Lectura y escritura de ficheros con Java

Adolfo Sanz De Diego

Junio de 2011

# Ficheros en Java

Java es multiplataforma, e intenta aislarse del sistema de ficheros, aunque no consigue evitarlo del todo.

Utiliza el clase **File** del paquete java.io:

* Está pensada para realizar operaciones de **información** sobre archivos.
* No proporciona métodos de acceso a los archivos.
* Sólo proporciona operaciones a nivel de sistema de archivos:
  + listado de archivos,
  + crear carpetas,
  + borrar ficheros,
  + cambiar nombre,
  + etc.

# Uso de la clase File

Utiliza como único argumento una cadena que representa una **ruta** (ya sea absoluta o relativa) en el sistema de archivos. El problema es que a veces se utiliza / otras \ etc.

La clase File tiene variables estáticas que nos facilitan la tarea:

* **separator**: El carácter separador de nombres de archivo y carpetas.
  + / para Linux
  + \ para Windows (que debe escribirse \\).
* **pathSeparator**: El carácter separador de rutas de archivo que permite poner más de un archivo en una ruta.
  + : para Linux
  + ; para Windows.

# Métodos de File

|  |  |  |
| --- | --- | --- |
| \*\* | boolean isDirectory() **boolean isFile()** boolean exists() **boolean canRead()** boolean canWrite() **boolean isHidden()** boolean isAbsolute()\*\* | Devuelve true si es una carpeta Devuelve true si es un archivo Devuelve true si existe la carpeta o archivo Devuelve true si se puede leer Devuelve true si se puede escribir Devuelve true si es oculto Devuelve true si la ruta es absoluta |

|  |  |  |
| --- | --- | --- |
| \*\* | String getAbsolutePath() **File getAbsoluteFile()** String getName() **String getParent()** File getParentFile() **boolean setReadOnly()** boolean delete()\*\* | Devuelve una cadena con la ruta absoluta Devuelve un objeto File con la ruta absoluta Devuelve una cadena con el nombre de la carpeta o archivo Devuelve el nombre de la carpeta superior o null si no tiene Devuelve un objeto File la carpeta superior o null si no tiene Activa el atributo de sólo lectura Devuelve true si borra la carpeta o archivo |

|  |  |  |
| --- | --- | --- |
| \*\* | boolean mkdir() **boolean mkdirs()** String[] list() **File[] listfiles()** static File[] listRoots()\*\* | Devuelve true si consigue crear una carpeta Devuelve true si consigue crear una carpeta y si hace falta toda la estructura Devuelve un array con los nombres de los ficheros Devuelve un array de objetos File con los ficheros del directorio Devuelve un array de objetos File con los ficheros de la carpeta raíz |

|  |  |  |
| --- | --- | --- |
| \*\* | boolean renameTo(File f2) **long length()** boolean createNewFile()\*\* | Devuelve true si consigue cambiar el nombre por el de f2 Devuelve el tamaño del archivo en bytes Devuelve true si consigue crear el archivo, false si ya estaba creado, y IOEXception si falla Borra el archivo cuando finaliza la ejecución del programa |

# Ejemplo lectura fichero

BufferedReader fichero = null;  
StringBuilder texto = null;  
try{  
 fichero = new BufferedReader(new FileReader("texto.txt"));  
 String linea = fichero.readLine();  
 while (linea != null) {  
 texto.append(linea);  
 linea = fichero.readLine();  
 }  
} catch(Exception e){  
 System.out.println("Error"+e.toString());  
} finally {  
 try {  
 if (fichero != null) fichero.close();  
 } catch(Exception e){  
 System.out.println("Error"+e.toString());  
 }  
}  
// salida por pantalla  
System.out.println(texto);

# Ejemplo escritura fichero

PrintWriter fichero = null;  
try  
{  
 // si append == true escribe al final del fichero  
 // si append == false sobrescribe el fichero  
 boolean append = false;  
 fichero = new PrintWriter(new FileWriter("texto.txt", append));  
 for (int i = 0; i < 10; i++) {  
 fichero.println("Linea " + i);  
 }  
} catch (Exception e) {  
 e.printStackTrace();  
} finally {  
 try {  
 if (null != fichero) fichero.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
}

# XML en Java

Existen numerosas librerías para procesar XML en Java.

**XStream** nos permite parsear **POJO** (Plain Old Java Objects) a XML y viceversa.

Caracteristicas de XStream:

* **Fácil de usar**.
* **No requiere mapeos**.
* **Alto rendimiento**.
* **XML Limpio**.
* **No requiere modificar los objetos**.
* **Soporte completo para objetos complejos**.
* **Integración con otras API de XML**.
* **Estrategias de conversión personalizables**.
* **Mensajes de error detallados**.
* **Permite otros formatos de salida como JSON**.

# Ejemplo XStream: POJO

public class PhoneNumber {  
 private int code;  
 private String number;  
 // ... constructores y métodos  
}  
public class Person {  
 private String firstname;  
 private String lastname;  
 private PhoneNumber phone;  
 private PhoneNumber fax;  
 // ... constructores y métodos  
}

# Ejemplo XStream: XML

<person>  
 <firstname>Joe</firstname>  
 <lastname>Walnes</lastname>  
 <phone>  
 <code>123</code>  
 <number>1234-456</number>  
 </phone>  
 <fax>  
 <code>123</code>  
 <number>9999-999</number>  
 </fax>  
</person>

# Ejemplo XStream: Conversión

POJO to XML

XStream xstream = new XStream();  
xstream.alias("person", Person.class);  
Person joe = new Person("Joe", "Walnes");  
joe.setPhone(new PhoneNumber(123, "1234-456"));  
joe.setFax(new PhoneNumber(123, "9999-999"));  
String xml = xstream.toXML(joe);

XML to POJO

Person newJoe = (Person) xstream.fromXML(xml);