邮件和分页

1.短信验证

很多业务需要使用手机验证码服务，这里以阿里云短信服务为例，说明python如何集成第三方短信服务，之所以选择阿里云短信服务，是因为阿里云短信服务针对个人，只要账号有钱就可以发送短信验证码，而很多短信平台是针对公司的，不支持个人。

1.短信验证码设置

首先登陆阿里云，到控制台，在"产品和服务"中选择：

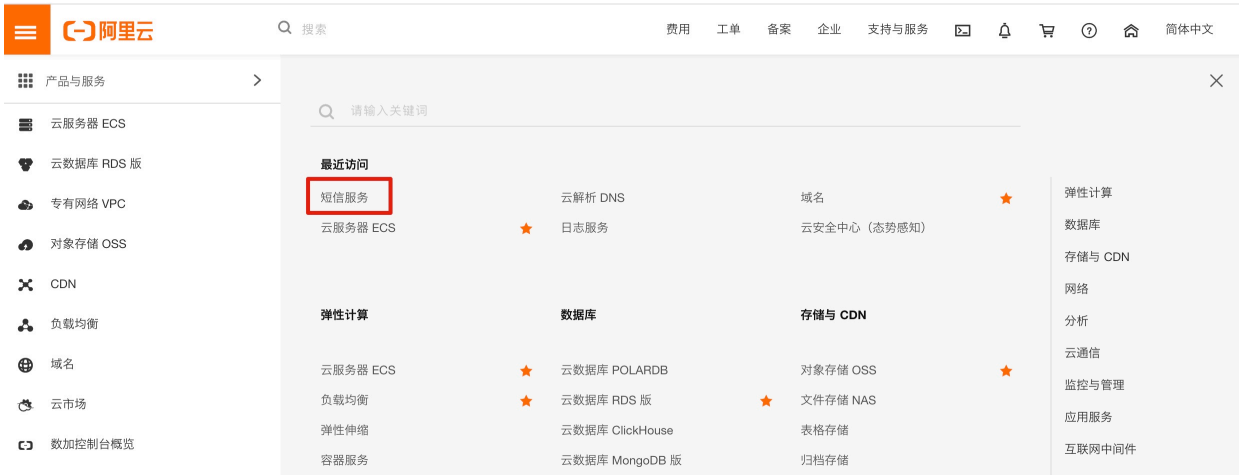

图2.1 访问阿里云短信服务

- 获取key和sceret

图2.2 短信服务的key

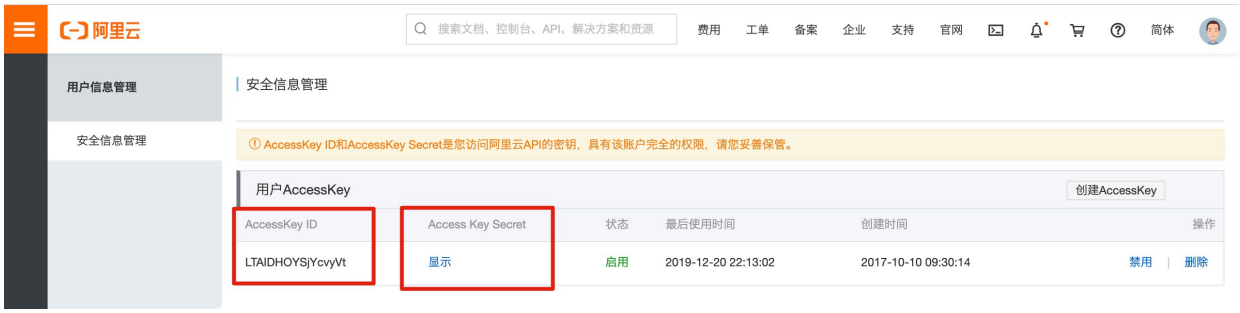

图2.3 获取短信服务的key

- 获取签名

图2.4 获取短信服务的签名

- 获取模板代码

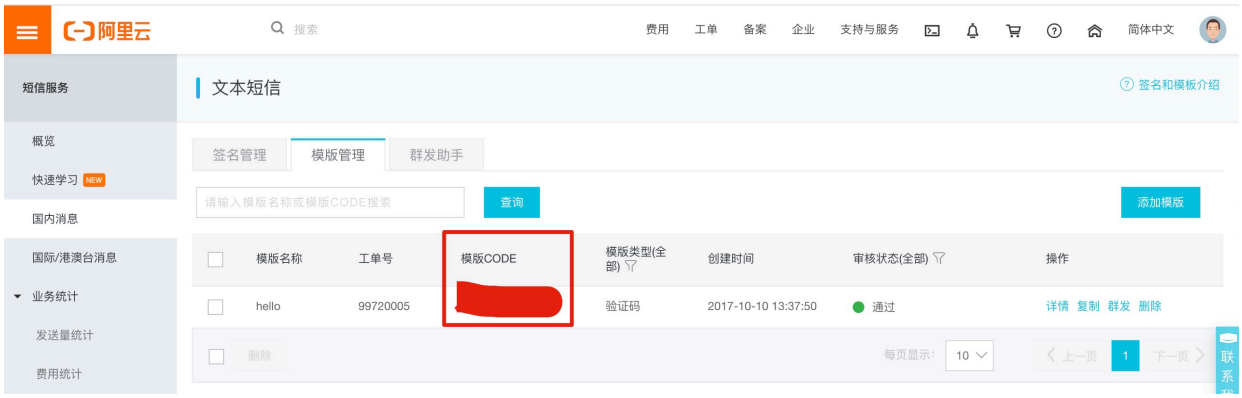

图2.5 获取短信服务的模板代码

2.安装阿里云短信验证码库

安装阿里云SDK核心库，在虚拟开发环境里，执行：

```
pip install aliyun-python-sdk-core
```

在安装完成后，您可以使用OpenAPI Explorer来生成相关API的Demo并应用在您的项目中。

- 短信验证码类

```
import random
from aliyunsdkcore.client import AcsClient
from aliyunsdkcore.request import CommonRequest

ACCESS_KEY_ID = "xxx"  #用户AccessKey  需要根据自己的账户修改
ACCESS_KEY_SECRET = "xxxxxx"  #Access Key Secret  需要根据自己的账
户修改

class SMS:
    def __init__(self,signName,templateCode):
        self.signName = signName  #签名
        self.templateCode = templateCode  #模板code
        self.client = client = AcsClient(ACCESS_KEY_ID,
ACCESS_KEY_SECRET, 'cn-hangzhou')

    def send(self,phone_numbers,template_param):
        request = CommonRequest()
        request.set_accept_format('json')
        request.set_domain('dysmsapi.aliyuncs.com')
        request.set_method('POST')
        request.set_protocol_type('https')  # https | http
        request.set_version('2017-05-25')
        request.set_action_name('SendSms')

        request.add_query_param('RegionId', "cn-hangzhou")
