FLASK框架

参考资料

- 官方文档http://flask.pocoo.org/

安装Flask-Script插件

- 满足从终端接受参数类似django中的终端启动
- 安装步骤
    - pip install Flask-Script
    - 创建Manager对象
        - manage = Manager(app)
    - 然后可以在终端使用命令
        - python manage.py runserver --help
        - python manage.py shell
    - 启动服务器后会产生环境不匹配警告只需在环境变量中添加FLASK_ENV=development字段即可，可通过代码控制environ["FLASK_ENV"] = "development"

文件拆分

- 安装flask-blueprint插件（对 路由的管理）
    - pip install flask-blueprint
    - 创建蓝图对象、向app中注册蓝图（可以有多个不同的蓝图进行文件的拆分）

      blue = Blueprint（"blue",__name__）、app.register_blueprint(blue)

- 安装flask-sqlalchemy
    - pip install flask-sqlalchemy
    - 通过app构建SQLAlchemy对象
    - 配置
        -

```
配置到APP上：
app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:///sqlite.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

连接数据库基本格式：
# app.config['SQLALCHEMY_DATABASE_URI'] = "mysql+pymysql://root:102487@localhost:3306/FLASK"
```

- 迁移系统
    - pip install flask-migrate
    - 创建migrate=Migrate对象
    - 关联app和db对象
        - migrate.init_app(app=app,db=db)

视图

- route规则
    - 写法：
        - <converter:variable_name>
        - ```
          @blue.route("/args/<uuid:name>/")
          @blue.route("/args/<any():name>/")
          ```
    - converter类型：
        - string：接收任何没有斜杠（‘/’）的文件（默认）
        - int：接受整形
        - float：接受浮点型
        - path：接受路径，与string区别：path可以接受“/”
        - uuid：接受uuid字符串
        - any：可以同时指定多种路径，进行限定
- 请求方法限定

```
@app.route(‘/rule/’,methods=[‘GET’,’POST’])
   def hello():
      return ‘LOL’
知识点：methods取值
   GET、POST、HEAD、PUT、PATCH、DELETE
```

- 反向解析
    - url_for("函数名"，参数名=value)
```

双R

- Request：由Flask框架创建，Request对象不可修改

```
url    完整请求地址
base_url    去掉GET参数的URL
host_url    只有主机和端口号的URL
path       路由中的路径
method     请求方法
remote_addr 请求的客户端地址
args       GET请求参数：类型ImmutableMultiDict
form       POST请求参数:类型ImmutableMultiDict
files      文件上传
headers    请求头
cookies    请求中的cookie

操作：
ImmutableMultiDict类型获取方式：
dict['uname'] 或 dict.get('uname)
获取指定key对应的所有值
dict.getlist('uname')
```

- Response对象：由程序员创建

1. 直接返回Response对象
2. 通过make_response（data,code）
   - data 返回的数据内容
   - code 状态码
3. 返回文本内容，状态码
4. 返回模板（本质和3一样）

重定向

redirect() url_for(‘函数名’,参数=value) 例如：
redirect(url_for("blue.index")) 带参数：
redirect(url_for("blue.getid",uid=1,name="fanding"))

返回json数据

- jsonify(字典)

终止执行

- 主动终止
    - abort(code)
    - abort(Response(response="hellow",status=200))

捕获异常

```
@app.errorhandler(404)
  def hello(e):
    return 'LOL'
```

@app.errorhandler(404)、@blue.errorhandler(404)、@blue.app_errorhandler(404)区别如下图：

补充说明

当我们不是使用的工厂模式创建app时，app.errorhandler(401)，即可捕捉全局401状态；若使用了create_app方式创建app，则无法进行捕捉，若想捕捉，可以在蓝图中写，如admin.errorhandler(401)，即捕捉admin蓝图下所有401状态码，admin.app_errorhandler(401)，则是捕捉的全局的401状态码，即其他蓝图中的401状态，也会被捕捉，进行处理

Flask有四大内置对象

- Request
    - request：请求对象
- Session
    - session：服务器端会话对象
- G
    - g：相当于一个全局对象
    - 作用：视图函数之间数据传递
    - 模板也能接收
- Config
    - 在模板中 config：app的配置信息
    - 在python代码中 app.config：app的配置信息
    - debugtoolbar：可能用此实现

消息闪现

- flash(“登录错误”):直接往模板中抛入消息
- flash获取：get_flashed_messages（）返回消息列表

会话技术

cookie

客户端端的会话技术

cookie本身由浏览器保存，通过Response将cookie写到浏览器上，下一次访问，浏览器会根据不同的规则携带cookie过来

