente
telnet est implemente
file est implemente
rmi n'est pas implemente
finger n'est pas implemente
daytime n'est pas implemente
nfs est implemente
```

Lire dans un URL

Quand un objet URL est créé, il est possible d'ouvrir son contenu directement par le biais de la méthode *openStream()* et de le lire sur le *stream* d'entrée par la méthode *read()*.

```
try {
URL u = new URL("http://www.ireste.fr");
InputStream in = u.openStream();
int car;
while ((car = in.read()) != -1)
        System.out.write(car);
```

```
}
catch (IOException e)
 {
System.err.println(e);
}
```

L'exemple suivant est plus complet. La méthode *openStream()* ouvre un flux *java.io.InputStream*. Le flux est ensuite traité par les classes *InputStreamReader* et *BufferedReader* pour y lire une chaîne de caractères.

```
import java.net.*;
import java.io.*;
public class URLReader {
   public static void main(String[] args) throws Exception {
              URL ireste = new URL("http://www.ireste.fr/");
   BufferedReader in = new BufferedReader( new InputStreamReader( ireste.openStream()));
              String inputLine;
           while ((inputLine = in.readLine()) != null)
              System.out.println(inputLine);
           in.close();
 }
}
```

Connexions URL

La classe *URLConnection* permet de contrôler la communication avec un serveur HTTP. Par le biais d'une connexion URL, l'utilisateur peut analyser les en-têtes MIME associés à un transfert HTTP et il peut envoyer ses propres données par les actions *POST* et *PUT*.

L'utilisation d'une connexion passe habituellement par plusieurs stades:

- la construction d'un objet URL,
- l'ouverture d'une connexion par *openConnection()* sur l'objet URL,
- la configuration de la connexion,
- la lecture de l'en-tête,
- la lecture des données,
- l'écriture des données,
- la fermeture de la connexion.

Certains de ces stades sont optionnels; c'est le cas de la lecture de l'en-tête ou de l'écriture des données. La lecture et l'écriture peuvent être exécutées dans l'ordre inverse: l'écriture puis la lecture.

Ci-dessous l'exemple d'établissement d'une connexion URL:

```
try {
  URL u = new URL("http://www.ireste.fr");
  URLConnection uc = u.openConnection();
  }
  catch (MalformedURLCOnnection e)
                    { System.err.println(e); }
  catch (IOException e)
                   { System.err.println(e);}
```

Lecture dans un URL

Le programme suivant ouvre une connexion URL et obtient en entrée un flux d'octets à tampon :

```
import java.net.*;
import java.io.*;
public class URLConnectionReader1 {
public static void main(String[] args) throws Exception {
  if(args.length>0)
  {
       URL u = new URL(args[0]);
       URLConnection uc = u.openConnection();
       InputStream rawin = uc.getInputStream();
                InputStream tampon = new BufferedInputStream(rawin);
       Reader r = new InputStreamReader(tampon);
            int c;
       while ((c = r.read()) != -1) System.out.println((char)c);
       in.close();
    }
}
```

La version suivante lit une chaîne de caractères ligne par ligne:

```
import java.net.*;
import java.io.*;

public class URLConnectionReader2 {
public static void main(String[] args) throws Exception {
  if(args.length>0)  {
       URL u = new URL(args[0]);
       URLConnection uc = u.openConnection();
       BufferedReader in = new BufferedReader(new InputStreamReader(uc.getInputStream()));
       String inputLine;
```

```
            while ((inputLine = in.readLine()) != null)
                System.out.println(inputLine);
            in.close();
        }
      }
    }
```

Lecture de l'en-tête MIME

A chaque transfert d'information, les serveurs HTTP génèrent un en-tête qui contient les caractéristiques de l'information envoyée.

Exemple d'un en-tête:

```
HTTP 1.1  200 OK    // version et code d'erreur
Date: Mon, 5 Nov 2000 12:05:32 GMT
Server: Apache/1.3.4 (Linux)
Last-Modified: Mon, 5 Nov 2000 09:12:41 GMT
ETag: "5e06f3-74aa- 320a357f"
Accept-Ranges: bytes
Content-Length: 23871
Connection: close
Content-Type: text/html
```

Les caractéristiques de l'en -tête peuvent être retrouvées par les méthodes *get*. Par exemple le type du contenu peut être obtenu par la méthode *getContentType()*; la taille de l'information par la méthode *getContentLenght()*.

Le programme suivant cherche un site WEB et en extrait un fichier (*args[0]*). Les paramètres MIME du fichier sont affichés sur l'écran. Le fichier lui même est stocké dans le répertoire de travail.