        request.add_query_param('PhoneNumbers', phone_numbers)
        request.add_query_param('SignName', self.signName)
        request.add_query_param('TemplateCode',
self.templateCode)
        request.add_query_param('TemplateParam', template_param)
        response = self.client.do_action_with_exception(request)
        return response


if __name__ == "__main__":
    # 使用自己的签名（xxx）和模板代码(xxxxxx)
    sms = SMS("xxx","xxxxxx")
    phone = input("请输入手机号：")
    #验证码
    num = random.randint(10000,99999)
```

```
    # 设置模板参数
    para = "{'number':'%d'}"%num

    # 发送验证码
    res = sms.send(phone,para)
    print(res.decode('utf-8'))
```

3.实现

```
#视图函数
@app.route("/send/",methods=["GET","POST"])
def send_sms():
    print(1111)
    para = "{'number':'122'}"
    session.permanent = True
    app.permanent_session_lifetime = timedelta(days=1)
    session['sms'] = '122'
    res = sms.send("15116905290",para)
    return str(res)
```

前端页面

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="{{ url_for('register') }}" method="post">
    {{ form.csrf_token }}
    手机号:  <input type="tel" name="phone" id="phone">
    {% for error in form.phone.errors %}
        <span>{{ error }}</span>
    {% endfor %}
    <br>

    <input type="text" name="sms"> <input type="button" value="发送短
信验证" id="send">
     {% for error in form.sms.errors %}
        <span>{{ error }}</span>
```

