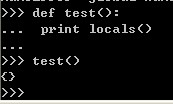
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\{H$]5UNO{S70ZEF2HL7[CF0.png交谈中请勿轻信汇款、中奖信息、陌生电话，勿使用外挂软件。  
  
Q.yuhen(1620443)  14:38:05  
x = a  和  x.member = a 是不同的  
前者是赋值，后者是改变对象的属性。  
要注意区别。  
如果明确需要修改 globals 里的 x，那么需要显式声明一下。  
Q.yuhen(1620443)  14:39:05  
glboas x;   x = "ssss"  
global x; x = "sss"  
有了 global 的声明，编译器就会使用 LOAD\_GLOBAL 代替 LOAD\_FAST 指令  
表示修改全局名字空间  
Q.yuhen(1620443)  14:40:06  
python 3 还提供了一个 nonlocal 用来修改外层函数的名字。  
不过 python 2 并没有  
想想看，我们随时可以在任何函数里面创建 module globals 全局名字。  
这是静态语言做不到的。  
还有一个细节  
Q.yuhen(1620443)  14:41:25  
每个函数只有一个 locals，这和静态语言通过 {} 区分更小的 code scope 不同。  
在 Python 函数中，你任何地方创建的名字都只会出现在 locals 当中。不可能存在同名的两个。  
C、golang 可以，Python 不可以。  
Q.yuhen(1620443)  14:42:31  
也因为名字空间是字典的缘故，所以 Python 不存在 overload 一说。  
字典里的名字总是唯一的。  
----- 休息一下，待会将另一套规则 -----  
goal<jefurry@qq.com>  14:44:07  
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\}$$SH8F5BOR8W96(N@`FVY4.gif鼓掌  
:“从今以后(154913283)  14:45:35  
D:\Program Files\Tencent\QQ\Users\443851281\Image\_0HJ2@ZO(_{8KZ$LFVV58ZQ.gif  
烈风<lhyhr@vip.qq.com>  14:44:58  
D:\Program Files\Tencent\QQ\Users\443851281\Image\BHNRI8WIQ[R6UMZ`U%5WPCP.gif  
121(443851281)  14:45:39  
以前只知道堆代码 不理解原理现在听来茅塞顿开呀  
奋斗人生(18180850)  14:45:44  
 D:\Program Files\Tencent\QQ\Users\443851281\Image\O]U8LT_6~6%I[DILQOAXX2N.jpg  
         願.(514493671)  14:46:04  
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\J]~G_ZF~_}14BZXB%N)Q51B.gif茅塞顿开  
奋斗人生(18180850)  14:46:11  
D:\Program Files\Tencent\QQ\Users\443851281\Image\UR[R}[E0X8I3OUC678057A5.jpg  
老大，locals中，也有\_\_builtins\_\_  
Q.yuhen(1620443)  14:46:55  
直接在 module 里面运行 locals 自然和 globals 是一回事了。  
121(443851281)  14:47:15  
在函数里运行是不一样的   
可以试试  
Q.yuhen(1620443)  14:47:27  
locals 返回当前的名字空间。  
当前！！！！！  
奋斗人生(18180850)  14:47:40  
好  
Q.yuhen(1620443)  14:48:33  
我去冲杯咖啡，5 分钟后继续。  
奋斗人生(18180850)  14:48:58  
  
