21.1mini-web 框架-路由支持正则

如果我们点击页面中的添加，希望得到服务器的响应怎么做，现在我们按图 21.1-1 中的添加会发现没有任何反应，那如何编写才能够有反应呢？

图 21.1-1

前面我们说过什么语言可以让网页有动态效果呢？没错，就是 JS，首先我们需要在 index.html 中添加如下代码

```
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>首页 - 个人选股系统 V5.87</title>
  <link href="/css/bootstrap.min.css" rel="stylesheet">
  <script src="/js/jquery-1.12.4.min.js"></script>
  <script src="/js/bootstrap.min.js"></script>
  <script>
      $(document).ready(function(){

              $("input[name='toAdd']").each(function(){
                  var currentAdd = $(this);
                  currentAdd.click(function(){
                      code = $(this).attr("systemIdVaule");
                      // alert("/add/" + code + ".html");
                      $.get("/add/" + code + ".html", function(data,
status){
                          alert("数据: " + data + "\n状态: " + status);
                      });
                  });
              });
```

```
        });
    </script>
</head>
```

上面 JS 代码的作用就是，当我们点击 toAdd 名称的 button 时，就会向服务器发送一个 html 页面的请求，如图 21.1-2 所示

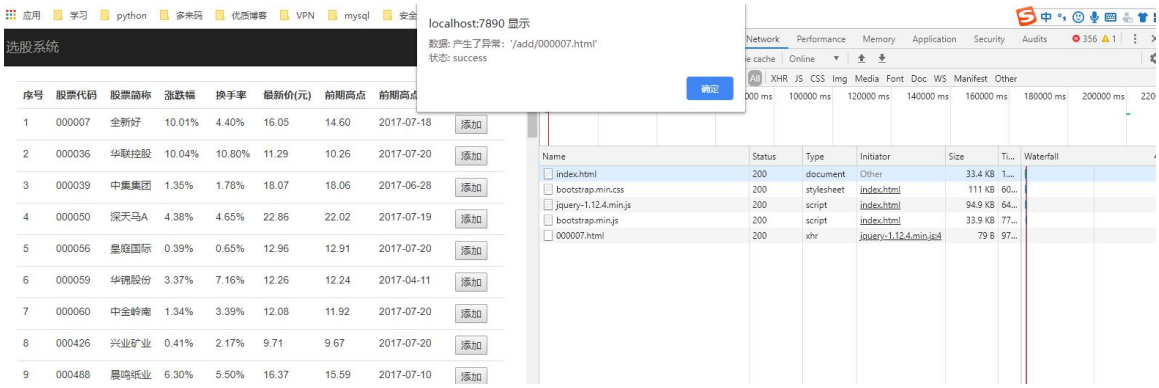


图 21.1-2

上图可以看到浏览器请求/add/000007.html 页面异常，原因是我们的 mini_frame.py 并没有针对该页面进行处理。

现在修改 mini_frame.py，通过正则对页面进行匹配，这样无论点击那个添加按钮我们可以走到一个函数，具体代码如下：

```
import re
from pymysql import connect

"""
URL_FUNC_DICT = {
    "/index.html": index,
    "/center.html": center
}
"""


URL_FUNC_DICT = dict()


def route(url):
    def set_func(func):
        # URL_FUNC_DICT["/index.py"] = index
        URL_FUNC_DICT[url] = func
        def call_func(*args, **kwargs):
```

```
            return func(*args, **kwargs)
        return call_func
    return set_func


@route(r"/index.html")
def index():
    with open("./templates/index.html",encoding="utf-8") as f:
        content = f.read()

    # my_stock_info = "哈哈哈哈 这是你的本月名称....."
    # content = re.sub(r"\{%content%\}", my_stock_info, content)
    # 创建 Connection 连接
    conn =
connect(host='192.168.0.111',port=3306,user='root',password='123',datab
ase='stock_db',charset='utf8')
    # 获得 Cursor 对象
    cs = conn.cursor()
    cs.execute("select * from info;")
    stock_infos = cs.fetchall()
    cs.close()
    conn.close()

    tr_template = """<tr>
        <td>%s</td>
        <td>%s</td>
        <td>%s</td>
        <td>%s</td>
        <td>%s</td>
        <td>%s</td>
        <td>%s</td>
        <td>%s</td>
        <td>
            <input type="button" value="添加" id="toAdd" name="toAdd"
systemidvaule="%s">
        </td>
        </tr>
    """

    html = ""
```

```
    for line_info in stock_infos:
        html += tr_template %
(line_info[0],line_info[1],line_info[2],line_info[3],line_info[4],line_i
nfo[5],line_info[6],line_info[7],line_info[1])

    # content = re.sub(r"\{%content%\}", str(stock_infos), content)
