断点调试的方式适合初学者，高效率的方式是打印日志。

打印日志的代码不需要删除，不必使用assert，assert一般用于unittest。

Django rest framework，简称drf

将和drf API有关一般放到api文件夹下。

联合索引：index\_together

index会存在包含关系，例如建联合索引(‘a’, ‘b’, ‘c’)后，不需要再建(‘a’, ‘b’)。

Mysql，Primary key使用自增主键id，一般不是uuid。一个重要的原因是历史原因，最初就是用整型自增id，非关系型数据库采用UUID。第二个原因是uuid比较长，而整型id比较短，存储更方便。第三个原因是整型id产生有时间序，id越大就越新。如果不需要拆分数据库，就不需要使用uuid。

自增主键整数id拆分数据库会产生一个问题，如何保证各个机器上产生的新数据的id是不重复的。

一个比较简单的方法：存数据的数据库不产生id，在存储数据前先去指定的中心机器上取id，中心机器上负责进行+1操作，可以使用Redis以保证这个操作是原子操作，然后把id返回给存数据的机器，在存数据的mysql就不设置自增id了，而是简单的Integer。缺点是中心机器必须高可用，不能出问题。

参考方法如下：<https://www.zhihu.com/question/30674667>

首先假设单库的目前的唯一id为3559，要拆分成3台机器，则第一台机器的索引从3559\*65536+1开始，索引一次为3559\*65536+1,2,3,…65535，第二台机器为3560\*65536+1,2,3,…+65535，第三台机器为3561\*65536+1,2,3,…+65535，当第一台机器主键自增量到65535时，就换一个自增id开始，如3562\*65535+1,2,3…65535，其它机器也是如此。

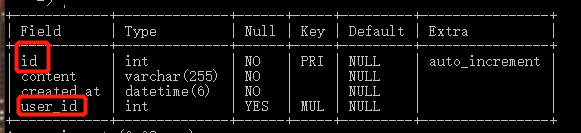
uuid比较适合分布式数据库，它是全球唯一的。

Django中创建了Tweet模型后，通过浏览器访问服务器时，发现其并不生效，这是因为没有修改数据库的原因。需要执行

python manage.py makemigrations

python mange.py migrate

执行后，查看mysql数据库：



即使是在一个app下，也可能存在多个tests.py用于测试，tweets下面有tests.py，它的api文件夹下也有tests.py。

Django中如果模型中创建了新的索引，也需要更新数据库，使用makemigrations和migrate

ModelViewSet尽可能不用，因为它默认提供了增删查改数据库的功能，但大多数情况下，设计的ViewSet其实并不具有这些权限。

设计Tweet相关的API

TweetViewSet，rest framework中ViewSet不再提供Django中View的get()和post()，取代的是list()和create()

接口list是获取tweet列表，访问时需要通过?操作符传入user\_id，因为我们不会要求列出所有tweet，而是某个用户的tweet

/api/tweets/user\_id=4

在浏览器执行：http://localhost/api/tweets/?user\_id=3

下图中加红框的部分，如果TweetSerializer中没有添加user = UserSerializer()，就不会出现，而是会变成”user”: 3，也就是只显示user\_id。



