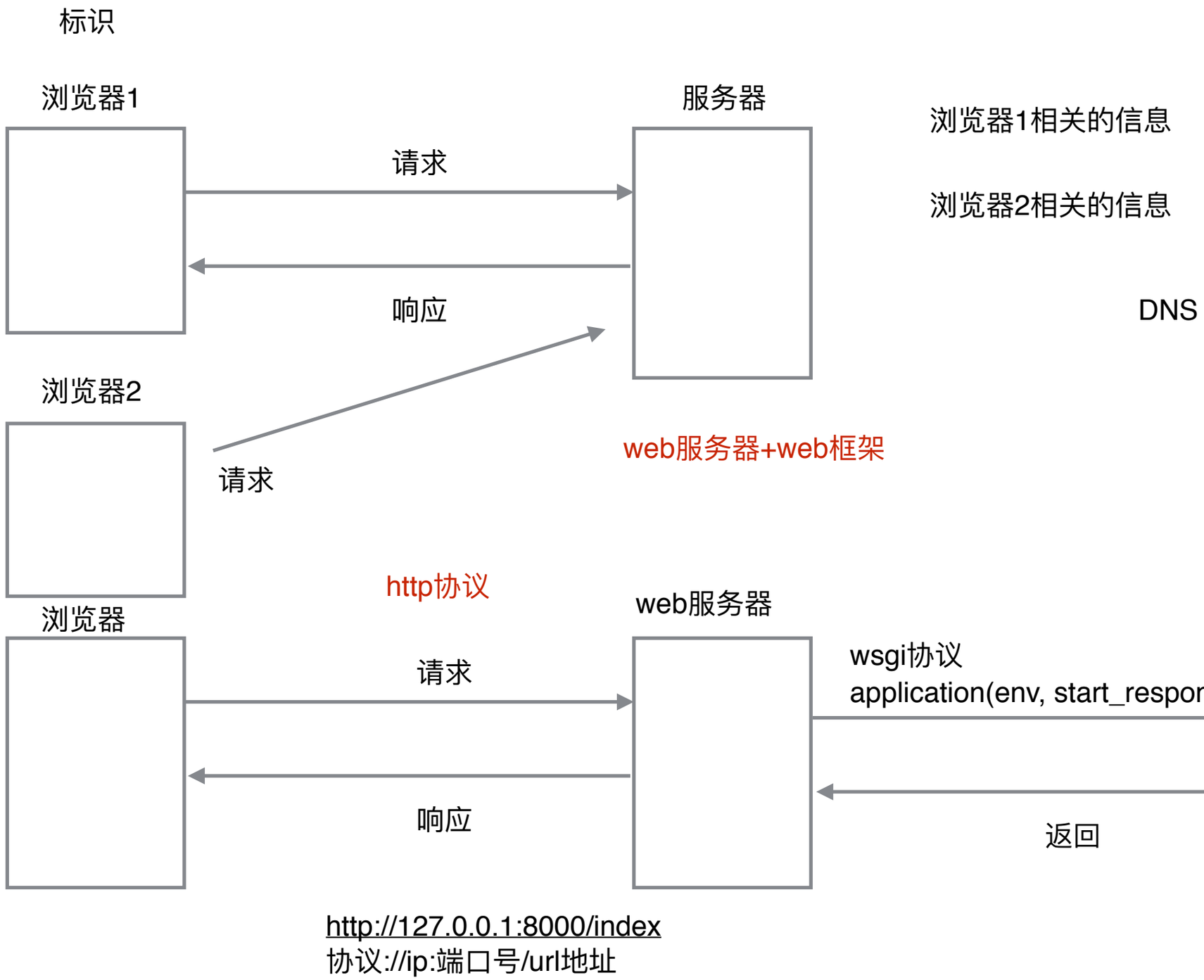
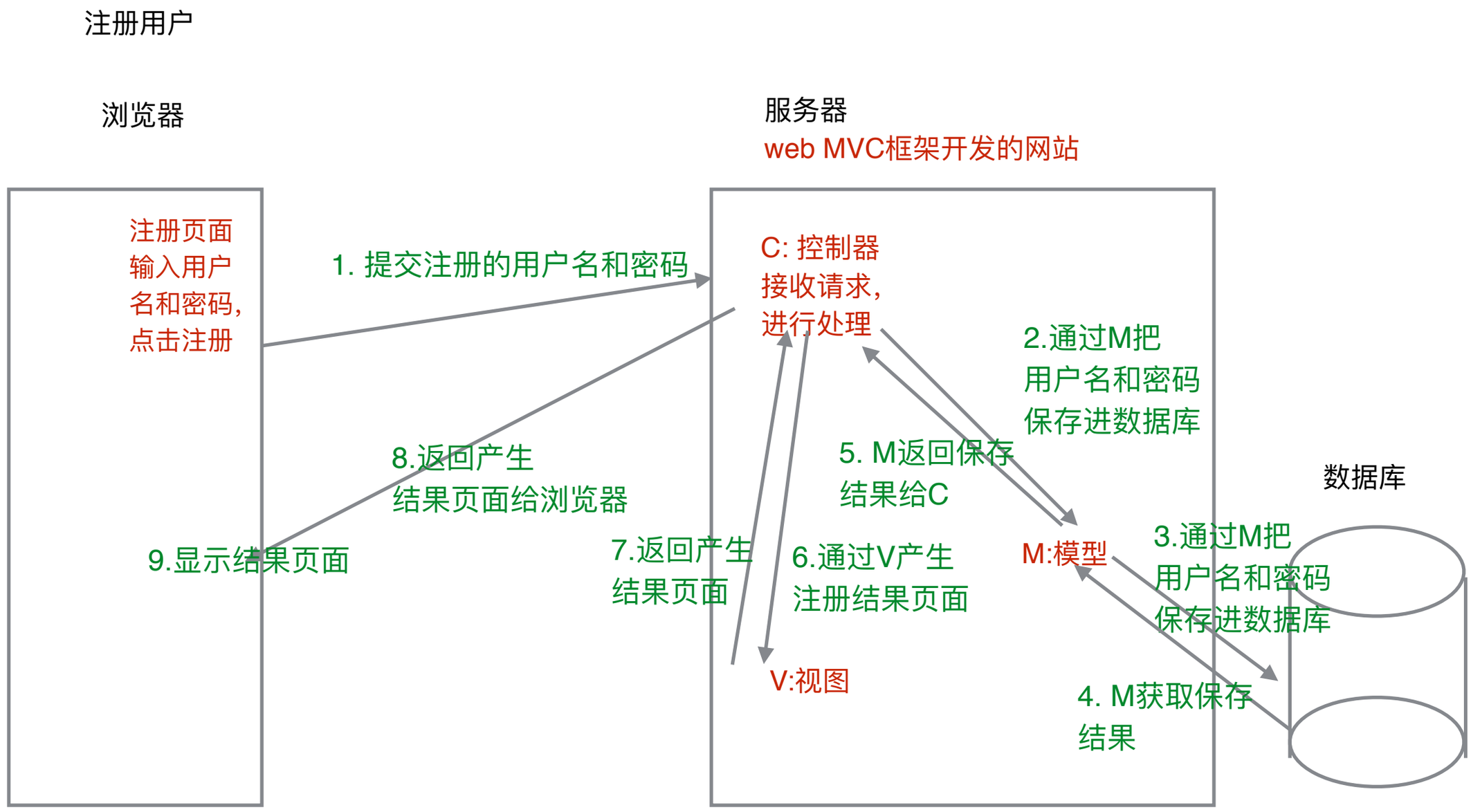
web开发

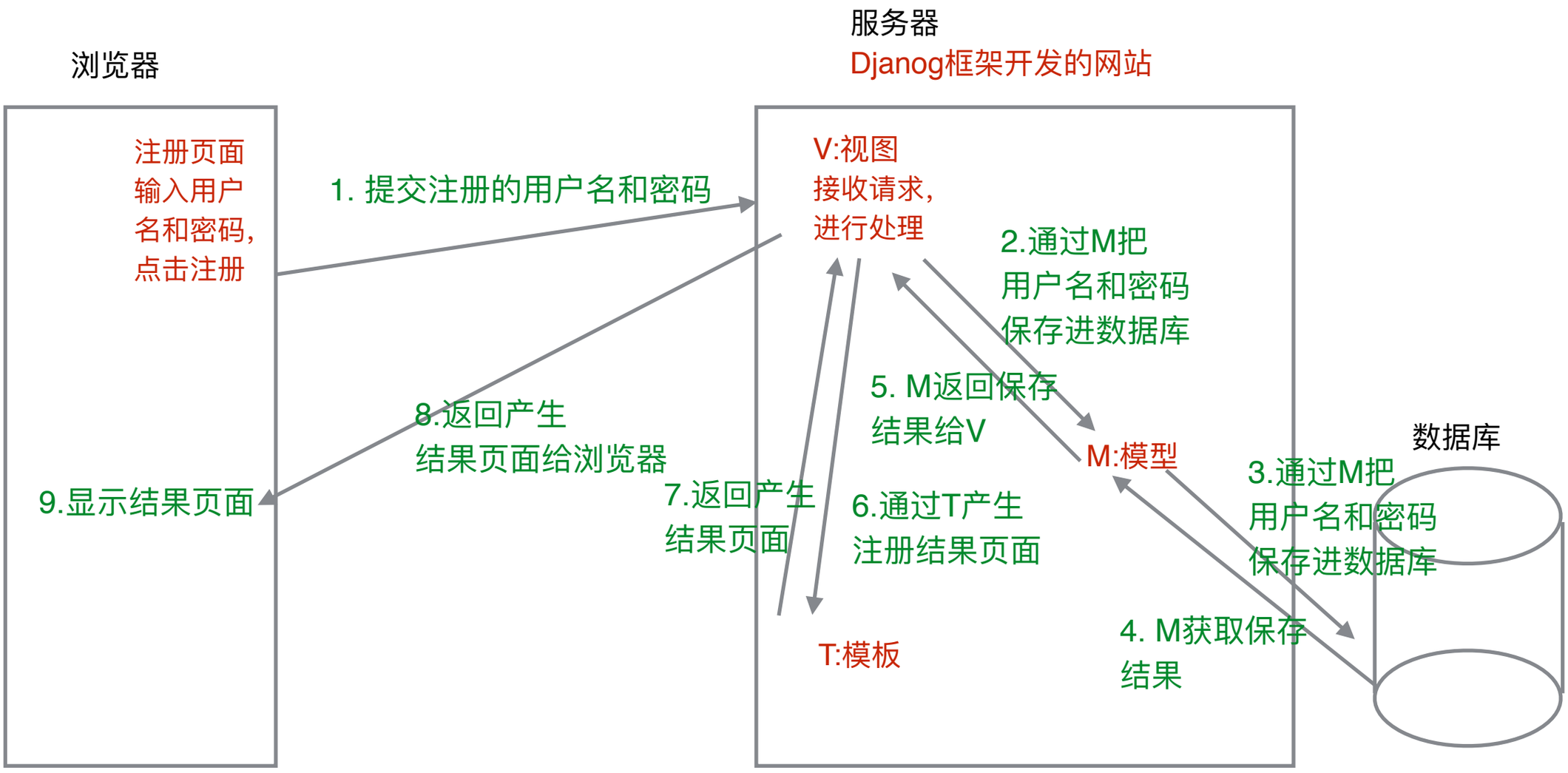

标识
浏览器1
服务器
浏览器1相关的信息
浏览器2相关的信息
DNS
请求
响应
浏览器2
请求
web服务器+web框架
http协议
浏览器
web服务器
web框架
请求
wsgi协议
application(env, start_response)
响应
返回
http://127.0.0.1:8000/index
协议://ip:端口号/url地址

