E/S de arquivos com Java NIO.2

ACH 2003 — Computação Orientada a Objetos

Daniel Cordeiro

20 de abril de 2016

Escola de Artes, Ciências e Humanidades | EACH | USP

SERIALIZABLE

- a interface não impõe método nenhum. Por padrão todos os campos da classe serão gravados, exceto estáticos e `transient`
- permite que a implementação da classe assuma o controle do processo de seriação com os métodos:
  - `private void writeObject(java.io.ObjectOutputStream out)`

    — usa `out` para gravar os dados no fluxo
  - `private void readObject(java.io.ObjectInputStream in)`

    — instancia a classe usando o construtor padrão (sem argumentos) e então chama esse método para que ele leia os parâmetros e os inicialize corretamente
  - `private void readObjectNoData()`

    — chamado caso não seja possível restaurar o objeto usando os dados do fluxo

EXEMPLO

```java
import java.io.Serializable;
public class User implements Serializable {

    /**
     * Serial version ID (denota a versão da classe)
     */
    private static final long serialVersionUID = -5585768630527384 3L;

    private String name;
    private String username;
    transient private String password;

    @Override
    public String toString() {
        String value = "name : " + name + "\nUserName : " + username
                + "\nPassword : " + password;
        return value;
    }
    // ...
}
```

EXEMPLO 2 – JAVA.UTIL.ARRAYLIST.WRITEOBJECT()

```java
/**
  * Save the state of the <tt>ArrayList</tt> instance to a stream (that
  * is, serialize it).
  *
  * @serialData The length of the array backing the <tt>ArrayList</tt>
  *             instance is emitted (int), followed by all of its elements
  *             (each an <tt>Object</tt>) in the proper order.
  */
private void writeObject(java.io.ObjectOutputStream s)
    throws java.io.IOException{

    // Write out all elements in the proper order.
    for (int i=0; i<size; i++) {
        s.writeObject(elementData[i]);
    }
}
```

EXEMPLO 2 – JAVA.UTIL.ARRAYLIST.READOBJECT()

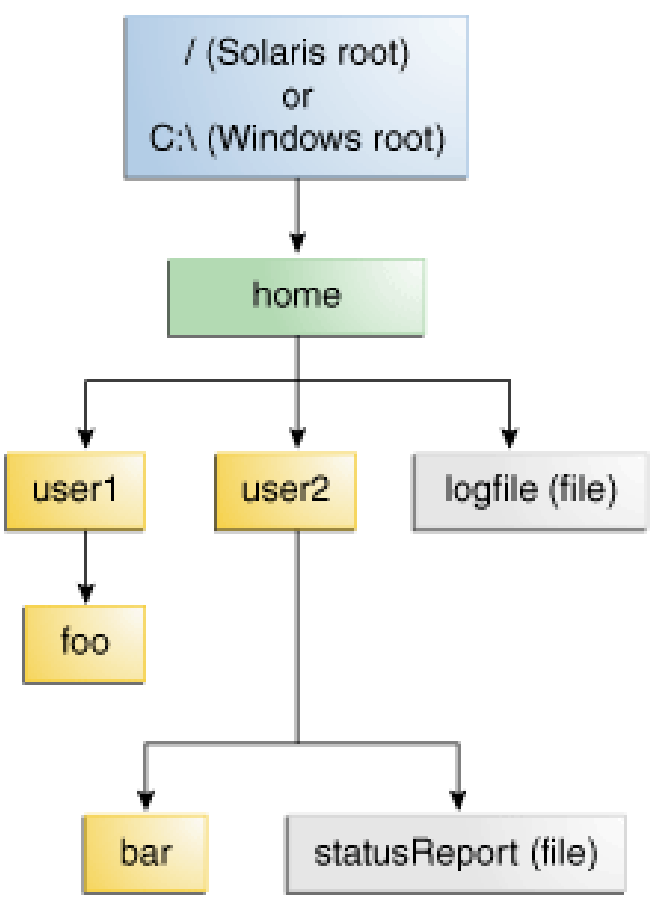
```
/**
 * Reconstitute the <tt>ArrayList</tt> instance from a stream (that is,
 * deserialize it).
 */
private void readObject(java.io.ObjectInputStream s)
    throws java.io.IOException, ClassNotFoundException {
    elementData = EMPTY_ELEMENTDATA;

    // Read in size, and any hidden stuff
    s.defaultReadObject();

    if (size > 0) {
        // be like clone(), allocate array based upon size not capacity
        ensureCapacityInternal(size);

        Object[] a = elementData;
        // Read in all elements in the proper order.
        for (int i=0; i<size; i++) {
            a[i] = s.readObject();
        }
    }
}
```