```
import java.net.*;
import java.io.*;
import java.util.*;


public class URLSaveFile {
public static void main(String[] args) {
for(int i=0; i<args.length;i++) {
try {  URL racine = new URL(args[0]);     saveFile(racine);
 }
 catch(MalformedURLException e) { System.err.println(args[i] + " n'est pas un URL"); }
 catch(IOException e) { System.err.println(e); }
 }
}
public static void saveFile(URL u) throws IOException{
URLConnection uc = u.openConnection();
            String contentType = uc.getContentType();
```

```
            int taille = uc.getContentLength();
            System.out.println("Type: " + contentType);
            System.out.println("Taille: " + taille);
            System.out.println("Date :" + new Date(uc.getDate()));
            System.out.println("Modification :" + new Date(uc.getLastModified()));
            System.out.println("Expiration :" + new Date(uc.getExpiration()));
            System.out.println("Encodage :" + uc.getContentEncoding());
            if(taille == -1 ) { throw new IOException("pas de fichier"); }
    InputStream rawin = uc.getInputStream();
            InputStream tampon = new BufferedInputStream(rawin);
            byte[] data = new byte[taille];
            int bytesRead = 0;
            int offset =0;
            while(offset <taille) {
              bytesRead = rawin.read(data,offset,data.length-offset);
              if(bytesRead == -1) break;
              offset +=bytesRead;
              }
    rawin.close();
            if(offset != taille) {
              throw new IOException("lus" + offset + "demandes" + taille); }
            String nomfichier = u.getFile();
            nomfichier = nomfichier.substring(nomfichier.lastIndexOf('/') + 1);
            FileOutputStream fout = new FileOutputStream(nomfichier);
            fout.write(data);
            fout.flush();
            fout.close();
            }
    }
```

Accès aux pages protégées

L'accès à plusieurs sites WEB nécessite une interaction entre l'utilisateur et le serveur. Par exemple la lecture d'une page protégée par un nom utilisateur et un mot de passe nécessite une interaction avec l'utilisateur.

Le mécanisme d'authentification repose sur un gestionnaire d'authentification, objet d'une sous-classe de la classe abstraite *Authenticator*. Sa méthode statique *setDefaut()* permet d'installer un tel gestionnaire.

Après l'installation du gestionnaire, les objets de la classe URL font appel à ses services chaque fois qu'une requête retourne un code de statut 401 *Unathorized* ou 407 *Proxy Authentication Required*. L'objet URL fait appel à la méthode statique *requestPasswordAuthentication()* de la classe *Authenticator*. Cette dernière appelle la méthode *getPasswordAuthentication()* du gestionnaire d'authentification courant et retourne un "user-ID" et un mot de passe. Ces résultats sont encapsulés dans un objet de la classe *PasswordAuthentication* et placés par la classe URL dans l'en-tête d'autorisation d'accès.

Les caractéristiques de l'en -tête peuvent être retrouvées par les méthodes *get*. Par exemple le type du contenu peut être obtenu par la méthode *getContentType()*; la taille de l'information par la méthode *getContentLenght()*.

Le programme suivant cherche un site WEB et en extrait un fichier (*args[0]*). Les paramètres MIME du fichier sont affichés sur l'écran. Le fichier lui même est stocké dans le répertoire de travail.

```
import java.net.*;
import java.io.*;
public class AuthenticatorTest extends Authenticator {
private String name;
private String passwd;
public AuthenticatorTest(String name, String passwd) {  // stockage du user-ID et mot de passe
this.name=name;
this.passwd=passwd;
}
public PasswordAuthentication getPasswordAuthentication() {  // appelee si statut retourne: 401 ou 407
// affiche des informations concernant la requête
System.out.println("Site: " + getRequestingSite());
System.out.println("Port: " + getRequestingPort());
System.out.println("Scheme: " + getRequestingScheme());
System.out.println("Prompt: " + getRequestingPrompt());
// retourne le user-ID et le mot de passe
return new PasswordAuthentication(name,passwd.toCharArray());
 }
public static void main(String[] args) throws Exception {
if(args.length!=1) {
  System.err.println("Usage: java AuthenticatorTest <url>");
  System.exit(1);
   }
AuthenticatorTest auth;   // creation d'un gestionnaire d'authentification
auth = new AuthenticatorTest("bako","123");
Authenticator.setDefault(auth); // installation du gestionnaire d'authentification
URL url = new URL(args[0]);
URLConnection uc = url.openConnection(); // ouverture d'une connexion
InputStream in = uc.getInputStream();
int c;
 while((c=in.read())!=-1){
  System.out.print((char)c);
   }
 }
}
```

Cache local

La plupart des communications avec le serveur WEB passe par la mémoire cache installée localement sur la machine du client. La méthode *setUseCaches()* permet de valider ou d'invalider l'utilisation du cache.

Exemple:

```
try {
URL u = new URL("http://www.serveur.fr/fichier.htm");
URLConnection uc = u.openConnection();
if(uc.getUseCaches())              // utilisation du cache autorise
 {
 uc.setUseCaches(false);          // utilisation du cache souprimee
  }
// lire directement dans l'URL
 }
catch (IOException e) { System.err.println(e); }
```

Envoyer les données au serveur

GET ou *POST*

Les programmes qui utilisent une connexion URL peuvent envoyer leurs données au serveur. La connexion peut être configurée pour les actions *GET* ou *POST*.

Les données envoyées par l'action *GET* sont attachées à l'URL; leur taille est normalement assez limitée, par exemple 1024 octets. Pour envoyer des données de taille plus conséquente, il faut utiliser l'action *POST*.

La méthode *setDoOutput(false)* implique l'utilisation de l'action GET; *setDoOutput(true)* change l'action *GET* en *POST*.

Exemple:

```
try {
URL u = new URL("http://www.serveurcommunicant.fr");
URLConnection uc = u.openConnection();
if(!uc.getDoOutput())              // test si le GET est positionne - false
      {
      uc.setDoOutput(true);       // connexion configuree pour l'action POST
      }
// ecrire dans la connection par POST
 }
catch (IOException e) { System.err.println(e); }
```

Des qu'une connexion est configurée pour l'action POST, les données peuvent être préparées dans un tampon de sortie: *BufferedOutputStream* ou *BufferedWriter*.

Exemple:

```
try {
URL u = new URL("http://www.serveurcommunicant.fr");
URLConnection uc = u.openConnection();
if(!uc.getDoOutput())              // test si le GET est positionne - false
 {  uc.setDoOutput(true);      // connexion configuree pour l'action POST  }
 OutputStream raw = uc.getOutputStream();
 OutputStream tampon = new BufferedOutputStream(raw);
OutputStreamWriter out = new OutputStreamWriter(tampon,"8859_1");
out.write("un=Jean&tp=sockets&exemple=url@connexion\r\n");
out.flush();
out.close();
 }
catch (IOException e) { System.err.println(e); }
```

Remarque: Le programme Java stocke les données dans le tampon jusqu'à la fermeture de la connexion par *close()*. Cela est nécessaire pour déterminer le paramètre *Content-length* dans l'en-tête *http*.

Le message envoyé est précédé par l'en-tête:

```
POST /cgi-bin/resultat.pl  HTTP 1.0
Content-type: application/x-www-form-urlencoded
Content-length: 42
un=Jean&tp=sockets&exemple=url%40connexion        // format encode
```

Le programme suivant envoie une chaîne de caractères au serveur *http java.sun.com*. Le programme à exécuter est logé dans le répertoire */cgi-bin/* et il s'appelle *backwards.pl*.

```
import java.io.*;
import java.net.*;


public class Reverse {
   public static void main(String[] args) throws Exception {
            if (args.length != 1) {
              System.err.println("Utilisation:  java Reverse " + "chaine_a_retourner");
              System.exit(1);}
            String stringToReverse = URLEncoder.encode(args[0]);         // encodage de la chaine
            URL url = new URL("http://java.sun.com/cgi-bin/backwards.pl");
            URLConnection connection = url.openConnection();
            connection.setDoOutput(true);          // initialisation de l'action POST
            PrintWriter out = new PrintWriter(connection.getOutputStream());
            out.println("string=" + stringToReverse);
```

```
            out.close();
            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null)
               System.out.println(inputLine);
            in.close();
      }
}
```

Chargeur de classes

Dans les exemples précédents nous avons utilisé les connexions URL pour recevoir des pages (fichiers) de données. La machine virtuelle Java permet également de lire le *codebyte* de classes distantes et de les exécuter localement. Pour réaliser cette opération il faut utiliser un chargeur de classes.

Le chargeur de classes installé par défaut recherche les classes localement, dans les répertoires et les archives d'installation. Dans ce chapitre nous nous intéressons à la classe*java.net.URLClassLoader* dédiée au chargement de classes distantes accessibles sur Internet.

La forme la plus simple d'un constructeur de chargeur de classes est *URLClassLoader(URL[] urls)*. Comme argument URL, l'utilisateur peut fournir une référence *http* ou un fichier local.

Exemples:

