Flask

ch-02

1. `/index` 和 `/index/` > flask会重定向
2. 唯一URL
   - `/index` 和 `/index/`其实是对应一个网页
   - 因此`flask`会将"/index"重定向到后者
3. 调试模式

```python
# debug 开启调试模式 - 可以修改代码后不重启服务器
app.run(debug=True)
```

4. 清除Chrome缓存
5. 注册路由 - 视图函数
   - 视图函数会将return值拿到并且自动生成**响应头**

```python
 # @app.route("/hello") # 不使用装饰器注册路由
 def hello():
    return "hello world"

# 另一种路由注册方式
app.add_url_rule("/hello", view_func=hello)
```

6. # TODO: 基于类的视图(即插视图)?
7. .run的参数

```python
# host 指定ip地址0.0.0.0
# port 指定端口
# debug 开启调试模式 - 可以修改代码后不重启服务器 (1. 性能低下 2. 只能使用flask自带服务器 3. 会将错误信息输出
app.run(host="0.0.0.0", port=81, debug=DEBUG)
```

8. 导入配置文件
   1. 将所需要的配置信息放在另一个Py文件中,当做模块导入
   2. 使用`app.config.fromobject()`导入,参数为模块的路劲(注意模块路劲的搜索)
      - 获取配置信息`app.config["参数名"]`
      - 本质上`app.config`是`字典`的子类

    - 所有的配置信息都要大写
    - `app.config.`还有几个导入的api
9. `if name == main`
    1. 主入口文件
    2. 在生产环境中并不需要调用.run方法,而是需要将flsk写的文件导入到`nginx + wsgi`服务器中
    3. `.run`只用于调试
10. 视图对象的返回值 - `response`对象
    - 返回response对象

```python
from Flask import make_response

def hello():
    # 会自动生成响应头 响应行+参数+响应体 始终返回response对象
    # 自定义response对象
    headers = {
        "content-type": "text/plain"
    }
    response = make_response("<html></html>", 404)
    response.headers = headers
    return response
```

    - ⭐视图函数可以有**多个返回值**来自动生成一个response对象
        - `return 响应体 + 状态码 + 参数`

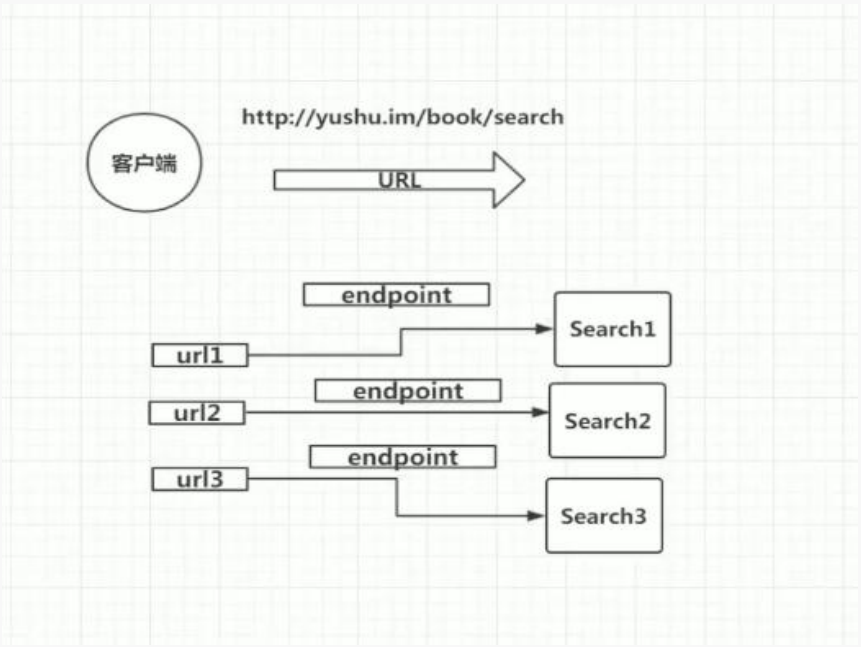
```python
def hello():
    # 会自动生成响应头 响应行+参数+响应体 始终返回response对象
    # 自定义response对象
    headers = {
        "content-type": "text/plain"
    }

    # return 响应体 + 状态码 + 参数
    return "<html></html>", 404, headers
```

