发的tweet是不允许修改的，原因是：tweet往往有评论和点赞，如果可以修改tweet，改变了tweet的意思，会导致评论和点赞被恶意使用，例如发tweet骂吴亦凡，会收到好评和点赞，但如果又把tweet改成支持吴亦凡，结果造成有大量好评和点赞，显然这是一个恶意的操作，所以tweet如果认为不合适，那就删掉，重新发tweet，被删除的tweet相关的评论和点赞也不会关联到新tweet上。

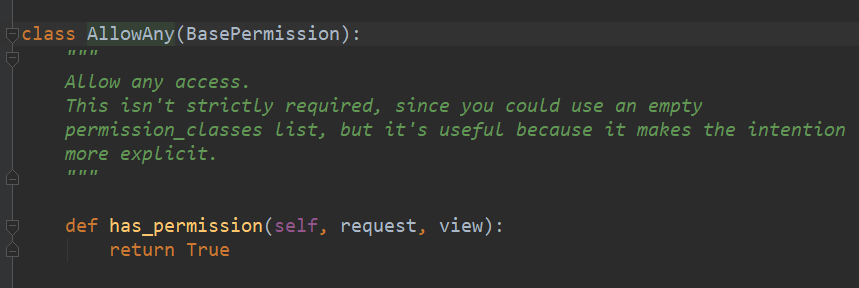
劝分不劝和，serializer在创建和展示应该用不同的，CommentSerializerForCreate和CommentSerializer。

权限管理：不在函数中添加所谓的if权限检测，原因：

1. views接收request，并且分发给其它模块，不应该做其它过多功能，导致代码可读性极差
2. permission可以复用，如果直接写在函数中，会导致重复代码很多。

IsAuthenticated在Web中就是是否登陆，AllowAny其实什么都没做，就是允许任何用户。

self.get\_object()：url中带有primary key，即detail=True的接口中才能这样用。



permission如果权限验证没有通过，就会抛出403错误，这是权限相关的状态码。不要抛出400错误，因为状态码在服务端会有专门的处理方式，例如抛403表示客户端在干坏事，想访问不该访问的资源，这是人有问题，而400则是表示客户端参数错误，例如名称，类型有问题，有可能是服务器没有兼容客户端，客户端没有升级等。

环境分不同种：测试环境，本地环境，alpha版(内测，假的数据库，自己造的，尽可能模拟现实环境)，beta(公测版，真实数据，相比production进行了某些限制，例如限制某项功能只有10%的人可用)，production(正式版，连接的数据库都是真实的)。

beta和production的区别：很多bug只有在真实数据才能测试，beta公测版是一点一点开放，这样有bug就能及时修复，不能一下就帮所有的功能都放上去，否则一旦有严重的bug，可能会导致系统完全崩溃。

Like模型中可以存储一个object\_id和一个content\_type，其中content\_type是一个字符串，例如’tweet’, ‘comment’等，但这有个较大的缺点，就是之后如果要修改Tweet，Comment的名字就会比较麻烦，所以写代码不要抱着一成不变的想法。

为什么Like不存成TweetLike和CommentLike？

1. 这两个Like模型会有重复代码，以后一旦需要在Like增加信息，总是需要改两份。
2. 这两种点赞在概念上相同的，只是某个参数不同，不适合区分表单。区分表单一定是结构不同，而不能是仅仅某个参数不同。例如，人和动物肯定要用不同的表单，而员工表单中领导和普通员工是不需要区分的，只需要在表单中区分某个或某几个属性就行了。

评论comments模块，一个评论只能对应一条tweet，而一条tweet可以有多条评论，这是一个一对多的关系，所有comment模型中，tweet就是外键，同样的一条评论只能由一个用户发出，而一个用户可以发多条评论。

评论可以进行修改，因为评论是对应tweet的，但tweet不应支持修改。

CommentViewSet的list api肯定不会去获取全部的comments，而是获取某条tweet的comments。

likes点赞应该既可以点赞tweet又可以点赞comment，这个应该设计在同一个表单中，应为这两个结构很类似。

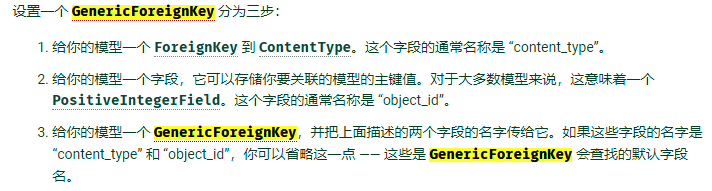
性能测试：一个api的代码从头执行到尾，所有的sql query的查询控制10这个量级，千万不要变成百这个量级，服务器好可以是几十个sql query，服务器差就控制在个位数sql query，而且sql query不能是较慢的数据库请求，例如没有index。

并发的数据库请求：不一定好，例如10个query，每个10ms，一个一个执行，需要100ms，如果并发执行则需要10ms，但这只是局部上有效果，对应整个系统而言，数据库的处理能力是有限的，如果使用并发，反而更有可能导致数据库崩溃掉，因为并发会导致数据库短时间的访问量更高。系统中使用Message Queue就是为了防止这种现象，进行“削峰”操作。

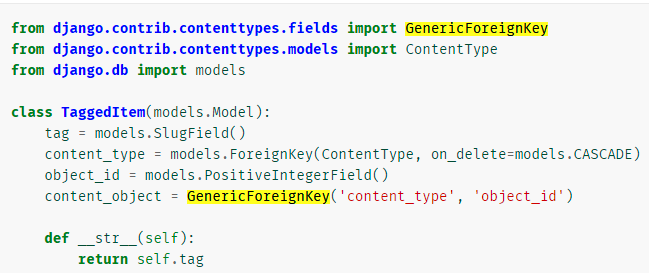
压力测试：用第三方工具测试。

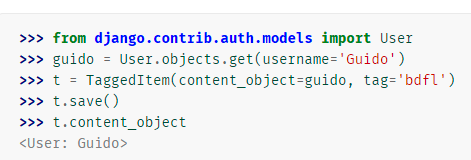
Go语言：继承了Python的易学易懂和C++的高效。

GenericForeignKey：类似Like模型，它的外键可以是Comment，也可能是Tweet，如果设置两个外键是不合适的，产生了冗余，另外如果以后Like还关联其它模型，则会在Like模型中不停的添加ForeignKey，这显然不合适。可以使用GenericForeignKey，类似一个通用外键或泛型外键，允许与任何模型建立关联。



例：

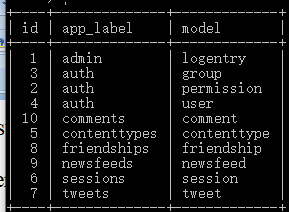




可以看出content\_object就是关联的模型对象，是根据object\_id和content\_type查出的模型对象。

注意：content\_object不会在数据库中生成，所以不能将其用在filter或get等类似查询过滤的方法中，不会产生效果。

ContentType，在mysql数据库中，有一个django\_content\_type表单，这是django自带的表单，其内容为工程中各个app对应的model



likes模块中，点赞是post请求调用了create api，在设计取消点赞时，如果使用delete请求调用destroy api就会有一个问题，因为客户端的点赞按钮当点赞后，就会设置为“取消点赞”状态，如果使用delete来取消点赞，就必须等待之前create api执行完毕并且返回like id，否则就无法知道要delete哪个赞，这种体验是不好的。

Comment和Tweet中都有点赞的相关信息。