Flask 项目实战：实现一个精美的智能提升优先级的计算器

day 1~day 27 总结了 Python 基础部分和进阶部分所有核心知识点。

如果你充分利用这 27 天，并掌握了主要知识点。

那么接下来，你在学习基于 Python 开发的常见框架和包时，将会事半功倍，学习速度会更快。

今天我们将利用过往所学的知识，使用 Web 主流框架之一 Flask，进行项目实战，实现一个精美的 Web 版计算器。

Web 版计算器

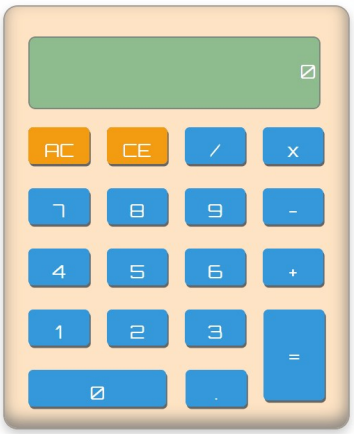
使用 Flask 作为后端，前端使用 Bootstrap 框架，语言 Python + html + css +javascript

计算器核心功能：具备括号自动补全功能，以此实现 `+-*/` 4 个操作优先级的对等性。

AC 键：清零屏幕

CE 键：删除屏幕上的最后一位字符

计算器操作的演示动画，如下所示：



代码目录结构

以下是框架目录结构图：

复制

```
|-- flask-calculator
    |-- manage.py
    |-- app
        |-- __init__.py
        |-- calc
        |   |-- views.py
        |   |-- __init__.py
        |-- static
        |   |-- css
        |   |   |-- bootstrap.min.css
        |   |   |-- orbitron.css
        |   |   |-- style.css
        |   |   |-- yMJMMIlzdpvBhQQL_SC3X9yhF25-T1nyGy6BoWgz.woff2
        |   |-- js
        |       |-- jquery-3.1.1.min.js
        |       |-- main.js
        |-- templates
            |-- index.html
```

`manage.py` ： app 启动相关模块；

app 的 `__init__.py` ：app 启动默认加载的文件，完成创建 app ，加载 bootstrap，跨站请求伪造保护功能等

`views.py`：app 的路由处理模块

calc 的 `__init__.py` ，完成实例化一个 Blueprint 类对象，创建蓝本

css：前端样式相关的处理，引用 bootstrap 框架

orbitron.css 和 `yMJMMIlzdpvBhQQL_SC3X9yhF25-T1nyGy6BoWgz.woff2` ：与字体相关

style.css ：自定义的 css 样式文件

jquery.js：引用库，与 html dom 操作相关的最常用的 js 库

main.js：自定义的前端计算器处理逻辑

index.html：自定义的前端 html 文件

界面设计

Flask 使用的前端模板引擎为 `jinja2` ，介绍此项目主要涉及的 `jinja2` 模板语法。

1) 导入基本的模板文件

复制

```
{% extends "bootstrap/base.html" %}
{% import "bootstrap/wtf.html" as wtf %}
```

2) 块标题

复制

```
{% block title %}Block Title{%endblock %}
```

`{% %}` 是 `jinja2` 的一种控制结构

紧接着， `index.html` 文件的 head 节点结构，如下所示：

复制

```
{% block head %}
<head>
    <title>Flask Calculator</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
 />
    <link href="../static/css/bootstrap.min.css" rel="stylesheet">
    <link href="../static/css/style.css" rel="stylesheet">
    <link href='../static/css/orbitron.css' rel='stylesheet' type='text/c
ss'>
    <!-- <link href='https://fonts.googleapis.com/css?family=Orbitron' re
l='stylesheet' type='text/css'> -->
</head>
{% endblock %}
```