```
response.set_cookie(key,value[,max_age=None,exprise=None)]
request.cookie.get(key)
```

cookie不能跨域名

cookie不能跨浏览器
response.set_cookie(key,value,max_age=None,exprise=None)

max_age: 整数，指定cookie过期时间
expries : 整数，指定过期时间，可以指定一个具体日期时间

max_age和expries两个选一个指定

过期时间的几个关键时间

过期的时的几个关键的时
max_age 设置为 0 浏览器关闭失效
设置为None永不过期

删除cookie
    response.delete_cookie(key)

注意：flask中的cookie默认对中文等进行了处理，直接可以使用中文，而django中不支持在cookie中直接写中文。

session

服务器端会话技术,依赖于cookie

默认存在客户端的cookie中
常用操作:
    内置对象直接用:
    session[‘key’] = ‘value’
    get(key,default=None) 根据键获取会话的值
    pop(key) 删除某一值
    clear()   清除所有

flask对存储在cookie中的数据做了如下处理:

    1、将session存储在cookie中
    2、对数据进行序列化
    3、还进行了base64

3、还进行了base64
4、还进行了zlib压缩
5、还传递了hash

默认存储时间是31天

使用redis存储session

```
安装:
    pip install flask-session
配置:
    配置文件中配置:
    SESSION_TYPE = 'redis'
    SESSION_REDIS =
redis.Redis(host='www.donogh.cn',port=6379,password='XXX')
ext文件:
    Session(app)
```

Token

待补充

模板

- templates和static文件配置

在创建蓝图的时候：
在创建蓝图的时候配置：
blue=Blueprint("blue",name,static_folder="../static/",template_folder="../templates/"):有时候不生效
在创建app的时候配置：
app = Flask(__name__,static_folder="../static",template_folder="../template")

- 模板中使用反向解析
  - href="{{ url_for("blue.get_students") }}?page={{ page }}&per_page={{ per_page }}"

分页处理

模板代码

```
{% block content %}
   <div class="container">
      <table class="table">
         <tr>
            <td>id</td>
            <td>姓名</td>
            <td>年龄</td>
         </tr>
         {% for student in pagination.items %}
```

```
            <tr>
                <td>{{ student.id }}</td>
                <td>{{ student.s_name }}</td>
                <td>{{ student.s_age }}</td>
            </tr>
        {% endfor %}



    </table>
    <nav aria-label="Page navigation">
        <ul class=pagination>
            {%- for page in pagination.iter_pages() %}
                {% if page %}
                    {% if page != pagination.page %}
                        <li>
                            <a href="{{ url_for("blue.get_students") }}?page={{ page
}}&per_page={{ per_page }}">{{ page }}</a>
                        </li>
                    {% else %}
                        <li class="active"><a href="#">{{ page }}</a></li>
                    {% endif %}
                {% else %}
                    <li><span class=ellipsis>…</span></li>
                {% endif %}
            {%- endfor %}
        </ul>
    </nav>
  </div>
{% endblock %}
```

视图函数代码：

```
def get_students():
    pagination = Student.query.paginate()
```

```
    pagination = Student.query.paginate()
    return
render_template("students.html",pagination=pagination,per_page=4)

注意：返回的pagination对象是当前页对象
```

与Django中的区别

- if标签：支持简单的算数表达式如{% if 1==2 %}　{% endif %}
- for循环有了else结构
- 有了宏定义

模板语法

