NIO.2 fájlmveletek

A Java7 megjelenését megelzen nagy várakozás övezte a NIO.2 újdonságait, aztán elmúlt a *hype*. Ennek oka valószínleg az lehet, hogy a Java jelenleg két ers területén – az Android és a Java EE platformon – nincs igazán szükség a fájlmveletek támogatására. Az Android esetén megtrt dolognak számít a fájlrendszer, az SQLite alapú tárterület a támogatott; Java EE esetén pedig deklaráltan nem alapozunk a fájlrendszerre.

Nem mindenki lehet Android vagy Java EE fejleszt – szükséges az IDE-k fejlesztése is :), ezért nézzük át, milyen elnyöket ad a NIO.2, amely alapveten a Java eddigi fájlrendszer modelljét bvíti ki az alábbiakkal:

- Platform független fájlrendszer modell
- Fájlrendszer-fa feltérképezés
- Az állománymveletek atomi támogatása (másolás, törlés, mozgatás)
- Soft- és hard-link támogatás
- Fájl attribútumok kezelése
- Változás követés
- SPI támogatás

Nézzük ezeket egyenként... 🙂

Platform független fájlrendszer modell

A Java7 NIO.2 kapcsán került bevezetésre a Path osztály, amely az eddigi megoldások mellett lehetvé teszi az elérési útvonalak platformra való fordítását. Emlékezzünk: egy csomó problémánk volt abból, hogy *Unix like* rendszereken a fájlrendszer kiinduló pontja a / volt, míg Windows és Windows fájlkezelését emuláló rendszerek (emlékeim szerint például a Symbian) esetén voltak meghajtók, így több gyökere is volt a fájlrendszereknek, illetve az elérési útvonalon a szinteket elválasztó jel a / helyett a \ volt, így a multiplatformos programjaink tele voltak platformfügg elágazásokkal, illetve File.separator összefzésekkel. Nem beszélve a soft- és hard-link támogatásról, a valódi elérési út felderítéserl és a többi apróságról... Na, ennek végre vége... 🙂

Fájlrendszer-fa feltérképezés

A fájlrendszer-fa feltérképezése azon igények egyike, amelyre eddig még nem volt szükségem, de örülök, hogy a NIO.2 ezt már támogatja, az alkalmazás szerverek, illetve a különféle szerver oldali Java megoldások fejleszti nyilván örülnek egy ilyen lehetségnek. A módszer a Visitor minta alapján épül fel, ezért hozzunk létre egy saját Visitor-t, amely egyszeren csak kiírja a kapott Path példányt:

```
public class FileVisitor extends SimpleFileVisitor<Path>
{

  @Override
  public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException
  {
    System.out.println(file);
    return FileVisitResult.CONTINUE;
  }
}
```

Ezek után már csak meg kell hívnunk:

```
Path basePath = Paths.get(".");
try
{
  Files.walkFileTree(basePath, new FileVisitor());
} catch (IOException ex)
{
  Logger.getLogger(App.class.getName()).log(Level.SEVERE, null, ex);
}
```

Az eredmény nem marad el:

```
./pom.xml
./src/test/java/hu/javaforum/testproject/AppTest.java
./src/main/java/hu/javaforum/testproject/App.java
./target/classes/hu/javaforum/testproject/App$FileVisitor.class
./target/classes/hu/javaforum/testproject/App$1.class
./target/classes/hu/javaforum/testproject/App.class
```

Az állománymveletek atomi támogatása

Sajnos a Java hetes verzióig kellett várnunk arra, hogy egy fájlt át tudjunk mozgatni vagy akár csak átnevezni... ez is már a múlté... 🙂

```
> ls -l /tmp/test*
-rw-r--r-- 1 auth.gabor users 5 aug   12 12.30 /tmp/test
```

Tekintsük a fenti egy darab állományt, amelyet szeretnénk átmásolni *test2* névre, majd átnevezni *test3* névre:

```
Path srcFile = Paths.get("/tmp/test");
Path dstFile = Paths.get("/tmp/test2");
Path moveFile = Paths.get("/tmp/test3");
Files.copy(srcFile, dstFile, StandardCopyOption.COPY_ATTRIBUTES);
Files.move(srcFile, moveFile, StandardCopyOption.ATOMIC_MOVE);
```

És az eredmény nem marad el:

```
> ls -l /tmp/test*
-rw-r--r-- 1 auth.gabor users 5 aug   12 12.30 /tmp/test2
-rw-r--r-- 1 auth.gabor users 5 aug   12 12.30 /tmp/test3
```

Ez Java7 eltt egy több tucat soros varázslatra volt szükség egy ilyen igény teljesítéséhez.