    content = re.sub(r"\{%content%\}", html, content)

    return content


@route(r"/center.html")
def center():
    with open("./templates/center.html",encoding="utf-8") as f:
        content = f.read()

    # my_stock_info = "这里是从 mysql 查询出来的数据。。。"
    # content = re.sub(r"\{%content%\}", my_stock_info, content)
    # 创建 Connection 连接
    conn =
connect(host='192.168.0.111',port=3306,user='root',password='123',datab
ase='stock_db',charset='utf8')
    # 获得 Cursor 对象
    cs = conn.cursor()
    cs.execute("select
i.code,i.short,i.chg,i.turnover,i.price,i.highs,f.note_info from info
as i inner join focus as f on i.id=f.info_id;")
    stock_infos = cs.fetchall()
    cs.close()
    conn.close()

    tr_template = """
        <tr>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
```

```
            <td>
                <a type="button" class="btn btn-default btn-xs"
href="/update/300268.html"> <span class="glyphicon glyphicon-star"
aria-hidden="true"></span> 修改 </a>
            </td>
            <td>
                <input type="button" value="删除" id="toDel"
name="toDel" systemidvaule="300268">
            </td>
        </tr>
    """

    html = ""
    for line_info in stock_infos:
        html += tr_template %
(line_info[0],line_info[1],line_info[2],line_info[3],line_info[4],line_i
nfo[5],line_info[6])

    # content = re.sub(r"\{%content%\}", str(stock_infos), content)
    content = re.sub(r"\{%content%\}", html, content)

    return content

# 给路由添加正则表达式的原因：在实际开发时，url 中往往会带有很多参数，例
如/add/000007.html 中 000007 就是参数，
# 如果没有正则的话，那么就需要编写 N 次@route 来进行添加 url 对应的函数 到
字典中，此时字典中的键值对有 N 个，浪费空间
# 而采用了正则的话，那么只要编写 1 次@route 就可以完成多个 url 例如
/add/00007.html /add/000036.html 等对应同一个函数，此时字典中的键值对个
数会少很多
@route(r"/add/\d+\.html")
def add_focus():
    return "add  ok ...."

def application(env, start_response):
    start_response('200 OK', [('Content-Type', 'text/html;charset=utf-
8')])

    file_name = env['PATH_INFO']
    # file_name = "/index.py"
```

```
    """
    if file_name == "/index.py":
        return index()
    elif file_name == "/center.py":
        return center()
    else:
        return 'Hello World! 我爱你中国....'
    """


    try:
        # func = URL_FUNC_DICT[file_name]
        # return func()
        # return URL_FUNC_DICT[file_name]()
        for url, func in URL_FUNC_DICT.items():
            # {
            #   r"/index.html":index,
            #   r"/center.html":center,
            #   r"/add/\d+\.html":add_focus
            # }
            ret = re.match(url, file_name)
            if ret:
                return func()
        else:
            return "请求的url(%s)没有对应的函数...." % file_name
    except Exception as ret:
        return "产生了异常：%s" % str(ret)
```