```
变量：
    {{ var }}
    来源：
        视图函数传递过来的参数
        前面定义出来的变量
        如果不存在默认不显示
标签：
    格式：{%  %}
    类型：
    结构标签：
        block、extends、include
        {{ super（）}}
    宏定义：
    marco：
        {% marco hello（name）  %}
            {{name}}

        {% endmarco %}
```

{% endmarco %}

导入：

{% from ‘xxx’ import ‘xxx’ %}

控制标签：

for标签：

{% for item in cols %}

AA

{% else %}

BB

{% endfor %}

获取循环信息：

也可以获取循环信息 loop

loop.first

loop.last

loop.index loop.index0

loop.revindex   loop.revindex0

if标签：

{% if a == b % }

{% endif % }

过滤器：

{{ 变量|过滤器|过滤器... }}

capitalize

lower

upper

title

trim

reverse

reverse
format
striptags 渲染之前，将值中标签去掉
safe
default
last
first
length
sum
sort

模板中资源定位类型：
herf = {% url_for("static"， filename="css/index.css") %}
herf = "/static/css.index.css"

反向解析带参数：
{{ url_for("blue.market_with_params",typeid=typeid ,childcid=childcid ,order_rule=order_rule.1) }}

模型

连接处理

项目中数据库优化

- 怎么连接
- 连接多个少

- Django和Flask
    - 默认都是有数据库连接池的
    - 可以定义连接池的大小，默认mysql可同时被100个客户端同时连接
    - 原理

> 运用了懒加载模式，当有请求过来，如果有闲置的连接，那么会使用闲置的连接操作数据库，如果没有闲置的连接，创建一个新的连接，前提是不超出连接池的大小，如果工作的连接等于了连接池的大小，再有请求过来，只能排队等待，只要有闲置的连接就会处理在等待中的请求。

注意

- 修改数据表名
    - __tablename__ = "axf_main"
- 模型继承注意事项
    - 父类添加属性__abstract__ = True

与Django区别

- 与Django的区别
    - 手动添加主键，主键默认自增
    - flask中没有了想django中一样的元类

模型操作

- flask-sqlalchemy、sqlalchemy官方文档
- 数据库连接的一般格式
    - dialect+driver://username:password@host:port/database
    - dialect：数据库类型如：mysql
    - sqlalchemy配置见文件拆分

- 操作

```
创建数据库表：db.create_all()
删除数据库表：db.drop_all()

添加数据：
db.session.add(object)
db.session.add_all(list[object])

删除数据：
db.session.delete(object)

修改数据：
修改后重新提交保存就可以

查询数据：
获取查询集：
   all()：特殊--->返回列表类型
   filter(类名.属性名.运算符('xxx'))
   filter(类名.属性 数学运算符  值)

运算符：
contains

startswith
```

startswith
endswith
in_
like
__gt__
__ge__
__lt__
__le__

筛选：
filter（）：
filter_by(name = “fanding”)：精准获取一般用在级联查询，省略模型名和用=代替==
offset()：
limit()：和offset一起使用时，顺序不是问题，都是先执行offset然后执行limit
order_by()
get():只能传入主键值，找不到返回None
get_or_404():找不到抛出404异常
first()：db.query.first()
paginate()

逻辑运算：
与
  and_
  filter(and_(条件),条件...)

或
  or_
  filter(or_(条件),条件...)

非

```
非
    not_
    filter(not_(条件),条件...)
```

声明外键关系

```
电影表1：m购买关系
电影表：
class Movie(BaseModel):
    __tablename__ = "movies"
    showname = db.Column(db.String(64))
    shownameen = db.Column(db.String(128))
    director = db.Column(db.String(64))
    leadingRole = db.Column(db.String(256))
    type = db.Column(db.String(64))
    country = db.Column(db.String(64))
    language = db.Column(db.String(64))
    duration = db.Column(db.Integer, default=90)
    screeningmodel = db.Column(db.String(32))
    openday = db.Column(db.DateTime)
    backgroundpicture = db.Column(db.String(256))
    flag = db.Column(db.Boolean, default=False)
    is_delete = db.Column(db.Boolean, default=False)

    hall_movies = db.relationship('HallMovie', backref='Movie',
lazy=True)


