Watching Directories for Modifications in Java 7

By Kishori Sharan Published on: March 20, 2012 at www.jdojo.com

---

NIO 2.0 in Java 7 added support for directory change notification known as a *watch service*. It lets you receive notifications, when a directory is modified. There are six steps needed to setup a watch service.

- Creating a watch service
- Registering a directory with the watch service
- Retrieving a watch key from the watch service queue
- Processing the events that occurred on the registered directory
- Resetting the watch key after processing the events
- Closing the watch service

A watch service is an instance of the `WatchService` interface. You can create a watch service for the default file system as:

```
WatchService ws = FileSystems.getDefault().newWatchService();
```

Registering a directory to a watch service is simply calling the `register()` method of the `Path` object that represents the directory to watch. You can specify the operations for which you would like to watch the directory. The `register()` method returns a `WatchKey` instance that serves as a token for the registration.

```
// Get a Path object for C:\kishori directory to watch
Path dirToWatch = Paths.get("C:\\kishori");

// Register the dirToWatch for create, modifiy and delete events
WatchKey token = dirToWatch.register(ws,
                                     ENTRY_CREATE,
                                     ENTRY_MODIFY,
                                     ENTRY_DELETE);
```

You can use the `take()` or `poll()` method of the `WatchService` object to retrieve and remove a signaled and queued `WatchKey`. The `take()` method waits until a `WatchKey` is available. The `poll()` method lets you specify a timeout for the wait. Typically, an infinite loop is used to retrieve a signaled `WatchKey`.

```
while(true) {
     WatchKey key = ws.take(); // Retrieve and remove the next available
                               // WatchKey from the watch service
}
```

Once you retrieve and remove a `WatchKey` from the watch service queue, you can retrieve and remove all pending events for that `WatchKey`. A `WatchKey` may have more than one pending events. The `pollEvents()` method of the `WatchKey` retrieves and removes all its pending events. It returns a `List` of `WatchEvent`. Each element of the `List` represents an event on the `WatchKey`. Typically, you will

need to use the `kind()`, `context()`, and `count()` methods of the `WatchEvent` object to know the details of the event. The following snippet of code shows the typical logic for processing an event.

```
while(true) {
     // Retrieve and remove the next available WatchKey
     WatchKey key = ws.take();

     // Process all events of the WatchKey
     for(WatchEvent<?> event : key.pollEvents()) {
          // Process each event here
     }
}
```

You must reset the `WatchKey` object by calling its `reset()` method, so it may receive event notifications and be queued to the watch service again. The `reset()` method puts the `WatchKey` into a *ready* state. The `reset()` method returns `true`, if the `WatchKey` is still valid. Otherwise, it returns `false`. A `WatchKey` may become invalid, if it is cancelled or its watch service is closed.

```
// Reset the WatchKey
boolean isKeyValid = key.reset();
if (!isKeyValid) {
     System.out.println("No longer watching " + dirToWatch);
}
```

When you are done with the watch service, close it by calling its `close()` method. You will need to handle the `java.io.IOException`, when you call its `close()` method.

```
// Close the watch service
ws.close();
```

Listing -1 has a complete program that watches a `C:\kishori` directory for changes. It uses a `try-with-resources` statement to work with the `WatchService` object. You can replace the directory path in the `Watcher` class with the directory path that you want to monitor. You will need to make changes to the watched directory, e.g., create a new file, and change an existing file, after you run the `Watcher` class. The output will show the details of the event that occur on an entry in the watched directory.

Listing -1: An example of implementing a watch service to monitor changes in a directory

```
// Watcher.java
package com.jdojo.chapter11;

import java.nio.file.WatchEvent.Kind;
import java.io.IOException;
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.WatchService;
import java.nio.file.WatchEvent;
import java.nio.file.WatchKey;
import static java.nio.file.StandardWatchEventKinds.ENTRY_CREATE;
import static java.nio.file.StandardWatchEventKinds.ENTRY_MODIFY;
import static java.nio.file.StandardWatchEventKinds.ENTRY_DELETE;
import static java.nio.file.StandardWatchEventKinds.OVERFLOW;

public class Watcher {
```

```
    public static void main(String[] args) {
        try (WatchService ws =
                        FileSystems.getDefault().newWatchService()) {

            // Get a Path object for C:\kishori directory to watch
            Path dirToWatch = Paths.get("C:\\kishori");

            // Register the dirToWatch with the watch service
            // for create, modifiy and delete events
            dirToWatch.register(ws, ENTRY_CREATE, ENTRY_MODIFY,
                                ENTRY_DELETE);

            System.out.println("Watching for events on " + dirToWatch);

            // Keep watching for events on the dirToWatch
            while(true) {
                // Retrieve and remove the next available WatchKey
                WatchKey key = ws.take();

                for(WatchEvent<?> event : key.pollEvents()) {
                    Kind<?> eventKind = event.kind();

                    if (eventKind == OVERFLOW) {
                        System.out.println("Event overflow" +
                                           " occurred");
                        continue;
                    }

                    // Get the context of the event, which is the
                    // directory entry on which the event occurred.
                    WatchEvent<Path> currEvent =
                                        (WatchEvent<Path>)event;
                    Path dirEntry = currEvent.context();

                    // Print the event details
                    System.out.println(eventKind +
                                        " occurred on " + dirEntry);
                }

                // Reset the key
                boolean isKeyValid = key.reset();
                if (!isKeyValid) {
                    System.out.println("No longer watching " +
                                        dirToWatch);
                    break;
                }
            }
        }
        catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Output: (You may get a different output.)

```
Watching for events on C:\kishori
ENTRY_CREATE occurred on test.txt
ENTRY_DELETE occurred on test.txt
```

Useful Links

Author's website	www.jdojo.com
Contacting the Author	ksharan@jdojo.com
Author's All Blogs	http://jdojo.com/feed/
Subscribing to Author's Blogs (RSS Feed)	http://jdojo.com/feed/
Downloading Sample Chapters of Harnessing Java 7 Book	http://jdojo.com/sample-chapters/
Purchasing Author's Books	http://www.jdojo.com/purchase