原作者: Jason Kirtland
日期: January 13th, 2009

原文链接(PPT): http://discorporate.us/jek/talks/defaultdict/
翻译作者：kodango <dangoakachan@foxmail.com>
翻译时间：March 17th, 2012

今天看到一篇讲defaultdict的PPT，同时第一次见到 `__missing__()` 这个方法，好奇之下，仔细阅读了这篇PPT。看完之后随手做笔记，分享给有需要的人。准确地说，这篇文章不是纯粹的翻译，因为原文本身只是一份PPT。文章的大多数文字内容，都是本人的阅读心得。

默认值可以很方便

众所周知，在Python中如果访问字典中不存在的键，会引发KeyError异常（JavaScript中如果对象中不存在某个属性，则返回undefined）。但是有时候，字典中的每个键都存在默认值是非常方便的。例如下面的例子：


```
strings = ('puppy', 'kitten', 'puppy', 'puppy',
           'weasel', 'puppy', 'kitten', 'puppy')
counts = {}
```

```
    counts[kw] += 1
```

该例子统计strings中某个单词出现的次数，并在counts字典中作记录。单词每出现一次，在counts相对应的键所存的值数字加1。但是事实上，运行这段代码会抛出KeyError异常，出现的时机是每个单词第一次统计的时候，因为Python的dict中不存在默认值的说法，可以在Python命令行中验证：

```
>>> counts = dict()
>>> counts
{}
>>> counts['puppy'] += 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'puppy'
```

使用判断语句检查

既然如此，首先可能想到的方法是在单词第一次统计的时候，在counts中相应的键存下默认值1。这需要在处理的时候添加一个判断语句：

```
strings = ('puppy', 'kitten', 'puppy', 'puppy',
           'weasel', 'puppy', 'kitten', 'puppy')
counts = {}

for kw in strings:
    if kw not in counts:
        counts[kw] = 1
    else:
        counts[kw] += 1

# counts:
# {'puppy': 5, 'weasel': 1, 'kitten': 2}
```

也可以通过 `dict.setdefault()` 方法来设置默认值：

```
strings = ('puppy', 'kitten', 'puppy', 'puppy',
           'weasel', 'puppy', 'kitten', 'puppy')
counts = {}

for kw in strings:
    counts.setdefault(kw, 0)
    counts[kw] += 1 # 原PPT中这里有一个笔误
```

`dict.setdefault()` 方法接收两个参数，第一个参数是健的名称，第二个参数是默认值。假如字典中不存在给定的键，则返回参数中提供的默认值；反之，则返回字典中保存的值。利用 `dict.setdefault()` 方法的返回值可以重写for循环中的代码，使其更加简洁：

```
strings = ('puppy', 'kitten', 'puppy', 'puppy',
           'weasel', 'puppy', 'kitten', 'puppy')
counts = {}

for kw in strings:
    counts[kw] = counts.setdefault(kw, 0) + 1
```

使用 `collections.defaultdict` 类

以上的方法虽然在一定程度上解决了dict中不存在默认值的问题，但是这时候我们会想，有没有一种字典它本身提供了默认值的功能呢？答案是肯定的，那就是 `collections.defaultdict` 。

defaultdict类就好像是一个dict，但是它是使用一个类型来初始化的：

```
>>> from collections import defaultdict
>>> dd = defaultdict(list)
>>> dd
defaultdict(<type 'list'>, {})
```

值：

```
>>> dd['foo']
[]
>>> dd
defaultdict(<type 'list'>, {'foo': []})
>>> dd['bar'].append('quux')
>>> dd
defaultdict(<type 'list'>, {'foo': [], 'bar': ['quux']})
```

需要注意的是，这种形式的默认值只有在通过 `dict[key]` 或者 `dict.__getitem__(key)` 访问的时候才有效，这其中的原因在下文会介绍。

```
>>> from collections import defaultdict
>>> dd = defaultdict(list)
>>> 'something' in dd
False
>>> dd.pop('something')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'pop(): dictionary is empty'
>>> dd.get('something')
>>> dd['something']
[]
```

该类除了接受类型名称作为初始化函数的参数之外，还可以使用任何不带参数的可调用函数，到时该函数的返回结果作为默认值，这样使得默认值的取值更加灵活。下面用一个例子来说明，如何用自定义的不带参数的函数zero()作为初始化函数的参数：

```
>>> from collections import defaultdict
>>> def zero():
...     return 0
...
```

```
defaultdict(<function zero at 0xb7ed2684>, {})
>>> dd['foo']
0
>>> dd
defaultdict(<function zero at 0xb7ed2684>, {'foo': 0})
```

