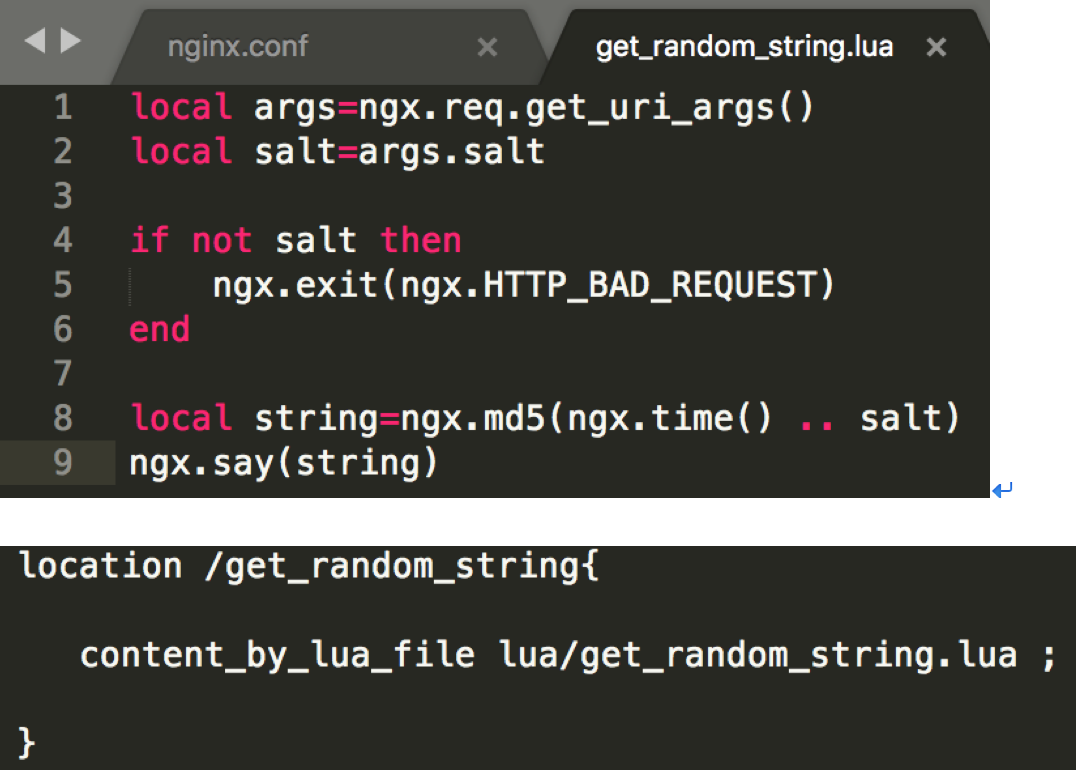
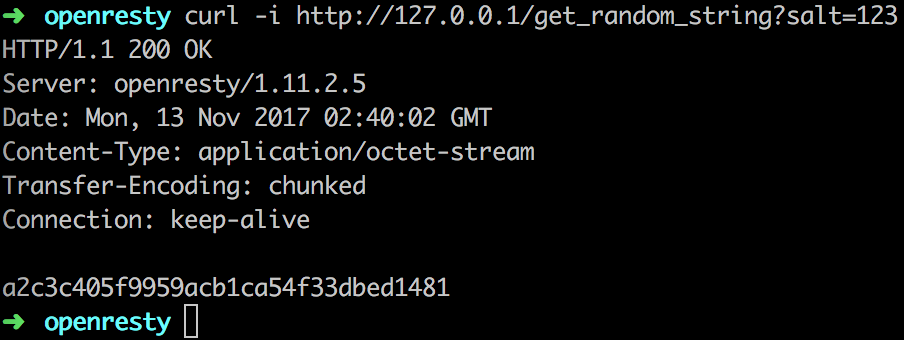
OpenResty学习系列（三）

OpenResty入门非常简单，我们上个系列的Hello World只用一行代码就实现了，并且压力测试她的qps会相当高。但是其入门虽然非常简单，却会有很多坑，她不像一般的开发语言。OpenResty结合了Nginx和LuaJIT，我们开发时不仅要熟悉Nginx、熟悉LuaJIT，还要熟悉Ngx Lua之间的API，她们之间有很多的细节，因此前期学习建议《OpenResty最佳实践》（参考系列一）。

