字典类

Python中的全能战士

《代码之美》第18章　　江涌

字典

在字典里，每个数据元素都有唯一的关键字与其对应，所有数据元素和关键字对构成映射表。

{(key1,obj1)，(key2,obj2)……(keyn,objn)}

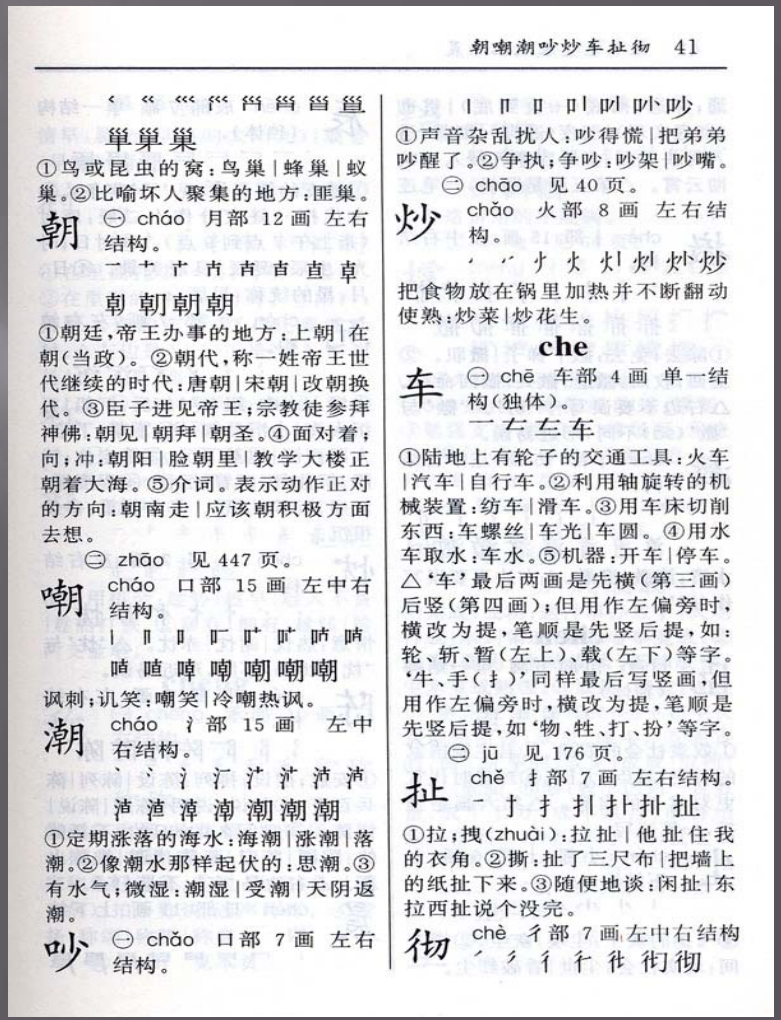
基本操作：

- 添加新的关键字—数据对
- 根据关键字访问对应数据元素
- 删除存在的关键字---数据对
- 遍历所有关键字、遍历所有数据元素或遍历所有关键字—数据对

字典

字典只是将互异的数据元素扩充成了互异的关键字—数据对，那么这种扩充有什么用呢？

比如在真实字典中：
〈车，“车的解释”〉
就是一个关键字—数据元素对。
在查找“车”的解释时我们先查询“车”字所对应的页码：
P(“车”)=41
这样就能很快的定位我们要的数据元素(“车”的解释)。
P------哈希函数



朝嘲潮吵炒车扯彻 41

单巢巢
①鸟或昆虫的窝：鸟巢|蜂巢|蚁巢。②比喻坏人聚集的地方：匪巢。

朝 ㊀ cháo 月部 12 画 左右结构。
朝朝朝朝
①朝廷，帝王办事的地方：上朝|在朝(当政)。②朝代，称一姓帝王世代继续的时代：唐朝|宋朝|改朝换代。③臣子进见帝王；宗教徒参拜神佛：朝见|朝拜|朝圣。④面对着；向；冲：朝阳|脸朝里|教学大楼正朝着大海。⑤介词。表示动作正对的方向：朝南走|应该朝积极方面去想。
㊁ zhāo 见 447 页。

嘲 cháo 口部 15 画 左中右结构。
嘲嘲嘲嘲嘲嘲嘲嘲
讽刺；讥笑：嘲笑|冷嘲热讽。

潮 cháo 氵部 15 画 左中右结构。
潮潮潮潮潮潮潮潮
①定期涨落的海水：海潮|涨潮|落潮。②像潮水那样起伏的：思潮。③有水气；微湿：潮湿|受潮|天阴返潮。

