WatchService实现对文件的监控

WatchService

1) 该类的对象就是操作系统原生的文件系统监控器！OS自己的文件系统监控器可以监控系统上所有文件的变化，这种监控是无需遍历、无需比较的，是一种基于信号收发的监控，因此效率一定是最高的；Java对其进行了包装，可以直接在Java程序中使用OS的文件系统监控器；

2) first获取当前OS平台下的文件系统监控器

 i. WatchService watcher = FileSystems.getDefault().newWatchService();

   ii. 从FileSystems这个类名可以看出这是属于OS平台文件系统的，这样可以直接可以得到一个文件监控器；

 3) 我们知道，操作系统上可以同时开启多个监控器，因此在Java程序中也不例外，上面的代码只是获得了一个监控器，我们可以用同样的代码同时获得多个监控器；

 4) 监控器就是一个后台线程，在后台监控文件变化所发出的信号，这里通过上述代码获得的监控器还只是一个刚刚初始化的线程，连就绪状态都没有进入，只是初始化而已；

3. second将监控器注册给指定的文件节点——开启监控线程来监控指定的节点：

    1) 首先需要有个监控目标，看我们需要指定监控哪个文件节点；

    2) 其次就是文件变化的种类，有创建、删除、修改等多种类型的文件变化；

    3) Path对象的register方法搞定一切：

         i. 原型：WatchKey Path.register(WatchService watcher, WatchEvent.Kind<?>... events);

         ii. 将指定的监控器注册给Path对象所代表的文件节点，而events则指定了监控器监控哪些类型的文件变化；

         iii. 该方法同时也会让监控器线程就绪并运行，该方法调用完后监控器就彻底开始监控了！

    4) WatchEvent.Kind<?> events：

         i. 一看到Kind就知道该类一定是一个枚举类，里面定义了文件变化的各种枚举类型；

         ii. 这些枚举变量都定义在StandardWatchEventKinds中，主要有三个：

ENTRY\_CREATE：创建

ENTRY\_DELETE：删除

ENTRY\_MODIFY：修改

    5) WatchKey：

         i. 监控键，就是该文件节点所绑定的监控器的监控信息池，即文件节点的监控池，简称监控池；

         ii. 所有监控到的信息都会放到监控池中；

         iii. register方法返回的就是节点的监控池；

    6) 监控池是静态的！！！

         i. 监控池只能表示某一个时间节点下的文件变化信息，并不能动态保存这些信息；像register方法，刚注册完是返回的监控池是一个空的监控池，因为刚刚开启线程，什么都还没有发生，即使后面发生了文件修改，那么该监控池对象的内容还是保持不变，仍然是空的；

Java要求，只有当你主动去获取新的监控池时才会将更新的内容放入获取到的监控池中！！这是因为应用程序往往有这样的需求，就是当文件发生变化的这个时间节点上进行一定的处理，那何不就刚好在这个时间点上去更新监控池呢？

当然Java也完全可以做到实时更新监控池，这样的话register返回的那个监控池就会实时变化，可以一直用到底，但是你要想，WatchService其实是操作系统的线程，而监控池是Java程序内的内存空间，要将文件变化实时反映出来那就需要Java进程和操作系统的监控进程实时交流才行，这样岂不是在进程间通信的时间会花费很多，那还是干脆只有在文件发生变化的时间点上程序通过主动获取监控池来将监控信息传入Java程序来的省时省力；

         ii. 获取下一个监控信息：其实就是获取新的监控池

             a. WatchKey WatchService.poll(); // 尝试获取下一个变化信息的监控池，如果没有变化则返回null

             b. WatchKey WatchService.take(); // // 尝试获取下一个变化信息的监控池，如果没有变化则一直等待

！！这两个方法都是WatchService的对象方法，从这点可以看出，获取更新的监控信息其实需要当前Java进程和操作系统进程之间进行交流，返回的WatchKey是当前Java程序的（内存空间位于当前Java程序中）；

         iii. 如果需要长时间一直监控要用take，而如果只是在某个指定的时间监控则用poll；

4. third获得WatchKey（监控池）中的具体监控信息：

    1) 一个文件变化动作可能会引发一系列的事件，因此WatchKey中保存着一个事件列表List<WatchEvent<?>> list，可以通过WatchKey的pollEvents方法获得该列表：

        i. 原型：List<WatchEvent<?>> WatchKey.pollEvents();

        ii. 获得列表后既可以通过for循环迭代遍历该变化动作所引发的所有具体事件（通常来说，大多数变化动作只会引发一个事件）；

    2) WatchEvent<?>：

        i. 表示监控时间对象，里面最常用的两个方法就是context和kind；

        ii. Path WatchEvent<?>.context(); // 返回触发该事件的那个文件或目录的路径

        iii. Kind<StandardWatchEventKinds> WatchEvent<?>kind(); // 返回事件类型（ENTRY\_CREATE、ENTRY\_DELETE、ENTRY\_MODIFY之一）；

5. forth完成一次监控就需要重置监控器一次：

    1) 直接调用WatchKey的reset方法就代表重置其关联的监控器了；

    2) 原型：boolean WatchKey.reset();

    3) 因为当你使用poll或take时监控器线程就被阻塞了，因为你处理文件变化的操作可能需要挺长时间的，为了防止在这段时间内又要处理其他类似的事件，因此需要阻塞监控器线程，而调用reset表示重启该线程；

App key简称API[接口](http://www.so.com/s?q=%E6%8E%A5%E5%8F%A3&ie=utf-8&src=internal_wenda_recommend_textn" \t "https://wenda.so.com/q/_blank)验证序号，是用于验证API接入合法性的。接入哪个网站的API接口，就需要这个网站允许才能够接入，如果简单比喻的话：可以理解成是登陆网站的[用户名](http://www.so.com/s?q=%E7%94%A8%E6%88%B7%E5%90%8D&ie=utf-8&src=internal_wenda_recommend_textn" \t "https://wenda.so.com/q/_blank)