# Django QueryDict对象

前述章节我们使用到了 request.GET 与 request.Post，在 HttpRequest 对象中，GET 与 POST 属性都是一个 QueryDict 的实例，而在 Django 中，QueryDict 被定义在 django.http.request 中可以使用如下方式引入。

from django.http.request import QueryDict
request.POST 和 request.GET 的 QueryDict 在一个正常的请求/响应循环中是不可变的。若要使其改变，需要使用 copy() 方法。

我们说过 QueryDict 是一个类字典，所以它实现了 Python 字典数据类型的所有标准方法。在 Python 的字典中，如果一个键对应多个值，那么，对应键只会保留最后一个值，但是若在 HTML 表单中，一个键对应多个值是很正常的事情，比如下拉复选框 <select> 就是这种情况。而 QueryDict 的存在就是为了解决这个问题的，它允许一个键对应多个值，且它在一次请求到响应的过程中是不可变的。

## 1. 初识QueryDict

QueryDict 继承自 MultiValueDict，而MultiValueDict 又继承自字典（dict），所以说它们都是字典的子类。接下来就让我们看一看 MultiValueDict 中定义的重要方法。

MultiValueDict 定义于 django/utils/datastructures.py 文件中，它也是 dict 的子类，用来处理多个值对应相同键的问题。同时，Django 在这个文件中还定义了一些其他的数据结构以适用于其他特定的场景。我们可以使用如下方式进行引入：

from django.utils.datastructures import MultiValueDict
在 MultiValueDict 类的注释中已经给出了常用的使用过程，如下所示：

```
>>> d = MultiValueDict({'name': ['Adrian', 'Simon'], 'position': ['Developer']})
>>> d['name']
'Simon'
>>> d.getlist('name')
['Adrian', 'Simon']
>>> d.getlist('doesnotexist')
[]
>>> d.getlist('doesnotexist', ['Adrian', 'Simon'])
['Adrian', 'Simon']
>>> d.get('lastname', 'nonexistent')
```

```
'nonexistent'
>>> d.setlist('lastname', ['Holovaty', 'Willison'])
```

对于一个字典来说，最重要的当然是根据键获取值，而 MultiVauleDict 提供了三个重要方法，分别是：__getitem__、get、getlist。下面我们对上述三个方法做简单的介绍。

### 1) __getitem __方法

__getitem__是 Python 中的魔术方法，在 Python 中具有双下画线的方法都称作魔术方法。当类对象中定义了这个方法时，类实例就可以通过[]运算符取值。在这里MultiValueDict 是通过重写其父类 dict 来实现的，源码如下：

```
def __getitem__(self, key):
    """
返回此键的最后一个数据值，如果是空列表，则返回[]；
如果没有找到，则引发键错误。
    """
    try:
        list_ = super().__getitem__(key)
    except KeyError:
        raise MultiValueDictKeyError(key)
    try:
        return list_[-1] #获取key对应值的最后一个
    except IndexError:
        return []
```

若果 key 不存在时，会抛出异常即 MultiValueDictKeyError 异常；若当一个 key 有多个值时，获取最后一个值。我们在视图中使用 request.GET['a'] 来获取 a 的值，就是使用了__getitem__方法。

### 2) get方法

我们也可以使用 request.GET.get('d',0) 获取 d 的值，但是它使用的是 MultValueDict 类中定义的 get 方法，其源码如下所示：

```
def get(self, key, default=None):
    """
返回传递的键的最后一个数据值。如果键不存在
或值为空列表，返回' default '。
    """
    try:
        val = self[key]
    except KeyError:
        return default
    if val == []:
        return default
    return val
```

分析源码可以得知，它首先尝试使用 val=self[key]获取 key 的值，如果获取不到则返回给定的默认值。但是你要注意这里捕获的异常是 KeyError，这个异常是 Python的标准异常类，而 MultiVauleDictKeyError 是它的一个子类。但是 get 和 __getitem__ 只能获取 key 的最后一个值，如果需要获取 key 的所有值就需要使用 getlist 方法。

### 3) getlist方法

这个方法同样也可以接受一个默认值，其内部实现了__getlist 方法，其源码如下所示：

```
def _getlist(self, key, default=None, force_list=False):
    """
返回键值的列表。用于在内部操作值列表。
如果force_list为真，返回值的新副本。
    """
    try:
        values = super().__getitem__(key)
    except KeyError:
        if default is None:
            return []
        return default
    else:
        if force_list:
            values = list(values) if values is not None else None
        return values
```

