Programación en red: URLs

URL

Definición:

- URL es un acrónimo para *Uniform Resource Locator*, y es una referencia (i.e. una dirección) a un recurso de internet. El recurso puede ser algo simple como un archivo o directorio, así como una referencia a un objeto más complicado como una consulta a una base de datos o un motor de búsqueda.

Una URL tiene dos componentes principales:

- ✓ Identificador de protocolo
  - Indica el nombre del protocolo a ser usado para recuperar el recurso.
- ✓ Nombre de recurso:
  - Es la dirección completa del recuso. El formato del nombre depende del protocolo utilizado

http://www.example.com

Para muchos protocolos, incluyendo HTTP, el nombre de recurso puede contener uno o más de los siguientes componentes:

- **Hostname.**
  - Nombre de la máquina en donde se encuentra el recurso.

    http://www.example.com/docs/resource1.html
- **Filename.**
  - El nombre de ruta del archivo en la máquina.

    http://www.example.com/docs/resource1.html
- **Número de puerto.**
  - El número de puerto al cual conectar (típicamente opcional).

    http://www.example.com:1080/docs/resource1.html
- **Referencia.**
  - Una referencia a un delimitador con nombre dentro de un recurso que usualmente identifica una ubicación específica dentro de un archivo (típicamente opcional indicado por un #).

    http://java.sun.com/index.html#chapter1

Por lo general, el hostname y filename es requerido, mientras que el número de puerto y la referencia son opcionales.

Creando URLs en Java

La forma más sencilla de crear un objeto URL en java es a través de un String representando la dirección URL.

✓public URL(String spec)

- Crea un objeto URL a partir de una representación con String.
- Puede arrojar un **MalformedURLException** si no se especifica un protocolo, o se encuentra un protocolo desconocido, o el argumento es null.

Por ejemplo:

```
URL myURL = new URL( "http://example.com" );
```

El objeto URL creado en el ejemplo representa una **URL absoluta**. Una URL absoluta contiene toda la información necesaria para encontrar el recurso en cuestión.

Nota: Una vez se crea un objeto URL, ninguno de sus atributos puede ser modificado (protocolo, hostname, filename, o número de puerto).

URL Relativas

Definición:

- Una URL relativa contiene sólo información suficiente para encontrar un recurso relativo a (o en el contexto de) otra URL.

Las URL relativas son por lo general usadas en archivos HTML. Por ejemplo, supongamos que se tiene un archivo JoesHomePage.html. Dentro de esta página, hay enlaces a otras páginas, PicturesOfMe.html y MyKids.html, las cuales se encuentran en la misma máquina y directorio que JoesHomePage.html. Los enlaces a estos dos archivos pueden ser especificados como nombres de archivos:

```
<a href="PicturesOfMe.html">Pictures of Me</a>
<a href="MyKids.html">Pictures of My Kids</a>
```

Estas direcciones URL son direcciones URL relativas, es decir, las URLs son especificadas de forma relativa al archivo en el que están contenidas.

Creando URLs relativo a otro

La clase URL de java cuenta con un constructor para permitir la creación de URL relativas a otro objeto URL.

✓public URL(URL context, String spec)

- Crea un URL analizando la especificación dada dentro del contexto indicado.
- Puede arrojar un MalformedURLException.
- Si el contexto es null, spec es tratado como un URL absoluto.

Supongamos que se conocen dos URL del sitio example.com

- http://example.com/pages/page1.html
- http://example.com/pages/page2.html

Podemos crear objetos URL para estas páginas en relación a su URL base común (http://example.com/pages/) de la siguiente forma:

```
URL myURL = new URL("http://example.com/pages/");
URL page1URL = new URL(myURL, "page1.html");
URL page2URL = new URL(myURL, "page2.html");
```

Constructores adicionales

Supongamos que se diseña un panel de búsqueda de red similar a un panel de exploración de archivos que permite al usuario elegir el protocolo, nombre de host, número de puerto y nombre de archivo. La clase URL cuenta con dos constructores para instanciar una URL a partir de sus componentes.

- ✓public URL(String protocol, String host, int port, String file)
  - El argumento host puede ser expresado como un nombre de host o una dirección IP literal (las direcciones IPv6 deben ser presentadas en corchetes).
  - Especificar -1 como puerto indica que el URL debe usar el puerto por defecto para el protocolo.
  - Puede arrojar un MalformedURLException si el protocolo es desconocido.

- ✓public URL(String protocol, String host, String file)
  - Es equivalente a usar el constructor anterior teniendo -1 como argumento para port.
  - Puede arrojar un MalformedURLException si el protocolo es desconocido.

Por ejemplo:

```
URL gamelan = new URL( "http", "example.com", "/pages/page1.html“ );
```

es equivalente a usar

```
URL gamelan = new URL( "http://example.com/pages/page1.html" );
```

Por otro lado,

```
URL gamelan = new URL( "http", "example.com", 80, "pages/page1.html“ );
```

genera el objeto URL para la siguiente dirección:

http://example.com:80/pages/page1.html

Es posible obtener un String conteniendo la dirección URL completa usando los métodos equivalentes **toString( )** o **toExternalForm( )**.