    购买关系：

    class HallMovie(BaseModel):
```

```
class HallMovie(BaseModel):

    h_movie_id = db.Column(db.Integer, db.ForeignKey(Movie.id))

    h_hall_id = db.Column(db.Integer, db.ForeignKey(Hall.id))

    h_time = db.Column(db.DateTime)
```

- 数据类型

```
Integer
SmallInteger
BigInteger
Float
Numeric
String
Text
Unicode
Unicode Text
Boolean
Date
Time
DateTime
Interval
LargeBinary
```

- 常用约束

```
primary_key
autoincrement
unique
index
nullable
default
ForeignKey()
```

高级

钩子函数

- 可以为app添加@app.before_request-->类似django中的中间件
- 可以为blue添加@blue.before_request
- 四种钩子函数
    - before_first_request
        - 在处理第一个请求之前运行
    - before_request
        - 在处理每次请求之前运行
    - after_request
        - 在每次请求之后运行，注册的函数至少有一个参数，这个参数实际上为服务器的响应，且函数中需要返回这个响应参数

- teardown_request
    - 注册一个函数,同样在每次请求之后运行.注册的函数至少需要含有一个参数,这个参数实际上为服务器的响应,且函数中需要返回这个响应参数

图片上传插件

```
1、安装 pip install flask-uploads
2、ext配置:
   photos = UploadSet('photos', IMAGES)
   configure_uploads(app, photos)
3、settings中配置
   UPLOADED_PHOTOS_DEST = BASE_DIR（这是图片保存路径)
4、视图函数使用:
file = request.files['icon']
filename =
photos.save(request.files['icon'],folder="img",name=file.filename)
url = photos.url(filename)//完整链接路径
```

flask中内置密码加密函数

- 密码加密generate_password_hash
- 密码核对check_password_hash(pwhash, password):
    -
    ```
    pwhash: a hashed string like returned by
                :func:`generate_password_hash`.
    ```

- password: the plaintext password to compare against the hash

缓存配置Redis

注意：
flask-cache不兼容现在的版本，所以使用flask-caching插件
安装：
pip install flask-caching

ext文件：
cache = Cache(config={'CACHE_TYPE': 'redis'})
cache.init_app(app)

settings文件：
CACHE_REDIS_HOST = "www.fand.wang"
CACHE_REDIS_PORT = 6379
CACHE_REDIS_PASSWORD = "fanding"
CACHE_REDIS_DB = "3"

为视图函数配置缓存：
注意：在蓝图和视图函数之间添加如下装饰器
@cache.cached(timeout=50)

使用cache临时存储数据：
添加：set(*args, **kwargs):set("name","fanding",timeout=60*60*24)
获取：get(*args, **kwargs)
删除：clear（）、delete(*args, **kwargs)

邮件发送

安装：
pip install flask-mail

ext文件：
mail = Mail()
mail.init_app(app)

settings文件：
MAIL_SERVER = "smtp.163.com"
MAIL_PORT = 25
# MAIL_USE_SSL = True
MAIL_USERNAME = "fand1024@163.com"
MAIL_PASSWORD = "fand102487"
MAIL_DEFAULT_SENDER = MAIL_USERNAME

邮件发送操作：
msg = Message(recipients=recipient_list, html=html_message,
subject=subject)
mail.send(msg)

Flask-restful

序列化

书写模板：

构建序列化模板

```
# 构建序列化模板
movie_user_field = {
    "username":fields.String,
    "phone":fields.String,
    "password":fields.String(attribute="_password")
}

single_movie_user_fields = {
    "status":fields.Integer,
    "msg":fields.String,
    "data":fields.Nested(movie_user_field)
}
```

视图函数中的使用：