利用 `collections.defaultdict` 来解决最初的单词统计问题，代码如下：

```
from collections import defaultdict

strings = ('puppy', 'kitten', 'puppy', 'puppy',
           'weasel', 'puppy', 'kitten', 'puppy')
counts = defaultdict(lambda: 0)  # 使用lambda来定义简单的函数

for s in strings:
    counts[s] += 1
```

defaultdict 类是如何实现的

通过上面的内容，想必大家已经了解了defaultdict类的用法，那么在defaultdict类中又是如何来实现默认值的功能呢？这其中的关键是使用了看 `__missing__()` 这个方法：

```
>>> from collections import defaultdict
>>> print defaultdict.__missing__.__doc__
__missing__(key) # Called by __getitem__ for missing key; pseudo-code:
  if self.default_factory is None: raise KeyError(key)
  self[key] = value = self.default_factory()
  return value
```

通过查看 `__missing__()` 方法的docstring，可以看出当使用 `__getitem__()` 方法访问一个不存在的键时(dict[key]这种形式实际上是 `__getitem__()` 方法的简化形式)，会调用 `__missing__()` 方法获取默认值，并将该键添加到字典中去。

文档中介绍，从2.5版本开始，如果派生自dict的子类定义了 `__missing__()` 方法，当访问不存在的键时，dict[key]会调用 `__missing__()` 方法取得默认值。

从中可以看出，虽然dict支持 `__missing__()` 方法，但是在dict本身是不存在这个方法的，而是需要在派生的子类中自行实现这个方法。可以简单的验证这一点：

```
>>> print dict.__missing__.__doc__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: type object 'dict' has no attribute '__missing__'
```

同时，我们可以进一步的做实验，定义一个子类Missing并实现 `__missing__()` 方法:

```
>>> class Missing(dict):
...     def __missing__(self, key):
...         return 'missing'
...
>>> d = Missing()
>>> d
{}
>>> d['foo']
'missing'
>>> d
{}
```

返回结果反映了 `__missing__()` 方法确实发挥了作用。在此基础上，我们稍许修改 `__missing__()` 方法,使得该子类同defautldict类一样为不存在的键设置一个默认值：

```
>>> class Defaulting(dict):
...     def __missing__(self, key):
...         self[key] = 'default'
...         return 'default'
...
```

```
{}
>>> d['foo']
'default'
>>> d
{'foo': 'default'}
```

在旧版本的 Python 中实现 defaultdict 的功能

defaultdict类是从2.5版本之后才添加的，在一些旧版本中并不支持它，因此为旧版本实现一个兼容的defaultdict类是必要的。这其实很简单，虽然性能可能未必如2.5版本中自带的defautldict类好，但在功能上是一样的。

首先，`__getitem__()`方法需要在访问键失败时，调用`__missing__()`方法：

```
class defaultdict(dict):
    def __getitem__(self, key):
        try:
            return dict.__getitem__(self, key)
        except KeyError:
            return self.__missing__(key)
```

其次，需要实现`__missing__()`方法用来设置默认值：

```
class defaultdict(dict):
    def __getitem__(self, key):
        try:
            return dict.__getitem__(self, key)
        except KeyError:
            return self.__missing__(key)

    def __missing__(self, key):
        self[key] = value = self.default_factory()
        return value
```

```
class defaultdict(dict):
    def __init__(self, default_factory=None, *a, **kw):
        dict.__init__(self, *a, **kw)
        self.default_factory = default_factory

    def __getitem__(self, key):
        try:
            return dict.__getitem__(self, key)
        except KeyError:
            return self.__missing__(key)

    def __missing__(self, key):
        self[key] = value = self.default_factory()
        return value
```

最后，综合以上内容，通过以下方式完成兼容新旧Python版本的代码：

```
try:
    from collections import defaultdict
except ImportError:
    class defaultdict(dict):
      def __init__(self, default_factory=None, *a, **kw):
          dict.__init__(self, *a, **kw)
          self.default_factory = default_factory

      def __getitem__(self, key):
          try:
              return dict.__getitem__(self, key)
          except KeyError:
              return self.__missing__(key)

      def __missing__(self, key):
```

更加完整的版本参见：http://code.activestate.com/recipes/523034/

与 defaultdict Python 字典 翻译 **相关的文章**

MySQL Python 教程（一）	26,300 人浏览过
MySQL Python 教程（二）	9,986 人浏览过
PySqlite 学习笔记	7,541 人浏览过
理解 Python 中的 *args 和 **kwargs	51,746 人浏览过
Shell 编码风格	9,051 人浏览过
Bash One-Liners Explained 译文（三）	8,060 人浏览过

- 理解 Python 中的 *args 和 **kwargs
- ArchLinux 安装笔记

团子的小窝

网站

有人回复时邮件通知我

提交回复