day23

内容回顾和补充

1. ajax操作

```
$.ajax({
    url:'...',
    type: "GET",
    data:{},
    dataType:"JSON",
    success:function(arg){
        // arg  ->  JSON.parse(arg)
    },
    error:function(){

    }
})
```

2. 队列

```
import queue

# 创建一个队列
q = queue.Queue()

# 往队列中放数据
q.put("冯涛1")
q.put("冯涛2")


# 去队列中获取数据
v1 = q.get()
v2 = q.get()
print(v1,v2)

try:
```

```
    v3 = q.get(timeout=3)
    print(v3)
except queue.Empty as e:
    pass
```

3. 关于递归

```
def func():
    func()
```

```
function f1(){
    $.ajax({
        url:'...',
        ...
        success:function(arg){
            f1()
        }
    })
}
f1()
```

今日概要

- 服务端向客户端主动推送消息
  - 轮询/长轮询
  - websocket
- gojs插件
- paramiko模块, 封装类
- git, 用Python去执行git命令, 封装类
- 项目开发: 基本增删改查

今日详细

1. 服务端向客户端主动推送消息

1.1 轮询

让浏览器定时通过ajax向后端偷偷发送请求,来说去数据.

以伪造服务端向浏览器推送数据的现象.

缺点:

- 延迟
- 请求次数多

1.2 长轮询 (兼容性好)

让浏览器通过ajax向后端偷偷发送请求, 来获取数据.

在此过程中会有阻塞, 最多阻塞30s.

详细见示例: poll.zip

1.3 websocket (主流浏览器支持)

```
websocket
    web, 写网站让浏览器和服务端进行交互.
    socket, 让网络上的两端创建链接并进行收发数据.
```

```
http是一个网络协议.(无状态短连接)
https是一个网络协议.(无状态短连接)
websocket是一个网络协议(让浏览器和服务端创建链接支持,默认不再断开,两端就可以完成相互之间的收发数据)

websocket协议的诞生,可以让我们真正实现服务端向客户端推送消息.
```

```
websocket实现原理:
    - 握手环节,验证服务端是否支持websocket协议.
        浏览器生成一个随机字符串,将随机字符串发送给服务端.
        服务端接收到随机字符串之后,让他跟 magic string 拼接,然后再进行sha1 / base64加密
        将密文返回到用户浏览器
        用户浏览器自动化会进行校验
    - 收发数据,密文
        数据解密时需要读取数据第2个字节的后7位,如果
            127
            126
            <=125
```

```
在Django中如果想要开发websocket相关的功能,就需要先安装:
    pip3 install channels==2.3

    建议:在python3.6的环境中去运行.

在channels的内部已经帮助我们写了握手/加密/解密等所有环节.

注意: 不是所有的服务端都支持websocket
    - django
        - 默认不支持
        - 第三方:channels
    - flask
        - 默认不支持
        - 第三方:geventwebsocket
    - tornado
        - 默认支持
```

1.4 django实现websocket

1.4.1 安装channels

```
pip3 install channels==2.3 -i https://pypi.tuna.tsinghua.edu.cn/simple
```

1.4.2 创建django项目

- 引入channles

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app01.apps.App01Config',
    # 项目中要使用channels做websocket了.
    "channels",
]
```

本质: channels会把原来只支持http协议的wsgi,换成支持http和websocket协议的asgi

- 配置application

```
# settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app01.apps.App01Config',
    # 项目中要使用channels做websocket了.
    "channels",
]

ASGI_APPLICATION = "channel_demo.routing.application"
```

```python
# channel_demo/routing.py

from channels.routing import ProtocolTypeRouter, URLRouter
from django.conf.urls import url

application = ProtocolTypeRouter({
    'websocket': URLRouter([
        # url(r'^chat/$', consumers.ChatConsumer),
    ])
})
```

Web聊天室案例

