day8:

1. 前端收到后端的`jsonify(resp)`的结果==`Content-Type:application/json`; 那么`resp`直接转化成了js文件中对应的对象
2. flask后端通过前端发送的request对象拿json格式的数据, **request.get_json()能讲请求体的json数据转化为dict格式**:

```
// GET 方式向flask发送的
request.args.get("")
// POST 方式发送的json数据, 后端接收
req_dict = request.get_json()
aaa = req_dict.get("aaa")
```

1. ® 注册页面一个部分作为另一个的premise:
    a. 拿到image_code判断作为sms_code的验证
    b. 拿到sms_code判断作为用户数据保存的前提
2. 保存用户的数据到数据库中: 之前不用判断"用户是否已经注册过", 因为之前设计数据库的时候已经设置`name, mobile`unique
3. 密码hash加密本来是调用方法, 但是用“@函数名.setter”的方式, 变为"设置属性"来调用方法

```
@property
def password(self):  # 这个方法名对应后端对象操作的属性
    # 对应password属性的读取操作
    pass

@password.setter  # 装饰器名就是之前@property修饰的函数名
def password(self, value):
    # 对应password属性的设置, value是用户设置的密码值
    self.password_hash = generate_password_hash(value)
```

1. 利用session记住用户的登录状态:

```
from flask import session
session["user_id"] = user.id # user是之前创建的对象
```

1. `try...except...else`: 执行`except`和`else`对应的内容是互斥的: 复习
2. **后端得到的json数据转化为dict: `request.get_json()`**

3. **后端python对象转化为json: `json.dumps()`**
4. **前端js中对象转化为json: `JSON.stringify(obj)`**
5. 前后端分离: 注册成功之后是前端`location.href="/"`导向不同的页面;
6. 而不是之前django那样的后端导向页面
7. ajax请求的完整格式:

```
$.ajax({
    url: "/api/v1_0/users", // 请求路径url
    type: "post",  // 请求方式
    data: req_json,  // 发送的请求体数据
    contentType: "application/json", //指明向后端发送json格式
    dataType: "json", // 从后端收到数据是json格式
    headers: {
        "X-CSRFToken": getCookie("csrf_token")
    }), //自定义请求头
    success: function (resp) {
        if (resp.errno == 0) {
            // 注册成功, 引导到主页面
            location.href = "/";
        } else {
            alert(resp.errmsg);
        }
    }
```

1. csrf_token从body请求体或者headers中提取:
    a. 如果请求是表单格式: 从body中提取
    b. 如果不是表单格式: headers中设置, 提取
2. 正则
    a. `\b`匹配一个word前或后边界的empty string
    b. `\B`匹配不是word前或后边界的empty string
3. 另外的API: 也都是先写后端再写前端
    a. `login()`: `User.query.filter(mobile=mobile).first()`查询有没有这个用户, `user.check_password()`检查输入的密码是否正确;
    b. `check_login()`: 检查用户的登录状态: 看**session**中能不能拿到这个用户的id
    c. `logout()`: 清楚session数据
4. 登录装饰器:
    a. 闭包外层函数的参数是被装饰的函数
    b. 闭包内层函数的参数是被装饰函数的参数
    c. 外层函数返回值是内层函数

```
def login_required(view_func):  # 修饰的就是视图函数
    def wrapper(*args, **kwargs):
        # 是否登录看session中能否拿到user_id
        user_id = session.get("user_id")
        if user_id is not None:
            # 用户已登录，执行视图函数
            return view_func(*args, **kwargs)
        else:
            resp = {
                "..."
        }
            return jsonify(resp)
    return wrapper
```

1. 第三方服务:
    a. 短信验证码: 用户->flask->云通讯
    b. 上传图片: 用户->flask->七牛云