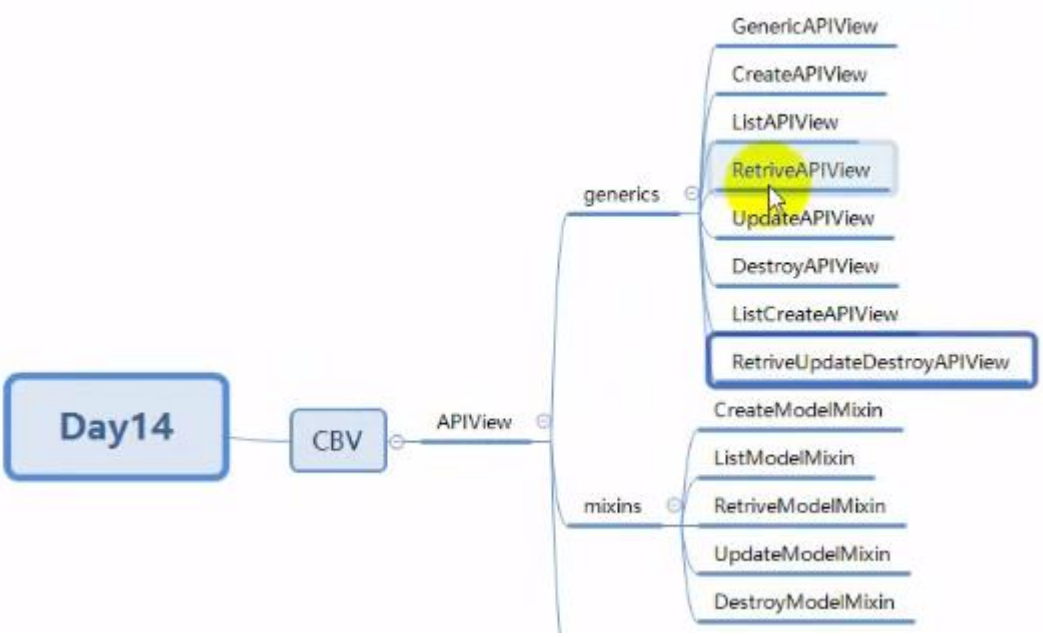
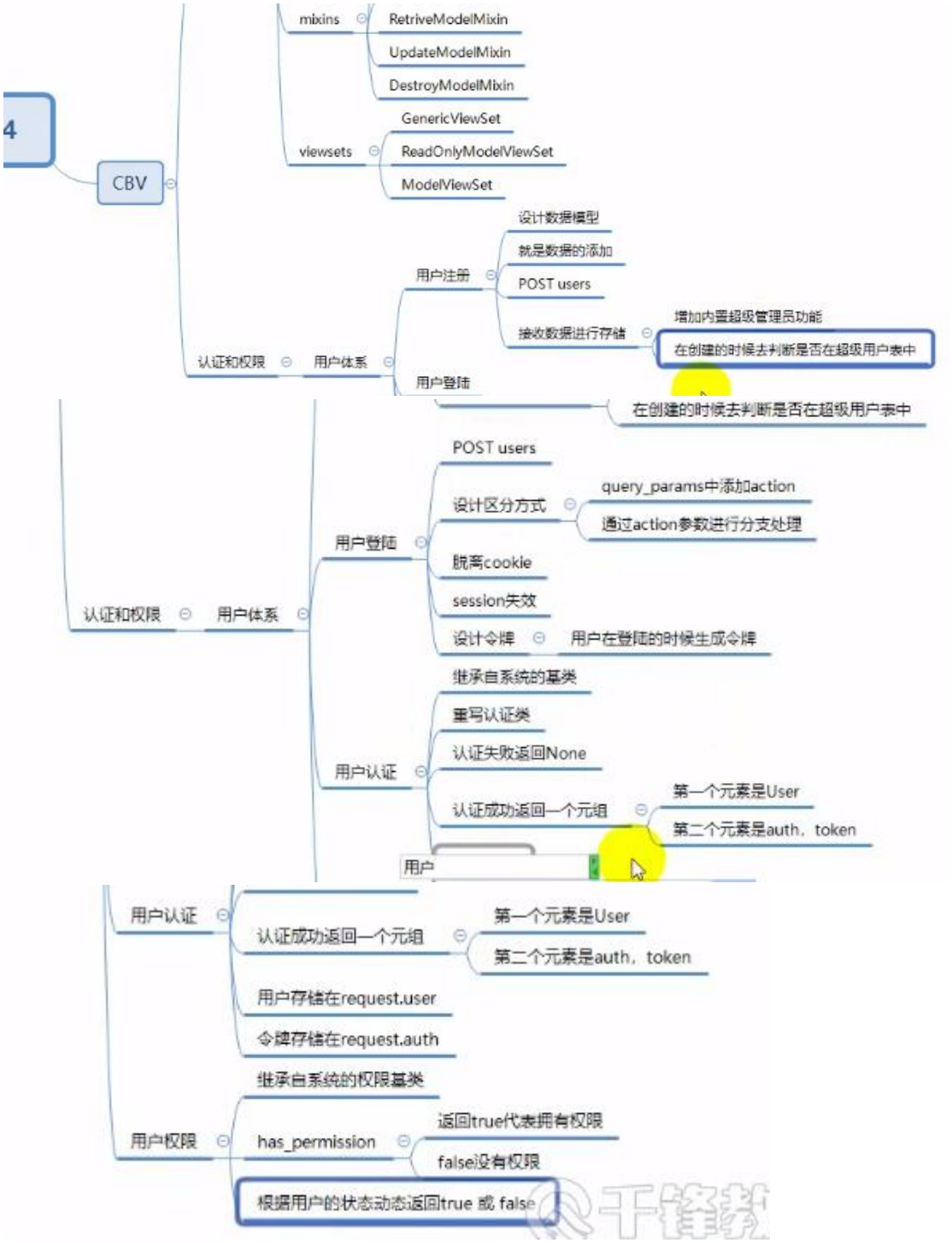
用户模块