Soft- és hard-link támogatás

A NIO.2 eltt erre nem volt lehetség, nem lehetett megállapítani egy elérési útról, hogy az tartalmaz-e linkeket vagy sem, most már vannak eszközeink.

```
> ls -l /tmp/test*
-rw-r--r-- 1 auth.gabor users 5 aug   12 12.30 /tmp/test
```

Hozzunk létre egy soft- (symbolic-) és egy hard-linket:

```
Path srcFile = Paths.get("/tmp/test");
Path softLinkFile = Paths.get("/tmp/test2");
Path hardLinkFile = Paths.get("/tmp/test3");
Files.createSymbolicLink(softLinkFile, srcFile);
Files.createLink(hardLinkFile, srcFile);

System.out.println("isSymbolicLink:");
System.out.println(srcFile + ": " + Files.isSymbolicLink(srcFile));
System.out.println(softLinkFile + ": " + Files.isSymbolicLink(softLinkFile));
System.out.println(hardLinkFile + ": " + Files.isSymbolicLink(hardLinkFile));
```

Az eredmény nem marad el:

```
isSymbolicLink:
/tmp/test: false
/tmp/test2: true
/tmp/test3: false
```

A fájlrendszernél is látszik az átlinkelés, illetve a link számláló a hard-link esetén:

```
> ls -l /tmp/test*
-rw-r--r-- 2 auth.gabor users 5 aug   12 12.30 /tmp/test
lrwxrwxrwx 1 auth.gabor users 9 aug   12 12.44 /tmp/test2 -> /tmp/test
-rw-r--r-- 2 auth.gabor users 5 aug   12 12.30 /tmp/test3
```

Fájl attribútumok kezelése

Ha le szeretnénk kérdezni egy adott fájl attribútumait, akkor a legtöbb példában megemlítésre kerül a JNI vagy a széttárt kéz, mivel erre túl sok eszközünk nem volt a Java platformot tekintve, s sajnos a megfelel megoldásra egészen a Java7 megjelenéséig várnunk kellett. A fentiekhez hasonlóan a megoldás egyszer, csak azt kell eldöntenünk, hogy milyen attribútumokat szeretnénk kiolvasni (a teljesség igénye nélkül):

- BasicFileAttributeView
- DosFileAttributeView
- PosixFileAttributeView
- UserDefinedFileAttributeView
- AclFileAttributeView
- FileOwnerAttributeView

Fogjunk egy megfelel áldozatot és vallassuk ki:

```
BasicFileAttributes bfa = Files.readAttributes(srcFile, BasicFileAttributes.class);
DosFileAttributes dfa = Files.readAttributes(srcFile, DosFileAttributes.class);
PosixFileAttributes pfa = Files.readAttributes(srcFile, PosixFileAttributes.class);

AclFileAttributeView afa = Files.getFileAttributeView(srcFile, AclFileAttributeView.class);
FileOwnerAttributeView foa = Files.getFileAttributeView(srcFile, FileOwnerAttributeView.class);

System.out.println("BasicFileAttributes:");
System.out.println(srcFile + ":creationTime " + bfa.creationTime());
System.out.println(srcFile + ":isDirectory " + bfa.isDirectory());
System.out.println(srcFile + ":isOther " + bfa.isOther());
System.out.println(srcFile + ":isRegularFile " + bfa.isRegularFile());
System.out.println(srcFile + ":isSymbolicLink " + bfa.isSymbolicLink());
System.out.println(srcFile + ":lastAccessTime " + bfa.lastAccessTime());
System.out.println(srcFile + ":lastModifiedTime " + bfa.lastModifiedTime());
System.out.println(srcFile + ":size " + bfa.size());

System.out.println("DosFileAttributes:");
System.out.println(srcFile + ":isArchive " + dfa.isArchive());
System.out.println(srcFile + ":isHidden " + dfa.isHidden());
System.out.println(srcFile + ":isReadOnly " + dfa.isReadOnly());
System.out.println(srcFile + ":isSystem " + dfa.isSystem());

System.out.println("PosixFileAttributes:");
System.out.println(srcFile + ":group " + pfa.group());
System.out.println(srcFile + ":owner " + pfa.owner());
System.out.println(srcFile + ":permissions " + pfa.permissions());

System.out.println("AclFileAttributeView:");
if (afa != null)
{
  System.out.println(srcFile + ":getAcl " + afa.getAcl());
}

System.out.println("FileOwnerAttributeView:");
if (foa != null)
{
  System.out.println(srcFile + ":getOwner " + foa.getOwner());
}
```