本小节会用几个简单的Ngx Lua API来实现一个小功能。创建修改过程参考系列（二），我们先看下代码：



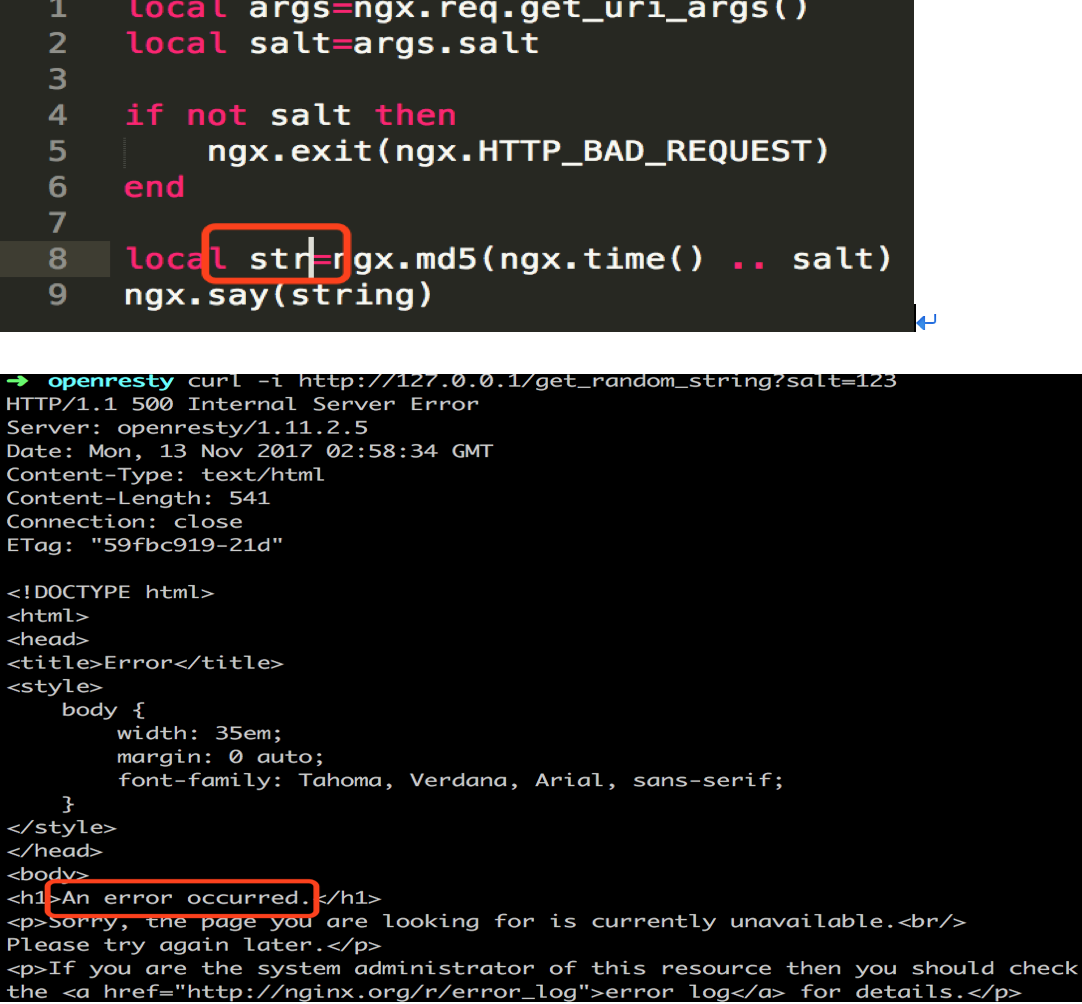
代码功能很简单：首先获取uri的参数，将其传给Lua的变量salt，然后做一个检测，如果终端没有传这个变量salt，我们会返回给他一个400错误，接着我们会获取系统的时间，在系统内部她是做了一个缓存的，然后将时间和客户端传上来的salt做一个字符串，这里的“..”表示字符串的加法，之后调用了Lua的md5API，将得到的md5值赋值给string，最后将string的值传递给客户端。整体流程很清晰。我们看下她的输出：