ch-03

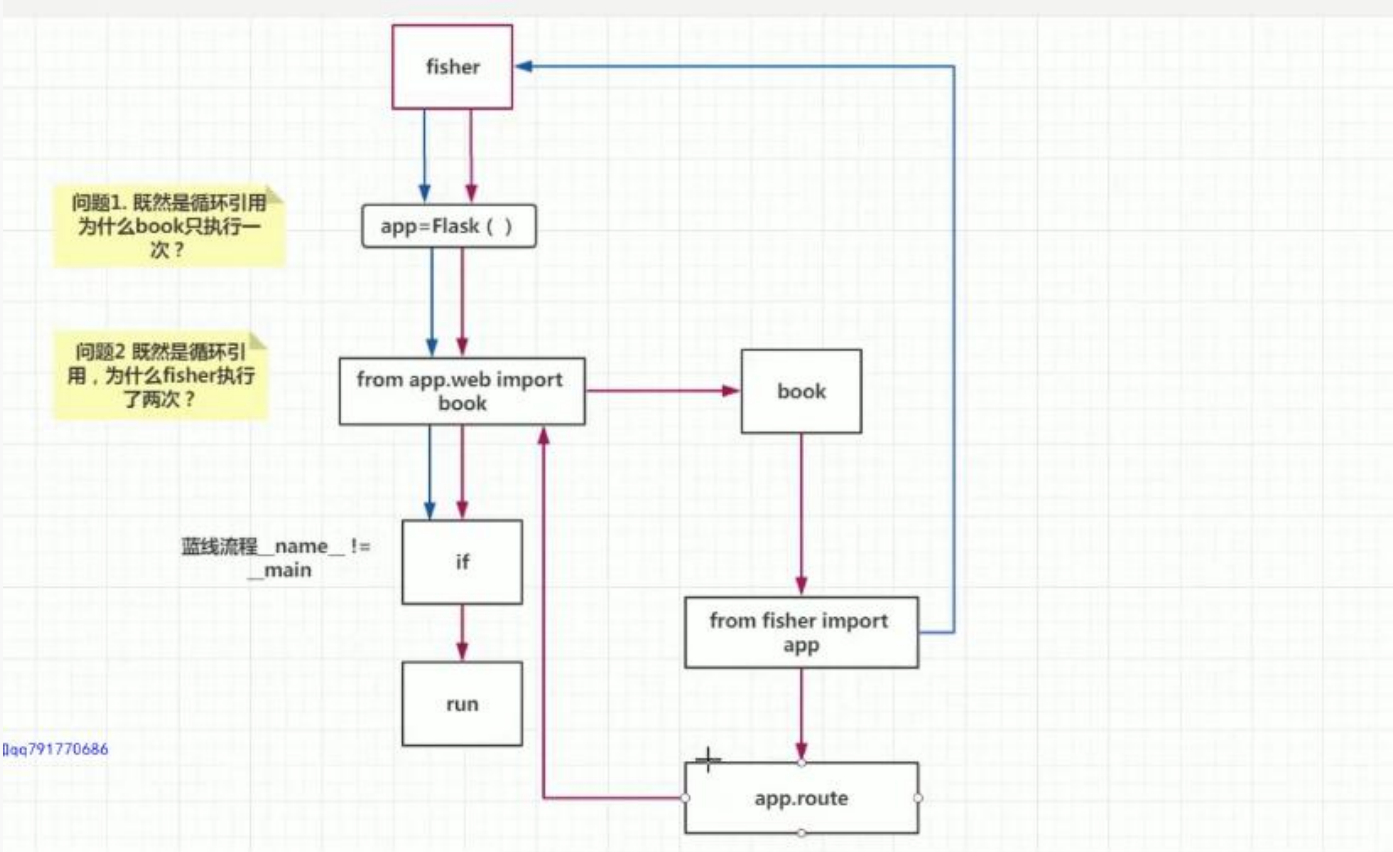
1. 逻辑判断的时候应该将最有可能为**假**的条件放在前面,这样减少资源的消耗
2. 视图函数原则:
    - 是项目的原点,不能含有很多的代码,将逻辑判断提取成函数
3. 读代码的步骤:

- 先看函数,再看细节

4. `restful` 标准 请求某网站api后返回的数据遵从一定的标准(一般含有建json)
5. flask路由的原理(`endpoint` 参数什么意思?)
    - 全局变量`url_map`和`view_function`里面都必须有视图函数的相关信息才算注册成功

