学懂元类，你只需要知道两句话：

* 道生一，一生二，二生三，三生万物
* 我是谁？我从哪来里？我要到哪里去？

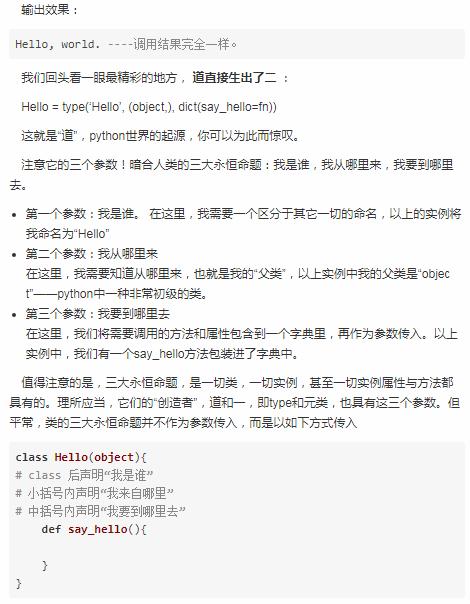
在python世界，拥有一个永恒的道，那就是“type”，请记在脑海中，type就是道。如此广袤无垠的python生态圈，都是由type产生出来的。

道生一，一生二，二生三，三生万物。

1. **道** 即是 type
2. **一** 即是 metaclass(元类，或者叫类生成器)
3. **二** 即是 class(类，或者叫实例生成器)
4. **三** 即是 instance(实例)
5. **万物** 即是 实例的各种属性与方法，我们平常使用python时，调用的就是它们。

道和一，是我们今天讨论的命题，而二、三、和万物，则是我们常常使用的类、实例、属性和方法，用hello world来举例：





* 造物主，可以直接创造单个的人，但这是一件苦役。造物主会先创造“人”这一物种，再批量创造具体的个人。并将三大永恒命题，一直传递下去。
* “道”可以直接生出“二”，但它会先生出“一”，再批量地制造“二”。
* type可以直接生成类（class），但也可以先生成元类（metaclass），再使用元类批量定制类（class）。

元类——道生一，一生二

一般来说，元类均被命名后缀为Metalass。想象一下，我们需要一个可以自动打招呼的元类，它里面的类方法呢，有时需要say\_Hello，有时需要say\_Hi，有时又需要say\_Sayolala，有时需要say\_Nihao。