当前！  
Q.yuhen(1620443)  14:51:33  
我们继续  
我们昨天说过，函数和方法的区别。  
方法、字段、属性这些总是和一个对象相绑定的。  
总是以组合体的方式出现  
Q.yuhen(1620443)  14:52:44  
如果我们只是访问一个对象实例或者一个类型对象，那么自然还是按照 LEGB 规则查找。  
但如果访问一个对象成员的时候，就不同了。  
因为这些成员不是直接保存在 globals/locals 里面。  
成员被实例或类型所引用。  
这涉及到的名字空间包括  
Q.yuhen(1620443)  14:53:49  
类型的名字空间 class.\_\_dict\_\_  
实例的名字空间 instance.\_\_dict\_\_  
什么叫实例，就是用类型对象通过构造函数创建的对象。  
那么这两个名字空间有什么区别呢？  
对象成员都有哪些？  
字段  
属性  
方法  
Q.yuhen(1620443)  14:54:51  
方法又可以分为实例方法和静态方法  
字段也有实例字段和静态字段一说。  
当编译器遇到 class 指令的时候。  
Q.yuhen(1620443)  14:56:04  
他创建对应的类型对象，类型对象 class.\_\_dict\_\_ 名字空间里包含了所有的实例方法、静态方法、属性和静态字段。  
为什么呢？因为这些都是所有实例所共享的。  
那么实例对象创建后，它的 instance.\_\_dict\_\_ 里面有什么呢？  
所有的实例字段。  
为什么是所有的？  
因为包括所有基类的实例字段。  
Q.yuhen(1620443)  14:57:07  
实例字段为什么要独立出来？  
因为每个实例的状态是不同的。  
字段就是对象实例的状态。  
任何方法默认都没有状态，它们通过 this、self 这个隐式的实例指针来访问他的实例字段。  
Q.yuhen(1620443)  14:58:15  
所以说，实例就是一堆实例字段的综合体。  
实例头部有类型对象的指针，实例字段 + 类型指针就构成了一个完整的对象。  
当我们要访问对象成员的时候，必须给出实例。  
注意类型对象自己也是实例，只是比较特殊而已。  
Q.yuhen(1620443)  14:59:29  
那么查找顺序是什么？  
如果用  instance.x 来访问，那么首先查找 instance.\_\_dict\_\_，如果没有就查找 class.\_\_dict\_\_。  
如果用 class.x 访问，那么就查找 class.\_\_dict\_\_。  
Q.yuhen(1620443)  15:00:40  
如果 class.\_\_dict\_\_ 也没有，那么就继续查找上一级的基类 base.\_\_dict\_\_。  
总结一下，对象成员的访问顺序。  
instance.\_\_dict\_\_ -> class.\_\_dict\_\_ -> bases\_.\_dict\_\_  
所以我们可以用 self 直接访问静态成员  
Q.yuhen(1620443)  15:01:45  
想象一下，因为实例方法也保存在 class.\_\_dict\_\_ 中。  
如果我们在 instance.\_\_dict\_\_ 人为插入一个同名的字段，也造成什么？  
显然，解释器优先命中这个伪造者。  
Q.yuhen(1620443)  15:03:00  
如果我们在 class.\_\_dict\_\_ 插入一个和  bases.\_\_dict\_\_ 中同名的方法，那么优先命中 class 里面的成员。  
这就是所谓的 override。  
因为解释器只查名字，而不关心具体的签名。  
所以 Python override 的方法参数完全可以不同。  
了解了这个查找规则，那么继承这块的东西就很清晰了。  
还有一个问题就是多重继承。  
Q.yuhen(1620443)  15:04:10  
虽然我们日常开发要避免多重继承  
但还是会用到，比如做 mixin 等处理。  
假设 class X(A, B)  
那么 X 的基类就有两个，这就有个问题。假设我们要访问的成员 test，A 和 B 都有。  
Q.yuhen(1620443)  15:05:13  
那么要选谁呢？  
这就要看一个称为 mro 的列表。  
解释器会将类型顺序放在这个列表里面。  
\_\_mro\_\_(X, A, B, object)  
goal<jefurry@qq.com>  15:31:08  
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\}$$SH8F5BOR8W96(N@`FVY4.gif  
奋斗人生(18180850)  15:32:05  
我刚才电脑有问题，所有的讨论组都消失了，谁能发我下面这句话到 15：20之间漏掉的讲义？  
  
这就要看一个称为 mro 的列表。  
解释器会将类型顺序放在这个列表里面。  
\_\_mro\_\_(X, A, B, object)  
In [1]: class A(object): pass  
In [3]: A.mro()  
Out[3]: [\_\_main\_\_.A, object]  
  
  
KEEP WALKING<huanganxin@yahoo.com.cn>  15:33:23  
 谁发下  
奋斗人生(18180850)  15:33:30  
求！  
KEEP WALKING<huanganxin@yahoo.com.cn>  15:33:47  
15：05以后的  
奋斗人生(18180850)  15:34:08  
求啊！

就这继续吧，打扰别人也顾不上了。  
Ling(124082922)  15:20:00  
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\X@8}U9MLE}EBUE273)]9PGF.gif  
Q.yuhen(1620443)  15:20:16  
mro 列表里保存了当前类型和所有基类，直到 object。  
关键是顺序，这影响到成员的搜索。  
默认 (new-style class) 规则是: 从下到上，从左到有。  
从下到上，自然是说从当前类型，往基类排。  
Q.yuhen(1620443)  15:21:19  
从左到右，则是你声明基类的顺序。  
比如 X(A, B) 和 X(B, A) 在 mro 里面的顺序就不一样。  
注意，我们这说的是 new-style class。  
Q.yuhen(1620443)  15:22:23  
因为 Python 2 里面有两套类型体系，但 classic 模式已经被 python 3 抛弃。  
我建议日常开发的时候最好都是用 new-style。  
现在了解了 mro 对成员搜索的影响。  
我们完全可以利用该特征动态做些事情。  
Q.yuhen(1620443)  15:23:27  
每个 class 对象都有几个属性。  
\_\_mro\_\_ 是一个，但我们不会直接处理这个。  
\_\_bases\_\_ 是个可以编辑的列表。保存了直接继承的所有基类。  
我们要动的就是这个，比如调整多个基类的顺序。  
从而影响 mro 列表  
或者动态增加基类  
减少基类  
Q.yuhen(1620443)  15:24:35  
\_\_bases\_\_ 和 \_\_base\_\_ 是两个  
后者总是指向 \_\_bases\_\_[0]  
而且只读。  
了解了 LEGB 和 mro 两套查找规则。  
我们完全可以在名字空间中自由发挥。  
连类型都可以自由变更，就没有什么做不到的。  
Q.yuhen(1620443)  15:25:49  
------ over，就这些吧。打字打得手都成鸡爪子了。-------

