Java7新特性 - AutoCloseable

授课老师：学相伴飞哥

01、课程大纲

01、概述

02、实战应用

02、概述

JDK在1.7之后出现了自动关闭类的功能，该功能的出现为各种关闭资源提供了相当大的帮助，这里我们谈一谈自动关闭类。
JDK1.7之后出现了一个重要的接口，以及改造了一个重要的方法结构：
1、AutoCloseable自动关闭接口
2、try(){}--catch{}--finally{}
相应的 一些资源也实现了该接口，如PreparedStatement、Connection、InputStream、OutputStream等等资源接口。

> 一句话： 实现*AutoCloseable*接口，覆盖*close*方法，把原来要写*finally*中释放资源的动作，放入到*close*方法中去执行，而这个执行是*jvm*自己去执行.
>
> 什么样子的情况下可以去做呢？--如果有*try(){}--catch{}--finally{}*

- 接口的实现类要重写close（）方法，
- 将要关闭的资源定义在try()中，这样当程序执行完毕之后，资源将会自动关闭。
- 自定义类如果要进行自动关闭，只需要实现AutoCloseable接口重写close（）方法即可，

同时也只有实现了AutoCloseable接口才能将，自定义类放入到try()块中，否则编译不能通过，举例说明

```
public class ReadTxt extends AutoClassable {
    @Override
    public void close() throws Exception {
        System.out.println("ReadTxt close");
    }

    public String readTextValue(String path)
{
        StringBuffer sb = new
StringBuffer();
        try(BufferedReader br = new
BufferedReader(new FileReader(path))){
            int line;
            while((line = br.read())!=-1){

sb.append(br.readLine()+"\n")
            }
        }
        return sb.toString();
    }
}

class MainTest {
    public static void main(String[] args) {
        try (ReadTxt rt = new ReadTxt()) {
            String line =
rt.readTextValue("G:\\学习文档\\test.txt");
            System.out.println(line);
        }
```

```
    }
}
```

03、案例分析

01、上锁为例

```
package com.kuangstudy.lock;

import java.util.concurrent.locks.Lock;
import
java.util.concurrent.locks.ReentrantLock;

public class MyLockDemo {
    static Lock lock = new ReentrantLock();

    public static void main(String[] args) {
        try {
            lock.lock();
            System.out.println("1-----加锁成
功!!!");
            System.out.println("2-----开始执
行业务逻辑");
            Thread.sleep(3000);
            System.out.println("3-----业务执
行完毕");
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            lock.unlock();
            System.out.println("4----释放锁资
源");
        }
    }
```

```
23
24 }
25
```

资源释放的问题？

通过上面的代码我们发现，在开发中锁是释放必须要做得事情，所以就把放在finally中来执行。但是在开发中往往很多的开发中，都会忘记释放锁或者忘记把锁的释放放入finally中，就造成死锁现象，这个很危险的操作和行为。

如何解决遗忘的问题呢？

- 使用AutoCloseable接口覆盖close方法

改进步骤

定义类MyLock类

```
1  package com.kuangstudy.lock;
2
3  import java.util.concurrent.locks.Lock;
4  import
   java.util.concurrent.locks.ReentrantLock;
5
6  public class MyLock implements
   AutoCloseable{
7
8      Lock lock = new ReentrantLock();
9
10     // 加锁
11     public void lock() {
12         lock.lock();
13     }
14
15     // 释放锁
```

```
    public void unlock() {
        lock.unlock();
    }


    @Override
    public void close() throws Exception {
        unlock();
        System.out.println("4----释放锁资源");
    }
}

```

改进

```
package com.kuangstudy.lock;

import java.util.concurrent.locks.Lock;
import
java.util.concurrent.locks.ReentrantLock;

public class MyLockDemo {
    public static void main(String[] args) {
        try (MyLock lock = new MyLock();){
            lock.lock();
            System.out.println("1-----加锁成
功!!!");
            System.out.println("2-----开始执
行业务逻辑");
            Thread.sleep(3000);
            System.out.println("3-----业务执
行完毕");
        } catch (Exception e) {
            e.printStackTrace();
        }//        finally {
//            // 不需要定义了.因为会自动去释放资源
//            lock.unlock();
//        }
```

```
20      }
21  }
22
```

```
1-----加锁成功!!!
2-----开始执行业务逻辑
3-----业务执行完毕
4----释放锁资源   -- 这里的执行就是AutoCloseable中的close方法自动执行的
```