如果每个内置的say\_xxx都需要在类里面声明一次，那将是多么可怕的苦役！ 不如使用 **元类**来解决问题。

以下是创建一个专门“打招呼”用的元类代码：



来！一起根据道生一、一生二、二生三、三生万物的准则，走进元类的生命周期吧！

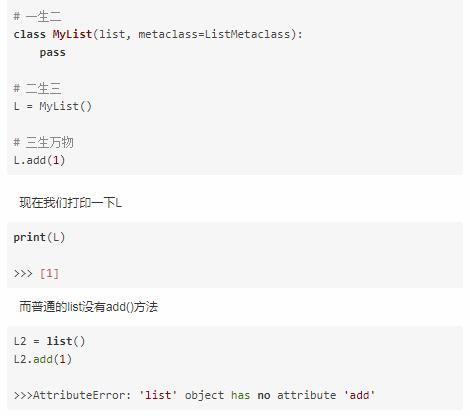


注意：通过元类创建的类，第一个参数是父类，第二个参数是metaclass

普通人出生都不会说话，但有的人出生就会打招呼说“Hello”，“你好”,“sayolala”，这就是天赋的力量。它会给我们面向对象的编程省下无数的麻烦。

现在，保持元类不变，我们还可以继续创建Sayolala， Nihao类，如下：





太棒了！学到这里，你是不是已经体验到了造物主的乐趣？

python世界的一切，尽在掌握。

年轻的造物主，请随我一起开创新世界。

我们选择两个领域，一个是Django的核心思想，“Object Relational Mapping”，即对象-关系映射，简称ORM。

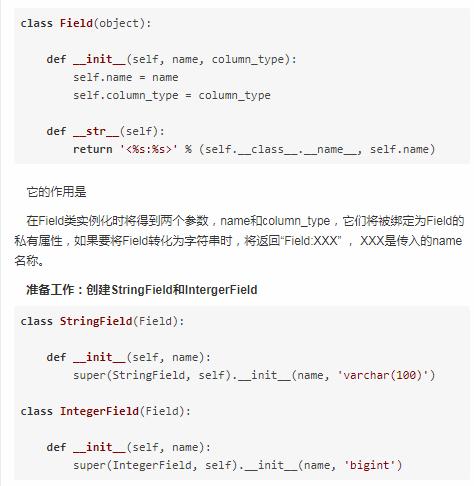
这是Django的一大难点，但学完了元类，一切变得清晰。你对Django的理解将更上一层楼！

另一个领域是爬虫领域（黑客领域），一个自动搜索网络上的可用代理，然后换着IP去突破别的人反爬虫限制。

这两项技能非常有用，也非常好玩！

挑战一：通过元类创建ORM

准备工作，创建一个Field类



它的作用是

在StringField,IntegerField实例初始化时，时自动调用父类的初始化方式。

道生一



它做了以下几件事

1. 创建一个新的字典mapping
2. 将每一个类的属性，通过.items()遍历其键值对。如果值是Field类，则打印键值，并将这一对键值绑定到mapping字典上。
3. 将刚刚传入值为Field类的属性删除。
4. 创建一个专门的\_\_mappings\_\_属性，保存字典mapping。
5. 创建一个专门的\_\_table\_\_属性，保存传入的类的名称。

一生二



IntegerField(‘id’)就会自动解析为：

Model.\_\_setattr\_\_(self, ‘id’, IntegerField(‘id’))

因为IntergerField(‘id’)是Field的子类的实例，自动触发元类的\_\_new\_\_，所以将IntergerField(‘id’)存入\_\_mappings\_\_并删除这个键值对。

二生三、三生万物

当你初始化一个实例的时候并调用save()方法时候

u = User(id=12345, name='Batman', email='batman@nasa.org', password='iamback')u.save()

这时先完成了二生三的过程：

1. 先调用Model.\_\_setattr\_\_，将键值载入私有对象
2. 然后调用元类的“天赋”，ModelMetaclass.\_\_new\_\_，将Model中的私有对象，只要是Field的实例，都自动存入u.\_\_mappings\_\_。

接下来完成了三生万物的过程：

通过u.save()模拟数据库存入操作。这里我们仅仅做了一下遍历\_\_mappings\_\_操作，虚拟了sql并打印，在现实情况下是通过输入sql语句与数据库来运行。



这里，我们利用request包，把百度的源码爬了出来。

试一试抓百度

把这一段粘在get\_page.py后面，试完删除

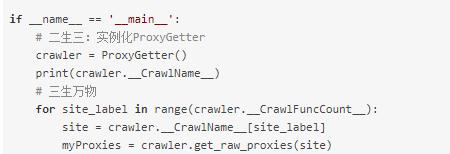


接下来进入正题：使用元类批量抓取代理

批量处理抓取代理







道生一：元类的\_\_new\_\_中，做了四件事：

1. 将“crawl\_”开头的类方法的名称推入ProxyGetter.\_\_CrawlName\_\_
2. 将“crawl\_”开头的类方法的本身推入ProxyGetter.\_\_CrawlFunc\_\_
3. 计算符合“crawl\_”开头的类方法个数
4. 删除所有符合“crawl\_”开头的类方法

怎么样？是不是和之前创建ORM的\_\_mappings\_\_过程极为相似？

一生二：类里面定义了使用pyquery抓取页面元素的方法

分别从三个免费代理网站抓取了页面上显示的全部代理。

如果对yield用法不熟悉，可以查看：

二生三：创建实例对象crawler

略

三生万物：遍历每一个\_\_CrawlFunc\_\_

1. 在ProxyGetter.\_\_CrawlName\_\_上面，获取可以抓取的的网址名。
2. 触发类方法ProxyGetter.get\_raw\_proxies(site)
3. 遍历ProxyGetter.\_\_CrawlFunc\_\_,如果方法名和网址名称相同的，则执行这一个方法
4. 把每个网址获取到的代理整合成数组输出。