与不带模板引擎的 普通 html 文件相比，所写的 `head` 部，只多出一对 `jinja2` 的控制结果：

复制

```
{% block head %}
......
{% endblock %}
```

注意，这对控制结构是必须要添加在 html 文件中的。

接下来，是 index.html 的 body 部分。

body 部分的 html 逻辑：

`<div class="screen">` 是计算器屏幕块

`<div class="container-fluid">` 是计算器按钮布局逻辑

最后 2个 js 脚本一个是引用的 jquery 脚本，另一个是自定义的 main.js 脚本

复制

```
{% block content %}
<body>
    <div class="box">
        <div class="screen">
            <div class="main-screen" id="output">0</div>
        </div>

        <div class="container-fluid">
            <div class="buttons">
                <button class="btn-clear btn btn-warning" id="clearButton"
>AC</button>
                <button class="btn btn-warning" id="deleteButton">CE</but
ton>
                <button class="btn-operate btn btn-info" value="/">/</but
ton>
                <button class="btn-operate btn btn-info" value="*">x</but
ton>
            </div>

            <div class="buttons">
                <button class="nums btn btn-info" value="7">7</button>
                <button class="nums btn btn-info" value="8">8</button>
                <button class="nums btn btn-info" value="9">9</button>
                <button class="btn-operate btn btn-info" value="-">-</but
ton>
            </div>

            <div class="buttons">
                <button class="nums btn btn-info" value="4">4</button>
                <button class="nums btn btn-info" value="5">5</button>
                <button class="nums btn btn-info" value="6">6</button>
                <button class="btn-operate btn btn-info" value="+">+</but
ton>
            </div>

            <div class="buttons">
                <button class="nums btn btn-info" value="1">1</button>
                <button class="nums btn btn-info" value="2">2</button>
                <button class="nums btn btn-info" value="3">3</button>
                <button class="btn-equal btn btn-info" id="resultButton">=
</button>
            </div>

            <div class="buttons">
                <button class="nums btn-zero btn btn-info" value="0">0</b
utton>
                <button class="nums btn btn-info" value=".">.</button>
            </div>
        </div>
    </div>
    <br>

    <script src="../static/js/jquery-3.1.1.min.js"></script>
    <script src="../static/js/main.js"></script>
</body>
{% endblock %}
```

有了 html 计算器的设计页面后，先不急于编写前端相关的 js 脚本逻辑。接下来写 Flask 后端逻辑。

Flask 后端逻辑

Step 1

在 calc 文件夹下，创建 `__init__.py` 文件，主要完成：

1) 创建一个蓝本对象

2) 导入 html 界面对应的路由处理函数：`views.py`

注意 `from . import views` 这行不能放在这个文件的开头

复制

```
from flask import Blueprint

# 创建一个 Blueprint 类对象
print('__name__')
calc = Blueprint('calc', __name__)

from . import views
```

Step 2

在 app 文件夹下，新建 `__init__.py` 文件，作为 app 启动的默认加载项。

导入 `Flask`，`Bootstrap`，`CORS` 三个类

分别创建 `CORS` ，`Bootstrap()` 对象，并加载到 app 中

注册蓝本对象 `calc` 到 app 中

复制

```
from flask import Flask
from flask_bootstrap import Bootstrap
from flask_cors import CORS

cors = CORS()  # 跨站请求伪造保护
bootstrap = Bootstrap()  # 引入著名的CSS前端框架


def create_app():
    app = Flask(__name__)

    bootstrap.init_app(app)
    cors.init_app(app, supports_credentials=True)

    from .calc import calc
    app.register_blueprint(calc)

    return app
```

Step 3

创建 `views.py` ，编写界面对应的路由处理函数。

复制

```
from . import calc
from flask import render_template, request
import re


@calc.route('/', methods=['GET'])
def index():
    return render_template('index.html')


# 返回计算结果的API
@calc.route('/api/getresult', methods=['POST'])
def get_calc_result():
    data = request.get_json()
    expr_val = data['expr']
    return str(eval(expr_val))
```

