HTTP的request请求到达router，router转换给Views，Views再发给Serializer。

Serializer的两个作用，序列化和反序列化。

反序列化：将用户post的数据(request.data，一般就是字符串，也可能是自定义的一个字典类型的data)转换为django的ORM格式对象，例如：



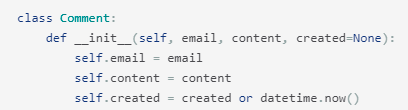
反序列化得到的也是一个字典对象serializer.validated\_data

序列化：Serializers可以把复杂的数据例如queryset,model instances转换为Python原生的数据类型，进而可以呈现为JSON,XML等内容。

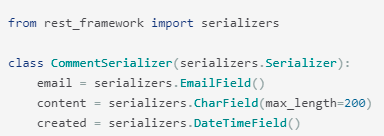


自己声明一个Serializer，例：

类：



Serializer：



使用Serializer来序列化对象：序列化后的结果是serializer.data



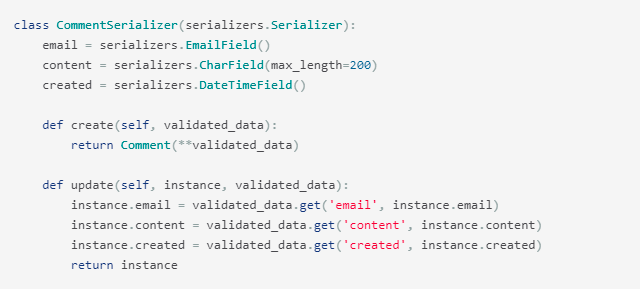


Serializer可以把数据库中存储的数据序列化为json或xml等，以方便之后的业务逻辑操作数据。

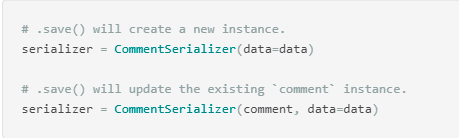
使用Serializer反序列化：反序列化会校验输入的数据，看是否可用，例如存储到数据库中。得到的validated\_data是一个dict，它的keys值就是Serializer中定义的字段名，如果指定了fields，则keys就是fields。



如果希望根据反序列化的validated\_data来创建类对象，需要在Serializer中实现create或update方法



调用.save()返回一个类对象。

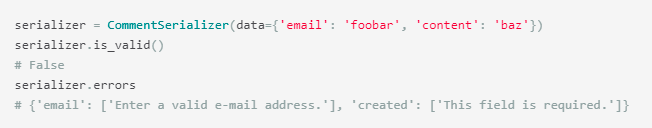




也可以重载.save()，自己实现。

校验(Validation)：

反序列化时，需要调用.is\_valid()进行校验(如果不调会抛出异常)，如果发生错误，在serializer的.errors会存储相关错误



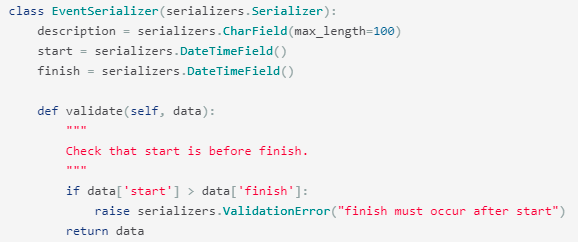
.is\_valid()会根据Serializer中定义的字段进行检测，如果定义了class Meta并且定义了model，例如model=User，is\_valid也会调用User相关的检测。Serializer也会调用默认的validate函数，也可以自己实现并重载。

字段级的校验：函数名为validate\_<field\_name>

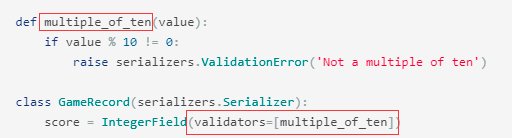


注意：如果你在serializer中声明字段时，required=False，假如该字段为空时，这个validate\_<field\_name>函数就不会执行。

对象级的验证：重载validate()函数

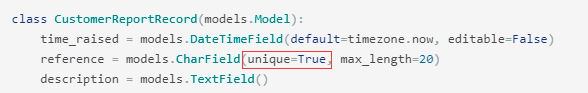


Serializer中的单个字段上可以使用validators(验证器)

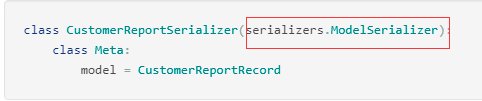


ModelSerializer：

如果serializer继承至ModelSerializer而不是直接继承至Serializer，则会自动处理校验，默认实现create和update。假如需要进行校验的类继承至Django的Model类，可以使用这样的serializer。例：



Serializer：



可以看出，与之前继承至serializers.Serializers相比，不需要在CustomerReportSerialzier实现相应的字段定义，只需要在嵌套类Meta中定义model=CustomerReportRecord就行了。

>>> serializer =CustomerReportSerializer()

>>>print(repr(serializer))