web MVC 框架



Model: 模型，操作数据库。
View: 视图，产生html页面。
Controller: 控制器，接收请求，进行处理，和M和V进行交互，返回应答。

Django框架



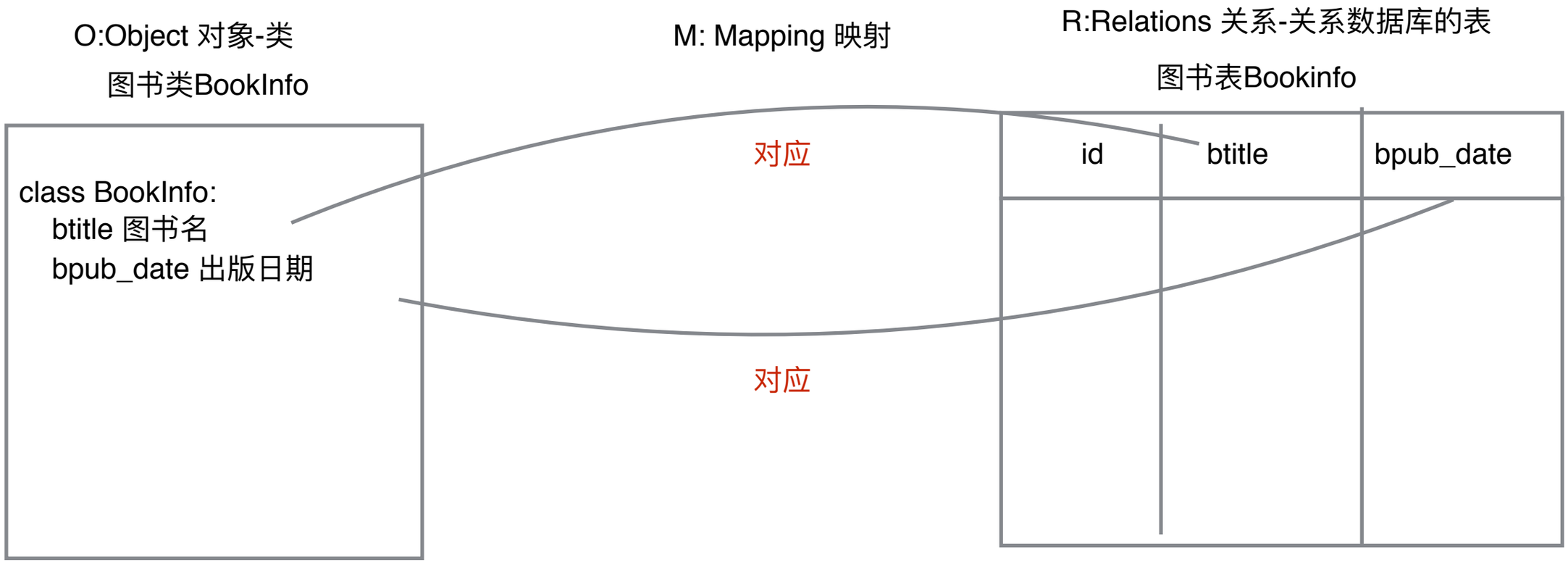
Django MVT

M: Model，模型, 和MVC中M功能相同，和数据库进行交互。

V: View, 视图， 和MVC中C功能相同，接收请求，进行处理，和M和T进行交互，返回应答。

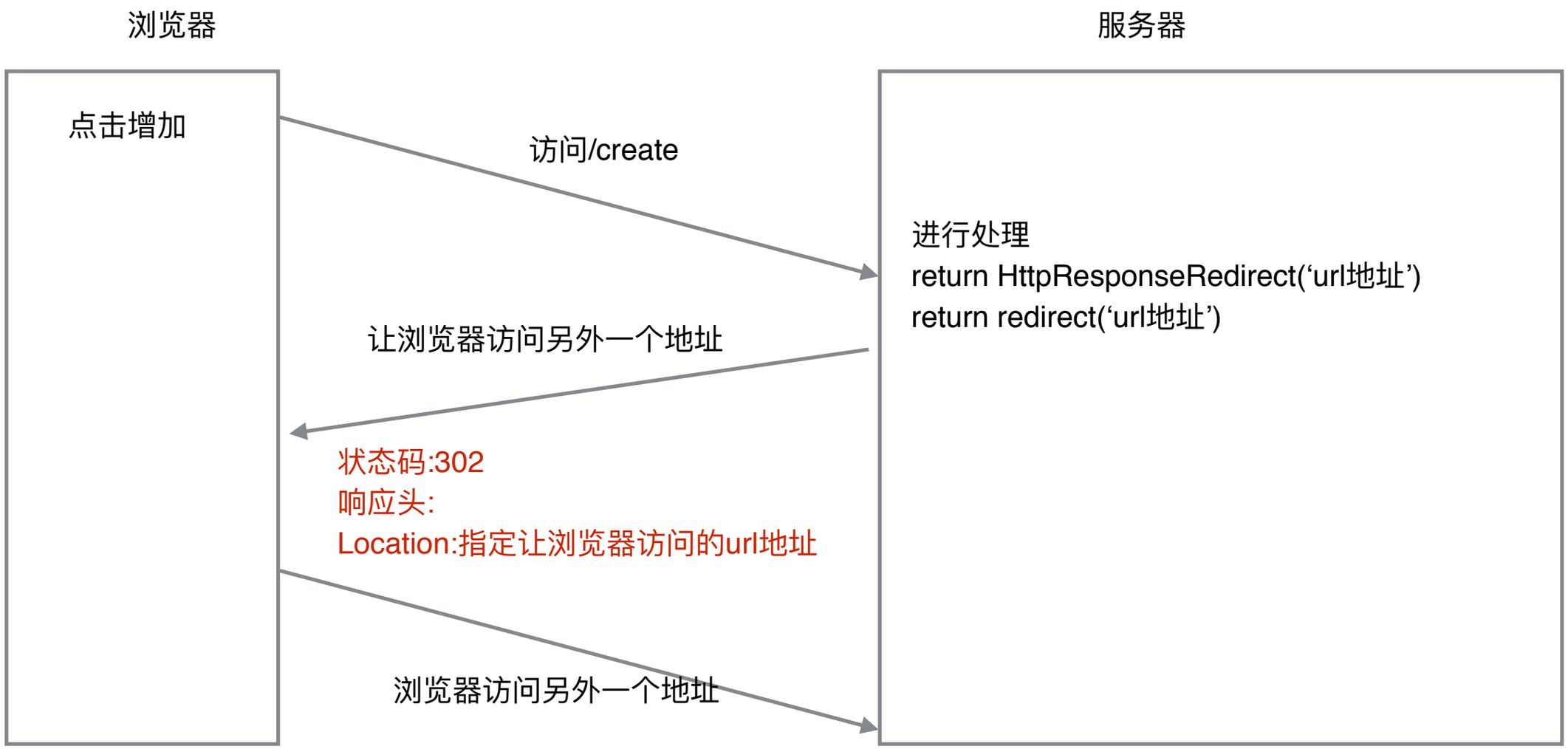
T: Template, 模板，和MVC中V功能相同，产生html页面。

ORM 框架

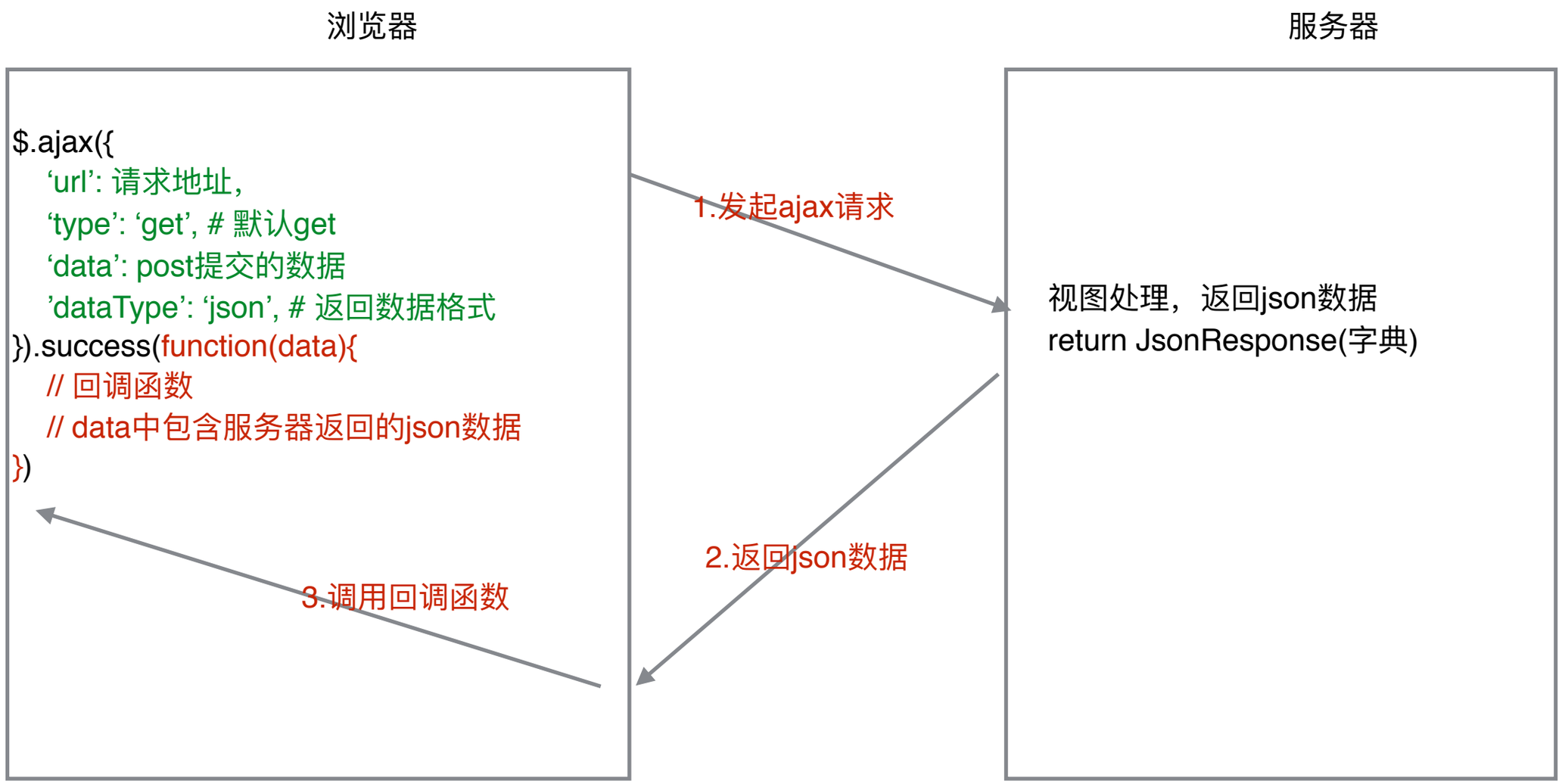


建立类和数据表之间对应的关系。
ORM框架帮助通过类和对象去操作数据表，不需要再写sql语句。

重定向


浏览器
服务器
点击增加
访问/create
进行处理
return HttpResponseRedirect(‘url地址’)
return redirect(‘url地址’)
让浏览器访问另外一个地址
状态码:302
响应头:
Location:指定让浏览器访问的url地址
浏览器访问另外一个地址

Ajax请求处理流程



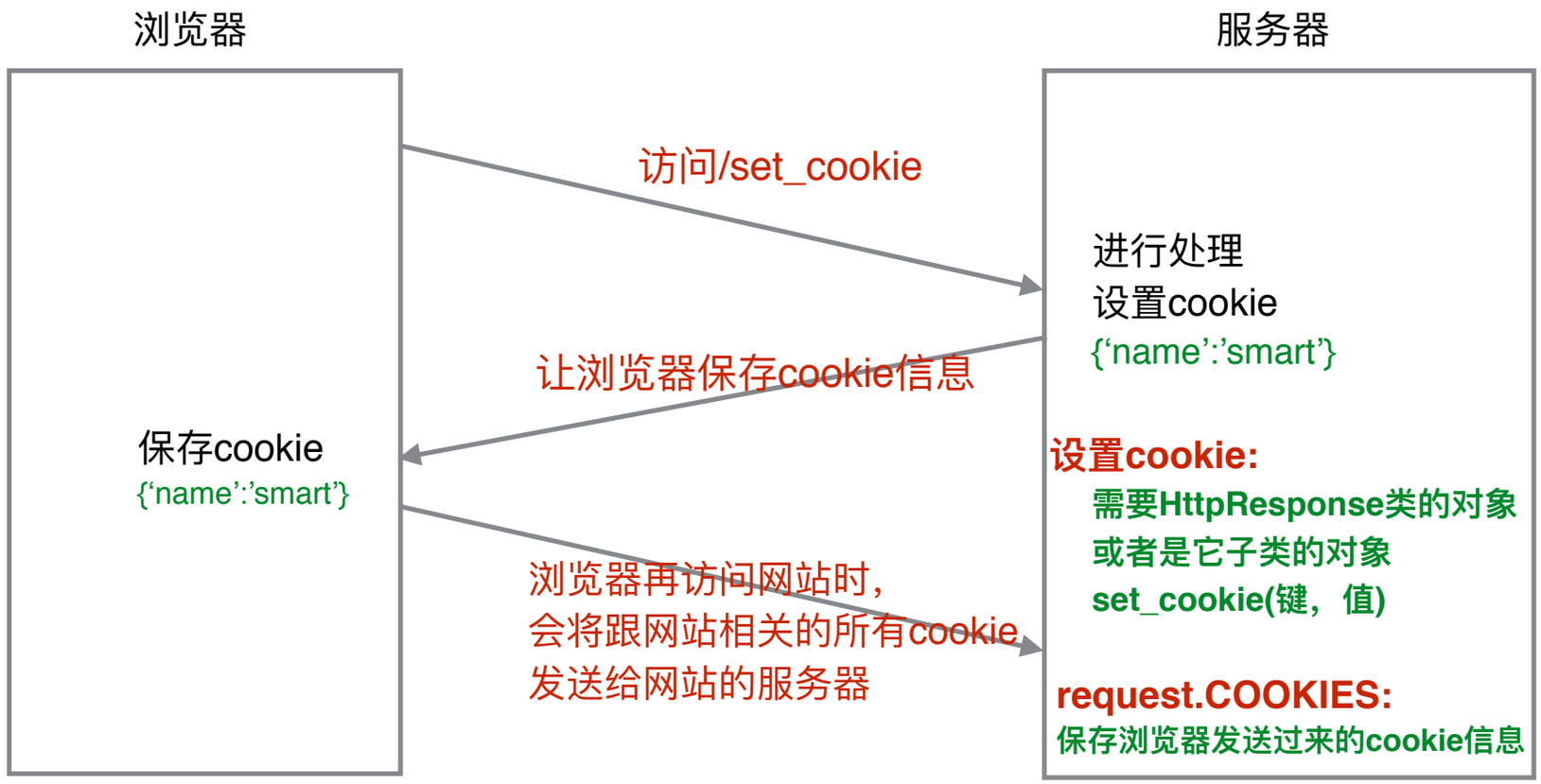
ajax请求的处理的流程:

1) 前端发起ajax请求。
2) 服务器进行处理，返回json数据。
3) 前端调用回调函数。

$.get(‘url地址’， function(data){})
$.post(‘url地址’, 字典，function(data){})

状态保持:cookie

你(浏览器)->豆浆店老板(服务器)
给你订单(cookie)
拿订单找老板要豆浆

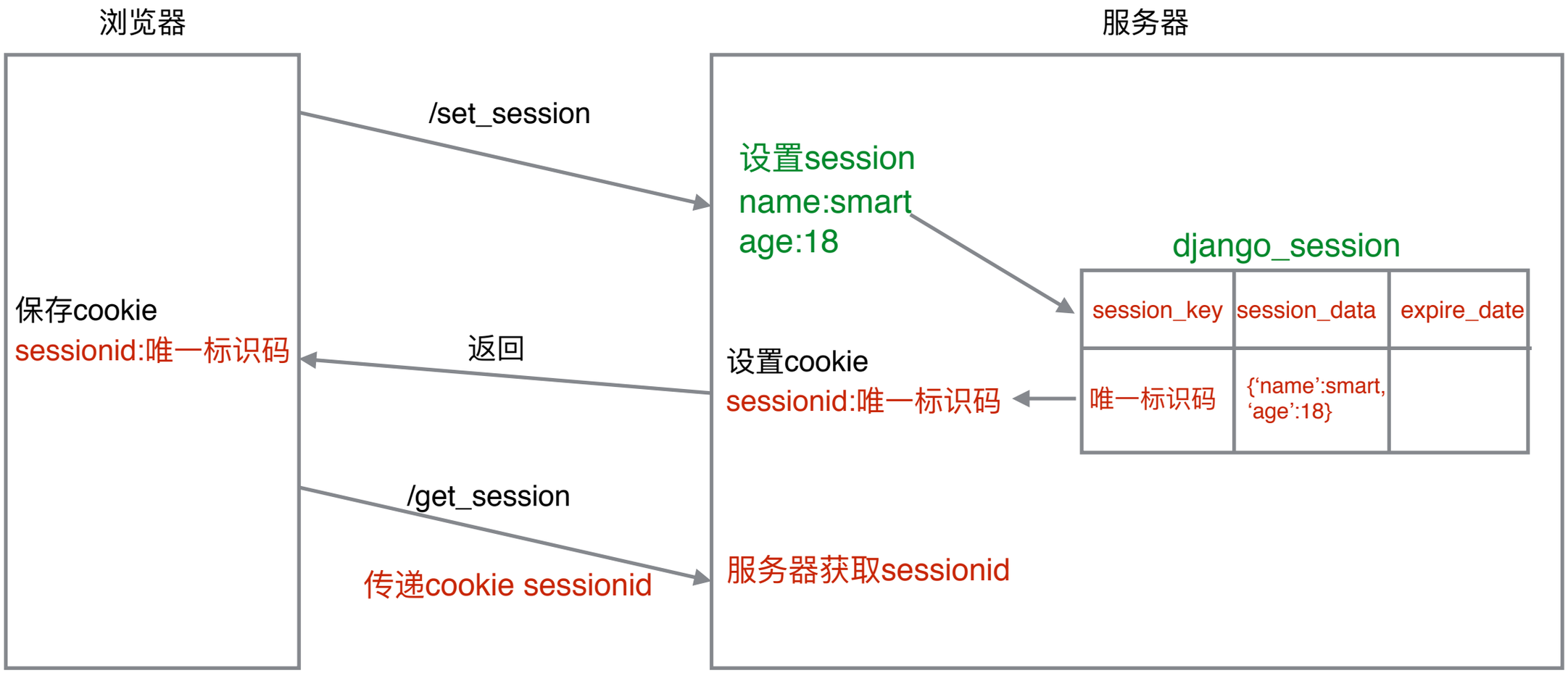


cookie: 由服务器生成，保存在浏览器端的一段文本信息。
特点:

1) cookie是以键值对来保存的。
2) 浏览器访问服务器的时候，会将跟服务器相关所有cookie信息发送给对应的服务器。
3) cookie是基于域名安全的。
4) cookie是有过期时间的，默认关闭浏览器之后过期。

cookie应用：记住用户名

状态保持:session



session存储在服务器端。

1）以键值对进行存储。

2）session依赖于cookie, 唯一标识码存在一个叫做sessionid的cookie中

3）session也是有过期时间，默认两周之后过期。

Django设置session和读取session:

```
request.session['键']=值
var = request.session['键']
```

html转义

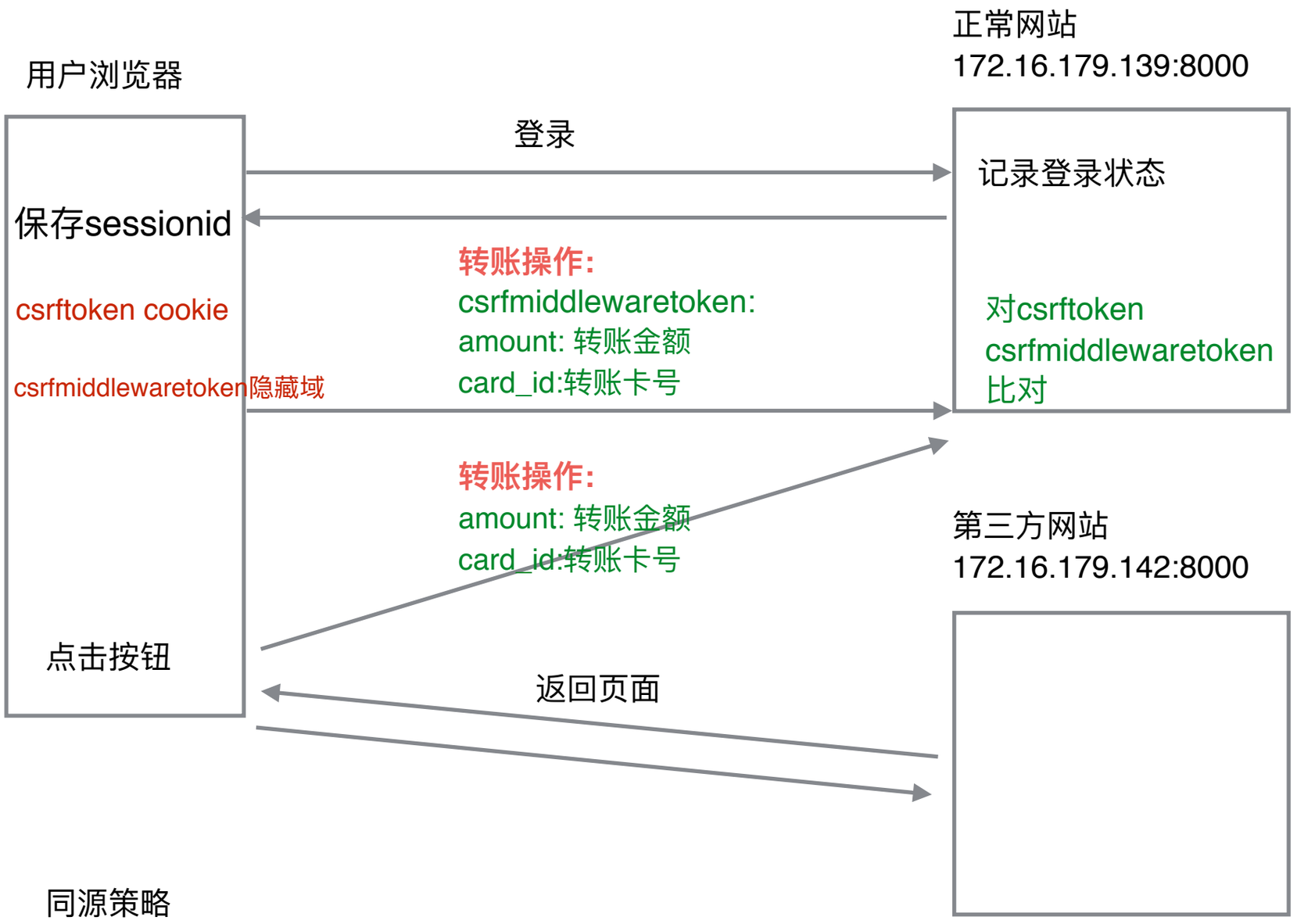
评论内容

<script>alert(‘hello’)</script>

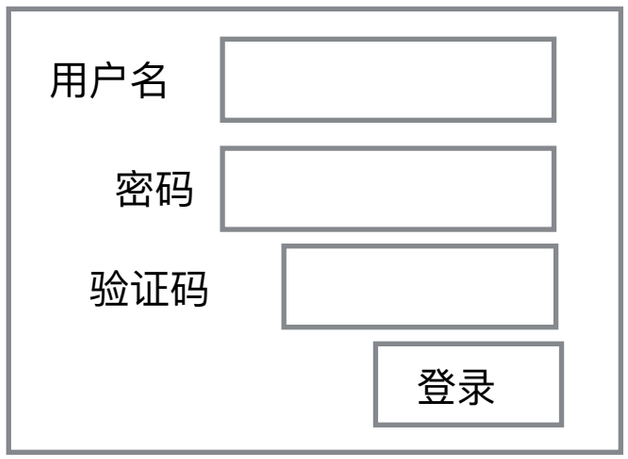
提交评论

展示评论

csrf(跨站请求伪造)


正常网站
172.16.179.139:8000
用户浏览器
登录
记录登录状态
保存sessionid
csrftoken cookie
csrfmiddlewaretoken隐藏域
转账操作:
csrfmiddlewaretoken:
amount: 转账金额
card_id:转账卡号
对csrftoken
csrfmiddlewaretoken
比对
转账操作:
amount: 转账金额
card_id:转账卡号
第三方网站
172.16.179.142:8000
点击按钮
返回页面
同源策略

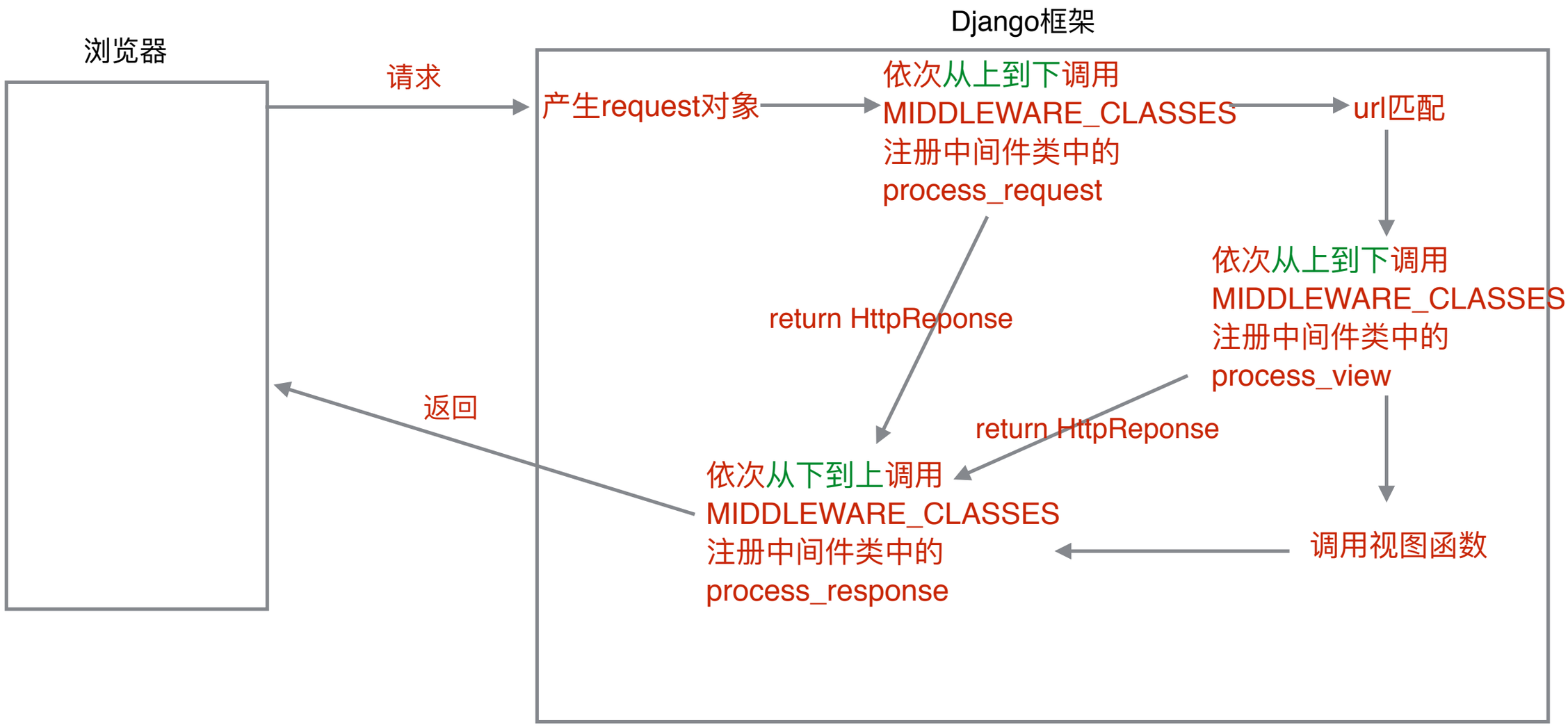
验证码



程序：暴力破解。

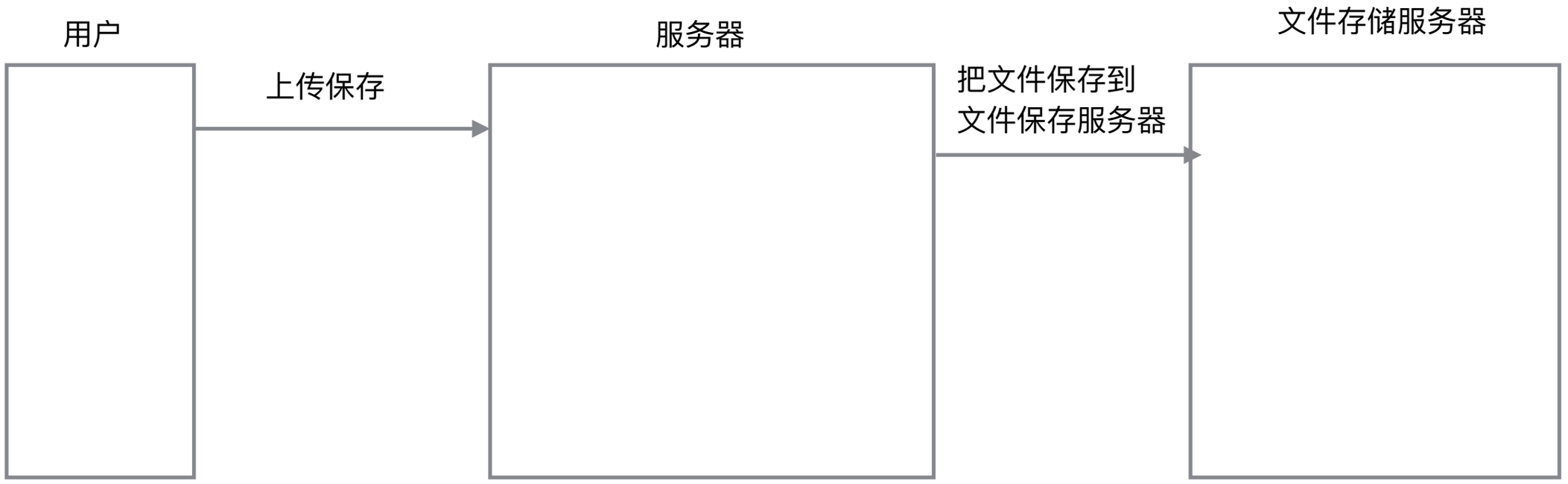
爬虫：urllib, requests

Django框架内容执行的流程



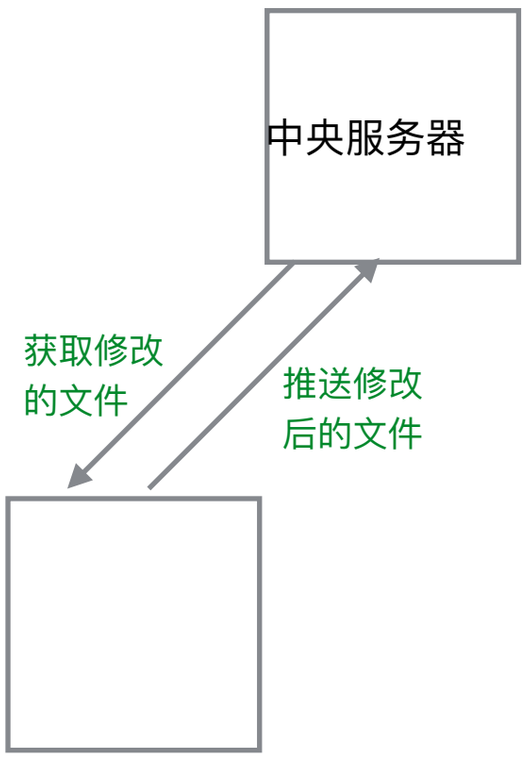
request.method
request.path
request.GET
request.POST