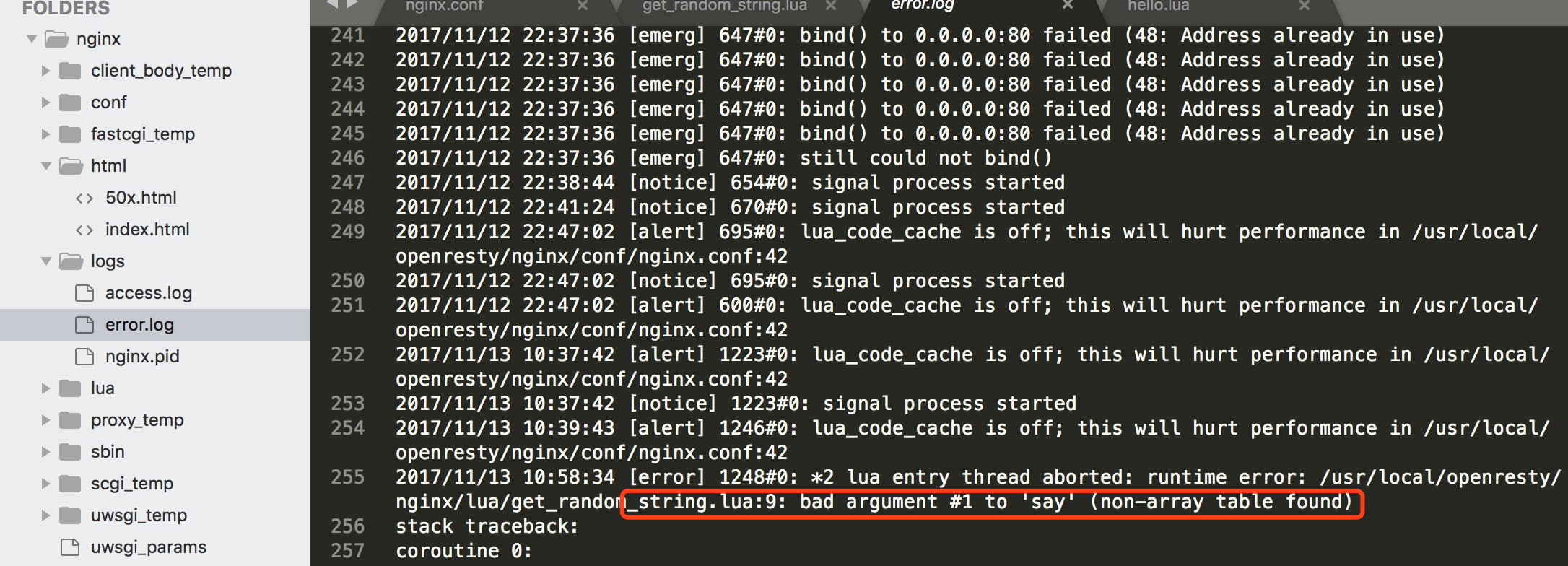
可以看到，我们向url做了一个简单的get请求，salt值是123，她的返回值是一个md5值。这里有几个地方需要大家注意就是在代码风格上，我们并没有返回一个400值，而是返回ngx里面的一个常量HTTP\_BAD\_REQUEST，我们可以在ngx官方文档上（地址参考系列一）看下她的定义：她会把HTTP的状态码都做一个常量的定义，所以我们在写Lua代码的时候，状态码会用这些常量来表示，这样代码看起来会显得特别的规范。



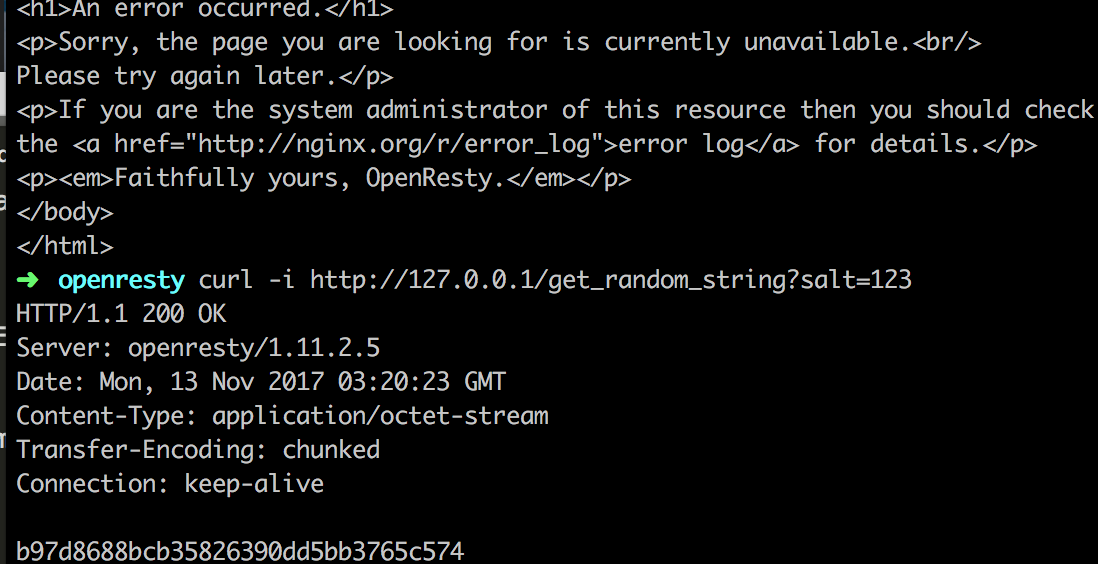
我们在写代码的时候，不可能一直都是对的，那出错的时候该怎么办呢？我们以我们的代码为例：我们将代码里的string改成str，重新执行。