Majd nézzük meg, mit mesélt magáról:

```
BasicFileAttributes:
/tmp/test:creationTime 2012-08-12T10:30:39Z
/tmp/test:isDirectory false
/tmp/test:isOther false
/tmp/test:isRegularFile true
/tmp/test:isSymbolicLink false
/tmp/test:lastAccessTime 2012-08-12T10:33:09Z
/tmp/test:lastModifiedTime 2012-08-12T10:30:39Z
/tmp/test:size 5
DosFileAttributes:
/tmp/test:isArchive false
/tmp/test:isHidden false
/tmp/test:isReadOnly false
/tmp/test:isSystem false
PosixFileAttributes:
/tmp/test:group users
/tmp/test:owner auth.gabor
/tmp/test:permissions [GROUP_READ, OTHERS_READ, OWNER_WRITE, OWNER_READ]
AclFileAttributeView:
FileOwnerAttributeView:
/tmp/test:getOwner auth.gabor
```

Változás követés

A fájlrendszerben történt változások figyelése fontos feladat, gondoljunk csak a kedvenc fejlesztkörnyezetünkre, amely szeretne értesülni arról, hogy a háta mögött módosítunk egy állományt. A Java7 eltt erre különféle praktikákra volt szükség, de ez is a múlté... 🙂

Tegyük fel, hogy szeretnénk figyelni a /tmp/nio.2/ könyvtár tartalmában beálló módosulásokat:

```
Path tmpDir = Paths.get("/tmp/nio.2/");
WatchService watcher = FileSystems.getDefault().newWatchService();

tmpDir.register(watcher,
        StandardWatchEventKinds.ENTRY_CREATE, StandardWatchEventKinds.ENTRY_DELETE,
        StandardWatchEventKinds.ENTRY_MODIFY);

while (true)
{
  WatchKey key = null;
  try
  {
    key = watcher.take();
  } catch (InterruptedException ex)
  {
    Logger.getLogger(App.class.getName()).log(Level.SEVERE, null, ex);
  }

  for (WatchEvent event : key.pollEvents())
  {
    System.out.println(event.context() + ": " + event.kind());
  }

  key.reset();
}
```

A fenti program szerkezete viszonylag egyszer, megmondjuk, hogy mit szeretnénk figyelni, regisztráljuk a figyel szolgáltatásba, majd egy *végtelen* ciklusban figyeljük az események bekövetkeztét. Próbáljuk ki:

```
> touch /tmp/nio.2/test
> rm /tmp/nio.2/test
> touch /tmp/nio.2/test2
> echo "abc" >/tmp/nio.2/test2
> rm /tmp/nio.2/test2
```

Ezzel egy idben az alábbit kell lássuk a programunk kimenetén:

```
test: ENTRY_CREATE
test: ENTRY_MODIFY
test: ENTRY_DELETE
test2: ENTRY_CREATE
test2: ENTRY_MODIFY
test2: ENTRY_MODIFY
test2: ENTRY_DELETE
```

Úgy gondolom, hogy sok Java fejleszt nem fogja használni a fenti megoldásokat, mivel ahhoz az kellene, hogy a Java platform desktop környezetben is terjedjen el, de jelenleg nem tudunk túl sok példát mondani multiplatformos programra.