```
URLClassLoader(http://www.ireste.fr/fdl/test/)
URLClassLoader(file:///home/test/)
```

Les URL de base utilisés par le chargeur de classes sont considérés comme répertoires s'ils se terminent par un *slash* ('/'), et comme archives Java dans tous les autres cas.

Pour un objet chargeur obtenu, la récupération d'une classe se fait par un appel à la méthode *loadClass()* avec en argument le nom de la classe demandée. Cette méthode hérite de la classe *ClassLoader*. Si la classe demandée n'est pas trouvée à partir des URL de base de ce chargeur, elle lève l'exception *ClassNotFoundException*.

Nous allons préparer deux classes à charger à distance. La première sera enregistrée sur le disque local (schéma: file), la deuxième sur le serveur WEB (schéma: *http*). Les deux classes implémentent la même interface *Chargeable*.

```
public interface Chargeable {            // a compiler avant la classe SimpleClassLoader
public void charge();
}
public class FichierLocal implements Chargeable {       // classe a charger
public void charge() {                                   // methode charge()
System.out.println("Ici un fichier local"); }
}
public class FichierWEB implements Chargeable {
public void charge() {                                  // methode charge()
```

```
System.out.println(“Ici un fichier WEB”); }
}
```

Ces deux classes peuvent être dynamiquement chargées à partir d’un URL indiqué dans la ligne de commande du programme ci-dessous. Ce programme utilise un chargeur *URLClassLoader()*.

```
import java.net.*;
public class SimpleClassLoader {
public static void main(String[] args) throws Exception {
if(args.length<1) {
   System.err.println("Utilisation: java SimpleClassLoader <classe>");
    System.exit(1); }
URL[] urls = new URL[] {
new URL("http://www.ireste.fr/fdl/ars/test/"),
new URL("file:///home/pbakowsk/test/")  };
URLClassLoader cl = new URLClassLoader(urls);           // creation du chargeur
Class c = null;  // declaration de la classe recherchee
try {
  c = cl.loadClass(args[0]);  // chargement de la classe
  Chargeable ic = (Chargeable)c.newInstance();           // creation de l'instance
   ic.charge();                                          // appel de la methode charge()
    } catch(ClassNotFoundException e) { System.err.println("Pas de classe " + args[0]);}
    } catch(ClassCastException e) { System.err.println("La classe " + args[0] + " n'implemente pas Chargeable");
}
  }
}
```

Recherche d’une ressource WEB

La classe *URLClasseLoader* contient la méthode *findResource(String nom)* permettant d’effectuer une recherche de la ressource dont le nom est passé en paramètre. La méthode *findResource(String nom)* retourne *null* si aucun URL de base ne permet d’accéder à une ressource de ce nom.

Attention: La méthode *findRessources()* requiert le nom complet de la ressource qui n’est pas forcement un *bytecode* d’une classe.

Exemple:

```
import java.net.*;
import java.util.Enumeration;
public class ChercheRessources {
public static void main(String[] args) throws Exception {
if(args.length<1) {
   System.err.println("Utilisation: java ChercheRessources <nomres>");
   System.exit(1); }
URL[] urls = new URL[] {
            new URL("http://www.ireste.fr/"),           // schema http
```

```
            new URL("file:///c:\\"),                    // schema file
            new URL("ftp://ftp.inria.fr/"),             // schema ftp  };
URLClassLoader cl = new URLClassLoader(urls);
System.out.println("La ressource " + args[0] + " existe a(ux) URL suivant(s) :");
Enumeration e = cl.findResources(args[0]);
while(e.hasMoreElements()) {
   System.out.println("\t"+((URL) e.nextElement()).toString()); }
 }
}
```

Résumé

Dans ce chapitre nous avons étudié la classe URL et différentes méthodes permettant de communiquer avec les serveurs WEB. Selon le mode d'initialisation, cette communication peut être uni- ou bi-directionnelle. L'information qui circule entre le client et le serveur peut être de nature très différente: fichier texte, fichier binaire (images, sons). L'utilisation des en-têtes MIME permet de caractériser la taille, la nature et d'autres caractéristiques de l'information transmise.

Les données très particulières sont des fichiers de classes Java. Leur lecture et chargement à distance nécessitent l'intervention de la classe *java.net.URLClassLoader* et de ses méthodes.