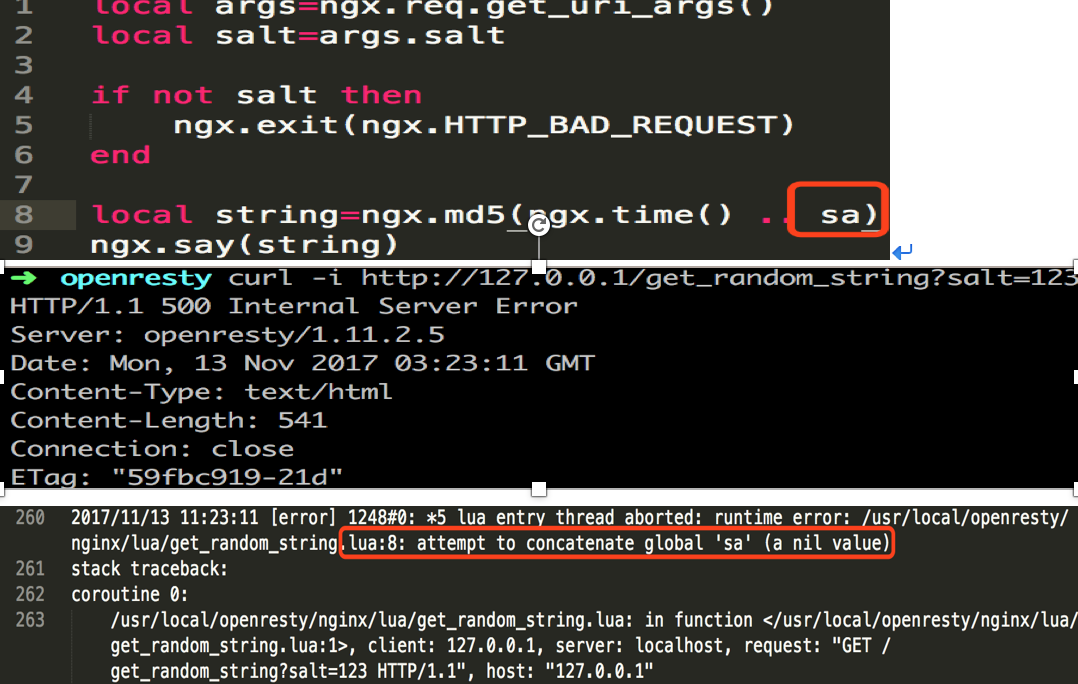
我们可以看到出现错误了，当OpenResty出现错误时，我们第一件要做的事就是看错误日志nginx/logs/error.log。错误很明显，她说ngx.say给了一个错误的参数，因为str这个参数我们没有定义。



我们将其改过来再次执行，结果恢复正常：



我们再次尝试修改，将salt改为没有定义的sa，看下结果：错误日志提示尝试去加一个空值sa，因为我们没有定义sa，在Lua里，未定义的变量默认为空值nil。



所以在出现错误的时候，我们第一时间应该做的就是去看错误日志，一般她都会给出比较明确的提示。

所以通过上面的例子我们可以看到她比较特殊的地方在于她需要调用自己的ngx lua API来实现自己的操作.

最后我们来简单介绍一下OpenResty的模块：她的模块并不是很丰富，差不多有十几个这样，常用模块如下图。这些模块涉及到如mysql数据库、redis、限流、模块渲染等常用功能组件；另外也有很多第三方的ngx\_lua组件供我们使用，对于大部分应用场景来说现在生态环境中的组件已经足够多了；如果不满足需求也可以自己去写来完成自己的需求。

