# Var val 和 def的区别

Var和val的区别不用我多说，val定义常量，而var定义变量。这次主要讨论def和val之间的区别。

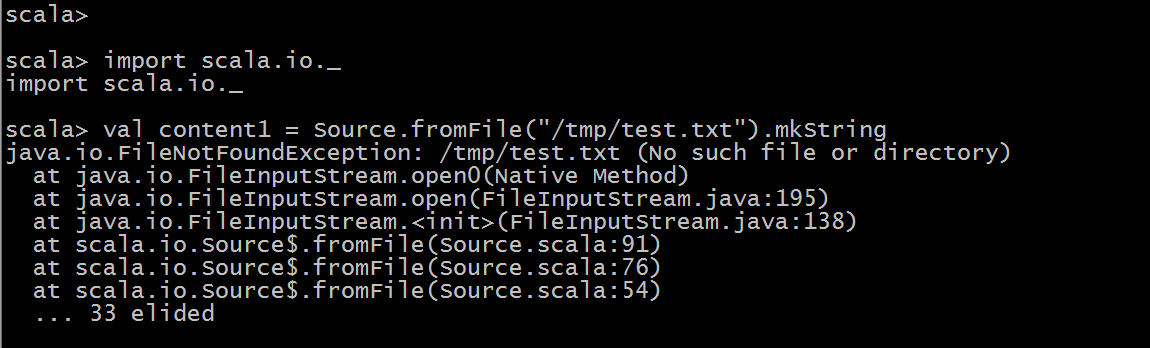
Val在定义变量时是call-by-value，我想这也就是为什么叫“val”的原因

首先在根目录下创建一个test.txt文件，文件内容“content1”，我这里是/tmp/test.txt，接下来我将会执行：

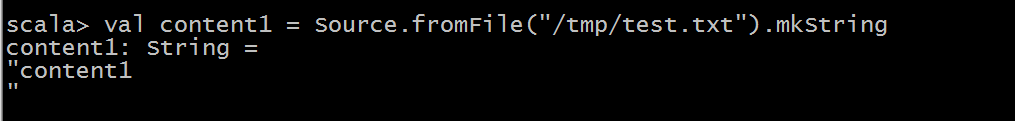
val content1 = Source.fromFile("/tmp/test.txt").mkString //将文件中的内容读出来，并生成一个字符串

看一下在REPL中的效果：

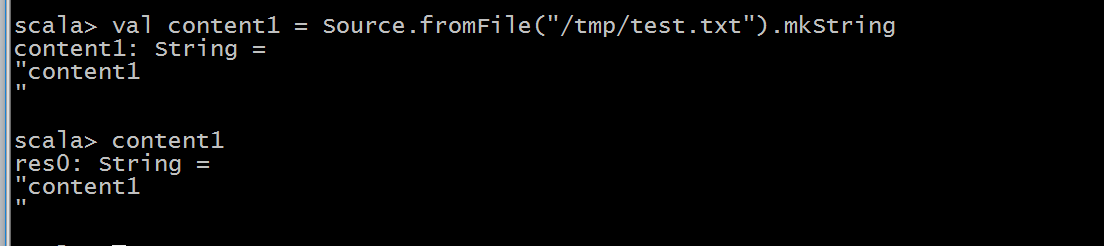
* 使用val时，如果/tmp/test.txt不存在



* 使用val时，/tmp/test.txt存在



* 接下来我修改test.txt的内容，改为“content2”再次使用content1的值

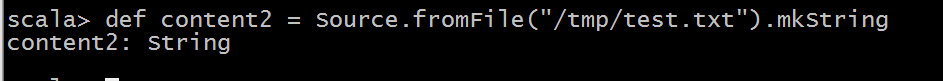


由此可以看出使用val会立即进行求值，如果出错也会立即报错，得到的是一个常量。

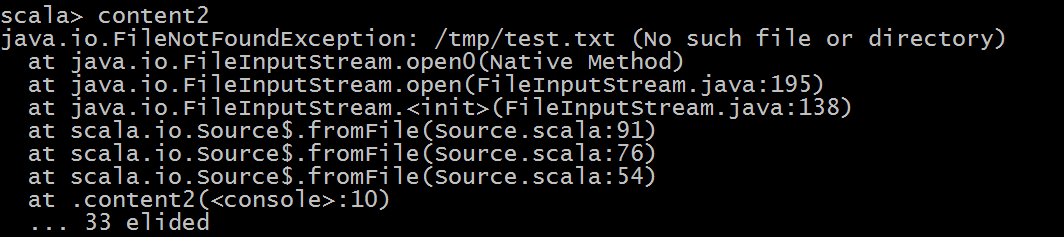
* def是call-by-name

下面我们讲 用def重新做以上操作

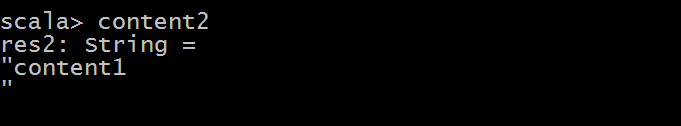
首先，当文件不存在时，可以正常执行：



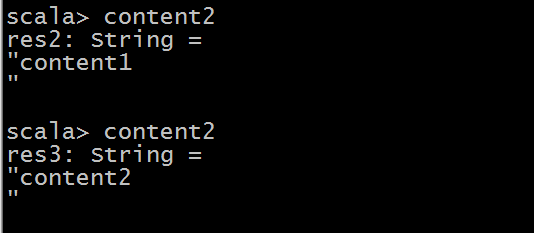
使用content2时：



当文件存在，且内容为“content1”时



当文件内容更改为“content2”时

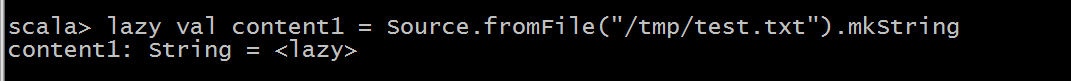


由此，可以看出def定义的是一个“方法”（这里我们暂且这么叫），会返回一个值（可以为空Unit），每次调用都会重新执行。

PS：你可以能会说def定义的就是一个无参的方法，方法当然每次使用都会重新计算（网上有这种说法），我认为可以这么理解。val和def的根本区别还是在于call-by-value和call-by-name，这是两种不同的定义方式。

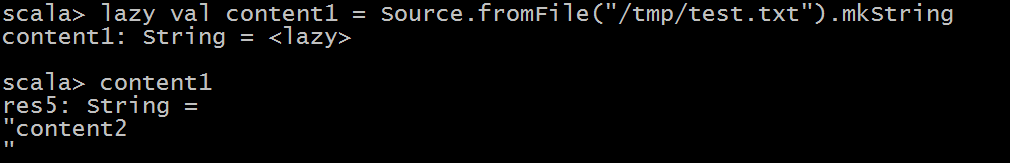
* lazy val

如果/tmp/test.txt文件不存在



此时并不会报错，只有在使用content1的时候才会报错

当/tmp/test.txt文件存在



此时改变test.txt的内容为“content3”，再次调用content1，输出的值不变。

Val 和lazy val 的不同之处：

Val会在定义的时候立即执行后面的语句，lazy val 会在调用的时候执行。有一点需要注意：如果使用 lazy val 定义变量，该变量并未使用，但此时文件内容改变，那么此后使用该变量，得到的是改变之后的文件内容。

添加lazy的内容。