21.2 增加关注功能

当我们点击关注按钮时，获得了对应股票编码，这时候我们需要将股票编码传递给函数 add_focus，在 add_focus 中，我们需要首先查询一下对应的股票编码在数据库中是否有？有同学说本来就是后端传递给前端的，为什么还要再次验证一下呢？这是因为黑客可以模拟在浏览器输入一个链接的，所以首先要看股票编码在数据库里有没有，如果有，再看关注过没有，已经关注过了，就提示关注过，没关注过，再往数据库中插入对应的数据，代码如下：

```
import re
from pymysql import connect

"""
```

```
URL_FUNC_DICT = {
    "/index.html": index,
    "/center.html": center
}
"""

URL_FUNC_DICT = dict()


def route(url):
    def set_func(func):
        # URL_FUNC_DICT["/index.py"] = index
        URL_FUNC_DICT[url] = func
        def call_func(*args, **kwargs):
            return func(*args, **kwargs)
        return call_func
    return set_func


@route(r"/index.html")
def index(ret):
    with open("./templates/index.html",encoding="utf-8") as f:
        content = f.read()

    # my_stock_info = "哈哈哈哈 这是你的本月名称....."
    # content = re.sub(r"\{%content%\}", my_stock_info, content)
    # 创建 Connection 连接
    conn =
connect(host='192.168.0.111',port=3306,user='root',password='123',datab
ase='stock_db',charset='utf8')
    # 获得 Cursor 对象
    cs = conn.cursor()
    cs.execute("select * from info;")
    stock_infos = cs.fetchall()
    cs.close()
    conn.close()

    tr_template = """<tr>
        <td>%s</td>
        <td>%s</td>
```

```
        <td>%s</td>
        <td>%s</td>
        <td>%s</td>
        <td>%s</td>
        <td>%s</td>
        <td>%s</td>
        <td>
            <input type="button" value="添加" id="toAdd" name="toAdd"
systemidvaule="%s">
        </td>
        </tr>
    """

    html = ""
    for line_info in stock_infos:
        html += tr_template %
(line_info[0],line_info[1],line_info[2],line_info[3],line_info[4],line_i
nfo[5],line_info[6],line_info[7],line_info[1])

    # content = re.sub(r"\{%content%\}", str(stock_infos), content)
    content = re.sub(r"\{%content%\}", html, content)

    return content


@route(r"/center.html")
def center(ret):
    with open("./templates/center.html",encoding="utf-8") as f:
        content = f.read()

    # my_stock_info = "这里是从 mysql 查询出来的数据。。。"
    # content = re.sub(r"\{%content%\}", my_stock_info, content)
    # 创建 Connection 连接
    conn =
connect(host='192.168.0.111',port=3306,user='root',password='123',datab
ase='stock_db',charset='utf8')
    # 获得 Cursor 对象
    cs = conn.cursor()
    cs.execute("select
i.code,i.short,i.chg,i.turnover,i.price,i.highs,f.note_info from info
```

```
as i inner join focus as f on i.id=f.info_id;")
    stock_infos = cs.fetchall()
    cs.close()
    conn.close()

    tr_template = """
        <tr>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>
                <a type="button" class="btn btn-default btn-xs"
href="/update/300268.html"> <span class="glyphicon glyphicon-star"
aria-hidden="true"></span> 修改 </a>
            </td>
            <td>
                <input type="button" value="删除" id="toDel"
name="toDel" systemidvaule="300268">
            </td>
        </tr>
    """

    html = ""
    for line_info in stock_infos:
        html += tr_template %
(line_info[0],line_info[1],line_info[2],line_info[3],line_info[4],line_i
nfo[5],line_info[6])

    # content = re.sub(r"\{%content%\}", str(stock_infos), content)
    content = re.sub(r"\{%content%\}", html, content)

    return content

# 给路由添加正则表达式的原因：在实际开发时，url 中往往会带有很多参数，例
如/add/000007.html 中 000007 就是参数，
# 如果没有正则的话，那么就需要编写 N 次@route 来进行添加 url 对应的函数 到
```

```
字典中，此时字典中的键值对有 N 个，浪费空间
# 而采用了正则的话，那么只要编写 1 次@route 就可以完成多个 url 例如 /add/00007.html /add/000036.html 等对应同一个函数，此时字典中的键值对个数会少很多
@route(r"/add/(\d+)\.html")
def add_focus(ret):
    # 1. 获取股票代码
    stock_code = ret.group(1)

    # 2. 判断试下是否有这个股票代码
    conn = connect(host='192.168.0.111', port=3306, user='root', password='123', database='stock_db', charset='utf8')
    cs = conn.cursor()
    sql = """select * from info where code=%s;"""
    cs.execute(sql, (stock_code,))
    # 如果要是没有这个股票代码，那么就认为是非法的请求
    if not cs.fetchone():
        cs.close()
        conn.close()
        return "没有这支股票，大哥 ，我们是创业公司，请手下留情..."

    # 3. 判断以下是否已经关注过
    sql = """ select * from info as i inner join focus as f on i.id=f.info_id where i.code=%s;"""
    cs.execute(sql, (stock_code,))
    # 如果查出来了，那么表示已经关注过
    if cs.fetchone():
        cs.close()
        conn.close()
        return "已经关注过了，请勿重复关注..."

    # 4. 添加关注
    sql = """insert into focus (info_id) select id from info where code=%s;"""
    cs.execute(sql, (stock_code,))
    conn.commit()
    cs.close()
    conn.close()

    return "关注成功...."
```

```
def application(env, start_response):
    start_response('200 OK', [('Content-Type', 'text/html;charset=utf-8')])

    file_name = env['PATH_INFO']
    # file_name = "/index.py"

    """
    if file_name == "/index.py":
        return index()
    elif file_name == "/center.py":
        return center()
    else:
        return 'Hello World! 我爱你中国....'
    """

    try:
        # func = URL_FUNC_DICT[file_name]
        # return func()
        # return URL_FUNC_DICT[file_name]()
        for url, func in URL_FUNC_DICT.items():
            # {
            #   r"/index.html":index,
            #   r"/center.html":center,
            #   r"/add/\d+\.html":add_focus
            # }
            ret = re.match(url, file_name)
            if ret:
                return func(ret)
        else:
            return "请求的url(%s)没有对应的函数...." % file_name
    except Exception as ret:
        return "产生了异常：%s" % str(ret)
```