中间件函数
通过自定义中间件类来干预请求和应答的过程。



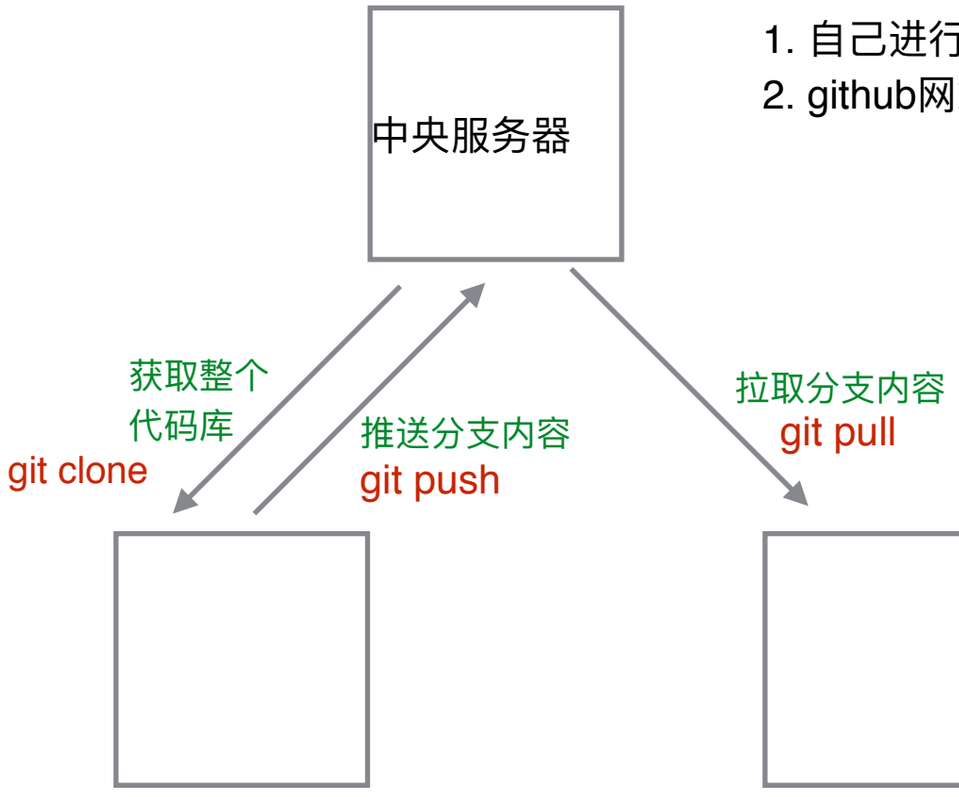
保存文件
MEDIA_ROOT指定目录下upload_to

集中式



员工1

分布式

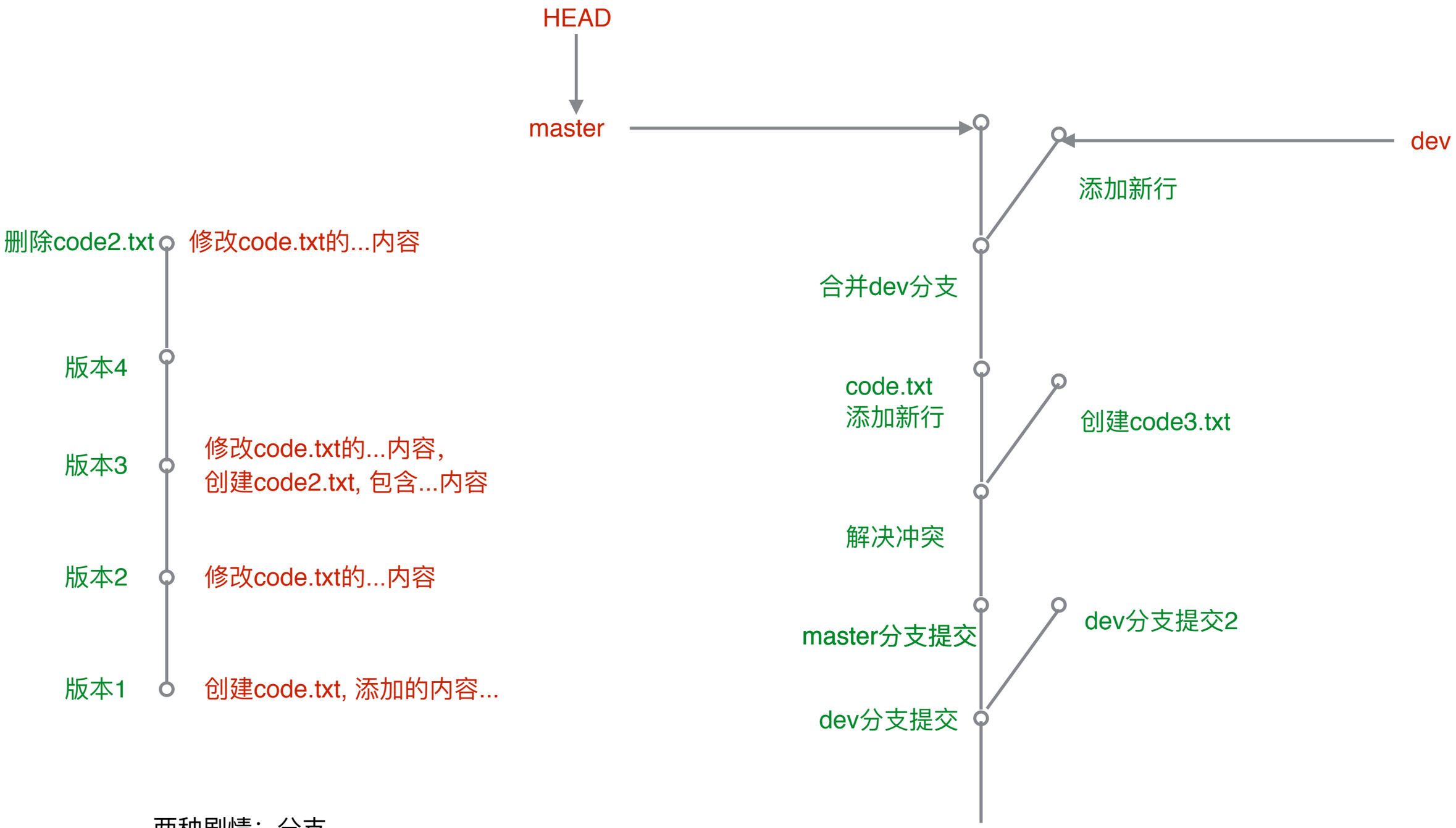


员工1

员工2

1. 自己进行搭建
2. github网站

添加ssh账户


HEAD
master
dev
添加新行
删除code2.txt
修改code.txt的...内容
合并dev分支
版本4
code.txt
添加新行
创建code3.txt
修改code.txt的...内容,
创建code2.txt, 包含...内容
版本3
解决冲突
版本2
修改code.txt的...内容
master分支提交
dev分支提交2
dev分支提交
版本1
创建code.txt, 添加的内容...

两种剧情：分支

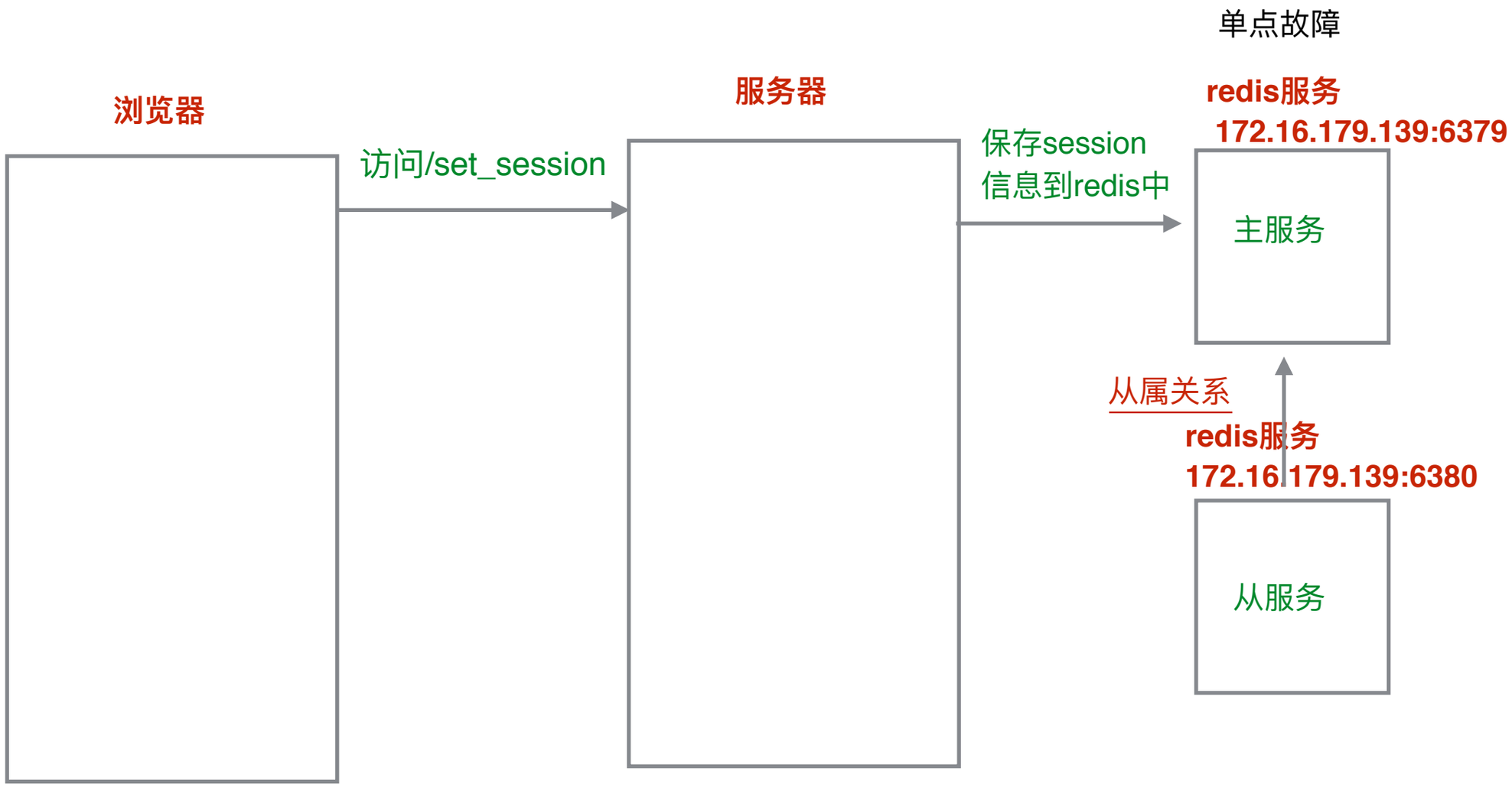
redis简介

关系型数据库: 数据存在表中, 表之间有关系。mysql, oracle, sql server sqlite
非关系型数据库: nosql, 采用key-value进行存储

