# Request和responses

## request

Request继承HttpRequest,提供更灵活的请求解析，核心功能是

Request.data属性，类似request.POST

request.POST # Only handles form data. Only works for 'POST' method.

request.data # Handles arbitrary data. Works for 'POST', 'PUT' and 'PATCH' methods.

## Response

 a Response object, which is a type of TemplateResponse that takes unrendered content and uses content negotiation to determine the correct content type to return to the client.

return Response(data) # Renders to content type as requested by the client.

## 状态码

such as HTTP\_400\_BAD\_REQUEST in the status module. It's a good idea to use these throughout rather than using numeric identifiers.

## 包装views

1. The @api\_view decorator for working with function based views.
2. The APIView class for working with class-based views.

These wrappers provide a few bits of functionality such as making sure you receive Requestinstances in your view, and adding context to Response objects so that content negotiation can be performed.

The wrappers also provide behaviour such as returning 405 Method Not Allowed responses when appropriate, and handling any ParseError exception that occurs when accessing request.datawith malformed input.

## 添加可选的url后缀

视图函数添加可选参数format=None,然后追加一些format\_suffix\_patterns

路由urlpatterns = format\_suffix\_patterns(urlpatterns)。客户端则可以指定返回数据的格式：

http http://127.0.0.1:8000/snippets/ Accept:application/json # Request JSON

http http://127.0.0.1:8000/snippets/ Accept:text/html # Request HTML

或者

http http://127.0.0.1:8000/snippets.json # JSON suffix

http http://127.0.0.1:8000/snippets.api # Browsable API suffix

也可以使用Content-Type控制请求数据格式

# POST using form data

http --form POST http://127.0.0.1:8000/snippets/ code="print(123)"

{

"id": 3,

"title": "",

"code": "print(123)",

"linenos": false,

"language": "python",

"style": "friendly"

}

# POST using JSON

http --json POST http://127.0.0.1:8000/snippets/ code="print(456)"

{

"id": 4,

"title": "",

"code": "print(456)",

"linenos": false,

"language": "python",

"style": "friendly"

}

# 基于类的视图

更利于重用代码，把各http方法分开

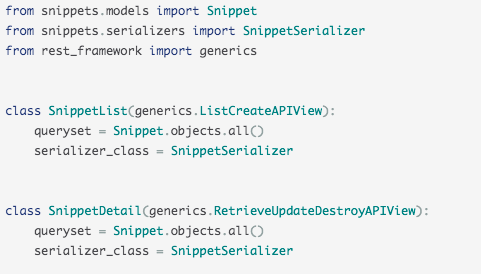
## 使用mixins

One of the big wins of using class-based views is that it allows us to easily compose reusable bits of behaviour.

The create/retrieve/update/delete operations that we've been using so far are going to be pretty similar for any model-backed API views we create. Those bits of common behaviour are implemented in REST framework's mixin classes.

## 范型views

REST framework provides a set of already mixed-in generic views that we can use



# 序列化器

<https://www.django-rest-framework.org/api-guide/serializers/#serializers>

序列化器可以把复杂的数据，例如querysets和模型实例或者非模型类转为python类型，然后渲染成json/xml或其他类型。也提供反序列化器。

也可以批量处理

# 测试