缓存框架的核心目标

- 较少的代码
  - 缓存应该尽可能快
  - 因此围绕缓存后端的所有框架代码应该保持在绝对最小值，特别是对于获取操作
- 一致性
  - 缓存API应该是提供跨越不同缓存后端的一致接口
- 可扩展性
  - 基于开发人员的需求，缓存API应该可以在应用程序级别扩展

缓存

- django内置了缓存框架，并提供了几种常用的缓存
  - 基于Memcached缓存
  - 使用数据库进行缓存
  - 使用文件系统进行缓存
  - 使用本地内存进行缓存
  - 提供缓存扩展接口

缓存配置

1. 创建缓存表

```
python manage.py createcachetable [table_name]
```

缓存配置

1. 创建缓存表

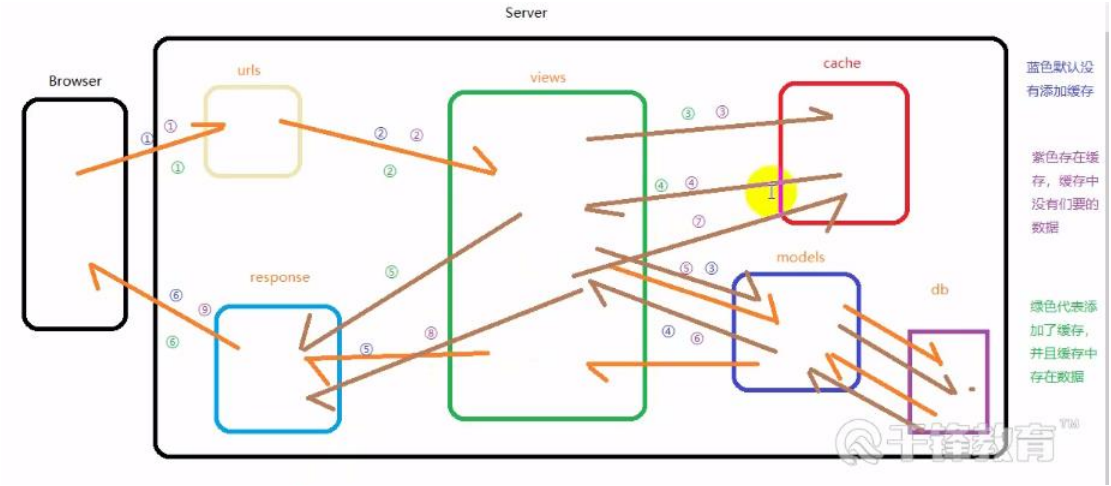
```
python manage.py createcachetable [table_name]
```