关系型数据库有通用操作语言: SQL语言
非关系数据库没有通用的操作语言。

redis是一个内存型的key-value类型非关系型数据库，支持数据的持久化。
redis经常用来做缓存。

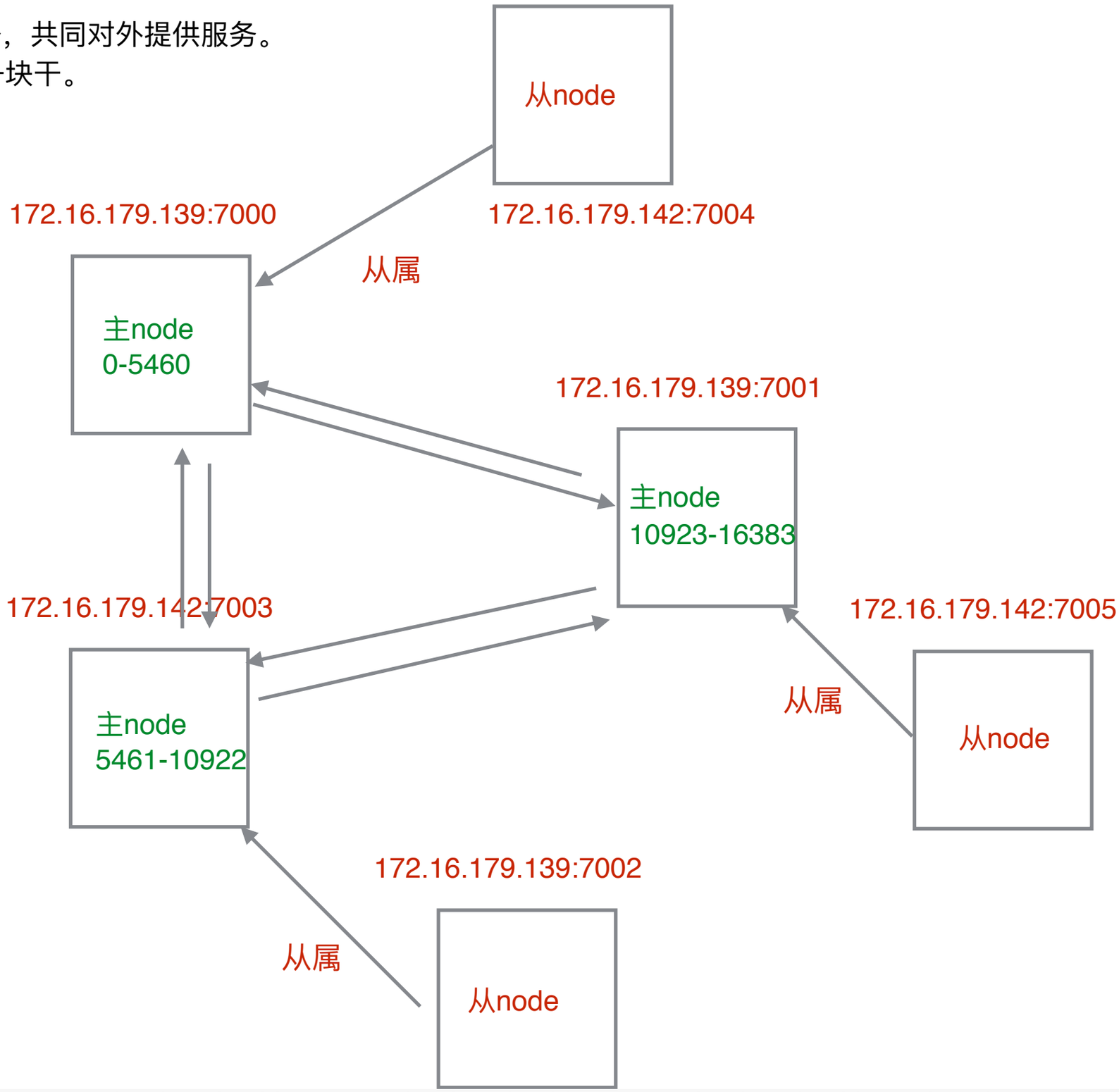
redis主从



1. 从服务会备份主服务中的数据。
2. 实现读写分离。

redis集群

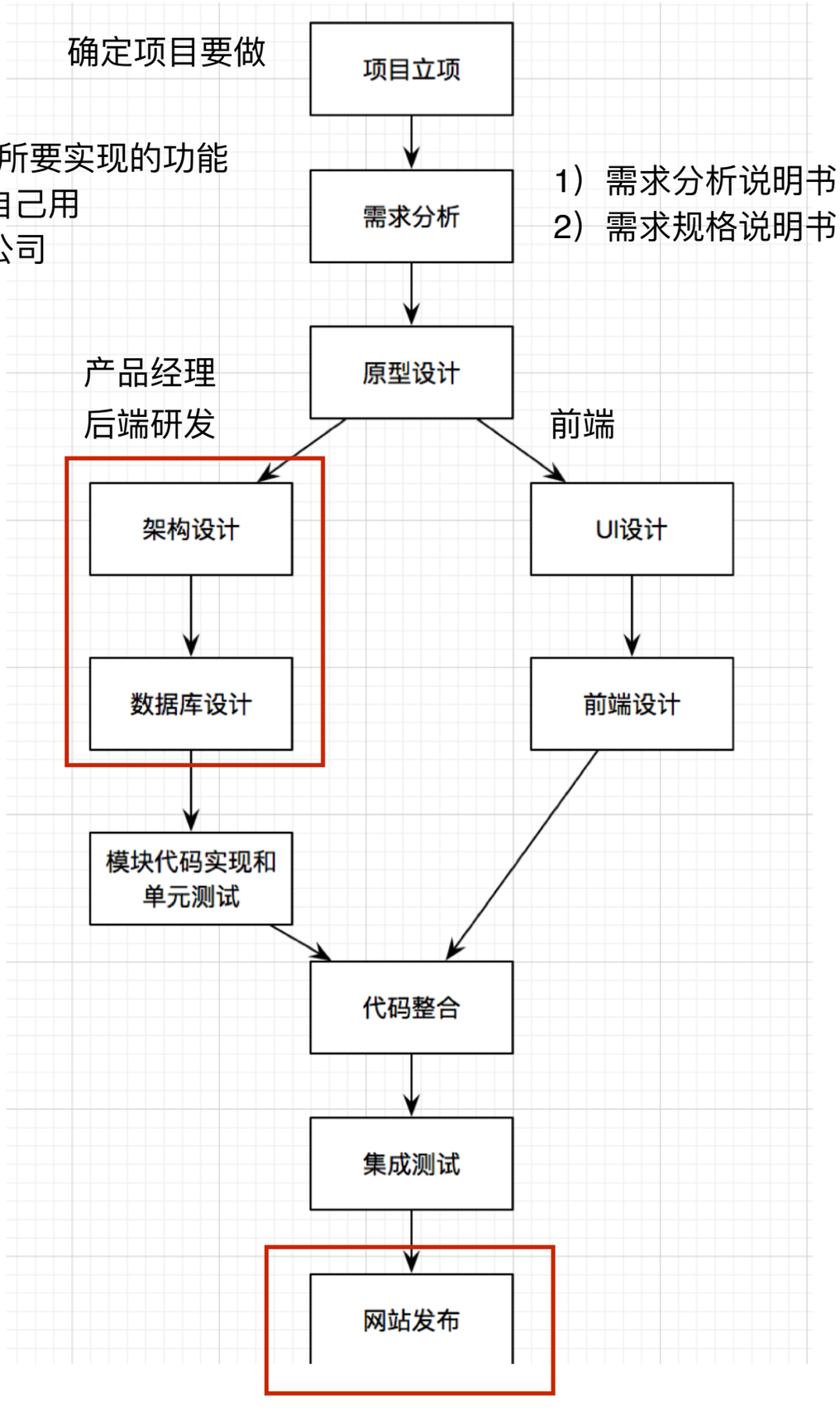
redis集群: 一组redis服务，共同对外提供服务。
一个人干活，找一帮人一块干。



16384 slots

```
redis-trib.rb create --replicas 1 172.16.179.139:7000 172.16.179.139:7001 172.16.179.139:7002 172.16.179.142:7003
172.16.179.142:7004 172.16.179.142:7005
```

web项目开发流程



确定项目要做

分析项目所要实现的功能
1）公司自己用
2）其他公司

1）需求分析说明书：给客户
2）需求规格说明书：给研发

产品经理
后端研发

前端

1. 项目功能模块设计
2. 开发环境选择
3. 项目部署架构设计
4. 项目用的技术
5. 前后端不分离

分析项目用到的数据表，表字段，表关系。

分工

前后端分离：
模板不使用
return json

项目架构分析

前端

注册 登录 用户中心	首页 详情页 列表页 搜索结果页	购物车页面	提交订单页	后台Admin页面

后端

用户模块	商品模块	购物车模块	订单模块	后台Admin 模块

mysql

缓存/session

redis

分布式文件存储系统
Fast DFS(余庆)

异步任务对列
celery

Admin后台管理页面

数据库设计

最近浏览: redis实现。
购物车功能: redis实现。

商品种类>SPU(小分类)>SKU(具体商品)

用户表

ID
用户名
密码
邮箱
激活标记
权限标记

地址表

ID
用户ID(外键)
收件人
地址
邮编
手机
是否默认

500g草莓
盒装草莓

商品SKU表

ID
名称
简介
价格
单位
库存
销量
状态(上下架)
图片(默认)
种类 ID(外键)
SPU ID(外键)

草莓

商品SPU表

ID
名称
详情

首页轮播商品表

ID
SKU ID(外键)
图片url
index(展示顺序)

首页促销活动表

ID
活动图片
活动页面url
index

首页分类商品展示表

ID
种类ID(外键)
SKU ID(外键)
展示类型
index

一个用户可以有多个收货地址。
一件商品可以有多条评论，评论跟订单相关。
一件商品可以有多张图片。
一个订单可以包含多个商品。

以空间换取时间。

关联查询

商品种类表

ID
种类名称
logo
种类图片

商品图片表

ID
SKU ID(外键)
图片url

订单信息表

订单ID
用户ID(外键)
地址ID(外键)
支付方式
商品总数(运营)
订单总额(运营)
订单运费
订单状态
创建时间

订单商品表

ID
订单ID(外键)
SKU ID(外键)
商品数目
商品价格(历史)
评论

ID	名称	简介	价格	单位	库存	详情	图片	状态	种类ID	SPU ID
1	500g草莓	500g草莓简介	10.00	500g	20	草莓详情	500g草莓图片	1	1	1
2	盒装草莓	盒装草莓简介	20.00	盒	10	草莓详情	盒装草莓图片	1	1	1

项目框架搭建

项目搭建:

1) 创建项目
2) 根据划分模块创建应用
3) 进行基本设置(注册应用，数据库，模板目录， 静态文件，包含应用urls)
4) 模型类创建
5) 迁移生成表。

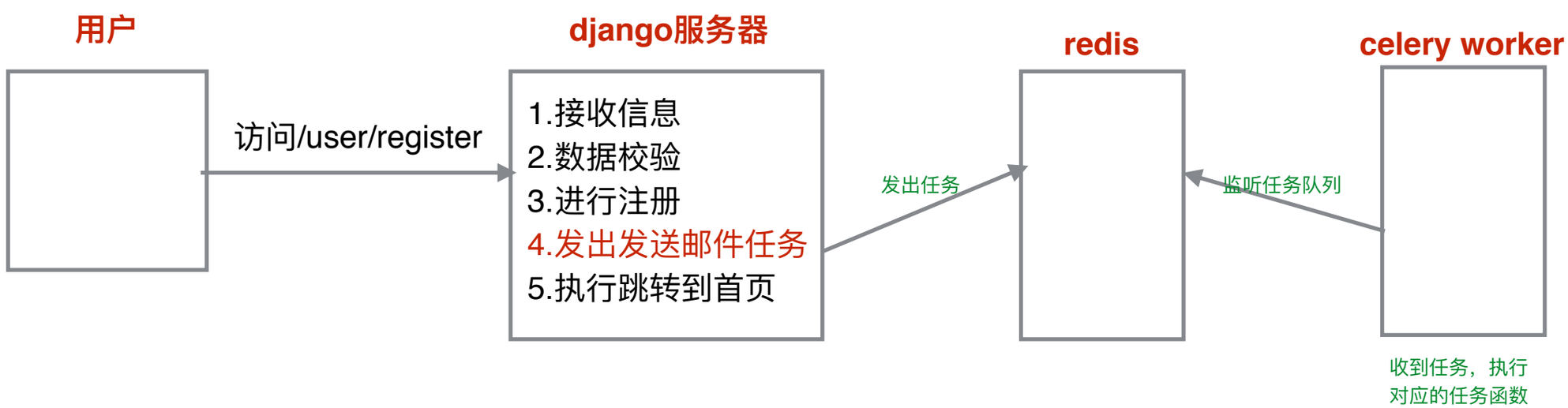
RegisterView中直接调用send_mail发邮件的问题

多线程，多进程：

1）注册处理进程和发送邮件进程需要在一台服务器上。

2）多进程和多线程调用顺序是不确定。

解决发送邮件问题



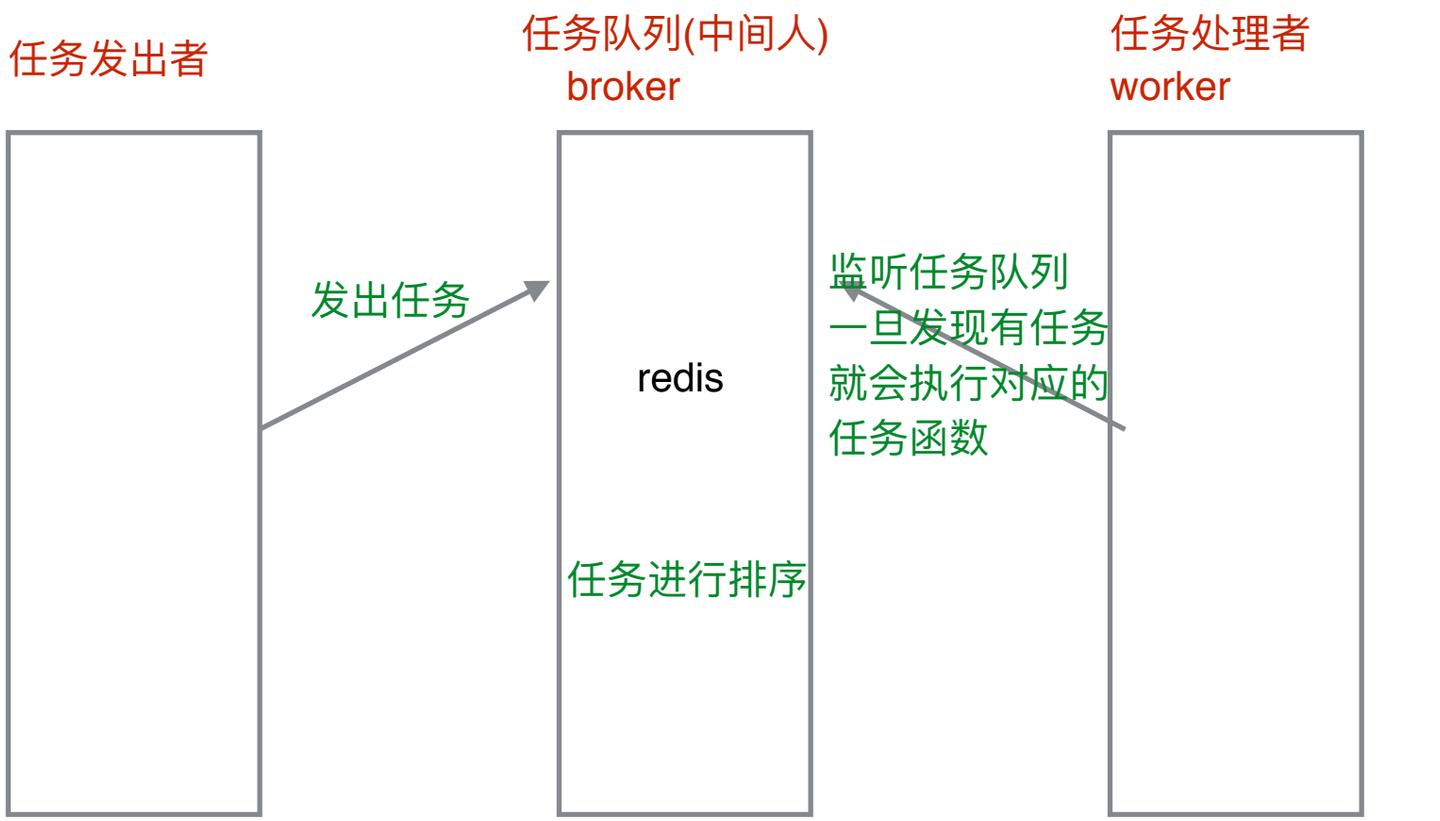
celery发出任务时不是发出的任务的代码，发出的是任务函数的名字和所需的参数。

Celery

任务发出者，中间人，任务处理者可以不在同一个电脑上，
任务发出者和处理者必须都能链接上中间人。

1.你打游戏
2.订外卖

你:
　任务发出者
订外卖:
　任务
美团:
　中间人
外卖小哥:
　任务处理者



任务就是函数。　　任务处理者就是进程。

1. 安装 pip install celery
2. 无论是发出任务还是启动工作进程，都需要Celery类的对象。

```
from celery import Celery
app = Celery('demo', broker='redis://172.16.179.142:3306/4')
```

3. 封装任务函数, 并使用app对象的task方法进行装饰

```
@app.task
def task_func(a, b):
    print('任务函数...')
```