```
marshal（）：
data = {
        "status":201,
        "msg":"注册成功",
        "data":user
       }
return marshal(data,single_movie_user_fields)
```

使用@marshal_with(single_movie_user_fieelds)装饰器

级联序列化：

序列化模板：

```
movie_fields = {
    "showname": fields.String,
    "shownameen": fields.String,
    "director": fields.String
```

```
    "director": fields.String,
    "leadingRole": fields.String,
    "type": fields.String,
    "country": fields.String,
    "language": fields.String,
    "duration": fields.Integer,
    "screeningmodel": fields.String,
    "openday": fields.DateTime,
    "backgroundpicture": fields.String,
}

multi_movies_fields = {
    "status": fields.Integer,
    "msg": fields.String,
    "data": fields.List(fields.Nested(movie_fields))
}
```

视图函数:

```
    @marshal_with(multi_movies_fields)
    def get(self):

        movies = Movie.query.all()

        data = {
            "msg": "ok",
            "status": HTTP_OK,
            "data": movies
        }

        return data
```

flask-restful中的abort

- abort(400,msg="消息提示")

提取请求参数

```
parse_base = reqparse.RequestParser()
parse_base.add_argument("password",type=str,required=True,help="请输入密码")
parse_base.add_argument("action",type=str,required=True,help="请确认请求参数")

register_parse = parse_base.copy()
register_parse.add_argument("phone",type=str,required=True,help="请输入手机号")
register_parse.add_argument("username",type=str,required=True,help="请输入用户名")

login_parse = parse_base.copy()
register_parse.add_argument("phone",type=str,help="请输入手机号")
register_parse.add_argument("username",type=str,help="请输入用户名")


视图函数中用法:
args = parse_base.parse_args()
username = args.get("username)
```

知识点

变量交换

- python中
    - a,b =b , a
- 加减法运算

```
a = a + b
b = a - b
a = a - b
```

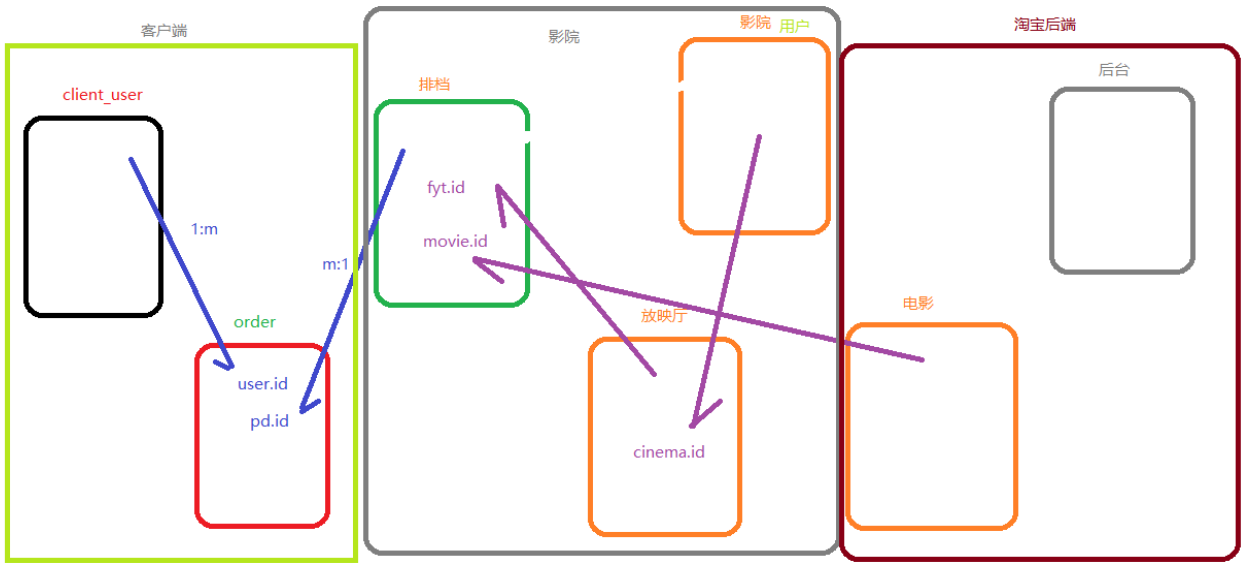
- 通过二进制转换

