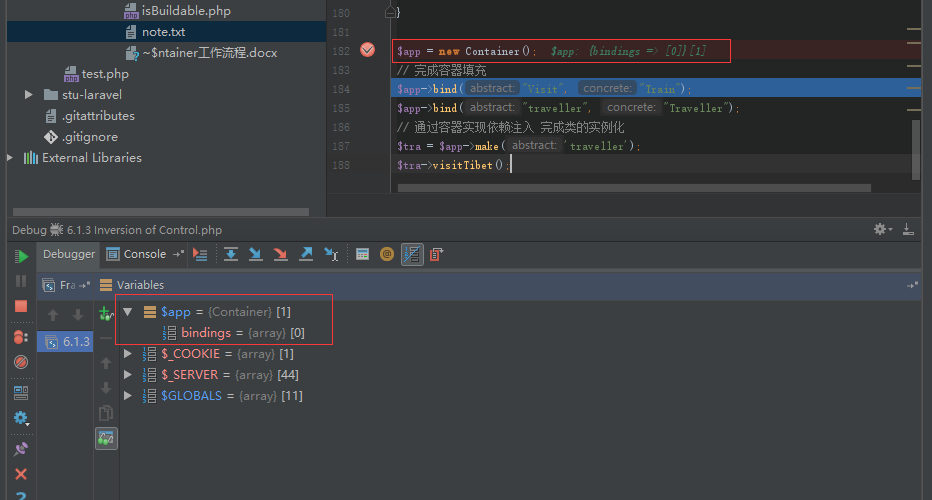
**第1行代码:**

**$app = new Container();**

**// step1. 初始化容器类Container**

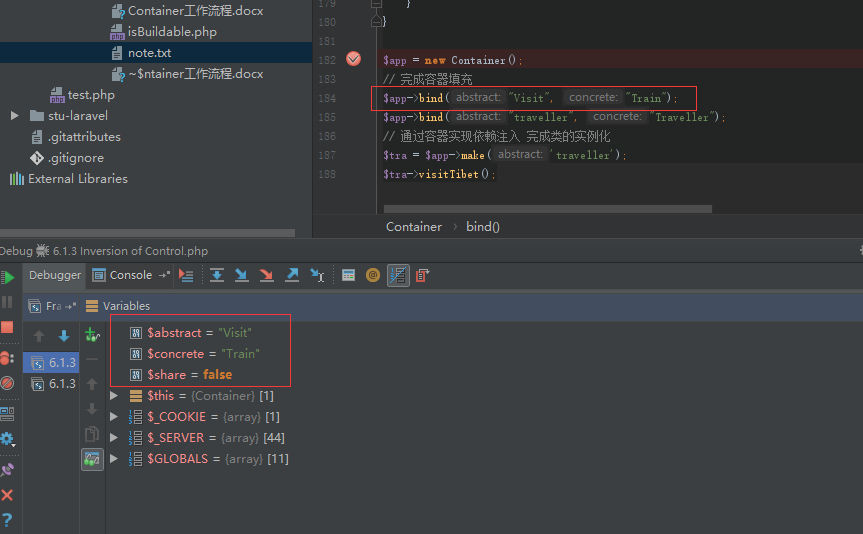
此时,可以看到$bindings = [];(空数组)



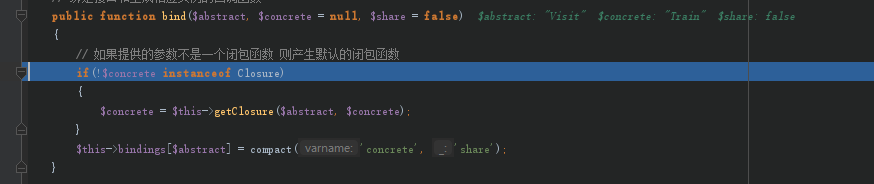
**第2行代码:**

**$app->bind(“Visit”, “Train”)**

**// step2.1 调用bind()方法**



bind()方法运行时图



此时:抽象类/接口 $abstract = “Visit” 具体类$concrete = “Train”

∵ $concrete不是闭包函数

∴ 执行了$concrete = $this->getClosure($abstract, $concrete);