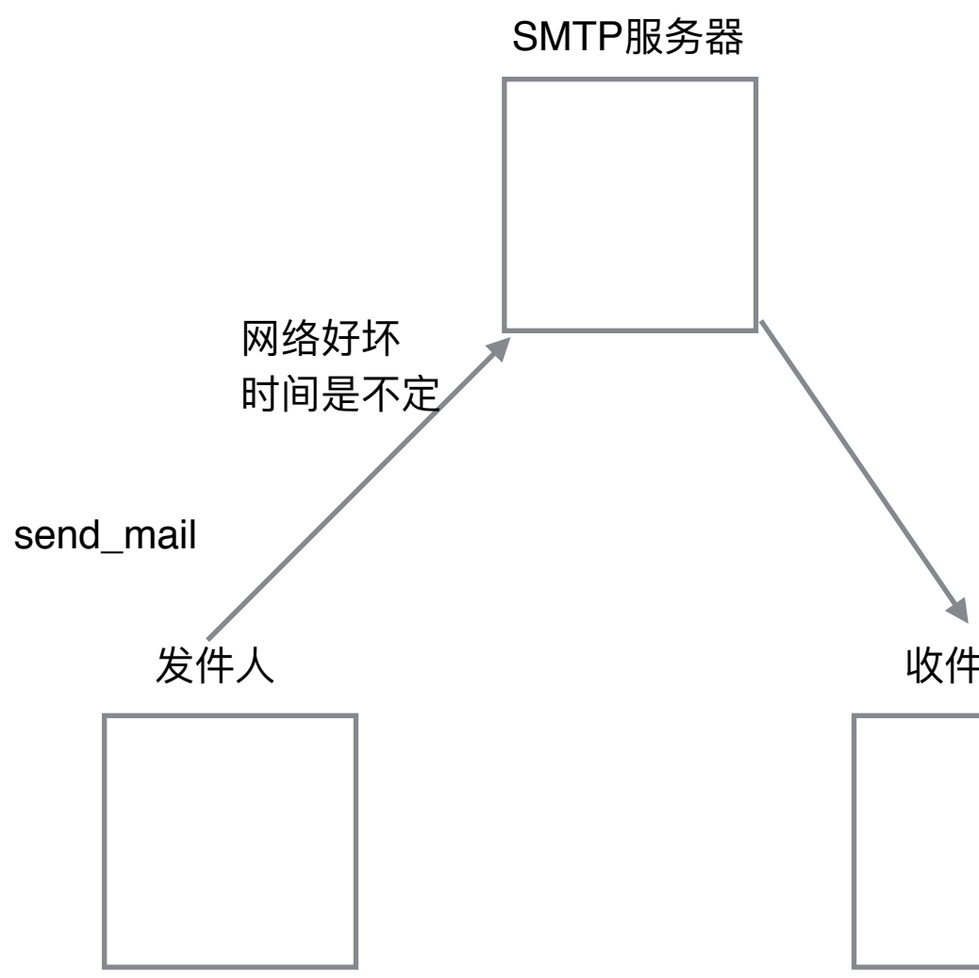
4. 启动工作进程

```
celery -A '任务函数所在文件路径' worker -l info
```

5. 发出任务

```
task_func.delay(a, b)
```

发送邮件



历史浏览记录存储分析

1. 什么时候需要去添加历史浏览记录?
答: 当用户访问商品详情页时，需要添加历史浏览记录(详情页视图中)。

2. 什么时候需要获取历史浏览的记录?
答: 当用户访问用户中心-信息页时，需要获取用户的历史浏览记录(信息页视图)。

3. 保存历史浏览记录时需要保存哪些数据?
答: 商品id, 添加历史浏览的时候需要保持浏览顺序。

4. 历史浏览记录存储?
答: 数据持久化存储: 文件， mysql, redis.
对于频繁操作数据，为了提高存储的效率，建议放在redis中。

5. 设置redis存储历史浏览记录的格式? key-value

方案1：所有用户的历史浏览记录采用一条数据来存储。**hash**

key: history
属性：user_用户id (区分每个用户)
值: '商品id1,商品id2,商品id3' (商品id以逗号进行分隔)
history = {user_1: '3,4,5', 'user_2':'1,2,3'}

获取id为1的用户的历史浏览记录：
hget history user_1
操作字符串

方案2：每个用户的历史浏览记录采用一条数据。**list**

key: history_用户id
value: 列表 [商品id1, 商品id2, 商品id3]

history_1: [1， 3, 2, 4, 5]

客户端

client.conf:

base_path=日志文件存储路径

tracker-server=ip:port

FDFS文件存储系统

tracker-server 追踪服务器

tracker-server 追踪服务器

tracker-server 追踪服务器

请求上传文件

返回给Client可用的storage-server的ip和port

请求下载文件

返回给Client可用的storage-server的ip和port

下载文件，传递文件ID

上传文件

返回保存文件ID

返回文件内容

tracker.conf:

bind_addr=监听的ip

port=端口号

base_path=日志文件存储路径

storage-server 存储服务器

storage-server 存储服务器

storage-server 存储服务器

相互备份

相互备份

相互备份

storage-server 存储服务器

storage-server 存储服务器

storage-server 存储服务器

1）tracker-server用于管理storage-server.

2）storage-server用于存储上传文件，存储服务器，会定期向对应tracker-server发送信息。

特点:

1） 实现海量存储。

2） 实现负载均衡。

web浏览器

nginx

172.16.179.131:8888

浏览器

http://172.16.179.131:8888/文件id

返回文件

去fdfs系统中获取文件

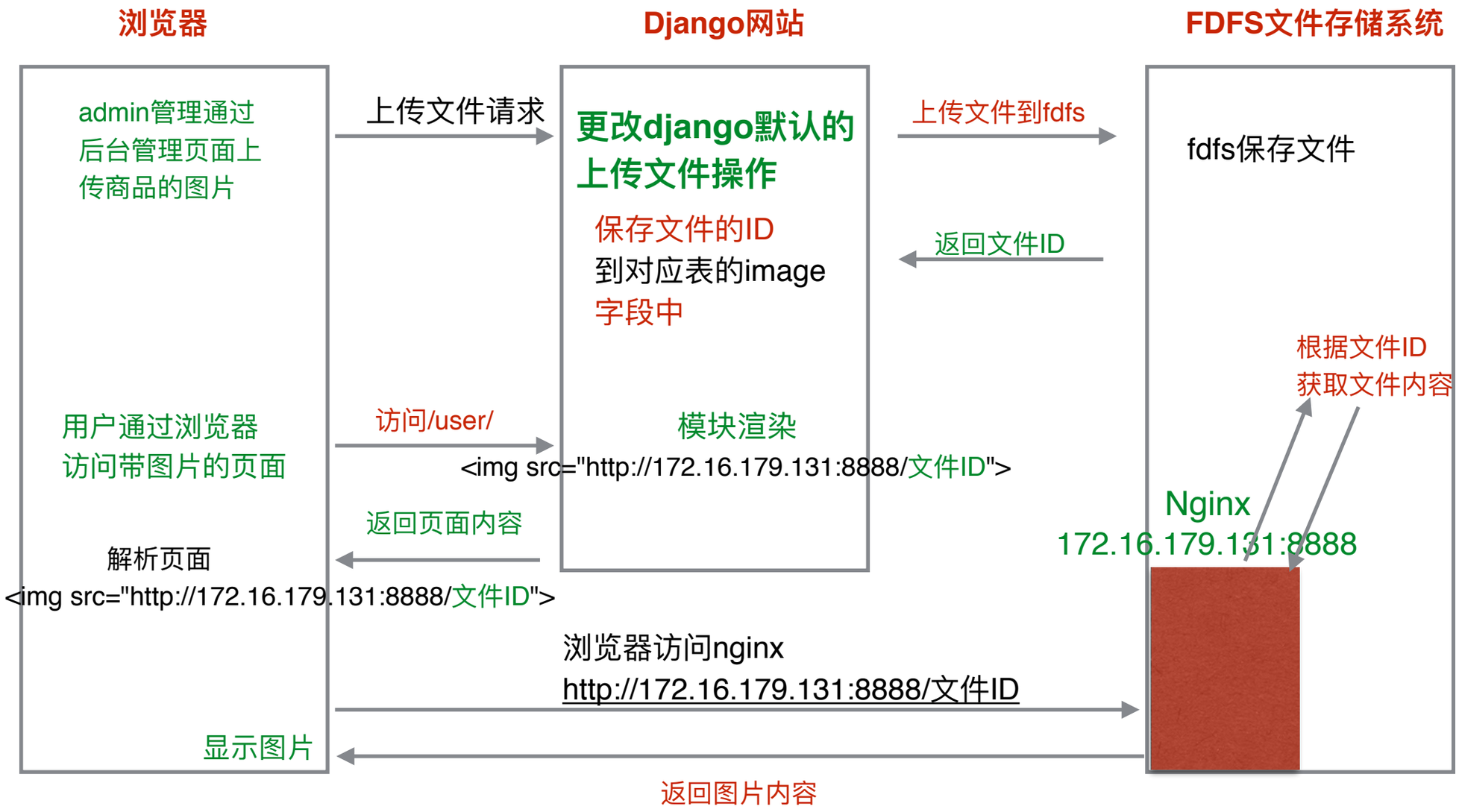
storage.conf:

base_path=日志文件存储路径

store_path0=保存上传文件的路径

tracker-server=对应tracker-serverip:port

项目上传图片和使用图片的流程



自定义文件存储类的流程

购物车记录存储分析的过程

1. 什么时候需要添加购物车记录?
答: 当用户点击加入购物车按钮时，需要添加用户的购物车记录。

2. 什么时候需要获取购物车记录?
答: 当用户访问购物车页面时，需要获取用户购物车记录。

3. 添加购物车记录需要保存哪些数据?
答: 商品id:添加数量

4. 购物车记录存储的方式?
答: 保存redis中。

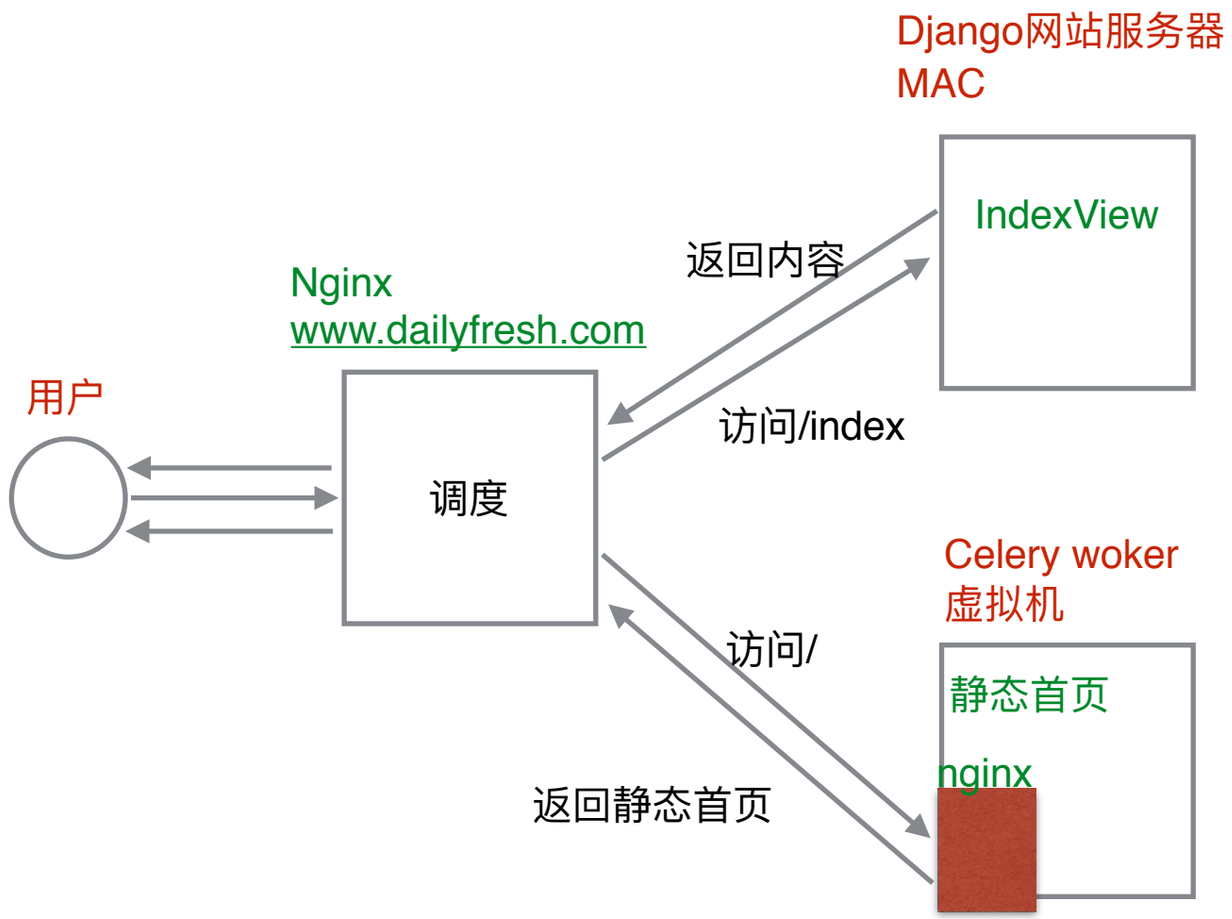
5. 购物车记录在redis中存储的数据格式?
答:
hash: 属性:值
存储方案: 每个用户的购物车记录对应一条数据。
key: cart_用户id （区分每个用户的购物车记录)
value: hash 商品id(属性):添加数量(值)

