Listening to Directory Events

José Paumard

PHD, JAVA CHAMPION, JAVA ROCK STAR

@JosePaumard https://github.com/JosePaumard

Agenda

How to observe what is happening in a directory

File / directory creation

File / directory deletion

File / directory modification

Understanding the Problem

We want to set up a system to observe the creations in a given directory

We know how to read the entries from a directory

So we could set up a special task

- that could be activated on a timer

- that would analyze the content of the dir

- and give us the information we need

What do we need?

- a thread

- to keep track of the entries

- to compare before & after contents

What if the rate of creations / deletions is greater than our sampling?

Suppose `file.log` is created then deleted then created again between two observations

What do we need?

- a thread

- to keep track of the entries

- to compare before & after contents

What if the rate of creations / deletions is greater than our sampling?

Suppose `file.log` is created then deleted then created again between two observations

This solution is:

- costly because we need to store the state of the directory

- cannot guarantee that we will not be missing events

Until Java 7, there is no proper way to solve this problem

Java 7 introduces the WatchService pattern

It still uses a special thread

But no scheduler

No events are missed (almost)

It is plugged on the native file system

Introducing the WatchService Pattern

The WatchService Pattern

Setting up a watch service is a four steps process:

1) create a watch service

2) register the watch service to a directory

3) gets the returned key

4) poll the events

```java
Path dir = Paths.get("D:/logs");
FileSystem fileSystem = dir.getFileSystem();
WatchService watchService = fileSystem.newWatchService();
```

As usual we need a starting directory

And the file system of this directory

Then we can get a new watch service

```java
WatchKey key = dir.register(watchService,
                            StandardWatchEventKinds.ENTRY_CREATE,
                            StandardWatchEventKinds.ENTRY_DELETE,
                            StandardWatchEventKinds.ENTRY_MODIFY);
```

Then we need to register this watch service to a path

And provide the events we want to listen to

This key is the object that will be notified on the events

There is one key per directory

The WatchKey Object

Valid as long as the directory is accessible

Three methods to poll events:

- take(): a blocking call

- poll(): non-blocking, can return null

- poll(long, TimeUnit): poll with a time out, can return null

```java
while (key.isValid()) {
    // or poll with / without a timeout
    WatchKey take = watchService.take();
    List<WatchEvent<?>> events = take.pollEvents();
    // work with the events
    take.reset();
}
```

The key may become invalid if the directory is no longer accessible

First we wait for available events

Then we can poll them from the key

A call to reset() is mandatory to empty the event queue

If there are too many events generated

An OVERFLOW is added to the queue

Some events may have been missed

```java
for (WatchEvent<?> event : events) {
    WatchEvent.Kind<?> kind = event.kind();
    if (kind == StandardWatchEventKinds.OVERFLOW) { // overflow
        continue;
    }
    // operations on the elements
}
```

The events are in a list, so we have several patterns to process them

We first get the kind of event: create / modify / delete

If it is an overflow there is not much we can do

Otherwise, we process the event normaly

Demo

Let us see some code!

Let us see this watch service in action

Module Wrap Up

What did you learn?

How to watch a directory for entries events

Java 7 brings the right API for that, plugged on the native file system

Course Wrap Up

The Java NIO API (non-blocking)

Allows for non-blocking reading and writing to disks and network

Also to access off heap memory

And asynchronous operations

Course Wrap Up

The Java NIO2 API

Access to native file systems

Patterns to visit very large directory trees efficiently

A very efficient API to listen to directory events, plugged on the native file system

Course Wrap Up

Thank you!

@JosePaumard

https://github.com/JosePaumard