SISTEMAS DE ARQUIVOS

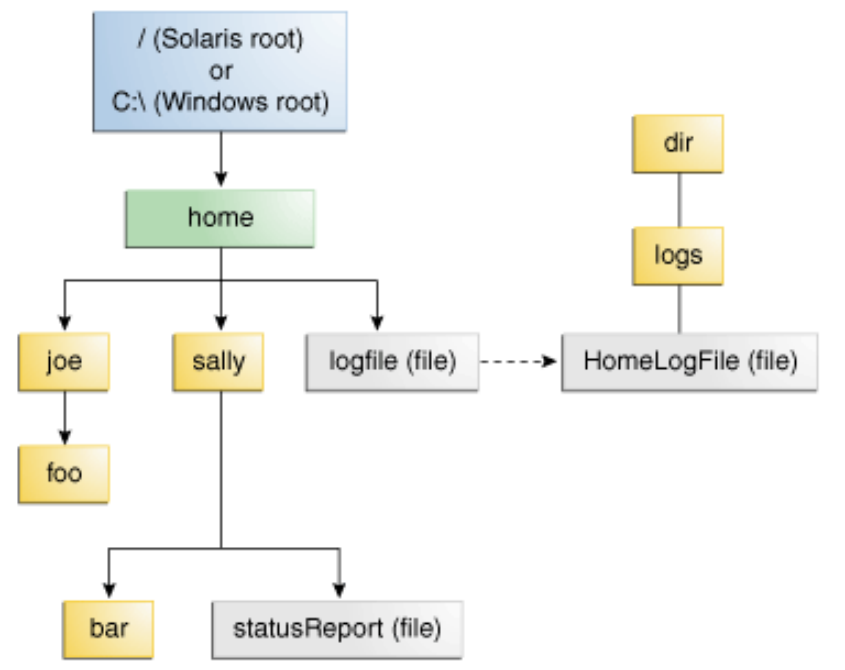


- no Windows cada nó raiz é um volume (`C:\`, `D:\`)
- sistemas Unix só possuem uma raiz, denotada pelo caractere “/”
- o arquivo é identificado pelo caminho a partir da raiz:
  - `/home/sally/statusReport` (Unix)
  - `C:\home\sally\statusReport` (Windows)
- caractere delimitador: separa o nome dos arquivos (“/” ou “\”)

Absoluto ou relativo

absoluto sempre contém a raiz no caminho. Ex: `/home/sally/statusReport`

relativo precisa ser combinado com outro caminho. Ex: `joe/foo`



Link simbólico

Um nó (que parece um arquivo comum) na verdade aponta para um outro caminho.

CLASSE PATH

- ponto de entrada para manipulação de arquivos
- classe `Path` é a forma programática de representar um caminho em Java
- contém o nome do arquivo e a lista de diretórios usados para construir o caminho
- permite examinar, localizar e manipular arquivos

CRIAÇÃO DE UM PATH

Instâncias de `Path` são criadas usando método `get()` da classe auxiliar `Paths`. Exemplos:

```
Path p1 = Paths.get("/tmp/foo");
Path p2 = Paths.get(args[0]);
Path p3 = Paths.get(URI.create("file:///Users/joe/FileTest.java"));

// Paths.get equivale a:
Path p4 = FileSystems.getDefault().getPath("/users/sally");

// /u/joe/logs/foo.log ou C:\joe\logs\foo.log
Path p5 = Paths.get(System.getProperty("user.home"),"logs", "foo.log");
```

