JAVA 7

New Features in JAVA 7

There are a number of features in JAVA 7 that will please developers.

- Strings in switch statements
- Diamond Operator
- Multi-catch exception handling
- Try-with-resource statements
- The new File System API
- Binary integral literals and underscores in numeric literals
- Support for dynamically-typed languages
- The fork and join framework for task parallelism

Strings in switch statements

Switch statements work either with primitive types or enumerated types. Java 7 introduced another type that we can use in Switch statements: the String type.

Eg:

```
String color="ORANGE";
switch(color)
{
case "RED":  System.out.println("STOP");
break;
case "GREEN":  System.out.println("GO");
break;
case "ORANGE": System.out.println("BE READY");
break;
}
```

Automatic resource management

Resources such as Connections, Files, Input/OutStreams , etc. should be closed manually by the developer by writing bog-standard code. Usually we use a try-finally block to close the respective resources. See the current practice of creating a resource, using it and finally closing it:

```
try(resources_to_be_cleant)
        {
                // your code
        }
```

Improved exception handling

There are a couple of improvements in the exception handling area. Java 7 introduced multi-catch functionality to catch multiple exception types using a single catch block.

```
public voidnewMultiCatch()
{
try{ methodThatThrowsThreeExceptions();

}catch(ExceptionOne | ExceptionTwo | ExceptionThree e) {
// log and deal with all Exceptions }
}
```

Numeric literals with underscores

Numerical literals are definitely eye strainers. I am sure you would start counting the zeroes like me if you've been given a number with, say, ten zeros. It's quite error prone and cumbersome to identify a literal if it's a million or a billion unless you count the places from right to left. Not anymore. Java 7 introduced underscores in identifying the places. For example, you can declare 1000 as shown below:

```
int thousand = 1_000;

int million = 1_000_000
```

New file system API (NIO 2.0)

Those who worked with Java IO may still remember the headaches that framework caused. It was never easy to work seamlessly across operating systems or multi-file systems. There were methods such as delete or rename that behaved unexpected in most cases. Working with symbolic links was another issue.

With the intention of solving the above problems with Java IO, Java 7 introduced an overhauled and in many cases new API.

The NIO 2.0 has come forward with many enhancements. It's also introduced new classes to ease the life of a developer when working with multiple file systems.

Working with Path

A new java.nio.file package consists of classes and interfaces such as Path, Paths, FileSystem,FileSystems and others.

A Path is simply a reference to a file path. It is the equivalent (and with more features) to java.io.File. The following snippet shows how to obtain a path reference to the “temp” folder:

```
public voidpathInfo() {
  Path path= Paths.get("c:\Temp\temp");
   System.out.println("Number of Nodes:"+ path.getNameCount());
   System.out.println("File Name:"+ path.getFileName());
   System.out.println("File Root:"+ path.getRoot()); System.out.println("File
   Parent:"+ path.getParent());
}
```

Files in NIO

Deleting a file or directory is as simple as invoking a delete method on Files (note the plural) class. TheFiles class exposes two delete methods, one that throws NoSuchFileException and the other that does not.

The following delete method invocation throwsNoSuchFileException, so you have to handle it:

Files.delete(path);

Where as Files.deleteIfExists(path) does not throw exception (as expected) if the file/directory does not exist.

You can use other utility methods such asFiles.copy(..) and Files.move(..) to act on a file system efficiently. Similarly, use thecreateSymbolicLink(..) method to create symbolic links using your code.

NIO.2 Enhancements – File System Change Notification

File System Change Notification

The java.nio.file package has a WatchService API to support file change notification.

WatchService API attempts to take advantage of native support available in FileSystem for file change notification.

Main goal here is to help with performance issues in applications that are currently forced to poll the file system.

User can register directory/directories to watch and specify type of events to monitor. Various types of events are -

ENTRY_CREATE - A directory entry is created.

ENTRY_DELETE - A directory entry is deleted.

ENTRY_MODIFY - A directory entry is modified.

OVERFLOW - Indicates that events might have been lost or discarded. You do not have to register for the OVERFLOW event to receive it.

Once the event occurs, it is forwarded to registered process which is thread or pool of threads for handling the event.

Cybage® Delivering Value Scientifically

NIO.2 Enhancements – File System Change Notification

- Example

```
FileSystem fileSystem = FileSystems.getDefault()
WatchService wService = fileSystem.newWatchService();
Path dir = ...;
try {
    dir.register(wService, ENTRY_CREATE, ENTRY_DELETE);
    WatchKey key = wService.take(); //or wService.poll();
    :
} catch (IOException x) {
    System.err.println(x);
}
```

- First step is to create WatchService using FileSystem.newWatchService() method.
- Next step is to registered instance of WatchService with object that implements Watchable interface. Java.nio.file.Path implement Watchable interface.
- WatchService has methods take(), poll() and close().

NIO.2 Enhancements – File System Change Notification

- Example

```
for (;;) {
  WatchKey key = watcher.take();
  for (WatchEvent event : key.pollEvents()) {
    String file = object.context().toString();
    if (event.kind() == ENTRY_DELETE) {
      System.out.println("Delete: " + file);
    } if (event.kind() == ENTRY_CREATE) {
      System.out.println("Created: " + file);
    }
  }
}
```

Infinite for loop waits for incoming events. When event occurs key is signaled and place into watcher queue.

Inner for loop retrieves pending events for the WatchKey & process as needed.

Reset key and resume waiting for the events.

Fork and Join

The effective use of parallel cores in a Java program has always been a challenge. There were few home-grown frameworks that would distribute the work across multiple cores and then join them to return the result set. Java 7 has incorporated this feature as a Fork and Join framework.

Basically the Fork-Join breaks the task at hand into mini-tasks until the mini-task is simple enough that it can be solved without further breakups. It's like a divide-and-conquer algorithm. One important concept to note in this framework is that ideally no worker thread is idle. They implement a work-stealing algorithm in that idle workers "steal" the work from those workers who are busy.

The core classes supporting the Fork-Join mechanism are **ForkJoinPool** and **ForkJoinTask**.

Cybage®
Delivering Value Scientifically

Any Questions?

ExcelShore®
Operational Excellence

Cybage®
Delivering Value Scientifically

Thank You!