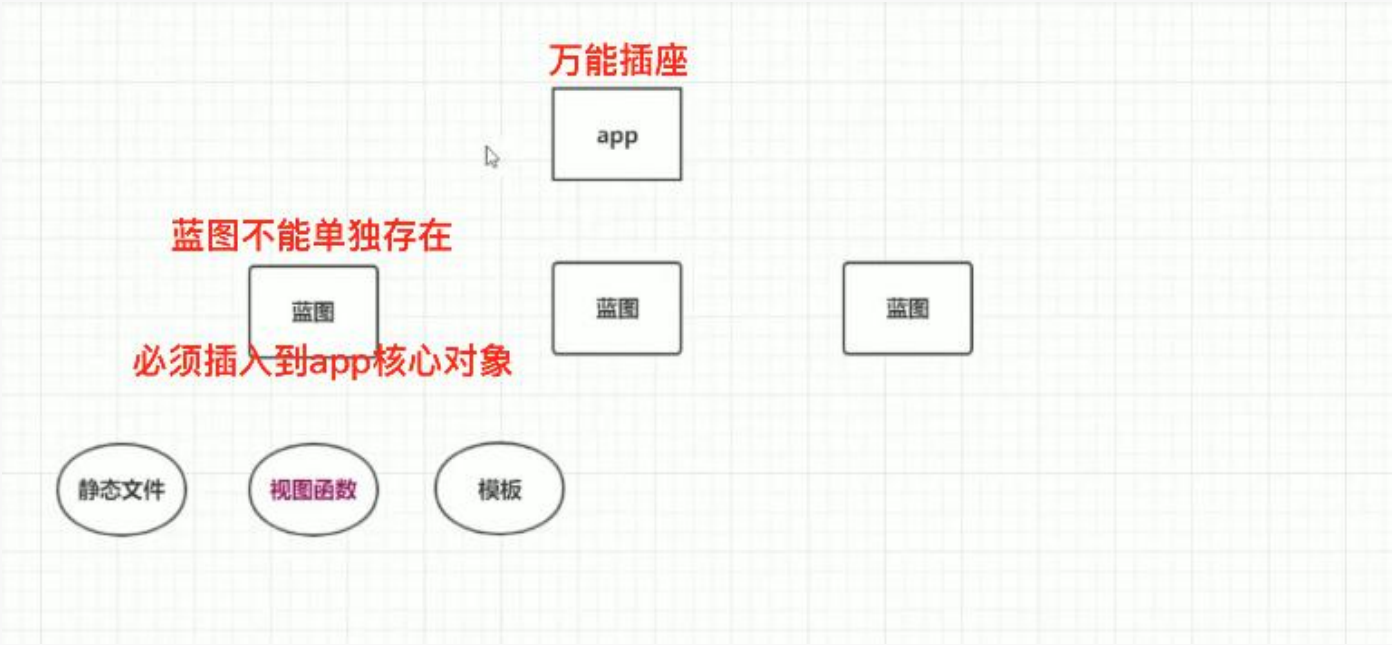


6. API的难点在于如何设计路由
7. 将**视图函数**分解放到不同文件中
    - 业务模型的分门别类
    - 视图函数逻辑不能全都写在视图函数内
    - *循环导入造成问题*: app被初始化了两次
    - 可以使用`id()`来观察实例对象被创建的过程



ch-04

- **蓝图(buleprint)**



  - 设置多个蓝图,插入到核心对象内
  - 将视图函数注册到**蓝图**上,再将蓝图注册到**app核心对象**上
  - 蓝图的使用范围?(web/cms/api才能创建一个蓝图)

- 分层(验证层)
  - 在web应用中很重要的**分层**
  - 对搜索的关键字进行验证就可以使用一个**验证层**
- `wtform`模块可以验证数据**有效性**

  > wtform怎么知道我前端提交的表单对应的哪一个的验证呢?是用`<input name=>`属性么?

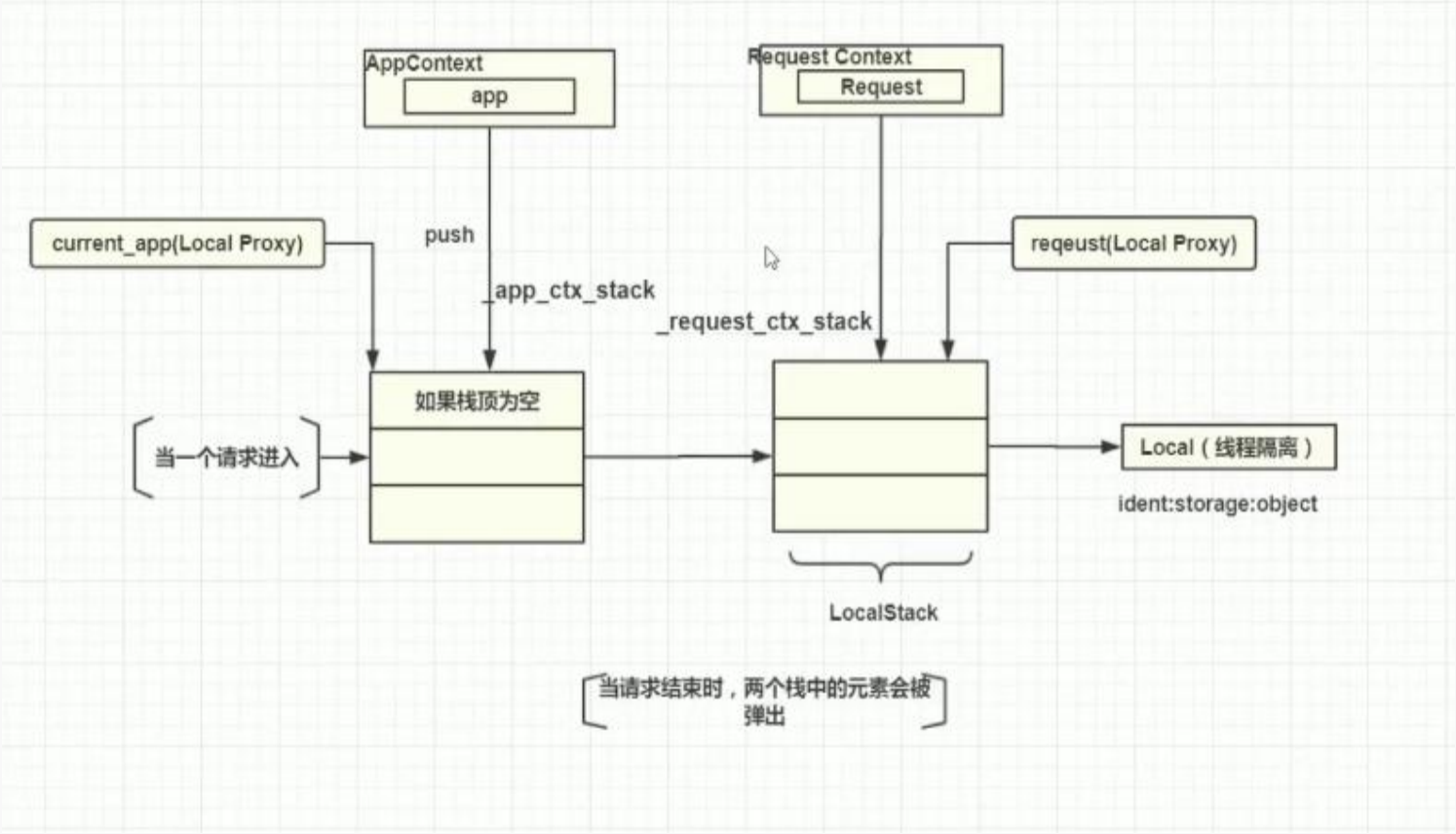
```python
# 验证搜索的参数的合法性
# wtforms 模块可以用在做验证

# 验证内容
from wtforms import Form, StringField, IntegerField

# 验证器 其中最后一个是用来验证空格的
from wtforms.validators import Length, NumberRange, data_required

class SearchForm(Form):
    """创建验证类,可以验证用户输入的搜索关键字的有效性"""

    # q要求长度不为0,并且不能够太长
    # message参数,自定义错误
    q = StringField(validators=[data_required(), Length(min=1, max=30)])

    # page要求正整数, 在1-99
    page = IntegerField(validators=[NumberRange(min=1, max=99)], default=1)
```