21.3 增加删除功能

现在我们点击删除按钮没有任何反应，为什么呢？首先我们需要在 center.html 代码中加入 JS 代码，实现点击后，向服务器发送请求的功能

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>个人中心 - 个人选股系统 V5.87</title>
    <link href="/css/bootstrap.min.css" rel="stylesheet">
    <script src="/js/jquery-1.12.4.min.js"></script>
    <script src="/js/bootstrap.min.js"></script>
    <script>
        $(document).ready(function(){

                $("input[name='toDel']").each(function(){
                    var currentAdd = $(this);
                    currentAdd.click(function(){
                        code = $(this).attr("systemIdVaule");
                        // alert("/del/" + code + ".html");
                        $.get("/del/" + code + ".html", function(data,
status){
                            alert("数据: " + data + "\n状态: " +
status);
                        });
                        window.location.reload()
                    });
                });
        });
    </script>
  </head>
```

增加了 JS 代码后，我们点击删除按钮，并不会删除，因为这时候浏览器给服务器发送的请求，服务器并没有响应，因此需要像 add_focus 一样，我们需要增加 del_focus 处理删除的请求，代码如下：

首先：center 函数中，构造页面的代码需要修改为下面内容，确保点击删除，或者下一节我们搞的修改，点击后，能够将对于的 code 值传递给服务器。

```
@route(r"/center.html")
def center(ret):
    with open("./templates/center.html",encoding="utf-8") as f:
        content = f.read()
```

```
    # my_stock_info = "这里是从 mysql 查询出来的数据。。。"
    # content = re.sub(r"\{%content%\}", my_stock_info, content)
    # 创建 Connection 连接
    conn =
connect(host='192.168.0.111',port=3306,user='root',password='123',datab
ase='stock_db',charset='utf8')
    # 获得 Cursor 对象
    cs = conn.cursor()
    cs.execute("select
i.code,i.short,i.chg,i.turnover,i.price,i.highs,f.note_info from info
as i inner join focus as f on i.id=f.info_id;")
    stock_infos = cs.fetchall()
    cs.close()
    conn.close()

    tr_template = """
        <tr>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>%s</td>
            <td>
                <a type="button" class="btn btn-default btn-xs"
href="/update/%s.html"> <span class="glyphicon glyphicon-star" aria-
hidden="true"></span> 修改 </a>
            </td>
            <td>
                <input type="button" value="删除" id="toDel"
name="toDel" systemidvaule="%s">
            </td>
        </tr>
    """

    html = ""
    for line_info in stock_infos:
        html += tr_template %
```

```
(line_info[0],line_info[1],line_info[2],line_info[3],line_info[4],line_i
nfo[5],line_info[6],line_info[0], line_info[0])

    # content = re.sub(r"\{%content%\}", str(stock_infos), content)
    content = re.sub(r"\{%content%\}", html, content)

    return content
```

然后，我们增加 del_focus 接口来处理请求，代码如下：

```
@route(r"/del/(\d+)\.html")
def del_focus(ret):

