一、异常Exceptions

想象一下这么个场景，用户发送过来一个请求，DRF的视图在处理请求的过程中发生了异常。这时，你希望看到的是什么？服务器直接弹出异常，然后卡住或结束进程，等待管理员处理？客户端堵塞等待服务器响应数据？

No！No!这肯定是错误的。正确的做法是，DRF的视图抓住弹出的异常，并根据异常的类型，返回对应的响应，响应内容中要包含异常的信息。客户端仍然能够获得响应，不过内容不是期望的数据，而是发生错误的提示和信息。客户端可以根据提示，重新修改并发送请求。

DRF的视图是如何处理异常的

DRF内置了许多种异常的处理方法，并返回对应的响应。

比如下面几类：

- `APIException` 的子类
- Django的 `Http404` 异常.
- Django的 `PermissionDenied` 异常

在每种情况下，REST框架都将返回带有适当状态代码和内容类型的响应。响应的内容将包含错误原因的详细信息。

大多数错误响应将在响应正文中包含一个 `detail` 键。

例如，对于下面的请求：

```
DELETE http://api.example.com/foo/bar HTTP/1.1
Accept: application/json
```

有可能你会收到一个响应，并告诉你不允许删除请求的对象：

```
HTTP/1.1 405 Method Not Allowed
Content-Type: application/json
Content-Length: 42

{"detail": "Method 'DELETE' not allowed."}  # 注意detail这个键
```

验证错误的处理方式略有不同，这时会将字段名作为键包含在响应的内容中。如果验证错误不属于特定字段，则使用 `NON_FIELD_ERRORS_KEY` 作为键名，或者使用settings文件中 `NON_FIELD_ERRORS_KEY` 配置项设置的键名。

一个验证错误的例子如下：

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: 94

{"amount": ["A valid integer is required."], "description": ["This field may not be blank."]}  # 注意返回的内容主体
```

自定义异常处理的方法

你可以自己创建处理函数来实现自定义异常处理，该函数将API视图中引发的异常转换为响应对象。这允许您控制API使用的错误响应样式。

函数必须接受两个参数，第一个是要处理的异常，第二个是包含额外上下文（如当前正在处理的视图）的字典。正常情况下，异常处理函数应该返回一个 `Response` 对象，如果无法处理异常，则返回None。如果处理程序返回None，则将重新引发异常，Django将返回标准的HTTP `500 'server error'` 响应。

例如，你可能想要所有的错误响应都在响应主体中包含HTTP状态代码，例如：

```
HTTP/1.1 405 Method Not Allowed
Content-Type: application/json
Content-Length: 62

{"status_code": 405, "detail": "Method 'DELETE' not allowed."}
```

要实现上面的功能，你可以这么做：

```python
from rest_framework.views import exception_handler

def custom_exception_handler(exc, context):
    # Call REST framework's default exception handler first,
    # to get the standard error response.
    response = exception_handler(exc, context)

    # Now add the HTTP status code to the response.
    if response is not None:
        response.data['status_code'] = response.status_code

    return response
```

默认的处理函数不使用上下文参数，但如果异常处理程序需要更多的信息（如当前正在处理的视图），则该参数很有用，该视图可以通过 `context['view']` 变量获得。

自定义异常处理程序写完了，还必须使用 `EXCEPTION_HANDLER` 在settings中配置异常处理程序：

```python
REST_FRAMEWORK = {
    'EXCEPTION_HANDLER': 'my_project.my_app.utils.custom_exception_handler'
#注意这个路径
}
```

如果你不做上面的配置，那么DRF默认使用的异常处理程序是：

```python
REST_FRAMEWORK = {
    'EXCEPTION_HANDLER': 'rest_framework.views.exception_handler'
}
```

注意，异常处理程序只由引发异常生成的响应进行调用。它不会用于任何视图直接返回的响应，例如序列化类验证失败时由常规视图返回的 `HTTP_400_BAD_REQUEST` 响应。

二、API参考

除了处理异常的函数，DRF当然还为我们内置了一些常见的异常。

APIException

签名: `APIException()`

所有从 `APIView` 或者 `@api_view` 视图中弹出的异常的基类。

要自定义异常，需要先继承 `APIException` ，然后设置 `status_code` 、 `default_detail` 和 `default_code` 这三个类属性。

例如，如果您的API依赖于有时可能无法访问的第三方服务，那么您可能想要自定义一个 `"503 Service Unavailable"` 的HTTP响应代码的异常。你可以这样做：

```
from rest_framework.exceptions import APIException

class ServiceUnavailable(APIException):
    status_code = 503
    default_detail = 'Service temporarily unavailable, try again later.'
    default_code = 'service_unavailable'
```

有一些属性可用于查看API异常的状态：

The available attributes and methods are:

- `.detail` - 返回错误的文本格式描述
- `.get_codes()` - 返回错误的标识
- `.get_full_details()` - 同时返回错误标识和文本描述

大多数情况下，错误的信息都比较简单：