- 配置文件
  - `secure.py`
    - 数据库密码等机密的配置可以放在secure里面
    - 还有关于 生产环境/开发环境 不同配置的信息
    - 永远不用上传到git上
  - `setting.py`
    - 一些正常的设置,不需要涉及到机密的信息
    - 比如每页显示多少条数据等
- 数据库的引入
  - 不能依赖以网页的`api`(多次访问会被封ip的)
  - 于是应该将用户访问的内容保存到本地数据库中
  - 下次用户再访问时先从数据库进行查找数据

- 数据库创建
  - `database first` 用图形化界面创建
  - `model first` 用E-R模型创建
  - `code first` 用代码创建数据库 - 数据库的设计应该围绕业务模型
    - 专注业务模型的设计,而不是专注数据库的设计
    - 数据库只是用来存储数据的,它的表关系就是由我们的业务来决定
    - 业务逻辑应该放在`model`里面
  - 使用`sqlalchemy`模块来创建数据库
- `ORM`对象关系映射 Object-Relational Mapping
  - 数据的增删改查

ch-05

- `flask`经典错误之
  - `RuntimeError: Working outside of application context.`
- `LocalProxy`



  - 如果没有request进来的话`AppContext`是不会自己入栈的

- `Flask`中的**上下文**
  - 应用上下文 对象 对Flask核心对象的封装：`AppContext`
  - 请求上下文 对象 对Request的封装：`RequestContext`
  - 一些对象需要操作用到这个对象的外部参数,于是将这些外部参数和核心对象放在一起重新封装成另一个对象.
- 错误解决方法(将AppContext对象推入栈中)

```python
from flask import Flask, current_app
# RuntimeError: Working outside of application context

# 离线应用
# 单元测试
app = Flask(__name__)
# 获得上下文对象
ctx = app.app_context()
# 入栈
ctx.push()
# current_app找到上下文对象,并且根据上下文对象返回对应的app核心对象的引用
a = current_app
d = current_app.config["DEBUG"]
ctx.pop()
```