Direcciones URL con Caracteres Especiales

Algunas URL contienen caracteres especiales como el caracter de espacio en la siguiente URL

http://example.com/hello world/

Para hacer estos caracteres legales, deben ser codificados antes de ser pasados al constructor de URL.

```
URL url = new URL("http://example.com/hello%20world");
```

La **codificación URL** reemplaza los espacios con un signo “+”, y algunos caracteres ASCII con un signo “%” seguido por su equivalente en hexadecimal.

Analizando un URL

La clase URL proporciona una variedad de métodos que permite consultar objetos URL.

- ✓String getProtocol( )
  - ➢Retorna el nombre de protocolo de esta URL.
- ✓String getAuthority( )
  - ➢Retorna la autoridad de esta URL.
- ✓String getHost( )
  - ➢Retorna el nombre de host de la URL.
- ✓int getPort( )
  - ➢Retorna el número de puerto de esta URL o -1 en caso no se haya especificado uno.

✓String getPath( )

- Retorna la ruta de esta URL o un String vacío si no existe una.

✓String getQuery( )

- Retorna la consulta de la URL o null en caso no haya una.

✓String getFile( )

- Retorna el nombre de archivo del URL. Este método tiene el mismo retorno que getPath( ) más la concatenación de getQuery( ) en caso exista.

✓String getRef( )

- Retorna la referencia de esta URL o null en caso no exista.

Nota: No todas las direcciones URL contienen estas componentes. La clase proporciona estos métodos debido a que las URLs HTTP suelen contener estas componentes y son las URL más usadas.

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

public class ParseURL {
    public static void main(String[] args) throws Exception {

        URL aURL = new URL("http://example.com:80/docs/books/tutorial"
                           + "/index.html?name=networking#DOWNLOADING");

        System.out.println("protocol = " + aURL.getProtocol());
        System.out.println("authority = " + aURL.getAuthority());
        System.out.println("host = " + aURL.getHost());
        System.out.println("port = " + aURL.getPort());
        System.out.println("path = " + aURL.getPath());
        System.out.println("query = " + aURL.getQuery());
        System.out.println("filename = " + aURL.getFile());
        System.out.println("ref = " + aURL.getRef());
    }
}
```

Leyendo directo de una URL

Podemos llamar al siguiente método para obtener un stream a partir del cual leer el contenido de una URL.

- ✓InputStream openStream( )
  - ➢Abre una conexión a la URL y retorna un InputStream para leer desde esa conexión.
  - ➢Puede arrojar un IOException.

```
public class URLReader {
    public static void main(String[] args) throws Exception {

        URL oracle = new URL("http://www.oracle.com/");
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(oracle.openStream()));

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

Estableciendo una Conexión con un URL

La clase abstracta **URLConnection** es una superclase de todas las clases que representen un enlace de comunicación entre una aplicación y un URL.

Las instancias de esta clase pueden ser usadas tanto para leer y escribir a un recurso referenciado por la URL. En general, crear una conexión a un URL es un proceso de múltiples pasos:

1. El objeto de conexión es creado invocando el método openConnection( ) de un objeto URL.
2. Los parámetros de configuración y propiedades generales de solicitiud sin creados.
3. La conexión actual al objeto remoto es hecha usando el método connect( ).
4. El objeto remoto se vuelve disponible. Los campos de cabecera y contenido del objeto remoto pueden ser accedidos.

Las siguientes líneas de código abren una conexión al sitio example.com:

```
try {
    URL myURL = new URL("http://example.com/");
    URLConnection myURLConnection = myURL.openConnection();
    myURLConnection.connect();
}
catch (MalformedURLException e) {
    // new URL() failed
    // ...
}
catch (IOException e) {
    // openConnection() failed
    // ...
}
```

No siempre será necesario llamar explícitamente al método connect( ) para iniciar la conexión. Las operaciones que dependen de estar conectado, como getInputStream( ), getOutputStream( ), etc, realizarán la conexión de forma implícita de ser necesario.

Modificando los Parámetros de Configuración

Los parámetros de configuración de una URLConnection pueden ser modificados usando los siguientes métodos:

- ✓void setAllowUserInteraction(boolean allowuserinteraction)
  - ✓Si es verdadero, la URL es examinada en un contexto en el cual tiene sentido permitir las interacciones de usuario como mostrar un cuadro de diálogo de autenticación. Caso contrario, no se permite la interacción de usuario.
- ✓void setDoInput(doinput)
  - ➢Establecer este parámetro como true indica que la aplicación pretende leer datos de la conexión URL.
  - ➢Valor por defecto: true.
- ✓void setDoOutput(boolean dooutput)
  - ➢Establecer este parámetro como true indica que la aplicación pretende escribir datos en la conexión URL.
  - ➢Valor por defecto: false.