id为1的用户的购物车记录，商品id为1加了2件，商品id为3加了4件。
cart_1: {‘1’:’2’, ‘3’:’4’}

cart_3: {‘3’:’1’, ‘5’:’2’}
id为3的用户的购物车记录，商品id为3加了1件，商品id为5加了2件。

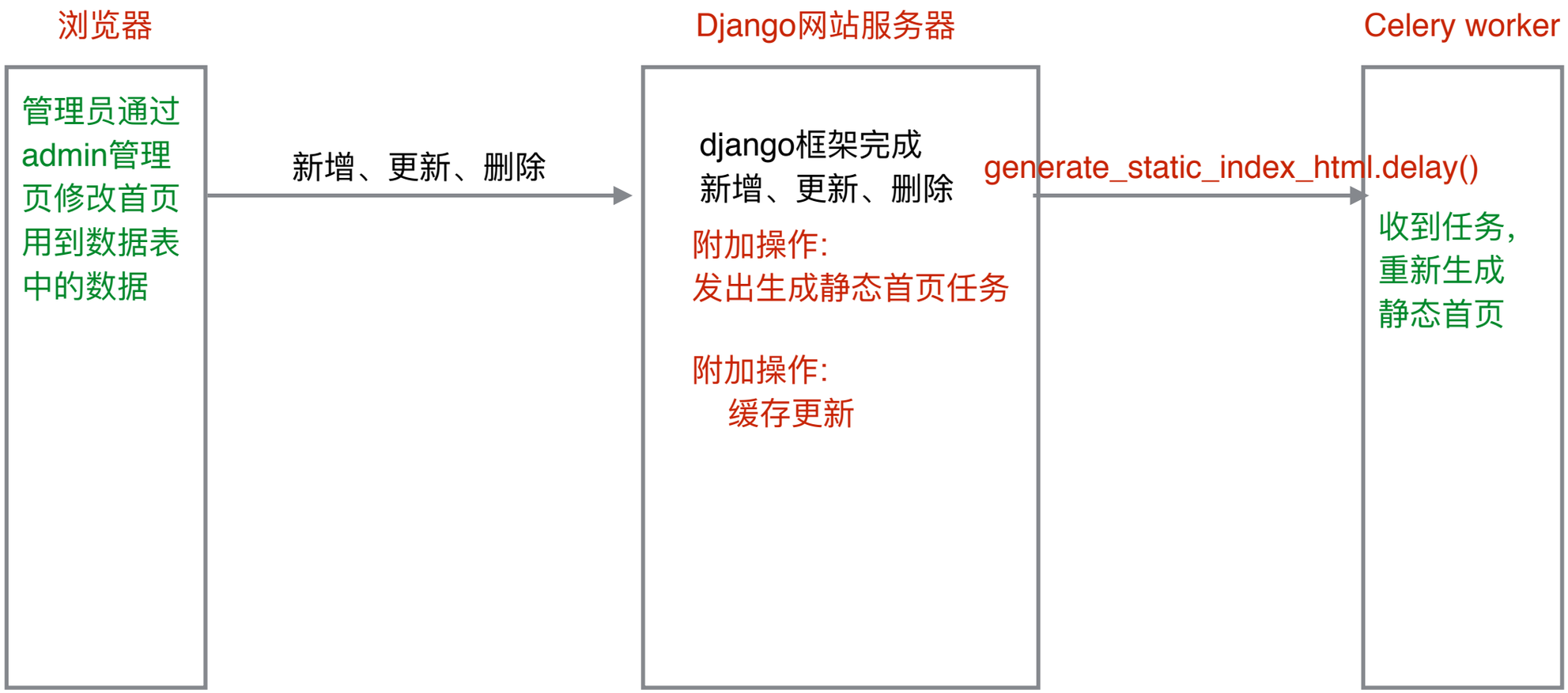
获取用户购物车中商品的条目数:
hlen cart_用户id

静态首页和IndexView的区分



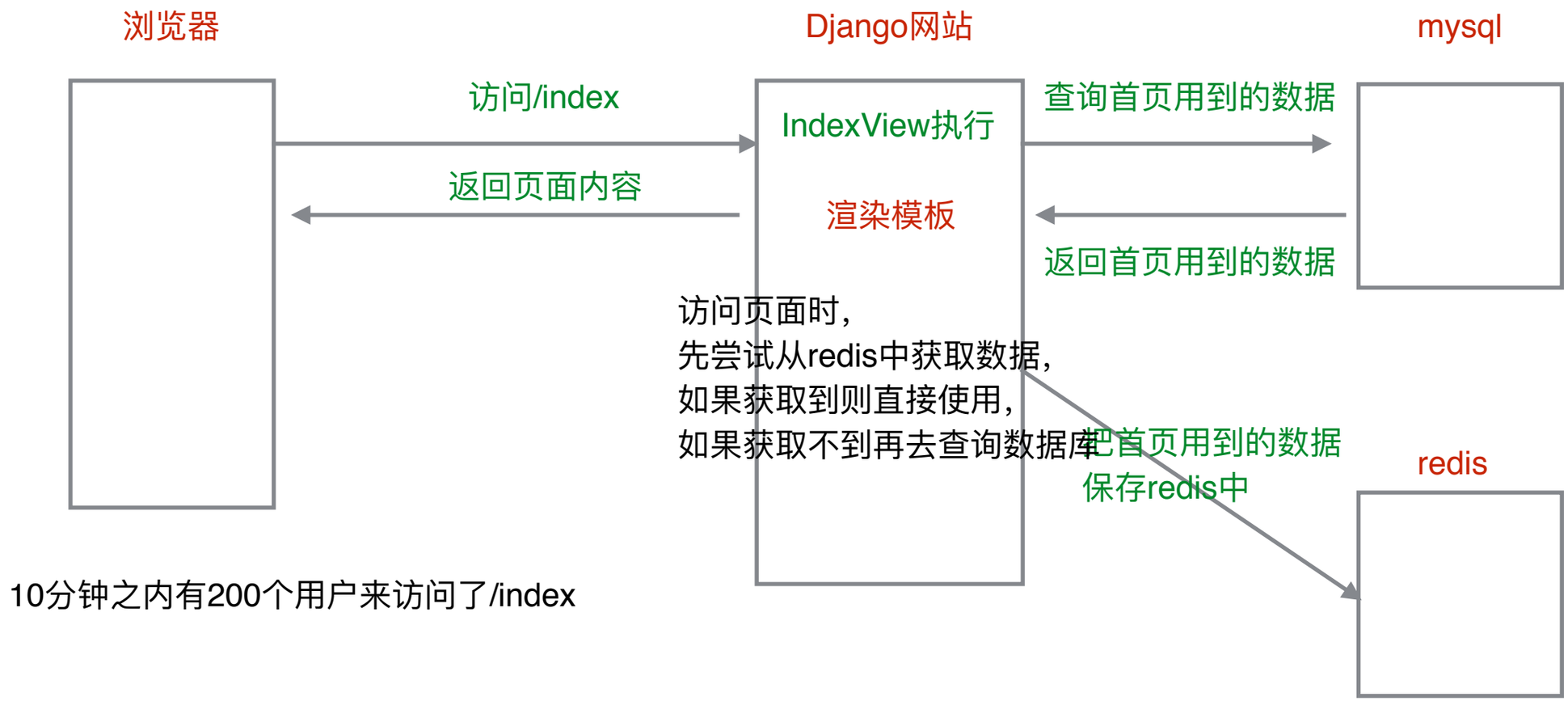
静态首页的重新生成

当首页用到的数据表中数据发生变化的时候，需要重新生成静态首页。



当通过admin管理页面修改了某个模型类对应的数据表的内容的时候，
如果是新增或更新，Django框架会调用模型admin管理类中的save_model, 由它实现新增或更新的操作。
如果是删除，Django框架会调用模型admin管理类中的delete_model, 由它实现删除的操作。

数据的缓存



10分钟之内有200个用户来访问了/index

缓存数据的更新：首页用到的数据表中数据发生变化的时候，需要更新首页的缓存。

DDOS攻击: 黑客组织很多电脑同一时间对网站发送访问。

服务器配置

127.0.0.1是本地的回环地址，除了本机的程序之外，
其他任何机器都不能连接。

smart虚拟机(172.16.179.142)

mysql服务:(172.16.179.142:3306)
redis服务:(172.16.179.142:6379)

MAC机器(172.16.179.1)

django服务器

fastdfs虚拟机(172.16.179.131)

tracker-server:（fdfs跟踪服务器）
172.16.179.131:22122)
storage-server:（fdfs存储服务器）
172.16.179.131:23000)

nginxweb服务器1：(提供fdfs系统中的图片):
172.16.179.131:8888
nginxweb服务器2：(提供生成的静态首页):
172.16.179.131:80

dailyfresh/settings.py文件配置项:

DATABASES:
HOST:172.16.179.142
PORT:3306

CACHES:
LOCATION:redis://172.16.179.142:6379/5

celery_tasks/tasks.py文件:

broker = redis://172.16.179.142:6739/4

utils/fdfs/client.conf文件:

tracker_server = 172.16.179.131:22122

商品搜索

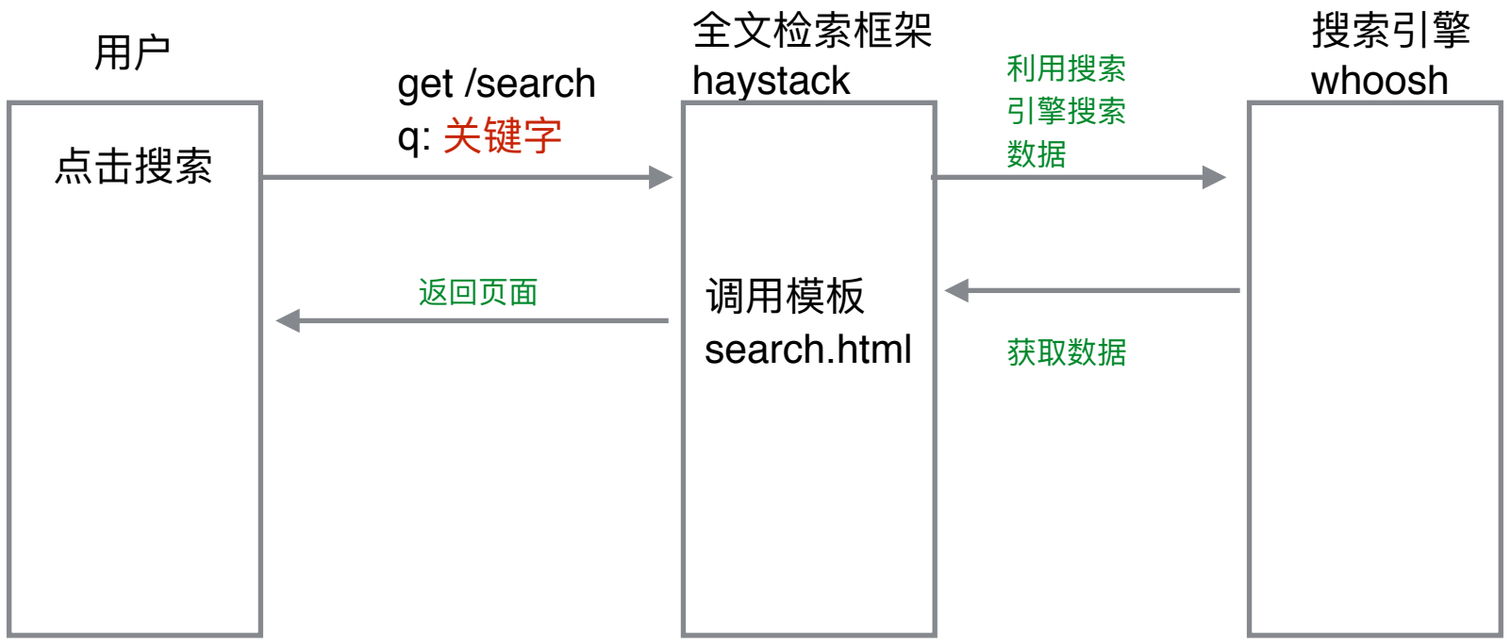
商品搜索: 根据商品的名称和简介搜索商品的信息。

```
select * from df_goods_sku where name like '%草莓%' or desc like  ' %草莓%';
```

搜索引擎: 搜索引擎可以根据表中某些字段的内容进行关键词的分析，建立关键词和表数据的对应关系(建立索引)。

'这是一个好吃的草莓':
这
是一个
好吃
的
草莓：1　3　5

全文检索框架: 支持多种搜索引擎，帮助用户来使用搜索引擎的功能。



订单创建

创建基本流程:

1) 用户每下一个订单，需要向df_order_info添加一条对应的记录。
2) 用户的订单中包含几个商品，需要向df_order_goods中添加几条记录。

订单ID：201803161117301
用户ID(外键): 1
地址ID(外键): 地址id
支付方式: 支付方式
商品总数(运营):
订单总额(运营):
订单运费: 10
订单状态: 1(待支付)
创建时间:

ID: 1
订单ID(外键): 201803161117301
SKU ID(外键): 5
商品数目: 1
商品价格(历史): 20

ID: 1
订单ID(外键): 201803161117301
SKU ID(外键): 4
商品数目: 2
商品价格(历史): 32

订单信息表
df_order_info

订单ID
用户ID(外键)
地址ID(外键)
支付方式
商品总数(运营)
订单总额(运营)
订单运费
订单状态
创建时间

订单商品表
df_order_goods

ID
订单ID(外键)
SKU ID(外键)
商品数目
商品价格(历史)
评论


网站用户
网站服务器
支付宝平台
点击去付款，
传递参数: order_id
组织调用
alipay.page.trade.pay
的参数信息
让用户浏览器重定向
访问调用支付宝aplpay.page.trade.py接口
调用alipay.page.trade.pay接口
让浏览器重定向访问支付页面地址
浏览器访问支付页面地址
返回支付宝支付页面
用户登录支付宝，选择支付方式，输入支付密码，确认支付
让浏览器重定向访问returl_url地址，再后面跟上同步参数

订单并发

当多个人同一时间够买同一件商品的时候，可能会出现订单并发问题。

商品库存: 10
A: 5 B:5
出现订单并发问题：两个人下单都成功，商品的库存变成了5件。

CPU：负责调度。

解决方案:

1) 悲观锁：在事务中查询数据的时候尝试对数据进行加锁(互斥锁), 获取到锁的事务可以对数据进行操作，获取不到锁的事务会阻塞，直到锁被释放。

2) 乐观锁: 乐观锁本质上不是加锁，查询数据的时候不加锁，对数据进行修改的时候需要进行判断，修改失败需要重新进行尝试。

1. 老师很好，东西都帮我们准备的很充分，就是项目要完了，老师要走了，我还是没学会升仙，哪位道友助我渡劫啊。

2. 支付宝里面的一些不是很清楚

3. celery和django-celery这两个包有什么区别

4. 支付宝沙箱的余额能提现么.....