2. 缓存配置

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'my_cache_table',
        'TIMEOUT': '60',
        'OPTIONS': {
            'MAX_ENTRIES': '300',
        },
        'KEY_PREFIX': 'rock',
        'VERSION': '1',
    }
}
```

缓存使用

- 在视图中使用（使用最多的场景）
- @cache_page()
  - time 秒 60*5 缓存五分钟
  - cache 缓存配置, 默认default,
  - key_prefix 前置字符串

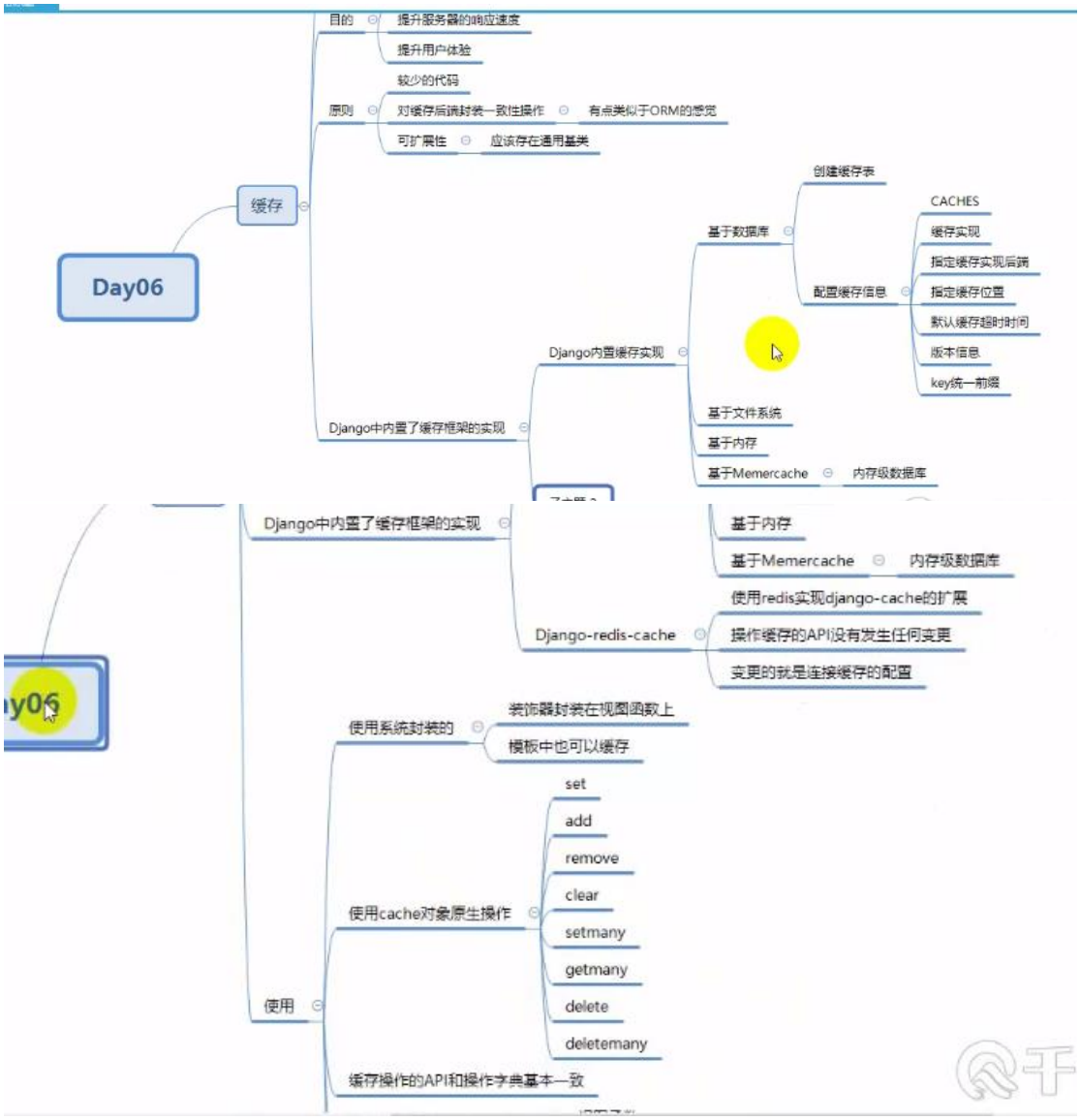


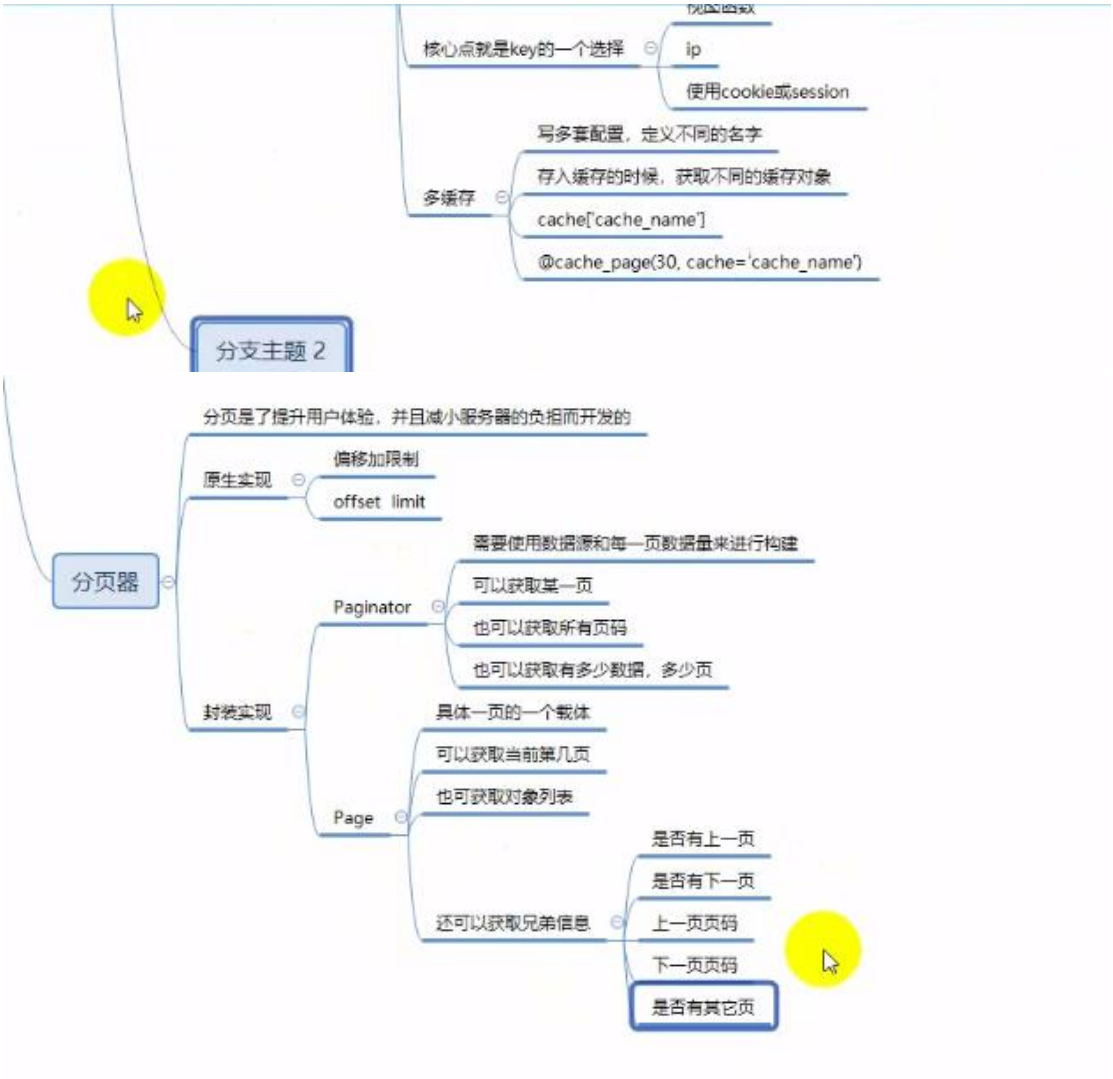
- 缓存操作
  - cache.set
    - key
    - value
    - timeout
  - get
  - add
  - get_*or*_*set*
  - get_many
  - set_many
  - delete
  - delete_many

  - clear
  - incr 增加
    - incr(key, value) key对应的值上添加 value
  - decr 减少
    - decr(key, value) key对应的值上减少value
    - 如果value不写，默认变更为1

```
CACHES = {
    'default': {
        # 'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
        # 'LOCATION': 'my_cache_table',
        # 'TIMEOUT': 60 * 5
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    }
}
```

配置 redis




视图函数
核心点就是key的一个选择
ip
使用cookie或session
多缓存
写多套配置，定义不同的名字
存入缓存的时候，获取不同的缓存对象
cache['cache_name']
@cache_page(30, cache='cache_name')
分支主题 2
分页器
分页是了提升用户体验，并且减小服务器的负担而开发的
原生实现
偏移加限制
offset limit
封装实现
Paginator
需要使用数据源和每一页数据量来进行构建
可以获取某一页
也可以获取所有页码
也可以获取有多少数据，多少页
Page
具体一页的一个载体
可以获取当前第几页
也可获取对象列表
还可以获取兄弟信息
是否有上一页
是否有下一页
上一页页码
下一页页码
是否有其它页