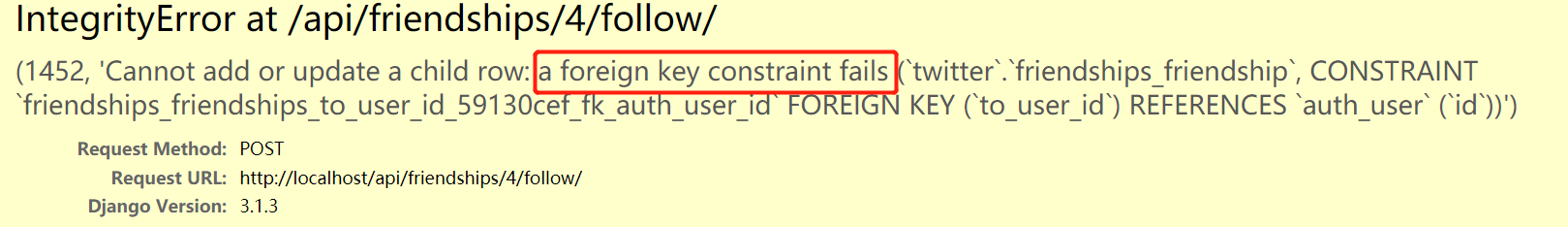
Django中unittest指定的测试文件：

python manage.py test tweets.api.tests -v2

更进一步只测试某个函数：

python manage.py test tweets.api.tests.TweetApiTests.test\_list\_api

Friendship模块中，在浏览器测试follow时，如果指定的to\_user\_id不存在，就会抛出外键失败的错误：



MySQL使用：

1. 不要使用JOIN
2. 不要用CASCADE
3. 数据量比较大的话，删除外键

Friendship模块中，可以把followers和followings都整合到list中

URL就是/api/friendships/?type=follower&to\_user\_id=1

FriendshipService中get\_followers()方法中，N+1次Queries产生的原因：

friendships = Friendship.objects.filter(to\_user=user)

return [friendship.from\_user for friendship in friendships]

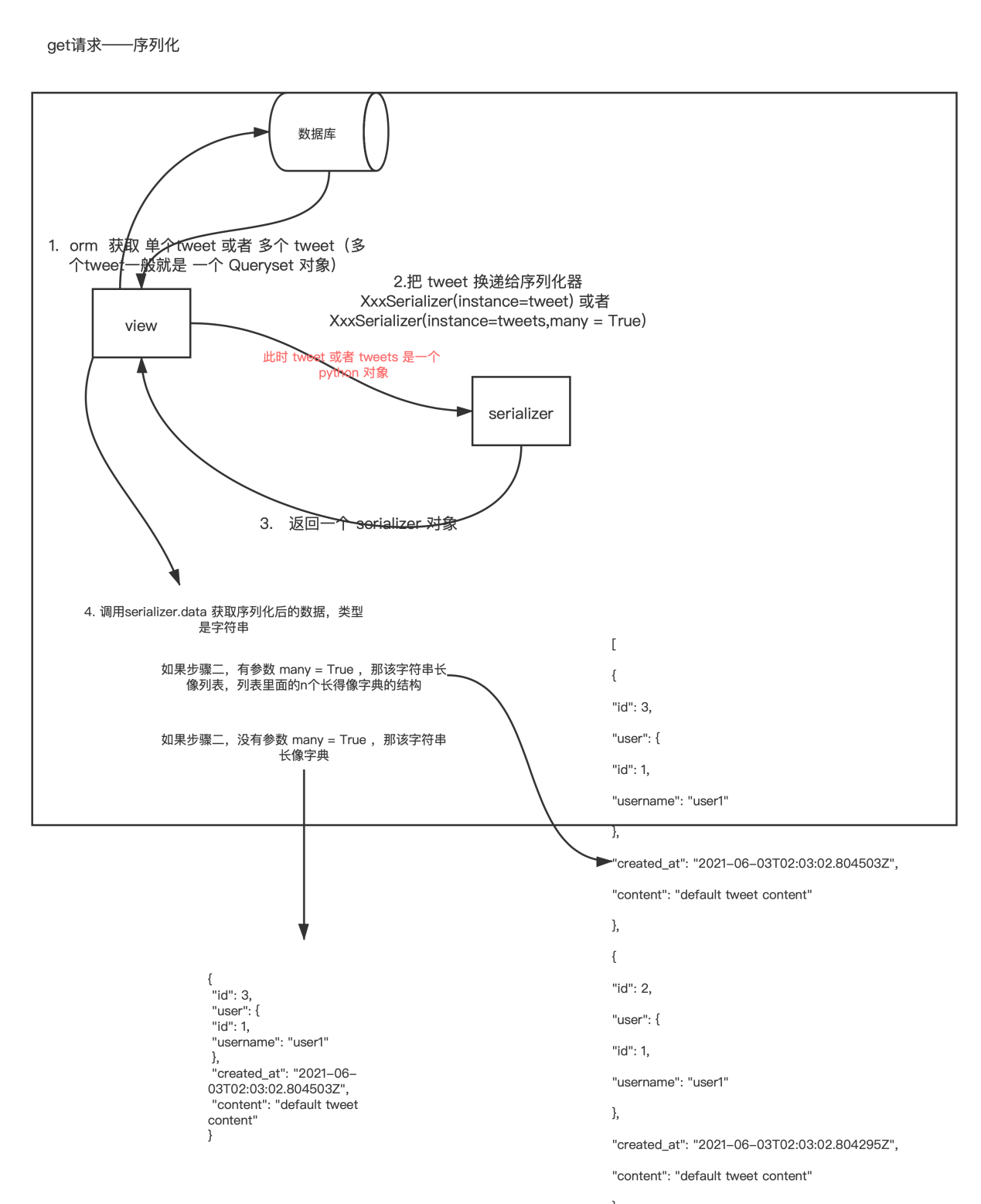
queryset是懒惰加载的，执行第一行代码时不会查询数据库，第二行代码对friendships进行迭代时，就会查询数据库得到所有to\_user=user的friendships，这是1次Query，之所以后面还有N次Queries，是因为from\_user是friendship表单中的外键，数据库表单中存储的是from\_user\_id，所以迭代执行friendship.from\_user时还要根据from\_user\_id从User表单中查找User对象，因此产生了N次Queries。如果访问的字段是friendship表单中其它非外键的字段，其实这个for循环不会产生N次Queries。