共有 2 个路由处理函数，分别处理：

1) URL：`/` ，对应处理函数 `index` ，`GET` 请求

2) URL：`/api/getresult` ，处理 `XMLHttpRequest` 的 `POST` 请求。

main.js

计算器按钮显示处理逻辑 写在 `main.js` 文件中，如下，分别处理：

1) `#clearButton` ：计算器上 `AE` 按钮，点击事件处理

2) `#deleteButton` ：计算器上 `CE` 按钮，点击事件处理

3) `.nums` ：`0-9 .` 按键点击事件处理

4) `. btn-operate` ：`+-*/` 按键点击事件处理

5) `#resultButton` ：`=` 键的点击事件处理

复制

```
$(document).ready(function() {
    var $mainOutput = $('#output');
    var op = ''
    var num1 = ''

    var clearData = function() {
        op = ''
        num1 = ''
    };

    var clearOutput = function() {
        $mainOutput.html('');
    };

    $('.nums').click(function() {
        num2 = $(this).val()
        content = $mainOutput.html()
        if(num1 == '' && num2 == '.') return;
        if(num2 == '.' && ('+-*/.').indexOf(content[content.length-1]) !=
-1) return; //不能连续输入小数点
        if(num1 == '') {
            $mainOutput.html('');
            $mainOutput.append(num2);
        }
        else{
            $mainOutput.append(num2);
        }
        num1 = num2
    });

    $('#clearButton').click(function() {
        $mainOutput.html('0');
        clearData();
    });

    $('#deleteButton').click(function() {
        input = $mainOutput.html()
        if (input != '0') {
            input = input.substring(0, input.length-1)
            if (input[input.length-1] == ')'){
                input = input.substring(1,input.length-1)
            }
            $mainOutput.html(input);
            if (input == '') {
                clearData();
                $mainOutput.html('0');
            }
        }
    });
```

```
    $('.btn-operate').click(function() { //+-*/
    var newOp = $(this).val();
    if(num1 == '') return;
    content = $mainOutput.html()
    if(('+-*/').indexOf(content[content.length-1]) != -1) return; //不能连续输入操作符

    if((op=='+' || op=='-') && (newOp=='*' || newOp=='/')){
        newStr = '(' + content + ')' + newOp //添加一对括号
    }
    else{
        newStr = content + newOp
    }
    $mainOutput.html(newStr)
    op = newOp
    });


    $('#resultButton').click(function() {
       content = $mainOutput.html()
       if (content == '') return ;
       if ('+-*/'.indexOf(content[content.length-1]) != -1) return;
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange=function() {
            if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
                var result = xmlhttp.responseText;
                $mainOutput.html(result);
                num1 = result
                op = ''
            }
        }
        xmlhttp.open("POST","api/getresult")
        xmlhttp.setRequestHeader('content-type', 'application/json');
        xmlhttp.send(JSON.stringify({'expr':$mainOutput.html()}));
    });
});
```

请求和响应

本实战项目，一个核心问题就是前、后端数据的交互问题。下面重点说说整个过程。

1) 按下计算器的 `=` 按钮

2) `XMLHttpRequest` 对象发送 post 请求

使用 `xmlhttp` 对象的 `send` 函数发送数据，数据封装到字典 `{'expr':$mainOutput.html()}` 中，然后 `JSON` 序列化后发送到后端。

复制

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.open("POST","api/getresult")
xmlhttp.setRequestHeader('content-type', 'application/json');
xmlhttp.send(JSON.stringify({'expr':$mainOutput.html()}));
```

3) 经过 URL ： `/api/getresult` 到 Flask 后端 ，API `get_calc_result` 处理请求。

复制

```
# 返回计算结果的API
@calc.route('/api/getresult', methods=['POST'])
def get_calc_result():
    data = request.get_json()
    expr_val = data['expr']
    return str(eval(expr_val))
```

`request` 对象的 `get_json` 方法获取到前端发送的数据，是一个字典，拿到 `expr_val` 后，经过 Python 的内置函数 `eval` 计算表达式的值，并 `str()` 后，返回结果到前端。

4) 响应结果到前端

main.js 根据 `readyState` 和 `status` 取值，判断是否请求并响应成功，然后通过 `xmlhttp.responseText` 得到返回结果，并且赋值到 dom 元素 `mainOutput` 上 ，也就是计算器的屏幕中。

复制