```
    {% endfor %}
    <br>
    <input type="text" name="code">
    <img src="{{ url_for('verify_code') }}" alt="" onclick="this.src=
{{ url_for('verify_code') }}+'?'+Math.random()">
     {% for error in form.code.errors %}
        <span>{{ error }}</span>
    {% endfor %}
    <br>
    <input type="submit">
</form>
</body>
</html>
<script src="https://cdn.bootcss.com/jquery/1.11.3/jquery.min.js">
</script>
<script>
     $("#send").click(function () {
        num = 5;
        $(this).attr('disabled',true);
        $(this).prop('value',""+num+"秒后发送");
        let _this = this;  //this表示当前按钮
        var timer = setInterval(function () {
            num -= 1;
            if (num <0){
                $(_this).attr('disabled',false);
                $(this).prop('value',"发送验证码")
                return
            }
            $(_this).prop('value',""+num+"秒后发送");
            console.log(num)
        },1000);
        let phone = $("#phone").val();
        let csrf = $("[name='csrf_token']").val();
        // 发送短信验证码
        //ajax
        $.post("/send/",{'phone':phone,'csrf_token':csrf},function
(data) {
            console.log(data)
        })

    })
</script>
```

2.邮件发送

- 说明：flask-mail专门用于邮件发送的扩展库，使用非常方便。
- 安装：`pip install flask-mail`
- 使用：

```
from flask_mail import Mail, Message
import os

# 邮件发送配置，一定要放在创建Mail对象之前
# 邮件服务器配置
app.config['MAIL_SERVER'] = "smtp.126.com"
# 用户帐号
app.config['MAIL_USERNAME'] = "landmark_csl@126.com"
# 授权码
app.config['MAIL_PASSWORD'] = "land123"

# 创建发送邮件的对象
mail = Mail(app)

@app.route('/send/')
def send():
    # 创建邮件消息对象
    msg = Message('账户激活',
                  recipients=['shuai_fmzj@163.com'],
                  sender=app.config['MAIL_USERNAME'])
    msg.html = '恭喜你，中奖了！！！'
    # 发送邮件
    mail.send(msg)
    return '邮件已发送'
```

- 封装函数发送邮件

```
def send_mail(subject, to, template, *args, **kwargs):
    if isinstance(to, (list,turple):
        recipients = to
    elif isinstance(to, str):
        recipients = to.split(',')
    else:
```

```
        raise Exception('邮件接收者参数类型有误')
    # 创建邮件消息对象
    msg = Message(subject,
                  recipients=recipients,
                  sender=app.config['MAIL_USERNAME'])
    # 将邮件模板渲染后作为邮件内容
    msg.html = render_template(template, *args, **kwargs)
    # 发送邮件
    mail.send(msg)
```

- 异步发送邮件

```
from flask import current_app

# 异步发送邮件任务
def async_send_mail(app, msg):
    # 邮件发送必须在程序上下文
    # 新的线程中没有上下文，因此需要手动创建
    with app.app_context():
        mail.send(msg)

# 封装函数发送邮件
def send_mail(subject, to, template, *args, **kwargs):
    if isinstance(to, list):
        recipients = to
    elif isinstance(to, str):
        recipients = to.split(',')
    else:
        raise Exception('邮件接收者参数类型有误')
    # 创建邮件消息对象
    msg = Message(subject,
                  recipients=recipients,
                  sender=app.config['MAIL_USERNAME'])
    # 将邮件模板渲染后作为邮件内容
    msg.html = render_template(template, *args, **kwargs)
    # 异步发送邮件
    # current_app是app的代理对象
    # 根据代理对象current_app找到原始的app
    app = current_app._get_current_object()
    # 创建线程
    thr = Thread(target=async_send_mail, args=(app, msg))
    # 启动线程
```

```
    thr.start()
    # 返回线程
    return thr
```

- QQ邮件发送额外配置：需要配置QQ邮箱开启smtp服务，然后设置授权码

```
# 邮箱端口
app.config['MAIL_PORT'] = 465
# 使用SSL(加密传输)
app.config['MAIL_USE_SSL'] = True
# 不是QQ邮箱的密码，而是授权码
app.config['MAIL_PASSWORD'] = '授权码'
```

3.分页显示

```
方法：paginate，分页查询
    参数：
        page：当前的页码
        per_page：每页的条数
        error_out：当查询出错时是否报错
    返回值：
        Pagination：分页对象，包含了所有的分页信息
Pagination：
    属性：
        page：当前页码
        per_page：每页的条数，默认为20条
        pages：总页数
        total：总条数
        prev_num：上一页的页码
        next_num：下一页的页码
        has_prev：是否有上一页
        has_next：是否有下一页
        items：当前页的数据
    方法：
        iter_pages：返回一个迭代器，在分页导航条上显示的页码列表，显示不完的时返回None
        prev：上一页的分页对象
        next：下一页的分页对象
```

- 封装分页显示的宏