INFORMAÇÕES SOBRE UM CAMINHO

```
// Nenhum dos métodos exige que o arquivo exista de fato
// sintaxe Windows
Path path = Paths.get("C:\\home\\joe\\foo");
// sintaxe Unix
Path path = Paths.get("/home/joe/foo");

System.out.format("toString: %s%n", path.toString());
 // /home/joe/foo ou C:\home\joe\foo
System.out.format("getFileName: %s%n", path.getFileName());
 // foo
System.out.format("getName(0): %s%n", path.getName(0));
 // home
System.out.format("getNameCount: %d%n", path.getNameCount());
 // 3
System.out.format("subpath(0,2): %s%n", path.subpath(0,2));
 // home/joe ou home\joe
System.out.format("getParent: %s%n", path.getParent());
 // /home/joe ou \home\joe
System.out.format("getRoot: %s%n", path.getRoot());
 // / ou C:\
```

CONVERSÃO DE CAMINHOS

para URI :

```
Path p1 = Paths.get("/home/logfile");
System.out.format("%s%n", p1.toUri());
  // file:///home/logfile
```

para caminho absoluto :

```
// cd $HOME/.config
Path mail = Paths.get("libreoffice");
Path fullPath = inputPath.toAbsolutePath();
// /home/danielc/.config/libreoffice
```

para o caminho real :

- resolve links simbólicos
- se o caminho for relativo, devolve absoluto
- se houver elementos redundantes, são removidos

Combinação parcial

```
Path p1 = Paths.get("/home/joe/foo");
System.out.format("%s%n", p1.resolve("bar"));
 // Resultado é /home/joe/foo/bar

Paths.get("foo").resolve("/home/joe");
 // Resultado é /home/joe já que o caminho era absoluto
```

CONCATENAÇÃO DE CAMINHOS

Combinação parcial

```
Path p1 = Paths.get("/home/joe/foo");
System.out.format("%s%n", p1.resolve("bar"));
 // Resultado é /home/joe/foo/bar

Paths.get("foo").resolve("/home/joe");
 // Resultado é /home/joe já que o caminho era absoluto
```

Combinação relativa

```
Path p1 = Paths.get("home");
Path p3 = Paths.get("home/sally/bar");

Path p1_to_p3 = p1.relativize(p3);
// Resultado é sally/bar

Path p3_to_p1 = p3.relativize(p1);
// Resultado é ../..
```

Tratamento de exceções

- **muitas** coisas podem sair errado em operações de E/S
- para facilitar o tratamento de exceções, muitas classes de fluxos ou canais implementam ou estendem a interface `java.io.Closeable`

```
Charset charset = Charset.forName("US-ASCII");
String s = ...;
try (BufferedWriter writer = Files.newBufferedWriter(file, charset)) {
    writer.write(s, 0, s.length());
} catch (IOException x) {
    System.err.format("IOException: %s%n", x);
}
```

Varargs

Vários métodos em `Files` foram definidos para receber um número arbitrário de parâmetros. Ex:

```
import static java.nio.file.StandardCopyOption.*;

// Path Files.move(Path, Path, CopyOption...)
Path source = ...;
Path target = ...;
Files.move(source,
           target,
           REPLACE_EXISTING,
           ATOMIC_MOVE);
```

Operações atômicas

Alguns métodos podem realizar operações atômicas, ou seja, operações que não podem ser interrompidas ou realizadas "parcialmente". Importante em programação concorrente.

Operações atômicas

Alguns métodos podem realizar operações atômicas, ou seja, operações que não podem ser interrompidas ou realizadas "parcialmente". Importante em programação concorrente.

Encadeamento de métodos

Muitos exemplos usarão uma técnica de programação orientada a objetos chamada encadeamento de métodos. Você chama um método e ele devolve um objeto, então você chama o método nesse objeto, e ele devolve outro objeto, etc.

```
String value = Charset.defaultCharset().decode(buf).toString();
UserPrincipal group =
    file.getFileSystem().getUserPrincipalLookupService().
        lookupPrincipalByName("me");
```

BIBLIOGRAFIA

- The Java™ Tutorials – Basic I/O: `https://docs.oracle.com/javase/tutorial/essential/io/`