```
xmlhttp.onreadystatechange=function() {
            if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
                var result = xmlhttp.responseText;
                $mainOutput.html(result);
                num1 = result
                op = ''
            }
        }
```

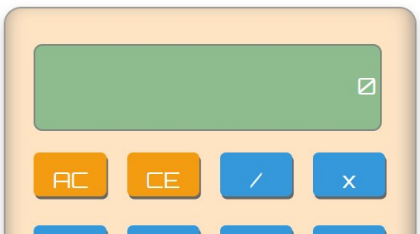
至此就完成整个前后端的数据交互逻辑。

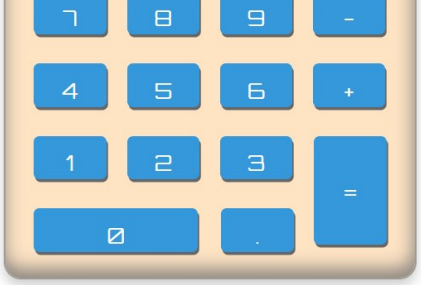
调试代码

Step 1 ：启动 `manage.py` 文件，如下表示，后端服务启动成功，并在 8080 端口监听，等待前端的调用。

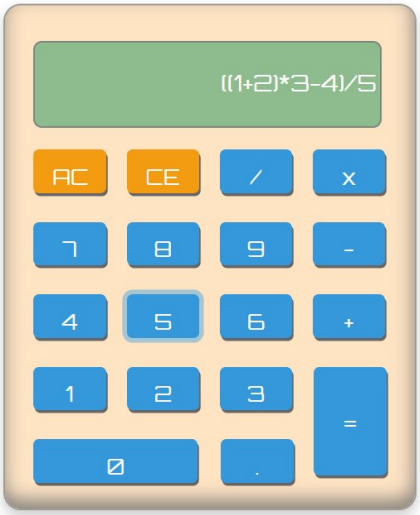
```
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
__name__
 * Debugger is active!
 * Debugger PIN: 663-788-611
 * Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
```

Step 2：打开浏览器，URL 栏输入： `127.0.0.1:8080` ，并回车，就能看到计算器的界面：

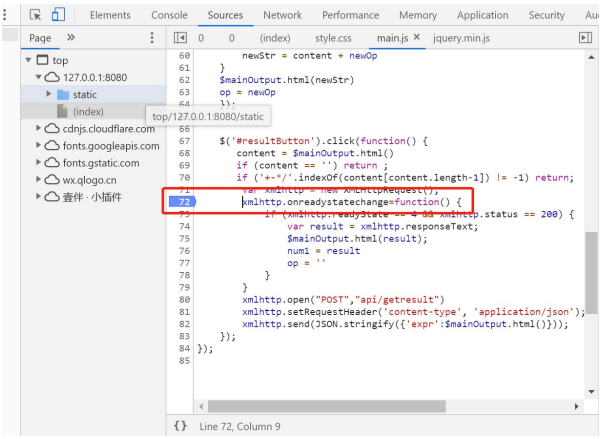




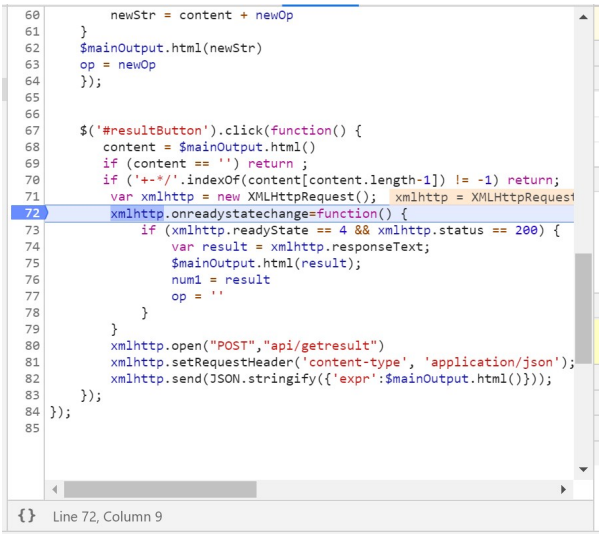
Step 3：依次输入 1, +, 2, * ,3,-,4,/,5，可看到屏幕中自动带出括号，并且实现了 + ， - 优先级与 * ，/ 的对等性。



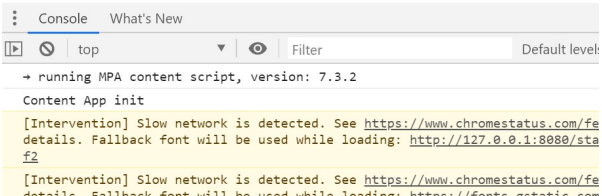
Step 4：点击 F12，看到前端的调试界面，并在 72 行，打一个断点，如下所示：



Step 5：点击 计算器的 = ，命中断点，并按行调试，逐次查看发送的数据结构。



Step 6：在浏览器的 console 中，查看要发送的数据，经过序列后的数据：

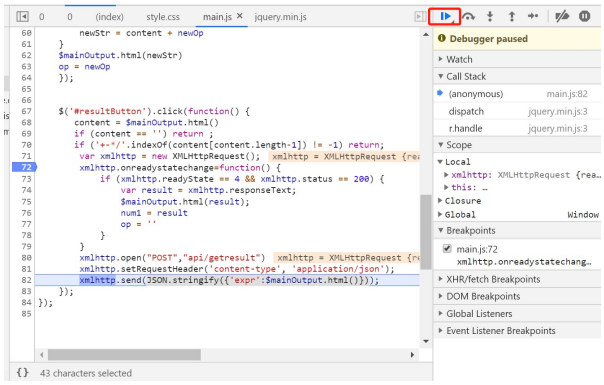