```
{% macro show_pagination(pagination, endpoint) %}
    <nav aria-label="Page navigation">
        <ul class="pagination">
            {# 上一页 #}
            <li {% if not pagination.has_prev %}class="disabled"
{% endif %}>
                <a href="{% if pagination.has_prev %}{{
url_for(endpoint, page=pagination.prev_num, **kwargs) }}{% else
%}#{% endif %}" aria-label="Previous">
                    <span aria-hidden="true">&laquo;</span>
                </a>
            </li>

            {# 中间页码 #}
            {% for p in pagination.iter_pages() %}
                {% if p %}
                    <li {% if pagination.page == p
%}class="active"{% endif %}><a href="{{ url_for(endpoint, page=p,
**kwargs) }}">{{ p }}</a></li>
                {% else %}
                    <li><a href="#">&hellip;</a></li>
                {% endif %}
            {% endfor %}

            {# 下一页 #}
            <li {% if not pagination.has_next %}class="disabled"
{% endif %}>
                <a href="{% if pagination.has_next %}{{
url_for(endpoint, page=pagination.next_num, **kwargs) }}{% else
%}#{% endif %}" aria-label="Next">
                    <span aria-hidden="true">&raquo;</span>
                </a>
            </li>
        </ul>
    </nav>
{% endmacro %}
```

4.登录管理

- 说明：flask-login是一个专门用来管理用户登录退出的扩展库
- 安装：`pip install flask-login`

- 使用：

```
# 第一步：添加扩展
from flask_login import LoginManager

login_manager = LoginManager()

def config_extensions(app):
    ...
    login_manager.init_app(app)
    # 设置登录端点
    login_manager.login_view = 'user.login'
    # 设置登录信息
    login_manager.login_message = '请先登录，然后才能访问'


# 第二步：继承自UserMixin类(也可以自己实现相关方法)
from flask_login import UserMixin

class User(UserMixin, db.Model):
    ...

# 第三步：实现回调
@login_manager.user_loader
def load_user(uid):
    return User.query.get(uid)
```

- 总结

```
状态切换：
    login_user      # 可以提供记住我的功能
    logout_user     # 退出登录
状态查询：
    is_authenticated    登录状态
    is_anonymous        匿名状态
路由保护：
    login_required      # 保护需要登录才能访问的路由
当前用户：
    current_user        # 哪里都可以使用，在模板中不需要分配
```

5.常用装饰器

使用before_request 和 after_request的方法都非常简单，使用@app.before_request 或者@app.after_request 修饰期望在请求前或请求后执行的函数即可

- before_request

  @app.before_request 也是一个装饰器,他所装饰的函数,都会在请求进入视图函数之前执行

  ```
  @app.before_request
  def before_req1(*args, **kwargs):
      # 如果是login,可以通过白名单
      if request.path == '/login':
          return None
      user = session.get('user_info')
      if user:
          return None
      return redirect("/login")
  ```

- after_request

  被app.after_request修饰过的函数会在请求得到响应后返回给用户前被调用，也就是说，这个时候，请求已经被app.route装饰的函数响应过了，已经形成了response，我们在这个时候做一些操作，flask有一个插件叫 flask-compress，是对响应结果进行压缩的，它就是用after_request的这个机制，在response返回前对数据进行了压缩，如果你有别的想要操作的事情，同样可以使用after_request来完成。

  ```
  @app.after_request
  def process_response1(response):
      # 返回值存在
      print("process_response1走了")
      return response
  ```

6.项目结构

目录结构

```
blog/                   # 项目根目录
    manage.py               # 启动控制代码
    requirements.txt        # 依赖包类表文件
    migrations/             # 数据库迁移目录
```

```
    tests/                  # 测试模块目录
    app/                    # 整个程序目录
        templates/              # 模板文件目录
            common/                 # 通用模板
            email/                  # 邮件模板
            。。。
        static/                 # 静态文件目录
            img/
            css/
            js/
            favicon.ico
        views/                  # 蓝本文件目录
        models.py               # 数据模型文件
        forms.py                # 表单类文件
        settings.py             # 配置文件
        extensions.py           # 扩展文件(存放所有扩展)
        email.py                # 邮件发送功能函数
        __init__.py             # 包文件
```

项目启动

- 根据目录结构，创建相关目录及文件
- 书写配置文件(就是书写各种环境的配置类)
- 使用工厂方法创建应用实例，并初始化配置
- 添加各种扩展(顺便粘贴邮件发送函数)
- 配置蓝本(添加各种蓝本文件，并注册)