router把request传给view，views接收request，然后调用serializer进行序列化和反序列化。用户post过来的数据通过serializer进行处理，其实是反序列化，要验证数据是否有效，合法，是否满足Model所需要的字段。当最后返回的Response内容包含查询的结果时，是一个序列化的操作，通常序列化为一个json格式，以方便传递给前端进行展示。

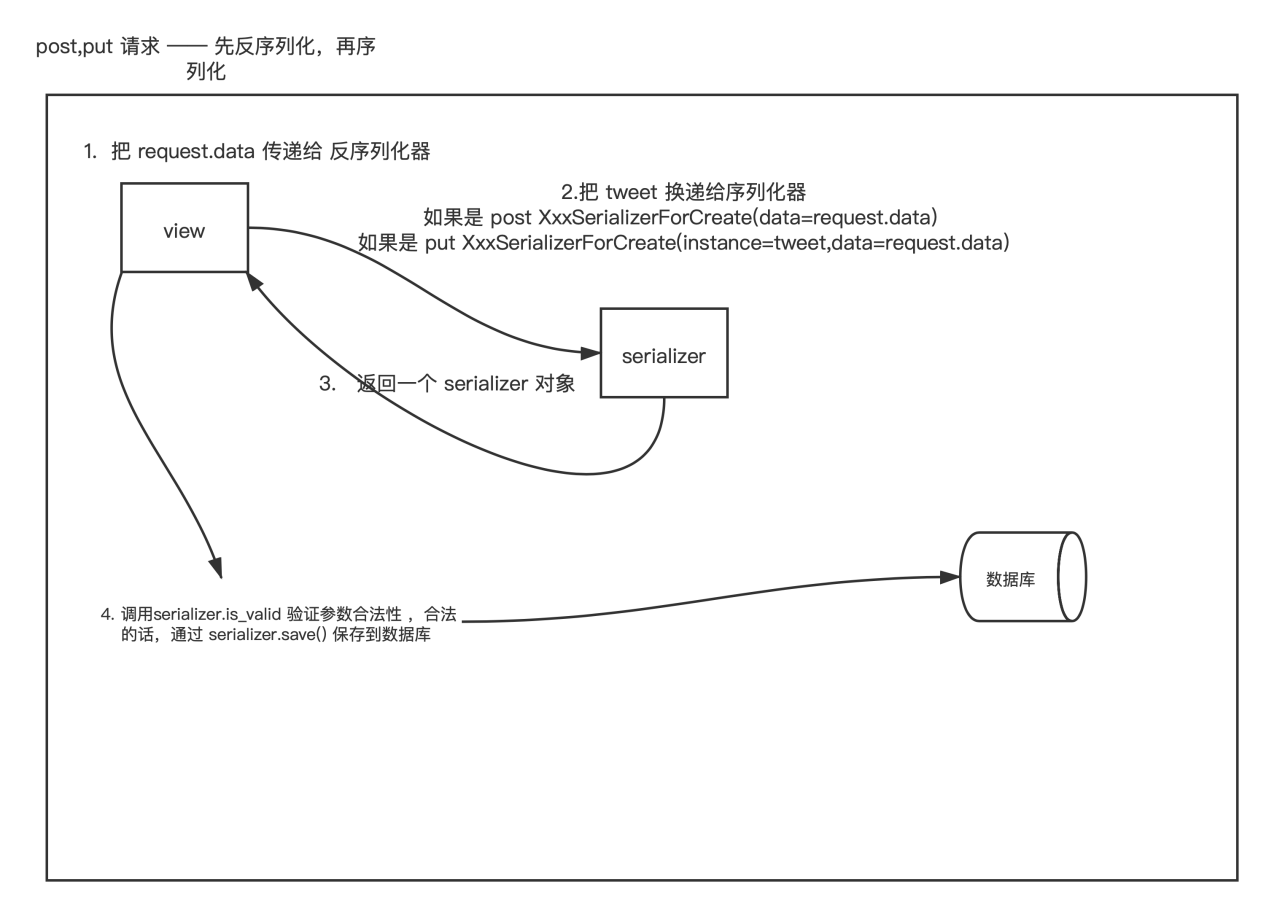
views就是接收request，返回response。其实可以不用serializer，但这样就需要自己对数据进行校验，并且自己对返回的response进行序列化。views就是一个联动各方的组件。