- 用户注册
  - RESTful
  - 数据开始
    - 模型，数据库
    - 创建用户
      - 用户身份
        - 管理员
        - 普通
        - 删除用户
- 用户登陆
- 用户认证
- 用户权限

        - 删除用户
  - 注册实现
    - 添加了超级管理员生成
- 用户登陆
  - 验证用户名密码
  - 生成用户令牌
  - 出现和注册公用post冲突
    - 添加action
    - path/?action=login
    - path/?action=register
  - 异常捕获尽量精确
- 用户认证
- 用户权限

        - path/?action=register
    - 异常捕获尽量精确
- 用户认证
    - BaseAuthentication
        - authenticate
            - 认证成功会返回 一个元组
                - 第一个元素是user
                - 第二元素是令牌 token，auth
- 用户权限
    - BasePermission
        - has_permission
            - 是否具有权限
            - true拥有权限
            - false未拥有权限
- 用户认证和权限
    - 直接配置在视图函数上就ok了




4
CBV
mixins
RetriveModelMixin
UpdateModelMixin
DestroyModelMixin
viewsets
GenericViewSet
ReadOnlyModelViewSet
ModelViewSet
认证和权限
用户体系
用户注册
设计数据模型
就是数据的添加
POST users
接收数据进行存储
增加内置超级管理员功能
在创建的时候去判断是否在超级用户表中
用户登陆
在创建的时候去判断是否在超级用户表中
认证和权限
用户体系
用户登陆
POST users
设计区分方式
query_params中添加action
通过action参数进行分支处理
脱离cookie
session失效
设计令牌
用户在登陆的时候生成令牌
用户认证
继承自系统的基类
重写认证类
认证失败返回None
认证成功返回一个元组
第一个元素是User
第二个元素是auth，token
用户
用户认证
认证成功返回一个元组
第一个元素是User
第二个元素是auth，token
用户存储在request.user
令牌存储在request.auth
用户权限
继承自系统的权限基类
has_permission
返回true代表拥有权限
false没有权限
根据用户的状态动态返回true 或 false

Day15

需求

- 存在级联数据
- 用户和收货地址
- 节流

分析

- 数据开始
  - 模型定义
  - 用户和收货地址 一对多
    - 用户表
    - 地址表
  - 用户表
  - 地址表
    - ForeignKey
- 序列化
  - 级联数据如何实现序列化???
- 节流

