# Grabber架构体系介绍

目录

[Grabber架构体系介绍 1](#_Toc37346577)

[1. 简介 1](#_Toc37346578)

[1.1 Grabber名称的由来 1](#_Toc37346579)

[1.2 为什么要创造Grabber 1](#_Toc37346580)

[1.3 Grabber的设计思想 3](#_Toc37346581)

# 1. 简介

## 1.1 Grabber名称的由来

斯坦福大学教授Edison Tse在他的书中《原创新》提到的，“Grabber”是任何可以吸引人兴趣的概念，它可能是一个美好的前景，一个新的价值诉求，一个令人兴奋的产品，一种时尚。

## 1.2 为什么要创造Grabber

在现代已有众多的低中高级的编程语言，我们为什么还要去费力创建Grabber这一门语言呢？因为我们想让中国人有一门自己的编程语言，因为我们用现在的语言写代码写的不爽。C艹、Java、Python等众多编程语言面向众多领域，但是这些语言各自都有各自的缺点。

比如C艹需要自己手动去管理内存，麻烦。但是这个麻烦的特性在面向底层开发的时候却化身成了一把利剑散发出它的寒芒。

又比如Java，Java拥有自己一套的内存管理机制，能够让开发人员的中心都放在业务层面上，而不是去关心这个对象我需不需要回收。但是也正因为这个优势让Java在底层的通信以及驱动编写领域丧失了生存的机会。虽然有，但是不大。

再说Python… 这们语言简化了很多东西，比如我们在打印HelloWorld的时候只需要像这样按下回车…

>>> print “Hello,World!”

File "<stdin>", line 1

print "Hello,World!"

SyntaxError: Missing parentheses in call to 'print'.

哦~我知道了

>>> print (“Hello,World!”)

Hello,World!

好了，只需要按下回车键即可输出一段话。

那么Grabber的重点就是要将这些高级语言的特性整合起来，并开创自己的设计思想让开发人员从老一套的OOP思想中走出来。OPP思想确实好，但是基本都没用过，OPP的思想是以接口为准。提供代码基本上都是接口从而达到调用者根本不需要关心内部代码实现。同时实现解耦合。

但是OOP写起来会非常繁琐，现在很多脚本语言都有自动生成Get\Set等操作，从而简化OOP的开发。

## 1.3 Grabber的设计思想

Grabber的设计思想也是基于OOP进行改造的，Gabber中没有接口，取而代之的（Macro definition）宏定义，它相当于是你的类抽象化的结构体，方便用于源码展示。

Grabber宏定义和C中的不同，C里面的宏定义是用于替换的而Grabber中的是用于抽象类的结构体而产生的。

比如，你这样规定一个类必须拥有的成员以及方法

文件名: people.ps

#define

set {

name,

parent,

age

},

fun {

buy(),

run()

}

#endif

然后使用#include引入文件

#include ’people.ps’

class Zs {

省略实现……

}

当你这样定义了一个Zs类的时候在其他地方声明

set zs = Zs()

可以看到我们并没有声明Zs()是哪个对象，但是编译器会自动查找并匹配符合Zs对象的结构体文件。