  - AppContext还支持**上下文管理**

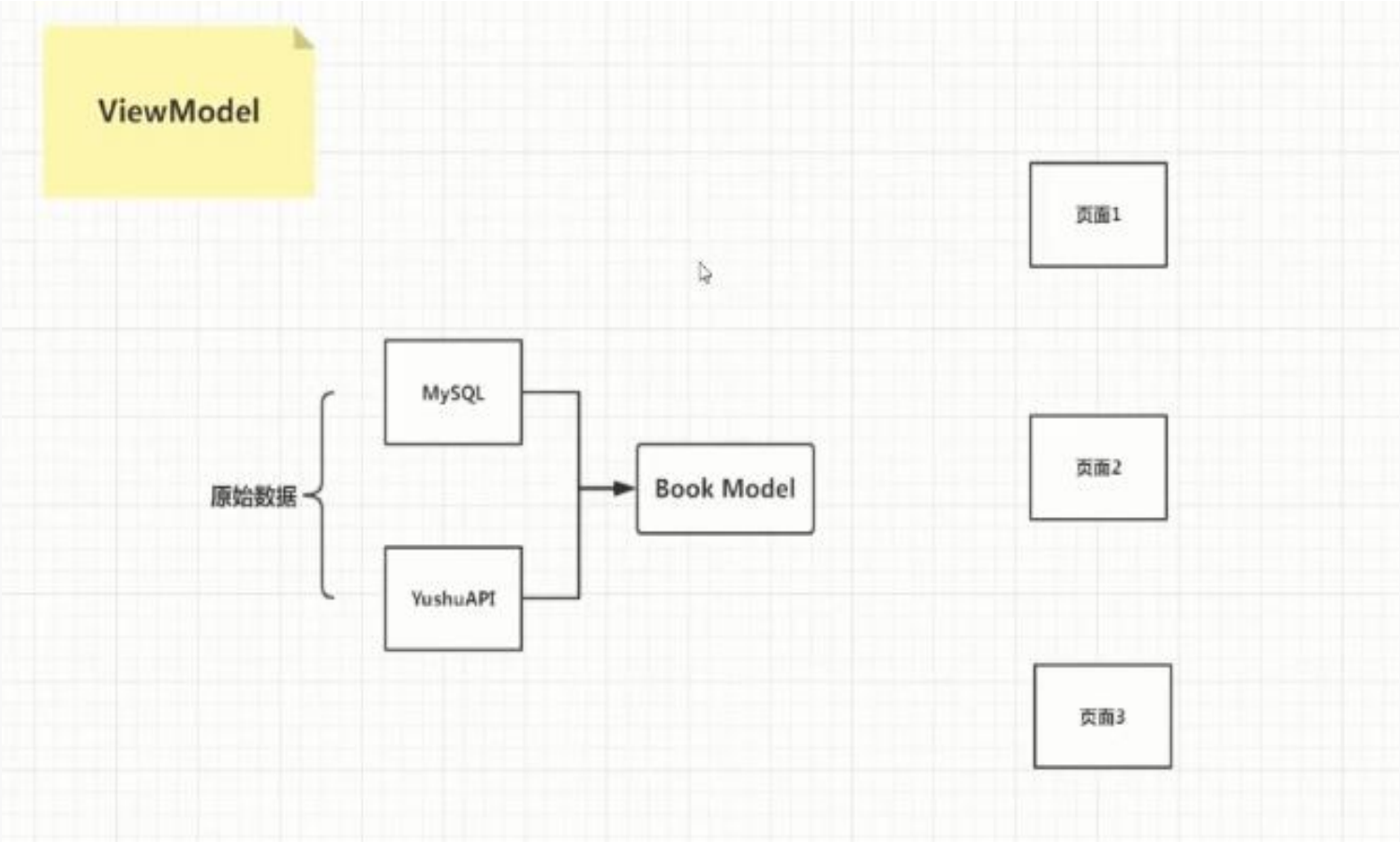
```python
from flask import Flask, current_app

app = Flask(__name__)

with app.app_context():
    a = current_app
    d = current_app.config["DEBUG"]
```