```
>>> print(exc.detail)
You do not have permission to perform this action.
>>> print(exc.get_codes())
permission_denied
>>> print(exc.get_full_details())
{'message':'You do not have permission to perform this
action.','code':'permission_denied'}
```

在验证错误的时候，错误内容可能是字典格式：

```
>>> print(exc.detail)
{"name":"This field is required.","age":"A valid integer is required."}
>>> print(exc.get_codes())
{"name":"required","age":"invalid"}
>>> print(exc.get_full_details())
{"name":{"message":"This field is required.","code":"required"},"age":
{"message":"A valid integer is required.","code":"invalid"}}
```

ParseError

签名: `ParseError(detail=None, code=None)`

解析请求发生错误!

当访问 `request.data` 时，如果请求包含了格式错误的数据，则引发该异常。

默认情况下，这会返回 `"400 Bad Request"` 类型的响应。

AuthenticationFailed

签名: `AuthenticationFailed(detail=None, code=None)`

认证失败!

当进入的请求包含不正确的认证身份时弹出该异常。

默认情况下返回 `"401 Unauthenticated"`，也有可能返回 `"403 Forbidden"`，这取决于你使用的认证模式。

NotAuthenticated

签名: `NotAuthenticated(detail=None, code=None)`

未认证。

当一个未认证的请求进行权限检查时失败了，弹出该异常。要和 `AuthenticationFailed` 区分开来。

默认情况下返回 `"401 Unauthenticated"`，也有可能返回 `"403 Forbidden"`，这取决于你使用的认证模式。

PermissionDenied

签名: `PermissionDenied(detail=None, code=None)`

无权访问!

当一个认证过的请求在权限检查时却未通过，弹出该异常。

默认情况下返回 `"403 Forbidden"` 。

NotFound

签名: `NotFound(detail=None, code=None)`

资源未找到!

当请求的资源对象不存在的时候，弹出该异常。 等同于Django标准的 `Http404` 异常。

默认情况下返回 `"404 Not Found"` 。

MethodNotAllowed

签名: `MethodNotAllowed(method, detail=None, code=None)`

不支持请求的方法!

视图不支持当前请求使用的HTTP方法时，弹出该异常。

默认情况下返回 `"405 Method Not Allowed"` 。

NotAcceptable

签名: `NotAcceptable(detail=None, code=None)`

客户端想要服务器返回的内容类型不支持!

当请求头部的 `Accept` 属性定义的类型，无法被视图渲染时，弹出该异常。

默认情况下返回 `"406 Not Acceptable"` 。

UnsupportedMediaType

签名: `UnsupportedMediaType(media_type, detail=None, code=None)`

不支持的媒体类型!

当后端所有的解析器都无法处理请求数据 `request.data` 中的内容时，弹出该异常。

默认情况下返回 `"415 Unsupported Media Type"`。

Throttled

签名: `Throttled(wait=None, detail=None, code=None)`

被限流了!

当请求被限流，不允许访问时，弹出该异常。

默认情况下返回 `"429 Too Many Requests"`。

ValidationError

签名: `ValidationError(detail, code=None)`

验证失败!

`ValidationError` 异常和其它的 `APIException` 异常有点区别:

- `detail` 参数是必须的，不可空缺。
- `detail` 可能是一个列表、字典或者嵌套的数据结构，用于容纳错误信息。
- By convention you should import the serializers module and use a fully qualified `ValidationError` style, in order to differentiate it from Django's built-in validation error. For example. `raise serializers.ValidationError('This field must be an integer value.')`
- 按照惯例，您应该导入序列化类并使用完全限定的 `ValidationError` 模块，以便将其与Django内置的验证错误区分开来。例如要这么写： `raise serializers.ValidationError('This field must be an integer value.')`，而不能这么写： `raise ValidationError('This field must be an integer value.')`

默认情况下，这个异常会返回 `"400 Bad Request"`。

三、通用的错误视图

DRF提供了两个错误视图，用于提供通用的500和400错误，并返回JSON类型的响应。（Django默认的错误视图，提供的是HTML类型的响应，这可能不适合那些只使用API接口的程序）

- `rest_framework.exceptions.server_error`

返回 `500` 响应，以 `application/json` 的内容类型。

作为 `handler500` 进行设置（参考liujiangblog.com的Django教程）：

```
handler500 = 'rest_framework.exceptions.server_error'
```

- `rest_framework.exceptions.bad_request`

返回 `400` 响应，以 `application/json` 的内容类型。

作为 `handler400` 进行设置：

```
handler400 = 'rest_framework.exceptions.bad_request'
```