# 迭代器与生成器

概念介绍:

迭代器可以看作一个特殊的对象, 每次调用该对象时会返回自身的下一个元素, 从实现上来看, 一个可迭代的对象必须是定义了\_\_iter\_\_()方法的对象, 而一个迭代器必须是定义了\_\_iter\_\_()方法和\_\_next\_\_()方法的对象.

生成器是能够返回一个迭代器的函数, 其最大的作用是将输入对象返回为一个迭代器.

可迭代对象:

可以返回一个迭代器的对象(元素之类)都可以称之为可迭代对象.

可迭代对象和容器一样是一种通俗的叫法, 并不是指某种具体的数据类型, list是可迭代对象, dict也是.

可迭代对象是实现了\_\_iter\_\_()方法的对象, 通过调用iter()方法可以获得一个迭代器..

迭代器:

迭代器(iterator)是一种对象, 它能够用来遍历标准模板库容器中的部分或全部元素, 每个迭代器对象代表容器中的确定的地址. 迭代器修改了常规指针的接口, 所谓迭代器是一种概念上的抽象: 那些行为上像迭代器的东西都可以叫做迭代器. 然而迭代器有很多不同的能力, 它可以把抽象容器和通用算法有机的统一起来.

迭代器是个所谓的复杂的指针, 具有遍历复杂数据结构的能力. 其下层运行机制取决于其所遍历的数据结构. 因此, 每一种容器型别都必须提供自己的迭代器. 事实上每一种容器都将其迭代器以嵌套的方式定义于内部. 因此各种迭代器的接口相同, 型号却不同. 这直接导出了泛型程序设计的概念: 所有操作行为都使用相同接口, 虽然它们的型别不同. 迭代器有一种具体的迭代器类型, 比如list\_iterator, set\_iterator.

迭代器内部有一个状态, 该状态用于记录当前迭代所在的位置, 以方便下次迭代时获取正确的元素.

迭代器是一个带状态的对象, 它能在你调用next()方法时返回容器中的下一个值, 任何实现了\_\_iter\_\_()和\_\_next\_\_()方法的对象都是迭代器, \_\_iter\_\_()返回迭代器自身, \_\_next\_\_()返回容器中的下一个值, 如果容器中没有更多元素了, 则抛出StopIteration异常.

迭代器与列表的区别在于, 构建迭代器的时候, 不像列表把所有元素一次性加载到内存, 而是以一种延迟计算(lazy evaluation)方式返回元素, 这正是它的优点, 它并没有把所有元素装载到内存中, 而是等到调用next()方法的时候才返回该元素(按需调用call by need的方式, 本质上for循环就是不断地调用迭代器的next()方法).

for … in … 的迭代实际是将可迭代对象转换成迭代器, 再重复调用next()方法实现的.

生成器:

生成器是一次生成一个值的特殊类型函数. 可以将其视为可恢复函数. 调用该函数将返回一个可用于生成连续x值的生成器.

简单的说就是在函数的执行过程中, yield语句会把你需要的值返回给调用生成器的地方, 然后退出函数, 下一次调用生成器函数的时候又从上次中断的地方开始执行, 而生成器内的所有变量参数都会被保留下来供下一次使用.

使用yield的函数叫生成器函数. 函数被调用时会返回一个生成器对象. 生成器其实是一种特殊的迭代器, 不过这种迭代器更加优雅, 它不需要像普通迭代器一样实现\_\_iter\_\_()和\_\_next\_\_()方法了, 只需要一个yield关键字. 生成器一定是迭代器(反之不成立), 因此任何生成器也是一种懒加载的模式生成值.

yield是生成器实现\_\_next\_\_()方法的关键。它作为生成器执行的暂停恢复点，可以对yield表达式进行赋值，也可以将yield表达式的值返回。

生成器是生成元素的, 迭代器是访问集合元素的一种方式

迭代器输出生成器的内容

迭代器对象表示的是一个数据流, 可以把它看做一个有序序列, 但我们不能提前知道序列的长度, 只有通过next()函数实现需要计算的下一个数据.