ch-06

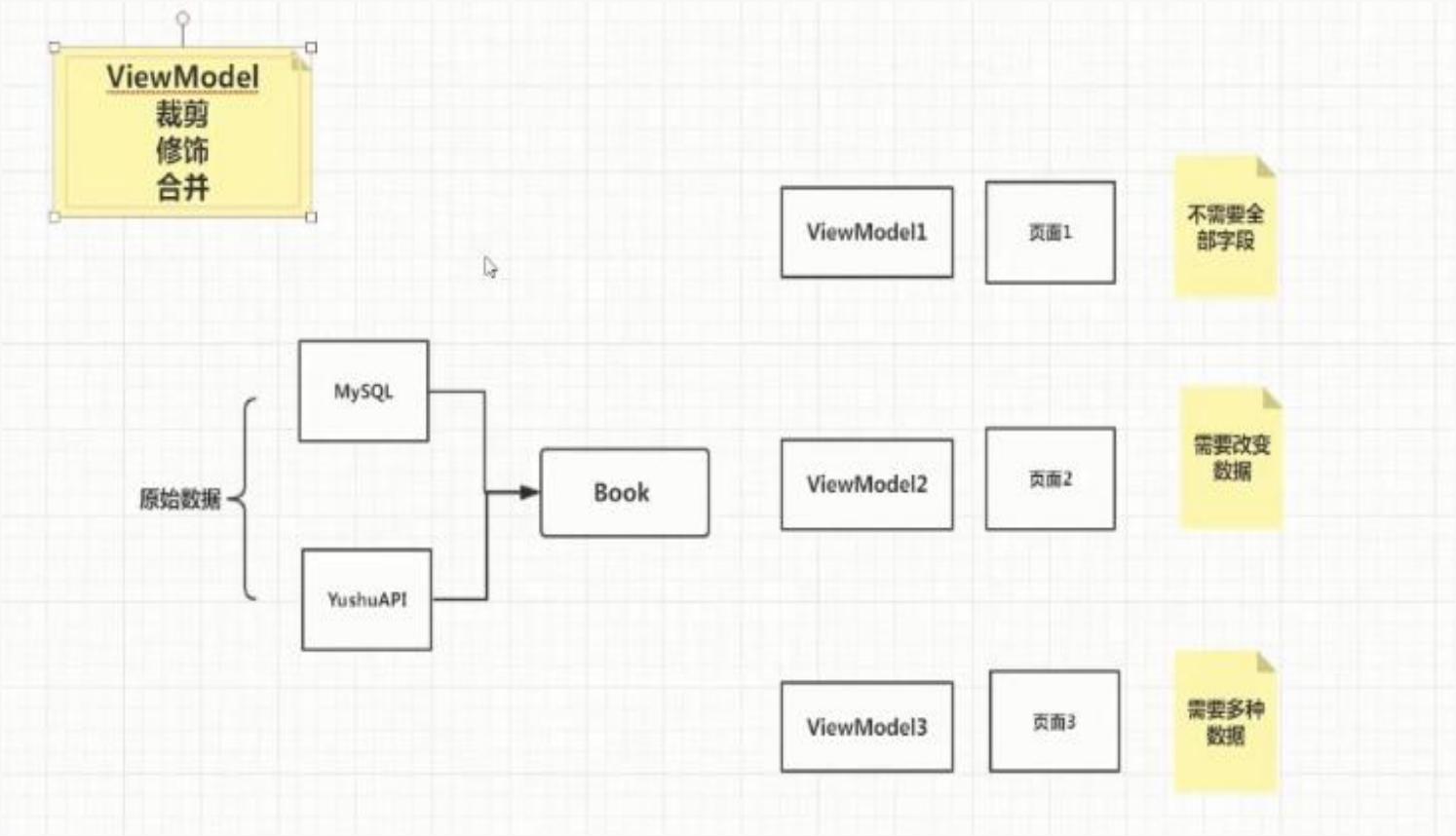
进程和线程

ch-07

- ViewModel
  - 页面需要的数据并不是**原始数据**，要根据**业务**来处理原始数据



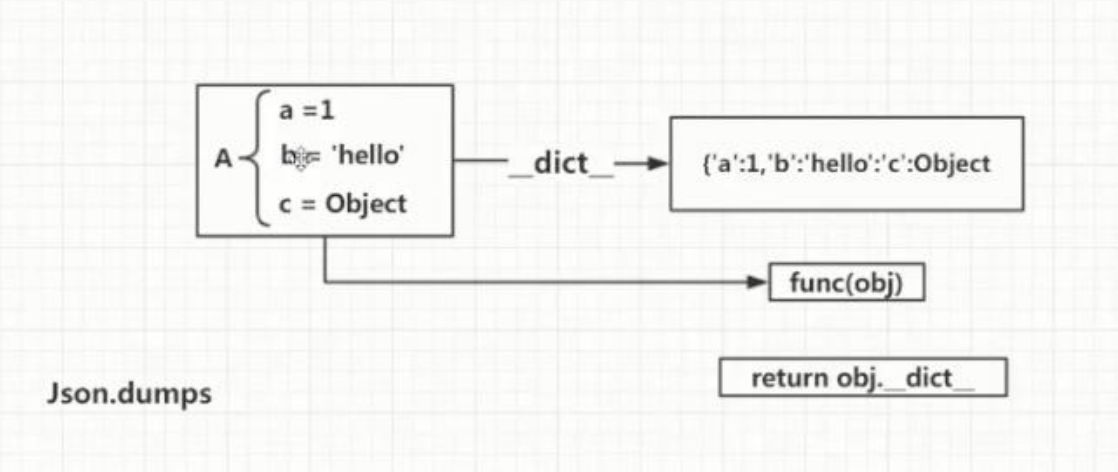
- 使用`ViewModel`来处理原始数据



- 类和伪类

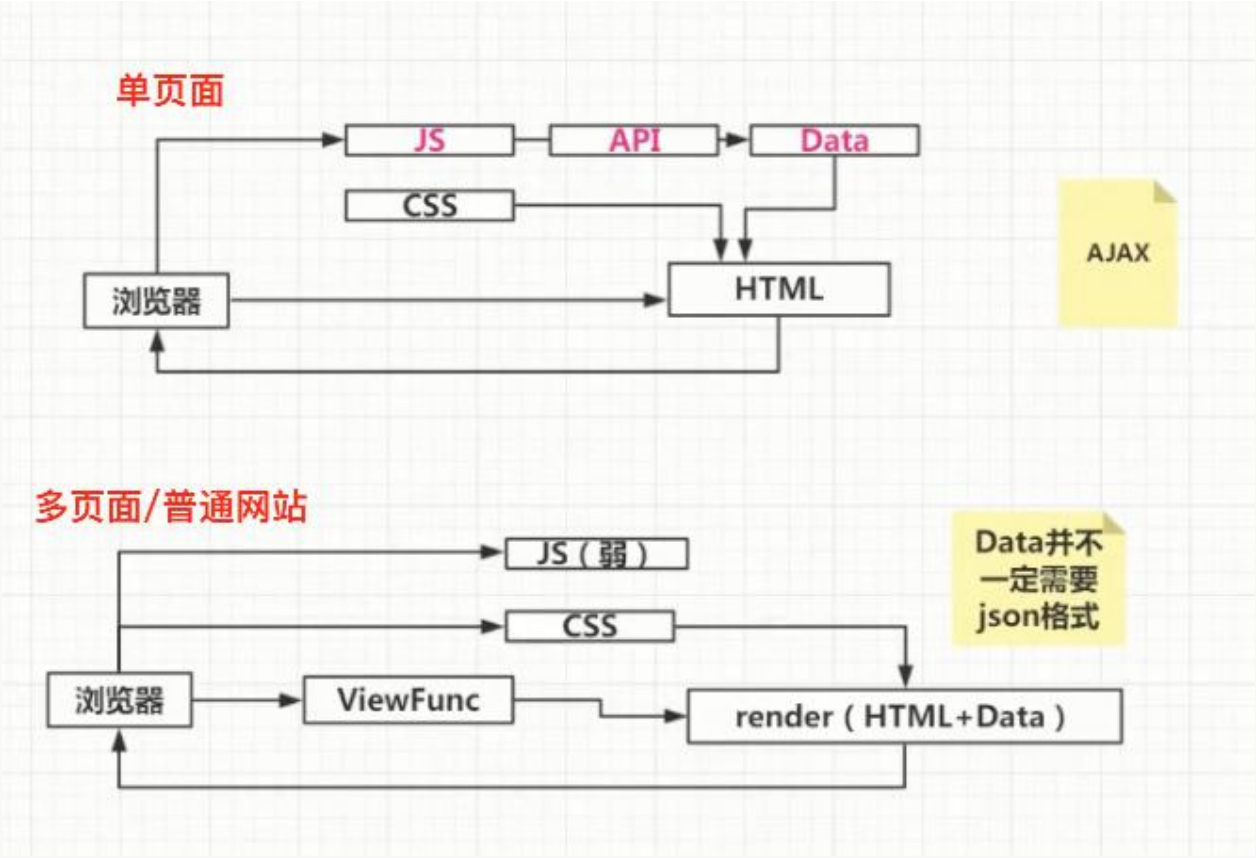
- 如果一个类大量的含有类方法和静态方法,则可以称之为伪类
- 如何真正的面向对象?