    # 1. 获取股票代码
    stock_code = ret.group(1)

    # 2. 判断试下是否有这个股票代码
    conn =
connect(host='192.168.0.111',port=3306,user='root',password='123',datab
ase='stock_db',charset='utf8')
    cs = conn.cursor()
    sql = """select * from info where code=%s;"""
    cs.execute(sql, (stock_code,))
    # 如果要是没有这个股票代码，那么就认为是非法的请求
    if not cs.fetchone():
        cs.close()
        conn.close()
        return "没有这支股票，大哥 ，我们是创业公司，请手下留情..."

    # 3. 判断以下是否已经关注过
    sql = """ select * from info as i inner join focus as f on
i.id=f.info_id where i.code=%s;"""
    cs.execute(sql, (stock_code,))
    # 如果没有关注过，那么表示非法的请求
    if not cs.fetchone():
        cs.close()
        conn.close()
        return "%s 之前未关注，请勿取消关注..." % stock_code

    # 4. 取消关注
    # sql = """insert into focus (info_id) select id from info where
```

```
code=%s;"""
    sql = """delete from focus where info_id = (select id from info
where code=%s);"""
    cs.execute(sql, (stock_code,))
    conn.commit()
    cs.close()
    conn.close()

    return "取消关注成功...."
```

21.4 修改功能

通过在 mini_frame 修改的位置可以看到，点击修改，会到 update 页面，所以我们首先需要编写 update.html 页面，update 页面如下，我们放在 templates 中

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>首页 - 个人选股系统 V5.87</title>
    <link href="/css/bootstrap.min.css" rel="stylesheet">
    <script src="/js/jquery-1.12.4.min.js"></script>
    <script src="/js/bootstrap.min.js"></script>
    <script>
        $(document).ready(function(){
            $("#update").click(function(){
                var item = $("#note_info").val();
                // alert("/update/{%code%}/" + item + ".html");
                $.get("/update/{%code%}/" + item + ".html",
function(data, status){
                    alert("数据: " + data + "\n状态: " + status);
                    self.location='/center.html';
                });
            });
        });
    </script>
  </head>
  <body>
```

```
<div class="navbar navbar-inverse navbar-static-top ">
    <div class="container">
    <div class="navbar-header">
        <button class="navbar-toggle" data-toggle="collapse" data-
target="#mymenu">
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
        </button>
        <a href="#" class="navbar-brand">选股系统</a>
    </div>
    <div class="collapse navbar-collapse" id="mymenu">
        <ul class="nav navbar-nav">
            <li><a href="/index.html">股票信息</a></li>
            <li><a href="/center.html">个人中心</a></li>
        </ul>
    </div>
    </div>
</div>
  <div class="container">
    <div class="container-fluid">
      <div class="input-group">
          <span class="input-group-addon">正在修改：</span>
          <span class="input-group-addon">{%code%}</span>
          <input id="note_info" type="text" class="form-control" aria-
label="Amount (to the nearest dollar)" value="{%note_info%}">
          <span id="update" class="input-group-addon" style="cursor:
pointer">修改</span>
      </div>
    </div>
  </div>
  </body>
</html>
```

结果出现如下错误

请求的url(/update/000007.html)没有对应的函数....

只放一个 update.html 是不行的，因为我们请求 html 时，需要 mini_frame 框架进行响应，因此我们需要编写 update 函数，update 函数用于打开 update.html 页面并读取内容，同时要讲股票 code 及股票备注信息 note_info 替换掉，代码如下：

```
@route(r"/update/(\d+)\.html")
def show_update_page(ret):
    """显示修改的那个页面"""
    # 1. 获取股票代码
    stock_code = ret.group(1)

    # 2. 打开模板
    with open("./templates/update.html",encoding="utf-8") as f:
        content = f.read()

    # 3. 根据股票代码查询相关的备注信息
    conn =
connect(host='192.168.0.111',port=3306,user='root',password='123',datab
ase='stock_db',charset='utf8')
    cs = conn.cursor()
    sql = """select f.note_info from focus as f inner join info as i on
i.id=f.info_id where i.code=%s;"""
    cs.execute(sql, (stock_code,))
    stock_infos = cs.fetchone()
    note_info = stock_infos[0]  # 获取这个股票对应的备注信息
    cs.close()
    conn.close()

    content = re.sub(r"\{%note_info%\}", note_info, content)
    content = re.sub(r"\{%code%\}", stock_code, content)

    return content
```