- ✓void setIfModifiedSince(long ifmodifiedsince)
  - ➢Algunos protocolos permiten omitir la obtención del objeto a menos este se haya modificado más recientemente que un tiempo determinado.
  - ➢Valor por defecto: 0.
  - ➢Un valor distinto de 0 representa un tiempo como el número de milisegundos desde enero 1 de 1970 GMT. El objeto es recuperado sólo si ha sido modificado más recientemente a dicho tiempo.

- ✓void setUseCaches(boolean usecaches)
  - ➢Si es verdadero, el protocolo tiene permitido usar almacenamiento en caché siempre que sea posible. Caso contrario, el protocolo siempre debe intentar obtener una nueva copia del objeto.

Nota: Ninguno de estos parámetros puede ser modificado una vez se ha realizado la conexión.

Accediendo al Contenido de un URL

Los siguientes métodos son usados para acceder a campos de cabecera y los contenidos luego de que una conexión es hecha con el objeto remoto.

- ✓Object getContent( )
  - ➢Recupera el contenido de la conexión URL.
  - ➢Se debe utilizar el operador instanceOf para determinar el tipo específico del objeto retornado.
- ✓String getHeaderField(int n)
  - ➢Retorna el valor del n-ésimo campo de cabezera.
  - ➢Retorna null si hay menos de n+1 campos.
  - ➢El parámetro n debe ser mayor o igual a 0.

Algunos campos de cabecera son usualmente accedidos. Los siguientes métodos proporcionan acceso a estos campos:

- ✓String getContentEncoding( )
  - Retorna el valor del campo de cabecera *content-encoding*.
  - En caso de no conocer el valor, retorna null.

- ✓int getContentLength( )
  - Retorna el valor del campo de cabecera *content-length*.
  - Retorna -1 si la longitud de contenido es desconocido o si el valor es mayor a Integer.MAX_VALUE.

- ✓String getContentType( )
  - Retorna el valor del campo de cabecera *content-type*.
  - Retorna null si no se conoce el tipo.

✓long getDate( )

- Retorna el valor del campo de cabecera *date*.
- Retorna la fecha de envío del recurso al cual referencia el URL.
- Si se desconoce el valor del campo, retorna 0.

✓long getExpiration( )

- Retorna el valor del campo de cabecera *expires*.
- Retorna la fecha de expiración del recurso referenciado por el URL.
- Si se desconoce el valor del campo, retorna 0.

✓long getLastModified( )

- Retorna el valor del campo de cabecera *last-modified*.
- Retorna la fecha en la que el recurso referenciado por el URL fue modificado por última vez.
- Si se desconoce el valor del campo, retorna 0.

Observación: Los valores de tiempo retornados son la cantidad de milisegundos transcurridos desde enero 1 de 1970 GMT.

Leyendo desde una Conexión URL

Una vez iniciada una conexión con el URL, es posible leer un stream de entrada de la misma haciendo uso del siguiente método:

- ✓InputStream getInputStream( )
  - ➢Retorna un stream de entrada que lee desde la conexión abierta del objeto URLConnection.

```
public class URLConnectionReader {

   public static void main(String[] args) throws Exception {
      URL oracle = new URL( "http://www.oracle.com/" );
      URLConnection yc = oracle.openConnection( );
      BufferedReader in = new BufferedReader(
                  new InputStreamReader( yc.getInputStream( ) ) );
      String inputLine;
      while ( ( inputLine = in.readLine( ) ) != null )
         System.out.println( inputLine );
      in.close( );
   }

}
```

Escribiendo a una Conexión URL

Para que un programa de java interactúe con un proceso en el lado del servidor usando URL simplemente debe ser capaz de proporcionar datos al servidor. Esto se puede lograr con los siguientes pasos:

1. Crear un objeto URL.
2. Recuperar el objeto URLConnection.
3. Permitir la salida sobre la URLConnection.
4. Abrir una conexión al recurso.
5. Obtener un stream de salida de la conexión.
6. Escribir al stream de salida.
7. Cerrar el stream de salida.

El método getOutputStream( ) de la clase URLConnection permite obtener el stream de salida de la conexión URL realizada. Luego se puede instanciar cualquiera de las clases ya vistas para realizar la escritura.

URLEncoder

Clase de utilidad para codificaciones de formularios HTML.

Cuando se codifica un String, se aplican las siguientes reglas:

- Los caracteres alfanuméricos se mantienen iguales.
- Los caracteres especiales “ . ”, “ - ”, “ * ” y “ _ ” se mantiene iguales.
- El caracter de espacio es reemplazado por un “ + ”.
- El resto de caracteres son inseguros y son convertidos en uno o más bytes primero usando el esquema de codificación. Luego, cada byte es representado por un String de 3 caracteres “%xy” donde *xy* es la representación de dos dígitos hexadecimales del byte. El esquema recomendado es UTF-8.

La clase cuenta con el siguiente método estático:

✓ static String encode(String s, String enc)

- Retorna la traducción del String s haciendo uso de esquema de codificación indicado.