```
http协议
    index       ->      index函数
    访问:浏览器发送请求即可

websocket协议
    chat        ->      ChatConsumer(3个方法)
    访问:new WebSocket对象
```

1.4.3 channel-layers [直播]

总结

- websocket是什么?
- django如果想要实现websocket协议,需要依赖 channels 模块.(dwebsocket)
- 轮询和长轮训都可以完成服务端主动向客户端推送消息 (伪)
- channels的用法和案例 (channel_demo案例)

2.gojs

是一个前端插件,用户帮助用户动态创建节点信息.

概念

- TextBlock,创建文本.

- Shap,图形
- Node,节点(文本和图形结合)
- Link,箭头

案例

- TextBlock

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<div id="myDiagramDiv" style="width:500px; height:350px; background-color: #DAE4E4;"></div>


<script src="js/go.js"></script>

<script>
    var $ = go.GraphObject.make;
    // 第一步：创建图表
    var myDiagram = $(go.Diagram, "myDiagramDiv"); // 创建图表，用于在页面上画图


    var node1 = $(go.Node, $(go.TextBlock, {text: "武沛齐"}));
    myDiagram.add(node1);

    var node2 = $(go.Node, $(go.TextBlock, {text: "武沛齐", stroke: 'red'}));
    myDiagram.add(node2);

    var node3 = $(go.Node, $(go.TextBlock, {text: "武沛齐", background: 'lightblue'}));
    myDiagram.add(node3);


```

```
</script>
</body>
</html>
```

- Shape

```
...
```

- Link

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div id="myDiagramDiv" style="width:500px; min-height:450px; background-color: #DAE4E4;"></div>
    <script src="js/go-debug.js"></script>

    <script>
        var $ = go.GraphObject.make;

        var myDiagram = $(go.Diagram, "myDiagramDiv",
            {layout: $(go.TreeLayout, {angle: 0})}
        ); // 创建图表，用于在页面上画图


        var startNode = $(go.Node, "Auto",
            $(go.Shape, {figure: "Ellipse", width: 40, height: 40, fill: '#79C900', stroke: '#79C900'}),
            $(go.TextBlock, {text: '开始', stroke: 'white'})
        );
        myDiagram.add(startNode);


        var downloadNode = $(go.Node, "Auto",
```

```
            $(go.Shape, {figure: "RoundedRectangle", height: 40, fill: 'red', stroke: '#79C900'}),
            $(go.TextBlock, {text: '下载代码', stroke: 'white'})
        );
        myDiagram.add(downloadNode);

        var startToDownloadLink = $(go.Link,
            {fromNode: startNode, toNode: downloadNode},
            $(go.Shape, {strokeWidth: 1}),
            $(go.Shape, {toArrow: "OpenTriangle", fill: null, strokeWidth: 1})
        );
        myDiagram.add(startToDownloadLink);
    </script>
</body>
</html>
```

更多案例见:https://www.cnblogs.com/wupeiqi/articles/11978547.html

3.paramiko模块

3.1 安装

```
pip3 install paramiko
```

3.2 快速上手

3.2.1 远程执行命令

```
import paramiko

# ######################### 用户名和密码
"""
# 创建SSH对象
ssh = paramiko.SSHClient()
```

```
# 允许连接不在know_hosts文件中的主机
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
# 连接服务器
ssh.connect(hostname='123.206.16.61', port=22, username='root', password='nidaye..!')
# 执行命令
stdin, stdout, stderr = ssh.exec_command('df')
# 获取命令结果
result = stdout.read()
print(result.decode('utf-8'))
# 关闭连接
ssh.close()
"""

# ######################### 公钥和私钥
"""
import paramiko

private_key = paramiko.RSAKey.from_private_key_file('/home/auto/.ssh/id_rsa')

# 创建SSH对象
ssh = paramiko.SSHClient()
# 允许连接不在know_hosts文件中的主机
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
# 连接服务器
ssh.connect(hostname='123.206.16.61', port=22, username='root', pkey=private_key)

# 执行命令
stdin, stdout, stderr = ssh.exec_command('df')
# 获取命令结果
result = stdout.read()
print(result.decode('utf-8'))
# 关闭连接
ssh.close()
"""

```

```
42
```

3.2.2 上传下载文件

```python
import paramiko

# ############### 用户名和密码
"""
transport = paramiko.Transport(('123.206.16.61', 22))
transport.connect(username='root',
password='nidaye..!')