吵 ㊀ chǎo 口部 7 画 左右结构。
①声音杂乱扰人：吵得慌|把弟弟吵醒了。②争执；争吵：吵架|吵嘴。
㊁ chāo 见 40 页。

炒 chǎo 火部 8 画 左右结构。
把食物放在锅里加热并不断翻动使熟：炒菜|炒花生。

che

车 ㊀ chē 车部 4 画 单一结构(独体)。
①陆地上有轮子的交通工具：火车|汽车|自行车。②利用轴旋转的机械装置：纺车|滑车。③用车床切削东西：车螺丝|车光|车圆。④用水车取水：车水。⑤机器：开车|停车。
△‘车’最后两画是先横(第三画)后竖(第四画)；但用作左偏旁时，横改为提，笔顺是先竖后提，如：轮、斩、暂(左上)、载(左下)等字。‘牛、手(扌)’同样最后写竖画，但用作左偏旁时，横改为提，笔顺是先竖后提，如‘物、牲、打、扮’等字。
㊁ jū 见 176 页。

扯 chě 扌部 7 画 左右结构。
①拉；拽(zhuài)：拉扯|他扯住我的衣角。②撕：扯了三尺布|把墙上的纸扯下来。③随便地谈：闲扯|东拉西扯说个没完。

彻 chè 彳部 7 画 左中右结构

Python

历史

Python的创始人Guido van Rossum为了打发圣诞节的无趣开发的一个脚本解释程序，做为ABC语言的一种继承。(Monty Python飞行马戏团)

特点

简单------Python是一种代表简单主义思想的语言。阅读一个良好的Python程序就感觉像是在读英语一样，尽管这个英语的要求非常严格！Python的这种伪代码本质是它最大的优点之一。它使你能够专注于解决问题而不是去搞明白语言本身。

Python

免费、开源

面向对象------Python既支持面向过程的编程也支持面向对象的编程。

可扩展性------如果你需要你的一段关键代码运行得更快或者希望某些算法不公开，你可以把你的部分程序用C或C++编写，然后在你的Python程序中使用。

丰富的库------Python标准库确实很庞大。它可以帮助你处理各种工作，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、 FTP、电子邮件、XML、XML-RPC、HTML、WAV文件、密码系统、GUI（图形用户界面）、Tk和其他与系统有关的操作。记住，只要安装了 Python，所有这些功能都是可用的。这被称作Python的“功能齐全”理念。

Python字典类

```
>>> d = {1: 'January', 2: 'February',...   'jan': 1, 'feb': 2, 'mar': 3}
{'jan': 1, 1: 'January', 2: 'February', 'mar': 3, 'feb': 2}
>>> d['jan'], d[1]
(1, 'January')
>>> d[12]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 12
>>> del d[2]
>>> for k, v in d.items(  ): print k,v   # Looping over all pairs.
jan 1
1 January
mar 3
feb 2
...
```

1.关键字和数据元素可以是各种数据类型；
2.关键字不是有序的。

Python字典类

Python中具有核心地位：字典是其他语言功能的基石

- 保存类和类实例属性

```
>>> obj = MyClass( )          # Create a class instance
>>> obj.name = 'object'       # Add a .name attribute
>>> obj.id = 14               # Add a .id attribute
>>> obj._ _dict_ _            # Retrieve the underlying dictionary
{'name': 'object', 'id': 14}
>>> obj._ _dict_ _['id'] = 12   # Store a new value in the dictionary
>>> obj.id                    # Attribute is changed accordingly
12
```

- 存储模块内容
- 函数调用前后，建立并释放一个字典实例

……

Python字典类

Python程序中会活跃着很多字典，这就要求字典的建立与销毁必须非常迅速。能够快速定位关键字是尤其重要的，通常用哈希表来实现这样的类型。

哈希函数：将关键字映射到其对应数据元素的存储位置（或中间结果）。

按这个函数建立的表就是哈希表。

因此在字典类的内部实现每个数据元素应该涉及关键字、数据元素与关键字对应的哈希值。

字典类的内部实现

结构PyDictObject实现字典类(在CPython中)

```
typedef  struct  _dictobject  PyDictObject;
struct _dictobject {
    PyObject_HEAD
    Py_ssize_t ma_fill;          /* # Active + # Dummy*/
    Py_ssize_t ma_used;          /* # Active */

/* The table contains ma_mask + 1 slots, and that's a power of 2.
 * We store the mask instead of the size because the mask is more
 * frequently needed.
 */
    Py_ssize_t ma_mask;
```

字典类的内部实现

```
/* ma_table points to ma_smalltable for small tables, else to
 * additional malloc'ed memory.  ma_table is never NULL!
 *  This rule  saves repeated runtime null-tests in the
 * workhorse getitem and  setitem calls.
 */
    PyDictEntry *ma_table;
    PyDictEntry *(*ma_lookup)(PyDictObject *mp, PyObject
    *key, long hash);
    PyDictEntry ma_smalltable[PyDict_MINSIZE];
};
```