首选对源码进行初步分析，_getlist调用父类的__getitem__方法获取 key 对应的 values，如果不存在 key，则考虑使用 default。最后，force_list 的作用是将 values 复制一份，并保存在 values 变量中。

# 2. QueryDict 常用方法

前面介绍了 QueryDict 继承自 MultiValueDict，那么它有实现了方法呢？首先看一下它的构造函数：

QueryDict.__init__(query_string=None, mutable=False, encoding=None)
其中 mutable=False 表示对象不可变，反之则可变。如果把 URL 中的查询字符串直接传递给 QueryDict，会得到如下结果：

```
In [1]: from django.http.request import QueryDict
In [2]: QueryDict("a=1&b=2&c=3")
Out[2]: <QueryDict: {'a': ['1'], 'b': ['2'], 'c': ['3']}>
```

这就是通过 QueryDict 构造函数实现，如果感兴趣的小伙伴可以研究一下它的源码，就可以知道它的代码逻辑了，在这里不做分析。下面我们介绍一下它的经常用到的方法。

### 1) QueryDict.fromkeys()

这个方法的作用是循环可迭代对象中的每个元素作为键值，并赋予同样的值，它的方法格式如下所示：

QueryDict.fromkeys(iterable, value='', mutable=False, encoding=None)
实例如下：

```
1. In [4]: QueryDict.fromkeys(['a','b','c'],value="1")
2. Out[4]: <QueryDict: {'a': ['1'], 'b': ['1'], 'c': ['1']}>
```

### 2) QueryDict.pop(key)

返回给定键的值的列表，并从QueryDict中移除该键。 如果键不存在，将引发KeyError。实例如下：

```
1. In [11]: QueryDict('a=1&b=2&c=3',mutable=True).pop("a")
2. Out[11]: ['1']
```

### 3) QueryDict.popitem()

删除 QueryDict 任意一个键，并返回二值元组，包含键和键的所有值的列表。默认移除最后一个键值对。假如是一个空的字典上调用时将引发 KeyError。实例如下：

```
1. In [12]: q = QueryDict('a=1&a=2&c=3', mutable=True)
2. In [13]: q.popitem()
3. Out[13]: ('c', ['3'])
```

### 4) QueryDict.items()

类似 dict.items()，如果有重复项目，返回最近的一个，而不是都返回，它的返回值是一个列表包含的二元组。如下所示：

```
1. In [26]: q = QueryDict('a=1&a=2&a=3')
2. In [27]: q.items()
3. Out[27]: <generator object MultiValueDict.items at 0x01594AF0>
4. In [28]: list(q.items())
5. Out[28]: [('a', '3')]
```

### 5) QueryDict.values()

类似 dict.values()，它的返回值是一个生成器对象，而且只返回最近的值，实例如下：

```
In [23]: q = QueryDict('a=1&a=2&a=3')
In [24]: q.values()
Out[24]: <generator object MultiValueDict.values at 0x09D578F0>
In [25]: list(q.values())
Out[25]: ['3']
```

### 6）QueryDict.urlencode(safe=None)

url 的编码格式返回数据字符串。如下所示：

```
In [29]: q = QueryDict('a=2&b=3&b=5')
In [30]: q.urlencode()
Out[30]: 'a=2&b=3&b=5'
```

提示：使用 safe 参数传递不需要编码的字符，如 q.urlencode(safe='?')。

### 7) QueryDict.update(dict)

用新的 QueryDict 或字典更新当前 QueryDict。类似 dict.update()，但是追加内容，而不是更新并替换它们。如下示例：

```
In [31]: q = QueryDict('a=1', mutable=True)
In [32]: q.update({'a': '2'})
In [33]:  q.getlist('a')
Out[33]: ['1', '2']
```

本节对 QueryDict 做了详细的讲解，分别从继承关系上进行了分析，大家看完本节，应该能够熟练掌握它的使用方法，而且对于类字典对象也不再感到陌生。当然还有一部分方法没有介绍，有兴趣的伙伴可以参阅官方文档《QueryDict Object》进行学习。