sftp = paramiko.SFTPClient.from_transport(transport)

# 上传文件
# sftp.put(r'D:\wupeiqi\s27\day01\4.基于paramiko操作
\xx', '/data/s27/xx')

# 下载文件
sftp.get('/data/s27/xx', 'log.txt')

transport.close()
"""


# ############# 公钥和私钥
import paramiko
private_key =
paramiko.RSAKey.from_private_key_file('/home/auto/.ssh
/id_rsa')
transport = paramiko.Transport(('hostname', 22))
transport.connect(username='wupeiqi',
pkey=private_key)
sftp = paramiko.SFTPClient.from_transport(transport)
# 将location.py 上传至服务器 /tmp/test.py
sftp.put('/tmp/location.py', '/tmp/test.py')
# 将remove_path 下载到本地 local_path
sftp.get('remove_path', 'local_path')
transport.close()
```

3.3 SSHProxy的封装

```python
import paramiko

class SSHProxy(object):
    def __init__(self, hostname, port, username, password):
        self.hostname = hostname
        self.port = port
        self.username = username
        self.password = password

        self.transport = None

    def open(self):
        self.transport = paramiko.Transport((self.hostname, self.port))
        self.transport.connect(username=self.username, password=self.password)

    def command(self, cmd):
        ssh = paramiko.SSHClient()
        ssh._transport = self.transport

        stdin, stdout, stderr = ssh.exec_command(cmd)
        result = stdout.read()
        return result

    def upload(self, local_path, remote_path):
        sftp = paramiko.SFTPClient.from_transport(self.transport)
        sftp.put(local_path, remote_path)
        sftp.close()

    def close(self):
        self.transport.close()

    def __enter__(self):
        self.open()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.close()
```

```
38
39 if __name__ == '__main__':
40
41     with SSHProxy("123.206.16.61",22,'root','nidaye..!') as ssh:
42         ssh.command('df')
43         ssh.upload(r'D:\wupeiqi\s27\day01\4.基于paramiko操作\xx', '/data/s27/xx')
44
```

补充:面向对象上下文

```
1  """
2  class Foo(object):
3      def __enter__(self):
4          print('进入')
5          return self
6
7      def __exit__(self, exc_type, exc_val, exc_tb):
8          print('出去')
9
10 with Foo() as f1:
11     print(1,f1)
12     print(2)
13 """
14
15 class Context:
16     def __enter__(self):
17         return self
18
19     def __exit__(self, *args,**kwargs):
20         pass
21
22     def do_something(self):
23         pass
24
25 with Context() as ctx:
26     ctx.do_something()
27
28 # 请在Context类中添加代码完成该类的实现.
```

4. Python操作git

4.1 安装模块

```
pip3 install gitpython
```

4.2 快速使用

```
import os
from git.repo import Repo

download_path = os.path.join('code', 'fuck')
Repo.clone_from('https://gitee.com/wupeiqi/fuck.git', to_path=download_path, branch='master')
```

4.3 封装Git相关类

```
import os
from git.repo import Repo
from git.repo.fun import is_git_dir


class GitRepository(object):
    """
    git仓库管理
    """

    def __init__(self, local_path, repo_url, branch='master'):
        self.local_path = local_path
        self.repo_url = repo_url
        self.repo = None
        self.initial(repo_url, branch)

    def initial(self, repo_url, branch):
        """
        初始化git仓库
        :param repo_url:
```

```
        :param branch:
        :return:
        """
        if not os.path.exists(self.local_path):
            os.makedirs(self.local_path)

        git_local_path = os.path.join(self.local_path, '.git')
        if not is_git_dir(git_local_path):
            self.repo = Repo.clone_from(repo_url, to_path=self.local_path, branch=branch)
        else:
            self.repo = Repo(self.local_path)

    def pull(self):
        """
        从线上拉最新代码
        :return:
        """
        self.repo.git.pull()

    def branches(self):
        """
        获取所有分支
        :return:
        """
        branches = self.repo.remote().refs
        return [item.remote_head for item in branches if item.remote_head not in ['HEAD', ]]