```
a = a ^ b
b = a ^ b
a = a ^ b
```

淘票票项目

项目分析

图解



端

- 淘票票后端
    - 淘票票公司自己管理
- 客户端
    - 看电影用户准备的
- 影院端
    - 电影放映

通用模块

- 用户体系
    - 用户权限
    - 用户角色
- 电影
    - 淘票票后端
        - 增删改查
    - 淘票票影院端
        - 查询
    - 淘票票客户端
        - 查询
- 排档
    - 淘票票后端
        - 增删改查

        - 影院端
            - 增删改查
        - 客户端
            - 查
- 影票
        - 淘票票后端
            - 增删改查
        - 影院端
            - 增删改查
        - 客户端
            - 增删查

可扩展

- 优惠卷
- 积分系统
- 影院会员系统
- 评价系统
- 套餐
- 定位（手机，浏览器）
- 地址
        - 后端规划好
        - 影院端添加

项目架构

- 一个项目入口
    - App/__init__
    - apis
        - admin
        - movie_admin
        - moive_user
    - models
        - admin
        - movie_admin
        - movie_user
- 另外一种架构
    - FalskTpp
        - __init__
        - settings
        - ext
    - Admin
        - apis
        - models
    - MovieAdmin
        - apis
        - models
    - MovieUser

- apis
- models

项目划分

淘票票后端

- 管理淘票票用户
- 淘票票影院管理
- 电影

淘票票影院端

- 后端购买来的电影播放权
    - 电影
- 放映厅
- 排档
    - 关系表
    - 电影和放映厅 + 时间

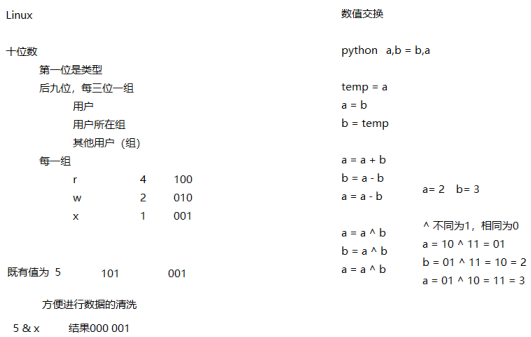
淘票票客户端

- 查看电影
- 查看影院
- 查看排档
- 下单
    - 找到具体排档
    - 选座（座位锁定），并发
        - 订单过期
    - 支付

系统设计

- 客户端
    - 用户系统
        - 字段
        - username
        - password
        - phone
        - email
        - is_delete
        - permission
    - 权限设计
        - 类似于Linux权限设计

          图解：



- 精髓，一个字段可以代表多种权限
- 还能分成两种
    - 完全和Linux中一样
        - 使用二进制
        - 所有的初始权限值都是2的n次幂
    - 只是给一个数值
        - 数值越大权限越高
        - 数值小的所有权限

- 多表设计权限
    - 用户表
    - 权限表
    - 用户权限表

项目总结

项目简单回顾

- 端
    - 用户端
    - 商家端
    - 后台管理端
- 用户模块
    - 用户权限
        - Linux
        - 多表
- 电影
- 商家电影授权
    - 商家和电影
- 电影院地址
    - 级联到商家
- 放映厅
    - 坐标系
    - 级联到电影院地址
- 排档
    - 放映厅和电影
    - 电影已经买了的，授权了的
- 订单
    - 排档和用户
    - 座位筛选

- 将买了的
- 锁定的

- 订单锁定
    - 懒处理方式
    - 通过一个过期时间搞定

- 订单生成
    - 座位表获取
    - 座位表确认
    - 并发
        - 锁
            - 乐观锁
                - 操作前和操作后对数据进行比对
                - 如果满足预期则是成功
            - 悲观锁
                - 真的给库，给表加锁
                    - 效率低