技能点

- HTTP_X_FORWARDED_FOR
  - 获取你的原始IP
    - 通过的普通的代理发送请求的请求
    - 如果获取REMOTE_ADDR获取到的是代理IP
- 代理
  - 普通代理
  - 高匿代理
    - 效率越低，请求速度越慢

```
DEFAULTS = {
    # Base API policies
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',
    ),
    'DEFAULT_PARSER_CLASSES': (
        'rest_framework.parsers.JSONParser',
        'rest_framework.parsers.FormParser',
        'rest_framework.parsers.MultiPartParser'
    ),
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication'
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.AllowAny',
    ),
    'DEFAULT_THROTTLE_CLASSES': (),
    'DEFAULT_CONTENT_NEGOTIATION_CLASS': 'rest_framework.negotiation.DefaultContentNegotiation',
    'DEFAULT_METADATA_CLASS': 'rest_framework.metadata.SimpleMetadata',
    'DEFAULT_VERSIONING_CLASS': None,
```

```
REST_FRAMEWORK = {
    "DEFAULT_THROTTLE_CLASSES": (
        "rest_framework.throttling.UserRateThrottle",
    ),
    "DEFAULT_THROTTLE_RATES":{
        "user": "5/m",
    }
}
```

节流器

- BaseThrottle
  - allow_request
    - 是否允许的请求的核心
  - get_ident
    - 获取客户端唯一标识
  - wait
- SimpleRateThrottle
  - get_cache_key
    - 获取缓存标识
  - get_rate
    - 获取频率
  - parse_rate
    - 转换频率
      - num/duration

- num/duration
  - duration
    - s
    - m
    - h
    - d
- allow_request
  - 是否允许请求
  - 重写的方法
- throttle_success
  - 允许请求，进行请求记录
- throttle_failure
  - 不允许请求
- wait
  - 还有多少时间之后允许

```python
class ScopedRateThrottle(SimpleRateThrottle):
    """
    Limits the rate of API calls by different amounts for various parts of
    the API.  Any view that has the `throttle_scope` property set will be
    throttled.  The unique cache key will be generated by concatenating the
    user id of the request, and the scope of the view being accessed.
    """
    scope_attr = 'throttle_scope'

    def __init__(self):
        # Override the usual SimpleRateThrottle, because we can't determine
        # the rate until called by the view.
        pass

    def allow_request(self, request, view):
        # We can only determine the scope once we're called by the view.
        self.scope = getattr(view, self.scope_attr, None)

        # If a view does not have a `throttle_scope` always allow the request
        if not self.scope:
            return True

        # Determine the allowed request rate as we normally would during
```

直接将 scope 加到 view 中，如下：

```python
class UserAPIView(RetrieveAPIView):

    serializer_class = UserSerializer

    queryset = UserModel.objects.all()

    authentication_classes = (LoginAuthentication,)

    permission_classes = (RequireLoginPermission,)

    throttle_scope = "10/m"

    def retrieve(self, request, *args, **kwargs):
        if kwargs.get('pk') != str(request.user.id):
            raise exceptions.AuthenticationFailed
        instance = self.get_object()
        serializer = self.get_serializer(instance)
        return Response(serializer.data)
```

总结

RESTful

- django-rest-framework
  - serializers
    - 序列化工具
      - 序列化与反序列化
  - APIView
    - CBV
    - 实现各种的请求处理
  - mixins
    - CRUDL
    - 对模型操作
  - viewsets
    - 对APIView和Mixins高度封装
    - 可以对接 router
  - router
    - DefaultRouter
    - 可以直接批量注册路由
  - authentication
- authentication
  - APIView中会自动认证
  - 自己创建认证类，实现认证方法
    - 认证成功返回元组，用户和令牌
- permission
  - 添加权限控制
  - 用户所拥有的权限
- throttle
  - 节流
  - 控制访问频率
- 级联模型
  - 添加级联字段
  - nested
  - 级联字段的key原来必须就是存在的
    - 隐性属性
    - 自定义related_name