链滴

Java 如何实现文件变动的监听

作者：someone10186

原文链接：https://ld246.com/article/1518277604651

来源网站：链滴

Java可以如何实现文件变动的监听

应用中使用logback作为日志输出组件的话，大部分会去配置 logback.xml 这个文件，而且生产环境，直接去修改logback.xml文件中的日志级别，不用重启应用就可以生效

那么，这个功能是怎么实现的呢?

I. 问题描述及分析

针对上面的这个问题，首先抛出一个实际的case，在我的个人网站中，所有的小工具都是通过配置文来动态新增和隐藏的，因为只有一台服务器，所以配置文件就简化的直接放在了服务器的某个目录下

现在的问题时，我需要在这个文件的内容发生变动时，应用可以感知这种变动，并重新加载文件内容更新应用内部缓存

一个最容易想到的方法，就是轮询，判断文件是否发生修改，如果修改了，则重新加载，并刷新内存所以主要需要关心的问题如下:

- 如何轮询?
- 如何判断文件是否修改?
- 配置异常，会不会导致服务不可用? （即容错，这个与本次主题关联不大，但又比较重要...）

II. 设计与实现

问题抽象出来之后，对应的解决方案就比较清晰了

- 如何轮询？ --》 定时器 Timer, ScheduledExecutorService 都可以实现
- 如何判断文件修改? --》根据 java.io.File#lastModified 获取文件的上次修改时间，比对即可

那么一个很简单的实现就比较容易了:

```
public class FileUpTest {

    private long lastTime;

    @Test
    public void testFileUpdate() {
        File file = new File("/tmp/alarmConfig");

        // 首先文件的最近一次修改时间戳
        lastTime = file.lastModified();

        // 定时任务，每秒来判断一下文件是否发生变动，即判断lastModified是否改变
        ScheduledExecutorService scheduledExecutorService = Executors.newScheduledThreadP
ol(1);
        scheduledExecutorService.scheduleAtFixedRate(new Runnable() {
            @Override
            public void run() {
                if (file.lastModified() > lastTime) {
                    System.out.println("file update! time : " + file.lastModified());
```

```
                lastTime = file.lastModified();
            }
        }
    },0, 1, TimeUnit.SECONDS);

    try {
        Thread.sleep(1000 * 60);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
  }
}
```

上面这个属于一个非常简单，非常基础的实现了，基本上也可以满足我们的需求，那么这个实现有什问题呢?

定时任务的执行中，如果出现了异常会怎样?

对上面的代码稍作修改

```
public class FileUpTest {

    private long lastTime;

    private void ttt() {
        throw new NullPointerException();
    }

    @Test
    public void testFileUpdate() {
        File file = new File("/tmp/alarmConfig");

        lastTime = file.lastModified();

        ScheduledExecutorService scheduledExecutorService = Executors.newScheduledThreadP
ol(1);
        scheduledExecutorService.scheduleAtFixedRate(new Runnable() {
            @Override
            public void run() {
                if (file.lastModified() > lastTime) {
                    System.out.println("file update! time : " + file.lastModified());
                    lastTime = file.lastModified();
                    ttt();
                }
            }
        }, 0, 1, TimeUnit.SECONDS);

        try {
            Thread.sleep(1000 * 60 * 10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

```
}
```

实际测试，发现只有首次修改的时候，触发了上面的代码，但是再次修改则没有效果了，即当抛出异之后，定时任务将不再继续执行了，这个问题的主要原因是因为 ScheduledExecutorService 的原因了

直接查看ScheduledExecutorService的源码注释说明

If any execution of the task encounters an exception, subsequent executions are suppressed. therwise, the task will only terminate via cancellation or termination of the executor. 即如果时任务执行过程中遇到发生异常，则后面的任务将不再执行。

所以，使用这种姿势的时候，得确保自己的任务不会抛出异常，否则后面就没法玩了

对应的解决方法也比较简单，整个catch一下就好

III. 进阶版

前面是一个基础的实现版本了，当然在java圈，基本上很多常见的需求，都是可以找到对应的开源工来使用的，当然这个也不例外，而且应该还是大家比较属性的apache系列

首先maven依赖

```
<dependency>
   <groupId>commons-io</groupId>
   <artifactId>commons-io</artifactId>
   <version>2.6</version>
</dependency>
```

主要是借助这个工具中的 FileAlterationObserver, FileAlterationListener, FileAlterationMonitor 个类来实现相关的需求场景了，当然使用也算是很简单了，以至于都不太清楚可以再怎么去说明了，接看下面从我的一个开源项目quick-alarm中拷贝出来的代码

```
public class PropertiesConfListenerHelper {

   public static boolean registerConfChangeListener(File file, Function<File, Map<String, Alar
Config>> func) {
      try {
         // 轮询间隔 5 秒
         long interval = TimeUnit.SECONDS.toMillis(5);