CustomerReportSerializer():

id =IntegerField(label='ID', read\_only=True)

time\_raised =DateTimeField(read\_only=True)

reference =CharField(max\_length=20, validators=[<UniqueValidator(queryset=CustomerReportRecord.objects.all())>])

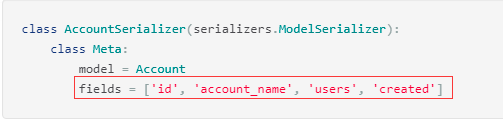
description =CharField(style={'type':'textarea'})

查看CustomerReportSerializer的类对象看出，其中自动包含了CustomerReportRecord中的字段，并且部分类型自动转换为rest\_framework中的类型，例如TextField转换为了CharField(style={'type': 'textarea'})，其中，字段reference自动添加了验证器UniqueValidator。Django中的Model会默认包含id字段，它是自增的主键。

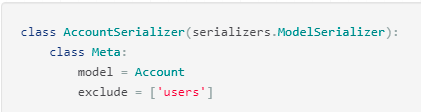
指定包含哪些字段：

ModelSerializer会默认生成一个基于Model的字段集，可以手动指定包含哪些字段或排除哪些字段。从3.3.0版本后，必须提供fields或exclude，如果希望包含所有字段，

包含哪些字段：



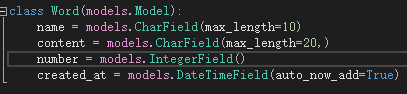
排除哪些字段：

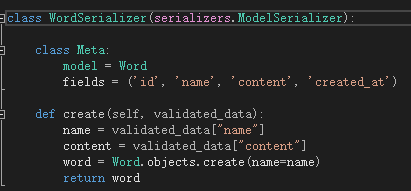


如果在Serializer中定义一个字段，但没有在fields或exlcude中指明，会抛出异常：

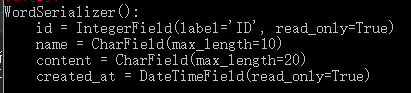


ModelSerializer中fields中的字段必须包含自定义的所有字段，然后可以把model模型中的字段添加进去。在浏览器中访问某个api接口时，如果接口对应的viewset的serializer\_class使用了ModelSerializer，则fields中指定的字段会渲染到html页面上，例如：





在python manage.py shell中，查看WordSerializer的对象：



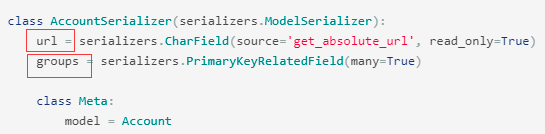
从上面可以看出，Word模型中有number字段，但WordSerializer中的fields中没有包含number，所以WordSerializer中最终也没有number属性。

浏览器访问时：可以看出渲染了name和content字段，fields中其它的字段没有进行渲染，从上述打印的WordSerializer对象可以看出，其它字段id和created\_at都是read\_only=True，只读字段



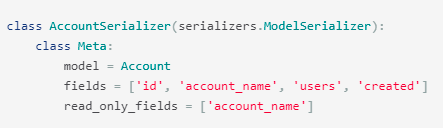
ModelSerializer的创建和渲染是两套逻辑

可以向ModelSerialzier中添加额外的字段，或覆盖ModelSerializer默认的字段：



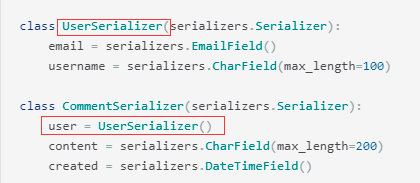
其中url是覆盖之前ModelSerializer自带的，groups是额外添加额字段

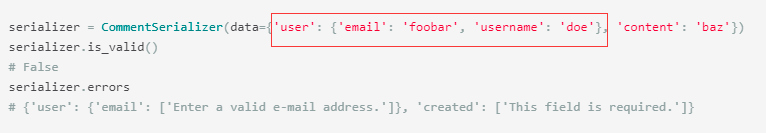
指定只读字段：可以把Modle的字段设置read\_only=True，也可以例如read\_only\_fields：



Model字段如果设置editable=False，和AutoField字段会自动设置read\_only=True，不需要添加到read\_only\_fields中。

Serializer中还可以包含另一个Serializer，例：





Serializer中字段构造的参数：

read\_only：read\_only=True的字段会包含在API输出中，即序列化的json数据会包含，但serializer进行create或update操作时，会忽略read\_only=True的字段，而且也不会校验它，无论输入参数中是否有该字段。

write\_only：write\_only=True的字段可用在(如果输入数据包含)create或update时使用，但序列化输出时不会显示该字段。

required：required=True的字段输入时必须传入，如果没有，校验时会抛出错误，如果设置为False，则为非比传字段，而且也允许序列化时忽略该字段。

default：字段的默认值

allow\_null：allow\_null=True的字段允许输入时传入None，反之，allow\_null=False，传入None时会抛出错误。

source：字段值来源，可以包含一个self的方法，，也可以是属性值。

validators：校验器