字典类的内部实现

下面的字典保存了从”aa”,’’bb’’…”mm”到整数1-13的一一映射。

int ma_fill　　13　　关键字存储单元与哑存单元总数

int ma_used　　13　　有关键字的存储单元数量

int ma_mask　　31　　对应哈希表大小的比特掩码

PyDictEntry ma_table[]:

[0]: aa, 1　　hash(aa) == -1549758592, -1549758592 & 31 = 0

[1]: ii, 9　　hash(ii) == -1500461680, -1500461680 & 31 = 16

[2]: null, null

[3]: null, null

[4]: null, null

[5]: jj, 10 hash(jj) == 653184214, 653184214 & 31 = 22
[6]: bb, 2 hash(bb) == 603887302, 603887302 & 31 = 6
[7]: null, null
[8]: cc, 3 hash(cc) == -1537434360, -1537434360 & 31 = 8
[9]: null, null
[10]: dd, 4 hash(dd) == 616211530, 616211530 & 31 = 10
[11]: null, null
[12]: null, null
[13]: null, null
[14]: null, null
[15]: null, null
[16]: gg, 7 hash(gg) == -1512785904, -1512785904 & 31 = 16
[17]: ee, 5 hash(ee) == -1525110136, -1525110136 & 31 = 8

[18]: hh, 8 hash(hh) == 640859986, 640859986 & 31 = 18
[19]: null, null
[20]: null, null
[21]: kk, 11 hash(kk) == -1488137240, -1488137240 & 31 = 8
[22]: ff, 6 hash(ff) == 628535766, 628535766 & 31 = 22
[23]: null, null
[24]: null, null
[25]: null, null
[26]: null, null
[27]: null, null
[28]: null, null
[29]: ll, 12 hash(ll) == 665508394, 665508394 & 31 = 10
[30]: mm, 13 hash(mm) == -1475813016, -1475813016 & 31 = 8
[31]: null, null

字典类内部实现

PyDictObject 包含一个8存储单元的哈希表，不多于5个元素的小字典(参数)可以直接放入，免去额外开销。

大多数Python程序涉及的内部字典只用字符串作为关键字，设计一种特别的字典，其关键字只能是字符串，可能更有用！

在Python的java版本中就有一个专用字符串的字典类Org.python.org.PyStringMap

字典类内部实现

在Cpython中不同：

```
struct PyDictObject {
    ...
    PyDictEntry *(*ma_lookup)(PyDictObject *mp,
                              PyObject *key, long hash);
};
```

刚开始ma_lookup指向函数lookdict_string，这个函数假定字典包含的关键字与待检索的关键字都是字符串类型。如果字典里有非字符串的关键字，ma_lookup就会指向更加通用的查表函数。

哈希表处理

- 冲突处理

拉链法：

每个存储单元都保存一个链表，链表上所有元素的哈希值对应当前存储单元。

缺点：需要申请内存空间，分配内存比较慢；降低缓存局部性。

开放地址法：H_i=(H(key)+d_i)mod m　i=1,2,…,k(k<=m-1)

H为哈希函数；m为哈希表表长，d_i增量序列，有下列3种取法：

1.线性探测d_i=1,2,3,...,m-1；

2.二次探测d_i=1^2,-1^2,2^2,-2^2,…,+-k^2(k<=m/2)；

3.伪随机数探测d_i=伪随机数序列。

哈希表处理

Python中的模式：

```
    /* 起始存储单元 */
slot = hash;

    /* 初始增量*/
perturb = hash;
while (<存储单元非空> && <单元内的值不等于关键字值>) {
    slot = (5*slot) + 1 + perturb;
    perturb >>= 5;
}
```

哈希表处理

- “保证哈希表最多只有2/3是满的。”
- “不超过5万个关键字的中小哈希表，尺寸应该是ma_used*4，超过5万的字典，尺寸大小为ma_used*2。”
- Python中有一个free_dicts数组保存不再使用的字典。当字典被删除时，会直接加到数组free_dicts中，如果数组已经满了，就直接释放字典对象。
  减少了字典的频繁创建与销毁，提高效率。

迭代与动态变化

字典的方法：

一次返回相应列表

Keys()、values()、items()；

逐个遍历

Iterkeys()、itervalues()、iteritems()

迭代器会记住字典中所有元素的个数，所以在遍历迭代中不允许改变字典大小(增加或删除存储单元)；

允许修改数据元素：

```
for k, v in d.iteritems( ):
                d[k] = d[k] + 1
```

总结

字典类在Python内部是举足轻重的，是位地道的全能战士而且它的实现总的来说还是干净利落的；

哈希表的处理直接影响到冲突率和运行效率。

官方主页：*http://www.python.org/*
参考教材：*http://docs.python.org/*