5.项目好快就

6.print ('\n'.join([' '.join(['%s*%s=%-2s' % (y,x,x*y) for y in range(1,x+1)]) for x in range(1,10)])) 帅哥,讲一下这个乘法口诀代码

```
li1=[]
fox x in range(1,10):
    li2 = []
    for y in range(1,x+1): (1,2)
        li2.append('%s*%s=%-2s' % (y,x,x*y))
    str = ‘’.join(li2)
    li1.append(str)
‘\n’.join(li1)
```

7. web开发的时候，什么时候用表单form提交数据，什么时候用ajax提交呢？两者在什么时候可以通用呢?
感觉这个有点模糊，创建订单的时候可以用form表单提交吗?
页面需要局部刷新: ajax
form使用的场景都可用户ajax来实现。

8. 很慌，自己根本不会敲，怎么找工作。。
需要努力提高敲代码能力。

9.早上我很努力的不打瞌睡听课了，下午自习就趴了二十分钟却被路过的老师拍了照...
节哀

订单并发解决
（悲观锁）

用户A: 进程1

向df_order_info中添加一条记录

select * from df_goods_sku where id=<sku_id> for update;

获取到锁，可以进行操作

获取商品的信息　　库存: 10

判断商品的库存　　5<10

向df_order_goods中添加一条记录

减少商品库存，增加销量　　库存:5

事务结束，锁被自动释放。

用户B: 进程2

向df_order_info中添加一条记录

select * from df_goods_sku where id=<sku_id> for update;

没拿到锁，事务阻塞，其他事务释放锁之后获取到锁才能进行操作

获取商品的信息　　库存:5

判断商品的库存　　5<=5

向df_order_goods中添加一条记录

减少商品库存，增加销量　　库存:0

事务结束，锁被自动释放。

订单并发解决
（乐观锁）

库存: 10
A: 5
B: 5

用户A: 进程1

向df_order_info中添加一条记录

select * from df_goods_sku where id=<sku_id> ;

获取商品的信息　　origin_stock: 原始库存

从获取商品的库存到更新商品的库存过程中，商品的库存要求没有发生变化

判断商品的库存

减少商品库存，增加销量

update from df_goods_sku
set stock=<new_stock>,sales=<new_sales>
where id=<sku_id> and stock=<orgin_stock>;

更新成功，进行接下来的操作，否则重新尝试

向df_order_goods中添加一条记录

更新成功，跳出循环break

用户B: 进程2

向df_order_info中添加一条记录

select * from df_goods_sku where id=<sku_id> ;

获取商品的信息　　origin_stock: 原始库存

更新失败需要重新进行尝试，最多尝试3次，否则下单失败

判断商品的库存

减少商品库存，增加销量

update from df_goods_sku
set stock=<new_stock>,sales=<new_sales>
where id=<sku_id> and stock=<orgin_stock>;

更新成功，进行接下来的操作，否则重新尝试

向df_order_goods中添加一条记录

更新成功，跳出循环break

使用场景:

1) 冲突比较多的时候建议悲观锁。例如: 秒杀
2) 冲突比较少的时候建议使用乐观锁。

私钥:
公钥:

django网站服务器和支付宝平台，通过网络请求进行交互。

支付宝网关地址

实际生产环境: https://openapi.alipay.com/gateway.do
沙箱环境(模拟): https://openapi.alipaydev.com/gateway.do

SDK: 软件开发工具包。

商品订单支付流程分析

网站用户

网站服务器

支付宝平台

点击去付款，访问/order/pay

传递参数: order_id(订单编号)

调用python SDK中

```
order_string = api_alipay_trade_page_pay()
```

业务参数:

订单id

订单实付款

订单标题

return_url: 同步访问地址，notify_url:异步访问地址

返回应答，让用户的浏览器重定向访问

http://支付宝网关地址?order_string

访问http://支付宝网关地址?order_string，让支付宝调用下单支付接口

处理

支付宝平台让用户的浏览器重定向访问订单支付的页面

用户的浏览器重定向访问订单支付的页面

返回订单支付的页面

登录支付宝，选择支付方式，输入支付密码，点击确认支付

同步访问

让用户的浏览器访问return_url, 并且会url的后面跟上一些参数

用户的浏览器访问return_url

调用Python SDK中 api_alipay_trade_query

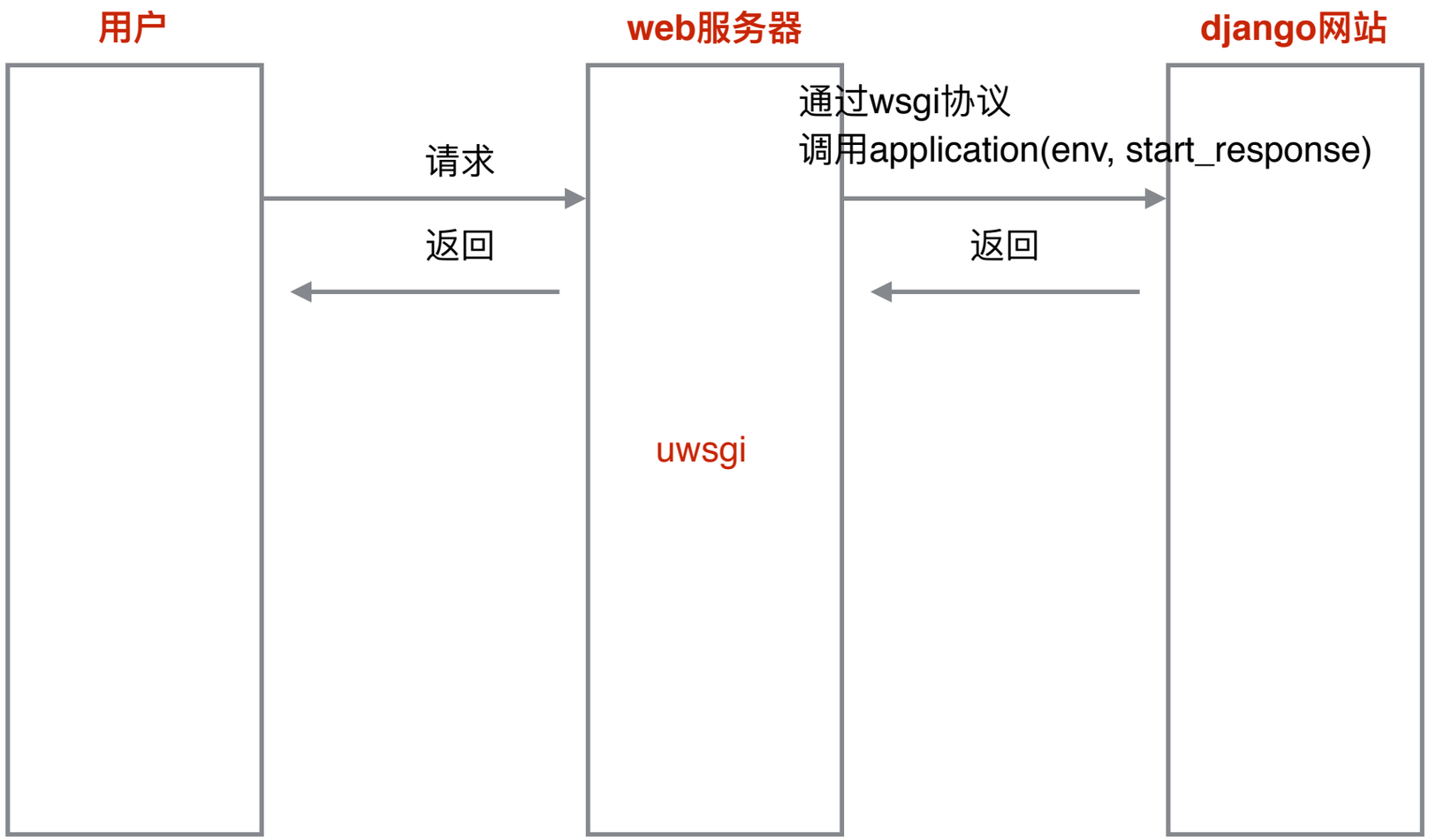
给用户显示支付结果

返回用户支付的结果

访问notify_url地址，告诉网站用户支付的结果

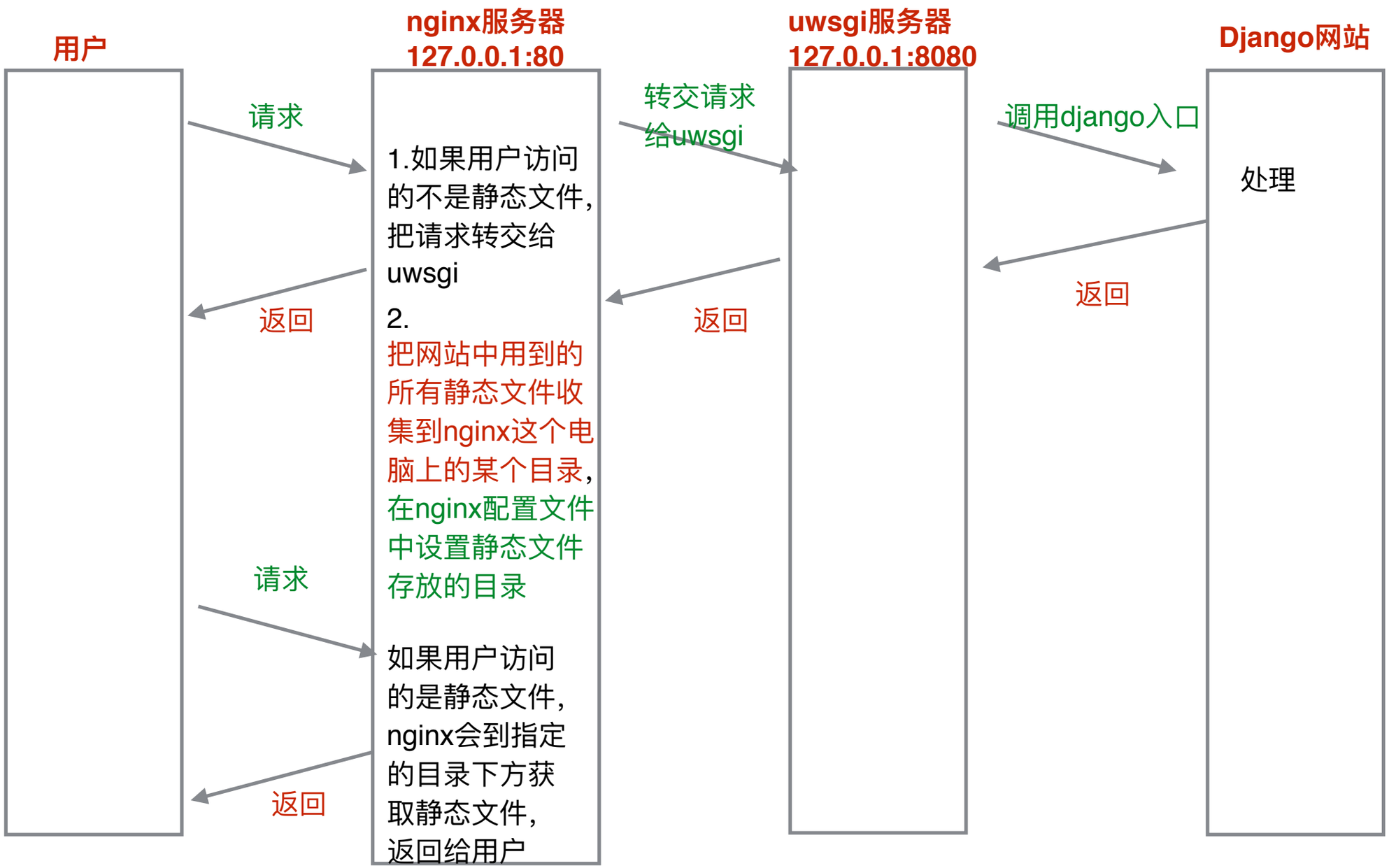
如同提供了notify_url, 网站服务器必须绑定公网ip;

uwsgi做web服务器



开发web服务器: python manage.py runserver

Nginx+uwsgi进行项目部署



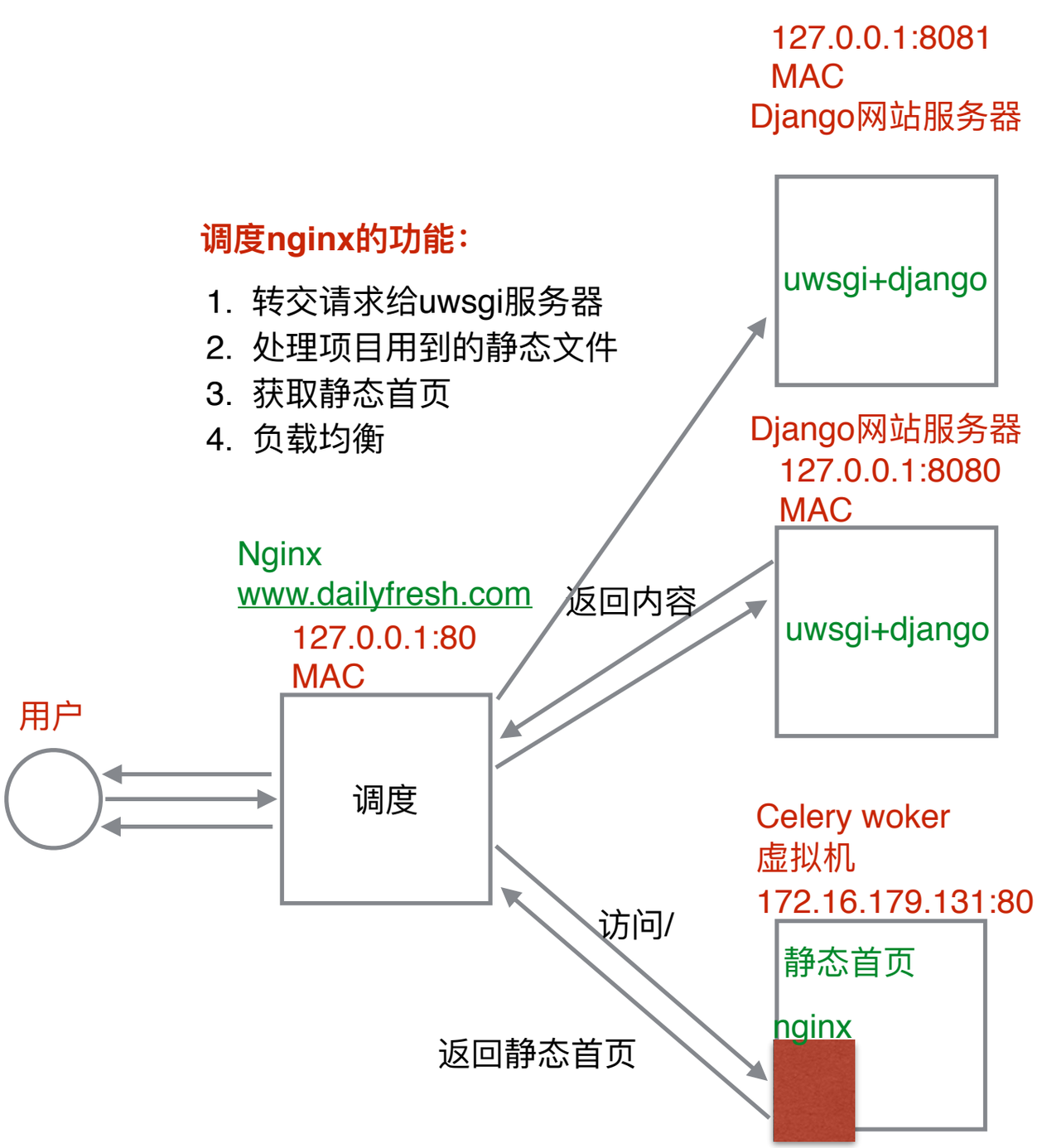
/static开头

1.配置nginx转发请求给uwsgi
2.配置nginx处理静态文件

生鲜项目部署架构

调度nginx的功能：

1. 转交请求给uwsgi服务器
2. 处理项目用到的静态文件
3. 获取静态首页
4. 负载均衡



mysql

172.16.179.142:3306

fdfs

tracker-server:
172.16.179.131:22122
storage-server:
172.16.179.131:23000
nginx:
172.16.179.131:8888

redis

redis:
172.16.179.142:6379
broker:
172.16.179.142:6379/4
caches:
172.16.179.142:6379/5