        - 事务
            - 默认orm开启事务
            - begin
            - commit
            - rollback 回滚
                - 多个操作才有意义

- 支付
    - 和django中一样
    - 支付注意配置支付账号的信息
    - 支付的时候调用指定客户端
        - web
        - phone
        - 区别是调用接口不一样，参数都一样
            - 使用的库封装了，对应的就是函数名不一样
    - 确保支付结果
        - 接收支付宝的回调

票房排行

- 电影和钱 来个排序
- 电影表
- 钱
    - 订单
- 电影
    - 排档
- 订单
    - 排档

开发流程

- 产品提出需求
- 开会、评审
- 要添加功能，周期（时间）
- 数据设计
    - 架构师设计
    - 后端经验丰富的工程师
    - 自己设计

- 建库建表
    - 建立模型
    - 对数据进行crud
    - 和前端进行联调
    - 前端需要请求接口
        - api文档
        - 接口功能
        - 接口地址
        - 接口所需要的参数
            - 那些是必须的、那些是非必须的、那些是通用字段

        - 接口返回数据
            - 数据字段
            - 状态码
            - 错误码

- 设计操作权限
    - 登录才能操作
    - 权限级别
- 送到测试手里
    - 测试工程师bug反馈
- 修复bug
    - 送测
- 上线

端的概念

- 一个项目至少有两段
    - 一段用户端
    - 后台管理端
- 大部分三端
    - 用户端
        - web端
        - 移动端
        - 微信
    - 商家端
        - web
        - 移动

        - 微信

    - 后台管理
        - web
        - 移动

我们实现使用RESTful

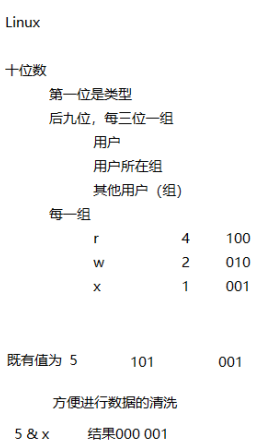
- 用户端的接口
    - 不需要再关注展示端的具体实现了
- 商家端的接口
- 后台管理接口

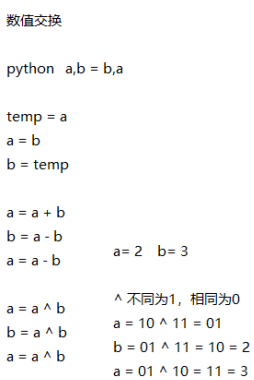
权限设计

权限设计

- 类似于Linux权限设计

  图解：





        - 精髓，一个字段可以代表多种权限
        - 还能分成两种
            - 完全和Linux中一样
                - 使用二进制
                - 所有的初始权限值都是2的n次幂
            - 只是给一个数值
                - 数值越大权限越高
                - 数值小的所有权限
- 多表设计权限
    - 用户表
    - 权限表
    - 用户权限表

登录设计

- 扫码登录
    - 前置条件
        - 有一个登录好的账号
    - 使用登录好的
- 跨应用登录
- 三方登录

给数据库加锁

悲观加锁：

db.session.with_lockmode(mode) db.session.commit() 对整个库加锁

db.session.Query(MovieOrder).with_lockmodel(mode)
db.session.commit() 对具体某张表进行加锁

乐观锁：

操作完成再读取一遍，如果和预期一样，则成功

orm中的事务

事物：保证代码完整执行

orm中默认开启事物

db.session.rollback() 事物的回滚

sqllite数据库的配置

```
SQLALCHEMY_DATABASE_URI = 'sqlite:///' + os.path.join(BASE_DIR, 'flask.sqlite')
```

支付完善

支付

- 如何确保付款
- 核心逻辑第三方平台不会暴漏给后端服务器的
- 直接和支付平台对接的是客户端
- 客户端可以返回给服务器一个支付状态
    - 客户端并不可信
    - 还要有一个确保策略
    - 接收支付宝的回调
        - notify_url
            - 已付款待确认就是客户端告诉你成功，但是还没有收到支付宝的回调