         // 因为监听是以目录为单位进行的，所以这里直接获取文件的根目录
         File dir = file.getParentFile();

         // 创建一个文件观察器用于过滤
         FileAlterationObserver observer = new FileAlterationObserver(dir,
               FileFilterUtils.and(FileFilterUtils.fileFileFilter(),
                     FileFilterUtils.nameFileFilter(file.getName())));

         //设置文件变化监听器
         observer.addListener(new MyFileListener(func));
         FileAlterationMonitor monitor = new FileAlterationMonitor(interval, observer);
         monitor.start();
```

```
            return true;
        } catch (Exception e) {
            log.error("register properties change listener error! e:{}", e);
            return false;
        }
    }

    static final class MyFileListener extends FileAlterationListenerAdaptor {

        private Function<File, Map<String, AlarmConfig>> func;

        public MyFileListener(Function<File, Map<String, AlarmConfig>> func) {
            this.func = func;
        }

        @Override
        public void onFileChange(File file) {
            Map<String, AlarmConfig> ans = func.apply(file); // 如果加载失败，打印一条日志
            log.warn("PropertiesConfig changed! reload ans: {}", ans);
        }
    }
}
```

针对上面的实现，简单说明几点：

- 这个文件监听，是以目录为根源，然后可以设置过滤器，来实现对应文件变动的监听
- 如上面 registerConfChangeListener方法，传入的file是具体的配置文件，因此构建参数的时候捞出了目录，捞出了文件名作为过滤
- 第二参数是jdk8语法，其中为具体的读取配置文件内容，并映射为对应的实体对象

一个问题，如果 func方法执行时，也抛出了异常，会怎样？

实际测试表现结果和上面一样，抛出异常之后，依然跪，所以依然得注意，不要跑异常

那么简单来看一下上面的实现逻辑，直接扣出核心模块

```
public void run() {
    while(true) {
        if(this.running) {
            Iterator var1 = this.observers.iterator();

            while(var1.hasNext()) {
                FileAlterationObserver observer = (FileAlterationObserver)var1.next();
                observer.checkAndNotify();
            }

            if(this.running) {
                try {
                    Thread.sleep(this.interval);
                } catch (InterruptedException var3) {
                    ;
```

```
            }
            continue;
        }
    }

    return;
  }
}
```

从上面基本上一目了然，整个的实现逻辑了，和我们的第一种定时任务的方法不太一样，这儿直接使线程，死循环，内部采用sleep的方式来来暂停，因此出现异常时，相当于直接抛出去了，这个线程跪了

JDK版本

jdk1.7，提供了一个WatchService，也可以用来实现文件变动的监听，之前也没有接触过，看到说明然后搜了一下使用相关，发现也挺简单的，同样给出一个简单的示例demo

```
@Test
public void testFileUpWather() throws IOException {
  // 说明，这里的监听也必须是目录
  Path path = Paths.get("/tmp");
  WatchService watcher = FileSystems.getDefault().newWatchService();
  path.register(watcher, ENTRY_MODIFY);

  new Thread(() -> {
    try {
      while (true) {
        WatchKey key = watcher.take();
        for (WatchEvent<?> event : key.pollEvents()) {
          if (event.kind() == OVERFLOW) {
            //事件可能lost or discarded
            continue;
          }
          Path fileName = (Path) event.context();
          System.out.println("文件更新: " + fileName);
        }
        if (!key.reset()) { // 重设WatchKey
          break;
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }).start();

  try {
    Thread.sleep(1000 * 60 * 10);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
```

```
}
```

IV. 小结

使用Java来实现配置文件变动的监听，主要涉及到的就是两个点

- 如何轮询： 定时器（Timer, ScheduledExecutorService），线程死循环+sleep
- 文件修改： File#lastModified

整体来说，这个实现还是比较简单的，无论是自定义实现，还是依赖 commos-io来做，都没太大的术成本，但是需要注意的一点是：

- 千万不要在定时任务 or 文件变动的回调方法中抛出异常！！！

为了避免上面这个情况，一个可以做的实现是借助EventBus的异步消息通知来实现，当文件变动之后发送一个消息即可，然后在具体的重新加载文件内容的方法上，添加一个 @Subscribe注解即可，这既实现了解耦，也避免了异常导致的服务异常 （如果对这个实现有兴趣的可以评论说明）

V. 其他

参考项目

- 项目： quick-alarm
- 测试类： FileUpTest.java

声明

尽信书则不如，已上内容，纯属一家之言，因本人能力一般，见解不全，如有问题，欢迎批评指正

作者：一灰灰

链接：https://juejin.im/post/5a7d08855188257a61322b18

来源：掘金