C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\{H$]5UNO{S70ZEF2HL7[CF0.png交谈中请勿轻信汇款、中奖信息、陌生电话，勿使用外挂软件。  
  
Q.yuhen(1620443)  17:25:12  
http://ejdb.org  这个发过吧？  
T.t.T!Ck.(121787333)  17:25:17  
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\X@8}U9MLE}EBUE273)]9PGF.gif刚看完他的演示视频。。貌似和我的理解没啥太大的区别啊  
Q.yuhen(1620443)  17:25:23  
https://github.com/shuttler/nessDB/  
T.t.T!Ck.(121787333)  17:25:35  
最终还是取决于数据是如何存储的  
Q.yuhen(1620443)  17:26:45  
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\}$$SH8F5BOR8W96(N@`FVY4.gif 大牛就是不同  
搂一眼就明白了，板载！  
T.t.T!Ck.(121787333)  17:27:46  
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\)QN1UH78VKP2T7)IA]ZM(FW.gif  
不许取笑  
Q.yuhen(1620443)  17:28:01  
 大拇指  
T.t.T!Ck.(121787333)  17:28:31  
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\QZTUPMQ{U[1R1LIF8A1RU(J.gif  
Q.yuhen(1620443)  17:28:50  
接下来发一些 Cloud 的资源，需要的接住了。  
http://www.openstack.org  
T.t.T!Ck.(121787333)  17:29:01  
好奇的就是。。他的数据具体的存储格式是怎样的。。。怎样做到将local这个特性最大化  
Q.yuhen(1620443)  17:29:05  
http://devstack.org  
http://trystack.org  
http://swiftstack.com  
渔(21457994)  17:29:25  
哇，好东西   
Q.yuhen(1620443)  17:29:29  
http://cloudstack.apache.org/cloudstack/index.html  
http://opennebula.org  
http://www.opennodecloud.com  
=黑仔(393324321)  17:29:55  
哪里找的都是  
Q.yuhen(1620443)  17:30:11  
平时研究留存  
看到好东西，随手就放到阅读列表，睡觉前整理一下。  
折腾一个方向，总归要找很多资料相互印证吧。  
慢慢就积累了很多东西  
Q.yuhen(1620443)  17:31:42  
------- 分隔线 ----------  
奋斗人生(18180850)  17:31:49  
可恨老大聪明一世，可怜竟也是个池中溺人。  
Q.yuhen(1620443)  17:31:55  
学习和研究 Python 的注意了  
奋斗人生(18180850)  17:31:58  
------- 继续分隔线 ----------  
Q.yuhen(1620443)  17:32:06  
发一些和 Python Debug 有关的资源。  
T.t.T!Ck.(121787333)  17:32:11  
pdb  
Q.yuhen(1620443)  17:32:21  
http://guppy-pe.sourceforge.net  
奋斗人生(18180850)  17:32:22  
接收  
Q.yuhen(1620443)  17:32:28  
http://pythonhosted.org/line\_profiler/  
https://github.com/fabianp/memory\_profiler  
http://mg.pov.lt/objgraph/  
https://github.com/mitotic/otrace  
https://pypi.python.org/pypi/pudb  
http://pyrasite.com  
https://github.com/wyplay/pytracemalloc  
有些年前发过了  
渔(21457994)  17:34:31  
恩，有些在你的bookmark里面   
Q.yuhen(1620443)  17:35:07  
@奋斗：你现在说话怎么都是红楼腔？  
咋了？  
奋斗人生(18180850)  17:35:51  
http://guppy-pe.sourceforge.net/ 这个打不开，其他都能打开  
Q.yuhen(1620443)  17:36:00  
翻墙  
=黑仔(393324321)  17:36:07  
求打包给吧  
奋斗人生(18180850)  17:36:10  
翻了啊  
我是我们公司翻墙小王子  
Qing (907751369)  17:36:33  
不如把它共享了  
=黑仔(393324321)  17:38:19  
求共享  
Q.yuhen(1620443)  17:39:13  
以前共享过了  
本群的惯例是年底送福利  
=黑仔(393324321)  17:39:35  
 我。。。。  
这年底得等死啊  
polaris(274768166)  17:39:49  
月底也得送  
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\X@8}U9MLE}EBUE273)]9PGF.gif  
=黑仔(393324321)  17:40:07  
Q.yuhen(1620443)  17:40:18  
这姑娘又出来了？