- 返回值的问题? - 如何将一个对象jsonfy()?
  - `jsonfy(object.__dict__)` ?
  - 如果 `__dict__` 下面包含了不能被序列化的**其他的对象**,则会序列化失败
  - 于是应该再访问**其他的对象.dict**属性再去序列化




```python
import json

# 遇到不能进行序列化的对象再访问其__dict__方法
json.dumps(books, default=lambda x: x.__dict__)
```

- HTML模板
  - 单页面与多页面的不同
  - 单页面是在**客户端**进行数据的渲染
  - 多页面则是在**服务器**进行数据的渲染



> 暂且不是特别明白

ch-08

- `static`文件夹默认放在`Flask(__name__)`下,可以直接访问静态文件
  - 如果要改变静态文件夹的话可以在实例化`Flask`核心对象时传入参数

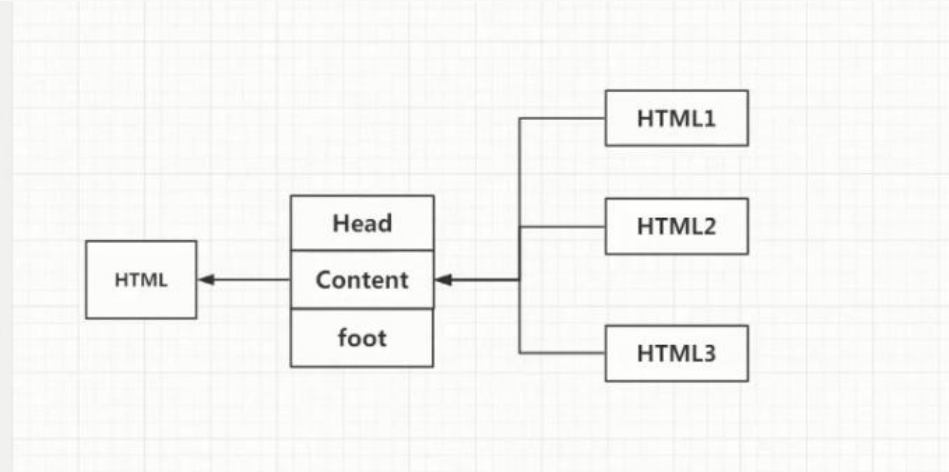
```python
app = Flask(__name__, static_folder="folder_name")
```

- **模板**(主要是前段写的,想做全站的话一定要学啊)
  - 就是HTML文件
  - `render_template`模块来处理模板
  - `jinja2`
- `jinja2`模板引擎
  - 一种**模板语言**
- `jinja2`的注释`{#jinja2#}`
- `jinja`的流程控制

```Python
{{# if 条件 #}}
{% if data.age < 18 %}
    {{ data.name }}
{% elif data.age == 18 %}
    do something
{% else %}
    {{ data.age }}
{% endif %}
```

```Python
{{# for 循环 #}}
{% for key, value in data.items() %}
{{ key }}
{{ value }}
{% endfor %}
```

- 模板的继承(直接看代码)




```Python
{ block content }
    {{ # 如果没有super()则会直接覆盖块中原先的内容 # }}
    {{ super()} }
    {% if data.age < 18 %}
        {{ data.name }}
    {% elif data.age == 18 %}
        do something
    {% else %}
        {{ data.age }}
    {% endif %}
{ endblock }
```

- `jinja`过滤器`|`
    - 其行为有点像**管道**

```Python
{{#只有在data.name变量不存在时,default内容才会显示#}}
{{#除非给default添加第二个参数 True#}}
{{ data.name | default("未名") }}

{{#将data的值传递给后面的函数, 有点像管道#}}
{{ data | length() }}
```

- `url_for` 反向生成 `URL`
  - 原理是使用 `endpoint` 反向构造

```Python
<link rel="stylesheet" href="{{ url_for('static', filename="test.css") }}">
```

