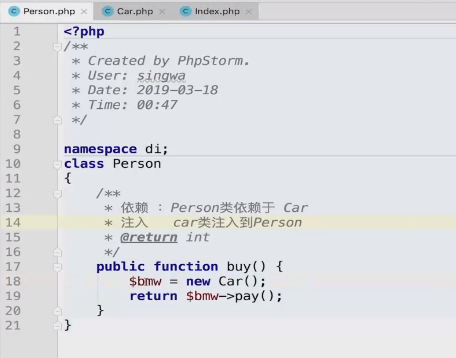
## 依赖对象、注入、控制反转



###### 依赖（事物）、注入（行为）

如上图的情况就是，

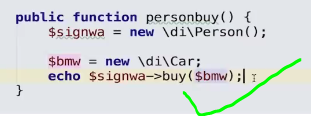
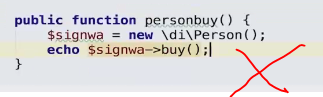
person类依赖car类，没了这个car类或者这个类改了，person类都要改。所以对于person类来说依赖对象是car类（实际还可能依赖几个类）。

car类注入了person类，这没什么好说的，person类里面有个car类。

###### 控制反转

**谁控制谁，控制什么？**

上图的情况就是person类控制car类（谁控制谁）被创建（控制它做了“被创建”’这个行为），但是实际项目中我们不能这样写代码，专业术语说就是耦合度依赖度太高，用我自己的话来说就是**可能会根据业务需求而改变的东西都不能放到类里面，要放到控制器**

**为什么叫反转，什么反转了？**

就想上面说的最直接的写法应该是person类创建car类，但是我们不能这样，我们要让控制器去创建car类。所以在这个过程中，person类主动创建car类，变成了被控制器创建了car类，主动变成了被动，所以叫反转。

（其实我个人是觉得这样来的名字真的很鸡肋，有时候真的不能根据这些英文名字直接翻译过来的中文名去理解它的含义）

参考：<https://blog.csdn.net/qq_42709262/article/details/81951402>

## 反射机制

###### 作用

可以通过反射机制，获取类的属性、方法、甚至注释，然后对其进行操作。

就像你对一个数组进行操作，都是用自带的数组函数吧，底层肯定也有一个数组的类去给这些函数调用。只不过数组的封装的好，那个类就不暴露出来用了。

一开始觉得没什么卵用的原因是php好像自带函数能对class进行操作，但仔细一想，这些自带函数的内部肯定就是调用反射类的。

而且提供的函数肯定不全，最多是常用的

**所以反射类就用来操作一个类的。**

## 容器类

###### 描述：

容器类本身使用单例模式，里面是注册树模式，存放运行流程用过的类。

容器类get函数逻辑：

注册树中是否有该类？

存在则直接返回

利用反射类实例该类

是否有构造函数

没有则直接new

利用反射类后去构造函数的参数

如果没有参数则直接new

循环检查参数，获取其类型（类？数组？字符串？）

根据其参数类型，用反射类附上参数。

用反射类实例化该类，并返回

