Java 7中的Try-with-resources

原文链接 作者：Jakob Jenkov 译者：fangqiang08(fangqiang08@gmail.com)

Try-with-resources是java7中一个新的异常处理机制，它能够很容易地关闭在try-catch语句块中使用的资源。

利用Try-Catch-Finally管理资源（旧的代码风格）

在java7以前，程序中使用的资源需要被明确地关闭，这个体验有点繁琐。

下面的方法读取文件，然后用System.out打印：

```
01 private static void printFile() throws IOException {
02     InputStream input = null;
03
04     try {
05         input = <strong>new FileInputStream("file.txt")</strong>;
06
07         int data = <strong>input.read()</strong>;
08         while(data != -1){
09             System.out.print((char) data);
10             data = <strong>input.read()</strong>;
11         }
12     } finally {
13         if(input != null){
14             <strong>input.close()</strong>;
15         }
16     }
17 }
```

上面代码中黑体字的程序可能会抛出异常。正如你所看到的，try语句块中有3个地方能抛出异常，finally语句块中有一个地方会能出异常。

不论try语句块中是否有异常抛出，finally语句块始终会被执行。这意味着，不论try语句块中发生什么，InputStream 都会被关闭，或者说都会试图被关闭。如果关闭失败，InputStream’s close()方法也可能会抛出异常。

假设try语句块抛出一个异常，然后finally语句块被执行。同样假设finally语句块也抛出了一个异常。那么哪个异常会根据调用栈往外传播?

即使try语句块中抛出的异常与异常传播更相关，最终还是finally语句块中抛出的异常会根据调用栈向外传播。

在java7中，对于上面的例子可以用try-with-resource 结构这样写：

```
01 private static void printFileJava7() throws IOException {
02
03     try(FileInputStream input = new FileInputStream("file.txt")) {
04
05         int data = input.read();
06         while(data != -1){
07             System.out.print((char) data);
08             data = input.read();
09         }
10     }
11 }
```

注意方法中的第一行：

```
try(FileInputStream input = new FileInputStream("file.txt")) {
```

这就是try-with-resource 结构的用法。FileInputStream 类型变量就在try关键字后面的括号中声明。而且一个FileInputStream 类型被实例化并被赋给了这个变量。

当try语句块运行结束时，FileInputStream 会被自动关闭。这是因为FileInputStream 实现了java中的java.lang.AutoCloseable接口。所有实现了这个接口的类都可以在try-with-resources结构中使用。

当try-with-resources结构中抛出一个异常，同时FileInputStreami被关闭时（调用了其close方法）也抛出一个异常，try-with-resources结构中抛出的异常会向外传播，而FileInputStreami被关闭时抛出的异常被抑制了。这与文章开始处利用旧风格代码的例子（在finally语句块中关闭资源）相反。

使用多个资源

你可以在块中使用多个资源而且这些资源都能被自动地关闭。下面是例子：

```
private static void printFileJava7() throws IOException {

    try(  FileInputStream     input         = new FileInputStream("file.txt");
          BufferedInputStream bufferedInput = new BufferedInputStream(input)
    ) {

        int data = bufferedInput.read();
        while(data != -1){
            System.out.print((char) data);
    data = bufferedInput.read();
        }
    }
}
```

上面的例子在try关键字后的括号里创建了两个资源——FileInputStream 和BufferedInputStream。当程序运行离开try语句块时，这两个资源都会被自动关闭。

这些资源将按照他们被创建顺序的逆序来关闭。首先BufferedInputStream 会被关闭，然后FileInputStream会被关闭。

自定义AutoClosable 实现

这个try-with-resources结构里不仅能够操作java内置的类。你也可以在自己的类中实现java.lang.AutoCloseable接口，然后在try-with-resources结构里使用这个类。

AutoClosable 接口仅仅有一个方法，接口定义如下：

```
public interface AutoClosable {

    public void close() throws Exception;
}
```

任何实现了这个接口的方法都可以在try-with-resources结构中使用。下面是一个简单的例子：

```
01 public class MyAutoClosable implements AutoCloseable {
02
03     public void doIt() {
04         System.out.println("MyAutoClosable doing it!");
05     }
06
07     @Override
08     public void close() throws Exception {
09         System.out.println("MyAutoClosable closed!");
10     }
11 }
```

doIt()是方法不是AutoClosable 接口中的一部分，之所以实现这个方法是因为我们想要这个类除了关闭方法外还能做点其他事。

下面是MyAutoClosable 在try-with-resources结构中使用的例子：

```
1 private static void myAutoClosable() throws Exception {
2
3     try(MyAutoClosable myAutoClosable = new MyAutoClosable()){
4         myAutoClosable.doIt();
5     }
6 }
```

当方法myAutoClosable.doIt()被调用时，下面是打印到System.out的输出：

```
1 MyAutoClosable doing it!
2 MyAutoClosable closed!
```

通过上面这些你可以看到，不论try-catch中使用的资源是自己创造的还是java内置的类型，try-with-resources都是一个能够确保资源能被正确地关闭的强大方法。