serializer的作用：可以不用serializer

序列化：



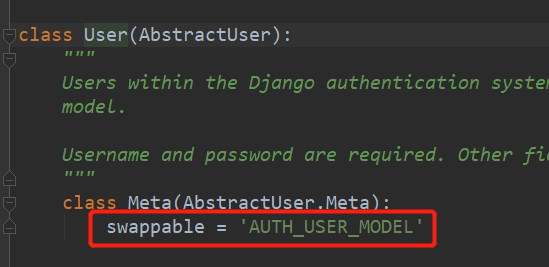
反序列化：



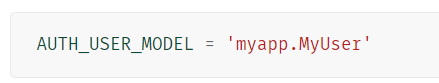
擅长使用Google，不要使用断点调试而是打印日志，学会看源码。

api文件夹里一般都存放和drf相关的文件，表示这一般都是restful风格的后端api，而和django相关的一般都放在tweets，friendships，newsfeeds下面。

用户权限的划分，django中默认的用户模型：



其中，swappable描述的是settings.py可以添加的一个字段，用来表示使用自定义的用户模型：



django提供了一些权限管理，但通常不够，需要自己写一个permission模型。django没有提供细微的行为权限。

对于数据库中的文本内容，例如tweet中的content进行内容搜索或排序，用到的就不是数据库的技术了，而是搜索引擎的技术，倒排索引。将content中的文本按照一定的规则进行拆分，例如拆分单词，中文的分词等。这样就形成某个词条出现在哪些tweet中，倒排索引。可以使用elastic search等成熟的全文搜索引擎。

Primary key的作用：

1. 表单中唯一标识一条记录，可以进行查询，mysql会自动添加一个primary key的index。
2. 在其他表单中作为外键。

django中MVC，M也是Model，View其实对应的C，而V则是前端来实现。后端一般都只做M和C。

mysql中的Key的作用有两种，一是约束作用，而是索引，所以Key不可以重复，而index可以重复，如果不想重复，就需要unique index，unique index就相当于Key了。