但是，当我们到达如下图的修改页面时，我们输入内容，点击修改，出现请求的 url 没有对应的函数的提示。

其实原因是我们点击修改时，再次向服务器提交了一个新的请求，这时是一个新的 url，因此我们需要编写新的函数进行处理，我们起名 save_update_page，由于点击修改时，浏览器传过来了两个值，一个是股票代码，一个是我们增加的备注信息，通过查找数据库中对应的股票代码，放入我们增加的备注信息即可，代码如下：

```
@route(r"/update/(\d+)/(.*)\.html")
def save_update_page(ret):
    """保存修改的信息"""
    stock_code = ret.group(1)
    comment = ret.group(2)

    conn =
connect(host='192.168.0.111',port=3306,user='root',password='123',datab
ase='stock_db',charset='utf8')
    cs = conn.cursor()
    sql = """update focus set note_info=%s where info_id = (select id
from info where code=%s);"""
    cs.execute(sql, (comment, stock_code))
    conn.commit()
    cs.close()
    conn.close()

    return "修改成功..."
```

21.5 修复乱码功能

上面虽然可以修改了，但是当我们加入中文时，会发现实际保存的是乱码，原因是浏览器会对**汉字，空格，斜杠**等进行转码，因此当我们服务器端接收到以后，需要对编码进行转换，在 save_update_page 加入下面语句即可。

```
comment = urllib.parse.unquote(comment)
```

或者

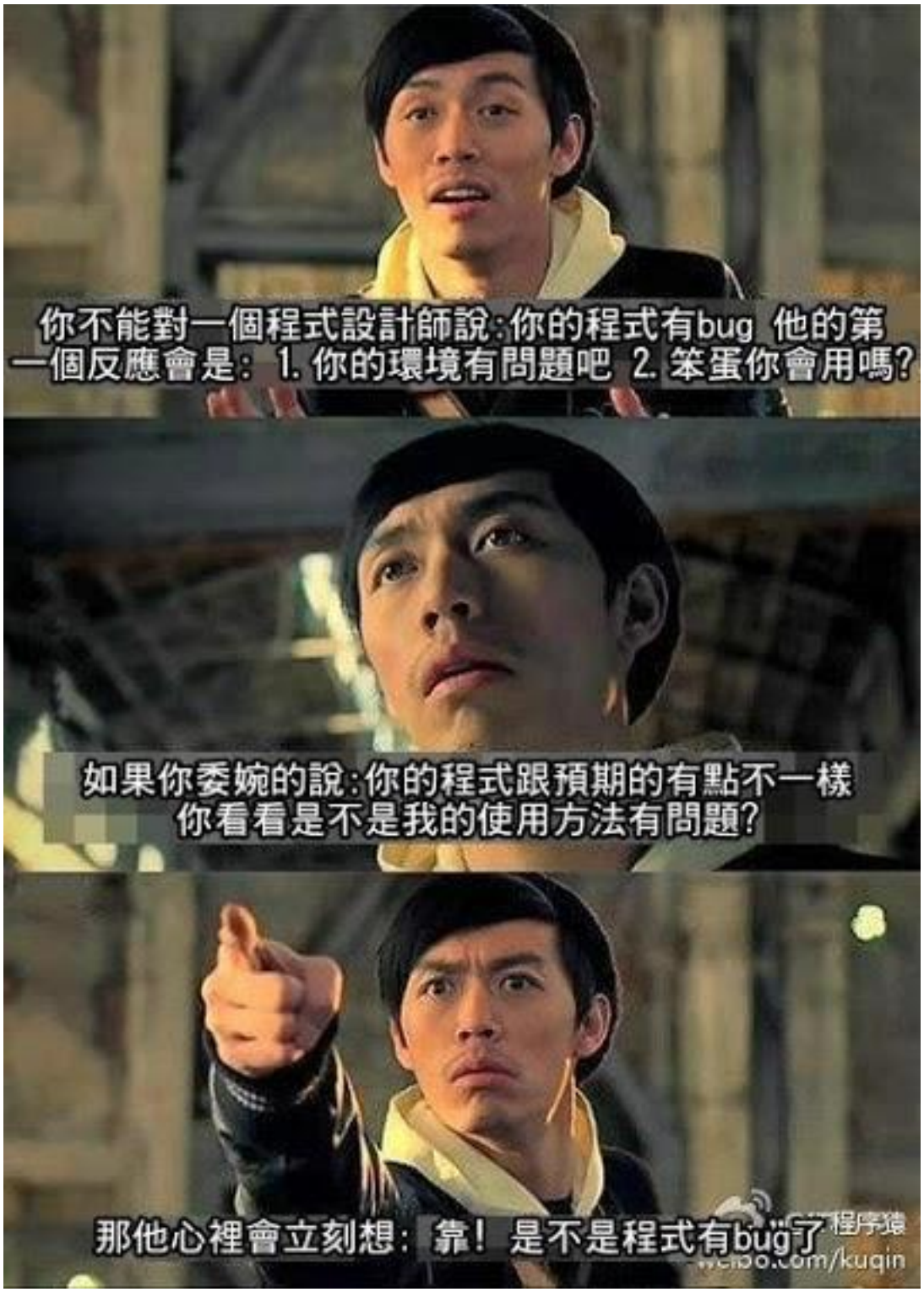
comment = parse.unquote(comment)