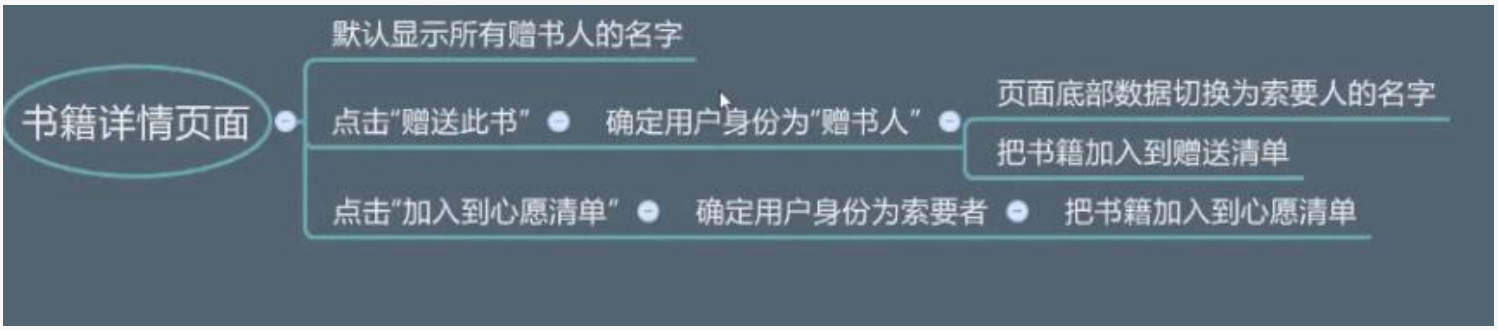
- 在模板中定义变量

```Python
{% set message = get_flashed_messages() %}
```

- 消息闪现机制
  - `from flask import flash`
  - 可以再模板中获得消息闪现 `get_flashed_messages()`
  - 得到的是数组
- `session` = 秘钥
  - 在 `secure.py` 中配置 `session` 的时候尽量保证独一无二

ch-09

- 开始业务逻辑
- 赠送图书的业务逻辑



- 运用code-first的思想去设计数据库
  - 如何在某个字段中表示与另一张表的**关系**
  - 在数据库中我们可以使用**外键**来约束表之间的关系

- 在Python中可以使用`SQLAlchemy.orm`中的`relationship()`

```python
from app.models.base import db
from sqlalchemy import Column, Integer, Boolean
from sqlalchemy.orm import relationships

class Gift(db.Model):
    id = Column(Integer, autoincrement=True, primary_key=True)
    # 将Gift表和user表连接起来再设置外键约束
    user = relationships("User")
    uid = Column(Integer, ForeignKey(user.id))      # 表示该书籍是否送出去了
    launched = Column(Boolean, default=False)
```

- 表的基类 每一张表都需要一个表示逻辑删除的字段

```python
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy import SmallInteger, Column

db = SQLAlchemy()

class Base(db.Model):
    # 告诉sqlalchemy这是一个基类,不需要创建表
    __abstract__ = True
    # 逻辑删除字段
    status = Column(SmallInteger, default=1)
```

- 使用orm来操作数据库
  - 添加数据到数据库

```python
# 添加orm
db.session.add(ORM对象)
# 确认提交
db.session.commit()
```

  - 查询数据库之**自定义验证器**

```python
def validate_email(self, filed):
    """检查数据库是否有匹配的数据, wtform知道要校验的是哪一个数据"""
    if User.query.filter_by(email=filed.data).first():
          # User是orm对象,其query方法可以查询数据,filter_by则是添加查询条件(参数是其字段),first返回第一个
          # 注意,该方法返回的是另外一个User对象(根据filter匹配到的)
        raise ValidationError("电子邮箱已经被注册")
```

- `cookie` - 用户携带的票据(包含有效期)

- 当用户登录网页的时候可以传入携带的`cookie`,服务器就能校验用户的身份

> 也可以用于广告的精准投放

- `cookie`失效
  1. 过了有效期
  2. 如果是一次性的则关闭浏览器就没了
- `flask`中的`flask-login`插件能够帮助管理`cookie`

  > 在使用前必须在创建核心对象时导入插件(就如同插入蓝图和数据库一样)

  ```python
  from flask_login import login_user

  # user是经过校验的orm对象
  # 而且必须在该对象中实现了get_id()的方法,然后login_user就能够通过这个方法获取到用户的id
  # 由于用户的id值是唯一的,因此可以用作生成独一无二的cookie
  login_user(user, remember=True, duration=datetime.timedelta(days=15))
  ```

  > 传入`login_uer()`的orm对象必须继承自`flask_login`的`UserMixin`类
  >
  > 其中定义了许多属性和方法是专门用来设置`cookie`的

- 什么时候用到`cookie`?
  - 当用户访问需要登录才能操作的页面时
  - `cookie`可以用作权限管理

  ```python
  # 如何限定一个视图函数,当用户登录时才能访问呢?
  from flask_login import login_required

  @web.route('/my/gifts')
  @login_required
  def my_gifts():
      """需要先验证用户是否登录"""

      return "My Gifts"
  ```

> 还要实现`get_user()`方法见代码
>
> 还有权限分级的知识

- 重定向攻击

ch-10

- 用户添加一本书到心愿单的逻辑(麻烦)
- 引入积分的思想