    def commits(self):
        """
        获取所有提交记录
        :return:
        """
        commit_log = self.repo.git.log('--pretty={"commit":"%h","author":"%an","summary":"%s","date":"%cd"}',
                                       max_count=50,

date='format:%Y-%m-%d %H:%M')
```

```
        log_list = commit_log.split("\n")
        return [eval(item) for item in log_list]

    def tags(self):
        """
        获取所有tag
        :return:
        """
        return [tag.name for tag in self.repo.tags]

    def change_to_branch(self, branch):
        """
        切换分值
        :param branch:
        :return:
        """
        self.repo.git.checkout(branch)

    def change_to_commit(self, branch, commit):
        """
        切换commit
        :param branch:
        :param commit:
        :return:
        """
        self.change_to_branch(branch=branch)
        self.repo.git.reset('--hard', commit)

    def change_to_tag(self, tag):
        """
        切换tag
        :param tag:
        :return:
        """
        self.repo.git.checkout(tag)


if __name__ == '__main__':
    local_path = os.path.join('codes', 'luffycity')
    repo = GitRepository(local_path,
'https://gitee.com/wupeiqi/fuck.git')
```

```
    branch_list = repo.branches()
    print(branch_list)
    repo.change_to_branch('dev')
    repo.pull()
```

总结

1. websocket
    - websocket是干嘛的?

      ```
      客户端和服务端建立链接,服务端就可以向客户度主动推送消息.
      ```

    - websocket的实现机制?

      ```
      - 握手环节
      - 数据解密环境
      ```

    - django如何用websocket

      ```
      - dwebsocket
      - channels (官方推荐)

      理解: django默认的wsgi会替换成asgi(达芙妮daphne)
      ```

    - websocket协议和http协议的区别?

      ```
      ...
      ```

    - 服务端向客户端推送消息其他解决方法
        - 轮询
        - 长轮询
2. gojs用于画图
3. paramiko
    - 理解他的作用
    - 代码保留
4. gitpython

    - 理解他的作用
    - 代码保留
5. ajax操作
6. 队列
7. django项目找模板的顺序: 项目根目录templates -> 按照app的注册顺序
8. 面向对象的上下文管理,with语法

作业

- channels
- gojs
- 结合以上两个示例实现

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div id="diagramDiv" style="width:100%; min-height:450px; background-color: #DAE4E4;"></div>

    <script src="js/go.js"></script>
    <script>
        var $ = go.GraphObject.make;
        var diagram = $(go.Diagram, "diagramDiv",{
            layout: $(go.TreeLayout, {
                angle: 0,
                nodeSpacing: 20,
                layerSpacing: 70
            })
        });

        diagram.nodeTemplate = $(go.Node, "Auto",
            $(go.Shape, {
                figure: "RoundedRectangle",
```

```
                fill: 'lightgray',
                stroke: 'lightgray'
            }, new go.Binding("figure", "figure"),
new go.Binding("fill", "color"), new
go.Binding("stroke", "color")),
            $(go.TextBlock, {margin: 8}, new
go.Binding("text", "text"))
        );

        diagram.linkTemplate = $(go.Link,
            {routing: go.Link.Orthogonal},
            $(go.Shape, {stroke: 'lightgray'}, new
go.Binding('stroke', 'link_color')),
            $(go.Shape, {toArrow: "OpenTriangle",
stroke: 'lightgray'}, new go.Binding('stroke',
'link_color'))
        );

        // 当客户端浏览器接收到服务端返回的数据时候
        ws.onmessage = function(event){
            var nodeDataArray =
JSON.parse(event.data);
            diagram.model = new
go.TreeModel(nodeDataArray);
        }

        // 动态控制节点颜色变化
        /*
        var node =
diagram.model.findNodeDataForKey("zip");
        diagram.model.setDataProperty(node,
"color", "lightgreen");
        */
    </script>
</body>
</html>
```

参考博客:

- websocket https://www.cnblogs.com/wupeiqi/p/6558766.html

- gojs https://www.cnblogs.com/wupeiqi/articles/11978547.html
- 发布功能 https://www.cnblogs.com/wupeiqi/articles/9805296.html
- paramiko https://www.cnblogs.com/wupeiqi/articles/5095821.html