不过如果这样，需要写为 from urllib import parse

#针对反斜杠 来看 POST 提交，JS 处理

21.6 logging 日志


脑残對話
什么？有bug!!不可能我都测过了！
你用的什么浏览器，什么版本，
你是最新代码吗?重启试试，能复现吗？
脑残對話
卧槽，还真是有bug!等我看看……
暴走漫画 baozoumanhua.com ©
@哒木子



开发过程中出现 bug 是必不可免的，你会怎样 debug？从第 1 行代码开始看么？还是有个文件里面记录着哪里错了更方便呢！！！log 日志

Python 中有个 logging 模块可以完成相关信息的记录，在 debug 时用它往往事半功倍

21.6.1. 日志级别

日志一共分成 5 个等级，从低到高分别是：

1. DEBUG
2. INFO
3. WARNING
4. ERROR
5. CRITICAL

说明:

- DEBUG：详细的信息,通常只出现在诊断问题上
- INFO：确认一切按预期运行
- WARNING：一个迹象表明,一些意想不到的事情发生了,或表明一些问题在不久的将来(例如。磁盘空间低”)。这个软件还能按预期工作。
- ERROR：更严重的问题,软件没能执行一些功能
- CRITICAL：一个严重的错误,这表明程序本身可能无法继续运行

这 5 个等级，也分别对应 5 种打日志的方法： debug 、info 、warning 、error 、critical。默认的是 WARNING，当在 WARNING 或之上时才被跟踪。

21.6.2. 日志输出

有两种方式记录跟踪，一种输出控制台，另一种是记录到文件中，如日志文件。

21.6.2.1、将日志输出到控制台

比如，log1.py 如下：

```
import logging

logging.basicConfig(level=logging.WARNING,
                    format='%(asctime)s - %(filename)s[line:%(lineno)d] -
 %(levelname)s: %(message)s')

# 开始使用 log 功能
logging.info('这是 loggging info message')
logging.debug('这是 loggging debug message')
logging.warning('这是 loggging a warning message')
logging.error('这是 an loggging error message')
logging.critical('这是 loggging critical message')
```

运行结果

```
2017-11-06 23:07:35,725 - log1.py[line:9] - WARNING: 这是 loggging a war
ning message
2017-11-06 23:07:35,725 - log1.py[line:10] - ERROR: 这是 an loggging err
or message
2017-11-06 23:07:35,725 - log1.py[line:11] - CRITICAL: 这是 loggging cri
tical message
```

说明

通过 logging.basicConfig 函数对日志的输出格式及方式做相关配置，上面代码设置日志的输出等级是 WARNING 级别，意思是 WARNING 级别以上的日志才会输出。另外还制定了日志输出的格式。