```
details. Fallback font will be used while loading. https://fonts.gstatic.com
> JSON.stringify({'expr':$mainOutput.html()})
< "{"expr":"((1+2)*3-4)/5"}"
>
```

Step 7 ：在后端 `get_calc_result` 函数中，打一个断点，如下：

```
10
11  #  返回计算结果的API
12  @calc.route('/api/getresult',  methods=['POST'])
13  def  get_calc_result():
14          data  =  request.get_json()
15          expr_val  =  data['expr']
16          return  str(eval(expr_val))
17
18
```

Step 8：在浏览器中，点击右上角的继续执行按钮，



Step 9：命中后端的断点：

```
11  #  返回计算结果的API
12  @calc.route('/api/getresult',  methods=['POST'])
13  def  get_calc_result():
14          data  =  request.get_json()
15          expr_val  =  data['expr']
16          return  str(eval(expr_val))
17
18
```

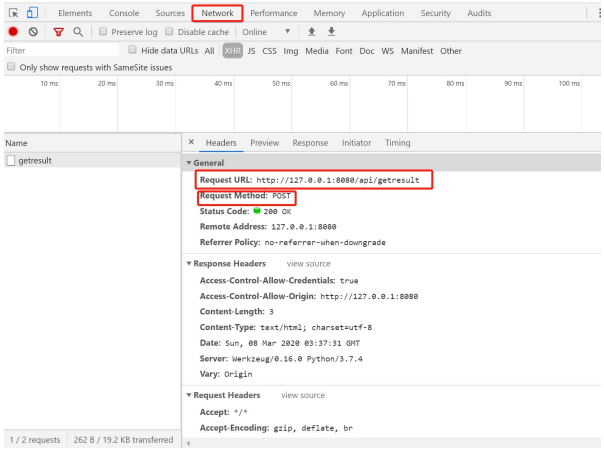
后端接收到的数据，与前端发送数据一致。

```
11  #  返回计算结果的API
12  @calc.route('/api/getr
13  def  get_calc_result()
14          data  =  reque
15          expr_val  =  data['expr']
16          return  str(eval(expr_val))
17
```

{'expr': '((1+2)*3-4)/5'}
'expr': '((1+2)*3-4)/5'
__len__: 1

继续 执行完成后端，返回计算结果的字符串到前端。

Step 10：浏览器，点击 `Network` ，查看发送的请求 Headers 等



并在 main.js 中，响应到结果后，处理函数 function(){}中，result 拿到结果 1.0.

```
66
67      $('#resultButton').click(function() {
68         content = $mainOutput.html()
69         if (content == '') return ;
70         if ('+-*/'.indexOf(content[content.length-1]) != -1) return;
71          var xmlhttp = new XMLHttpRequest();
72          xmlhttp.onreadystatechange=function() {
73              if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
74                  var result = xmlhttp.responseText;  result = "1.0"
75                  $mainOutput.html(result);
76                  num1 = result
77                  op = ''
78              }
79          }
80          xmlhttp.open("POST","api/getresult")
81          xmlhttp.setRequestHeader('content-type', 'application/json');
82          xmlhttp.send(JSON.stringify({'expr':$mainOutput.html()}));
83      });
84  });
85
```

以上，就是完整的前后端 10 个调试步骤。

小结

今天，与大家一起实战一个计算器 Flask 项目，实战项目的完整代码，会在 Python 专栏的交流群中发放，加群二维码在 day4 文章下面，欢迎进群领取。

这篇文章，完整介绍了本项目：

- 代码目录结构图
- 使用 Python 模板引擎 jinja2 + html + bootstrap框架 组合前端界面
- Flask 后端详细设计文档
- main.js 处理前端的按钮点击逻辑
- 重点介绍前、后端数据的交互流程，Flask API 接收前端数据还是非常方便
- 最后介绍了本项目的前后调试完整的 10 步流程

下一章