注意，只要用过一次 log 功能再次设置格式时将失效，实际开发中格式肯定不会经常变化，所以刚开始时需要设定好格式

21.6.2.2、将日志输出到文件

我们还可以将日志输出到文件，只需要在 logging.basicConfig 函数中设置好输出文件的文件名和写文件的模式。

log2.py 如下：

```
import logging

logging.basicConfig(level=logging.WARNING,
                    filename='./log.txt',
                    filemode='w',
                    format='%(asctime)s - %(filename)s[line:%(lineno)d] -
 %(levelname)s: %(message)s')
# use logging
logging.info('这是 loggging info message')
logging.debug('这是 loggging debug message')
logging.warning('这是 loggging a warning message')
logging.error('这是 an loggging error message')
logging.critical('这是 loggging critical message')
```

运行效果

```
python@ubuntu: cat log.txt
2017-11-06 23:10:44,549 - log2.py[line:10] - WARNING: 这是 loggging a wa
rning message
2017-11-06 23:10:44,549 - log2.py[line:11] - ERROR: 这是 an loggging err
or message
2017-11-06 23:10:44,549 - log2.py[line:12] - CRITICAL: 这是 loggging cri
tical message
```

21.6.2.3、既要把日志输出到控制台， 还要写入日志文件

这就需要一个叫作 Logger 的对象来帮忙，下面将对他进行详细介绍，现在这里先学习怎么实现把日志既要输出到控制台又要输出到文件的功能。

```
import logging

# 第一步，创建一个 logger
logger = logging.getLogger()
logger.setLevel(logging.INFO)  # Log 等级总开关

# 第二步，创建一个 handler，用于写入日志文件
logfile = './log.txt'
fh = logging.FileHandler(logfile, mode='a')  # open 的打开模式这里可以进行参考
fh.setLevel(logging.DEBUG)  # 输出到 file 的 log 等级的开关

# 第三步，再创建一个 handler，用于输出到控制台
ch = logging.StreamHandler()
ch.setLevel(logging.WARNING)   # 输出到 console 的 log 等级的开关

# 第四步，定义 handler 的输出格式
formatter = logging.Formatter("%(asctime)s - %(filename)s[line:%(lineno)d] - %(levelname)s: %(message)s")
fh.setFormatter(formatter)
ch.setFormatter(formatter)

# 第五步，将 logger 添加到 handler 里面
logger.addHandler(fh)
logger.addHandler(ch)

# 日志
logger.debug('这是 logger debug message')
logger.info('这是 logger info message')
logger.warning('这是 logger warning message')
logger.error('这是 logger error message')
logger.critical('这是 logger critical message')
```

运行时终端的输出结果:

```
2017-11-06 23:14:04,731 - log3.py[line:28] - WARNING: 这是 logger warning message
2017-11-06 23:14:04,731 - log3.py[line:29] - ERROR: 这是 logger error message
2017-11-06 23:14:04,731 - log3.py[line:30] - CRITICAL: 这是 logger critical message
```

在 log.txt 中，有如下数据：

```
2017-11-06 23:14:04,731 - log3.py[line:27] - INFO: 这是 logger info mess
age
2017-11-06 23:14:04,731 - log3.py[line:28] - WARNING: 这是 logger warnin
g message
2017-11-06 23:14:04,731 - log3.py[line:29] - ERROR: 这是 logger error me
ssage
2017-11-06 23:14:04,731 - log3.py[line:30] - CRITICAL: 这是 logger criti
cal message
```

21.6.2.4 日志格式说明

logging.basicConfig 函数中，可以指定日志的输出格式 format，这个参数可以输出很多有用的信息，如下:

- %(levelno)s: 打印日志级别的数值
- %(levelname)s: 打印日志级别名称
- %(pathname)s: 打印当前执行程序的路径，其实就是 sys.argv[0]
- %(filename)s: 打印当前执行程序名
- %(funcName)s: 打印日志的当前函数
- %(lineno)d: 打印日志的当前行号
- %(asctime)s: 打印日志的时间
- %(thread)d: 打印线程 ID
- %(threadName)s: 打印线程名称
- %(process)d: 打印进程 ID
- %(message)s: 打印日志信息

在工作中给的常用格式如下:

```
format='%(asctime)s - %(filename)s[line:%(lineno)d] - %(levelname)s: %
(message)s'
```

这个格式可以输出日志的打印时间，是哪个模块输出的，输出的日志级别是什么，以及输入的日志内容。