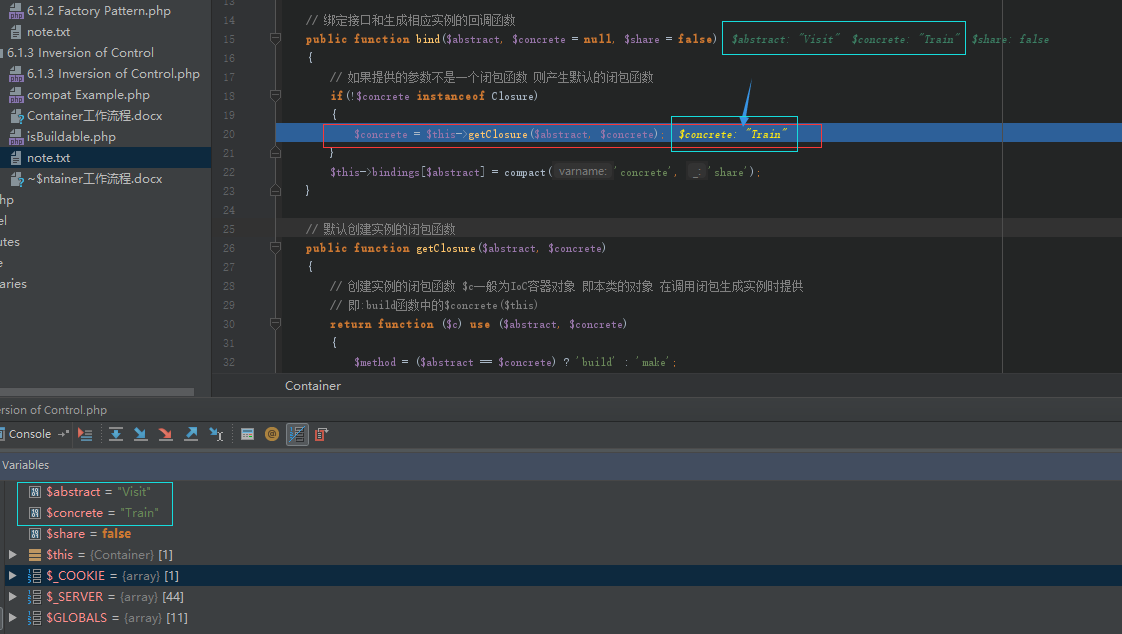
即:此时$concrete为容器默认的创建实例用闭包函数

**// step2.2 走进bind()方法的if{}代码块中**

此时:

$abstract = “Visit”

$concrete = “Train”



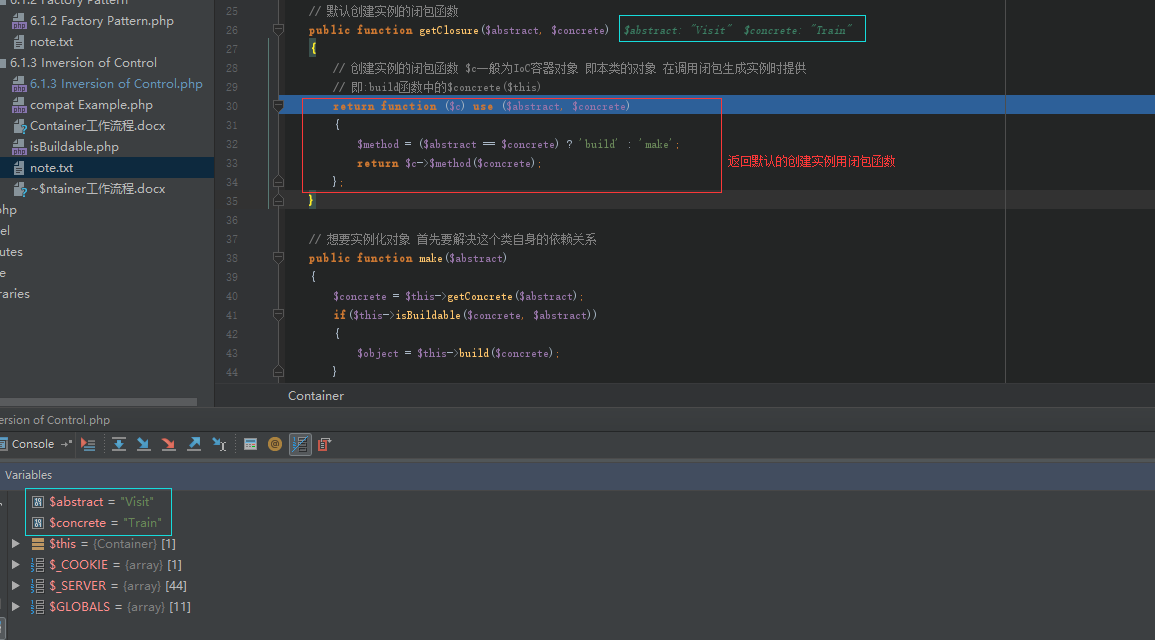
**// step2.3 调用本类的getClosure()方法**

此时:

$abstract = “Visit”

$concrete = “Train”

返回默认的创建实例用闭包函数 注意:此时闭包函数尚未执行!



**// step2.4 getClosure()方法返回默认的创建实例用闭包函数给bind()方法,然后**

**$this0>bindings[$abstract] = compat(‘concrete’, ‘share’);**

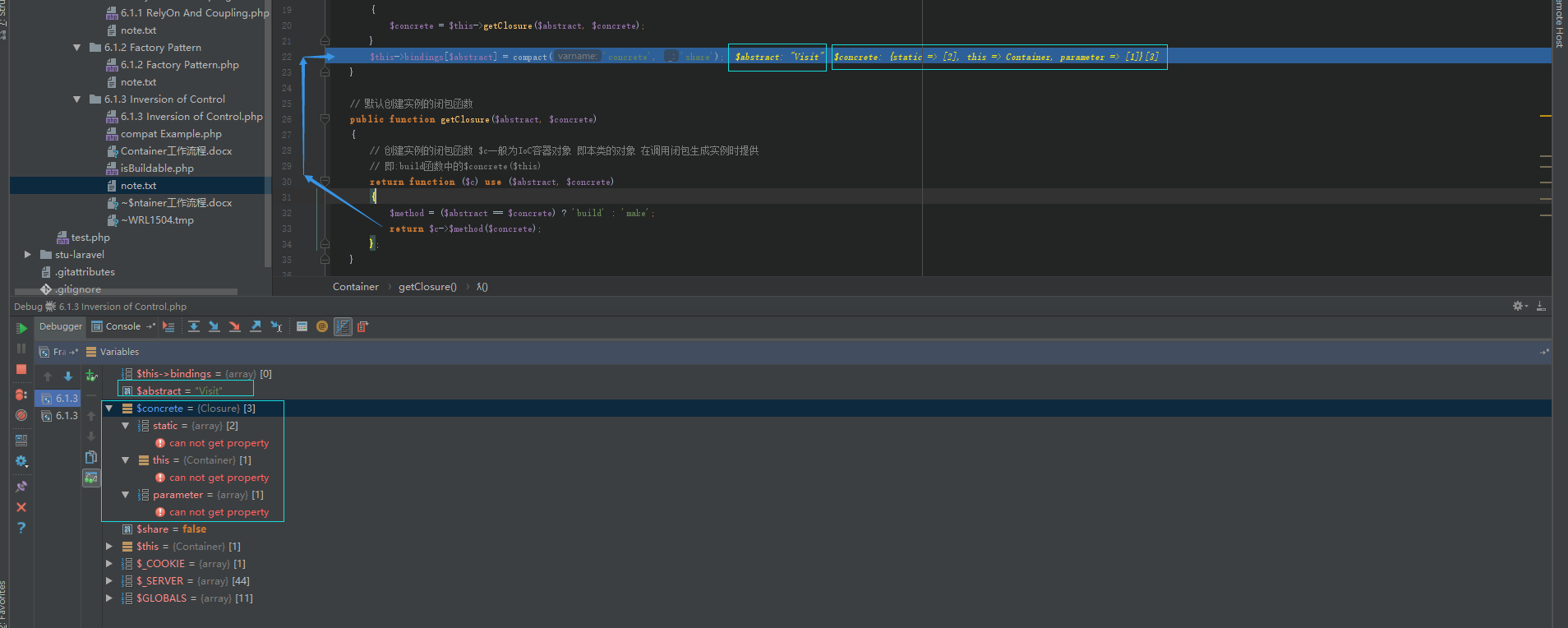
此时的$this->bindings[]数组的情况

$this->bindings[“Visit”] = [

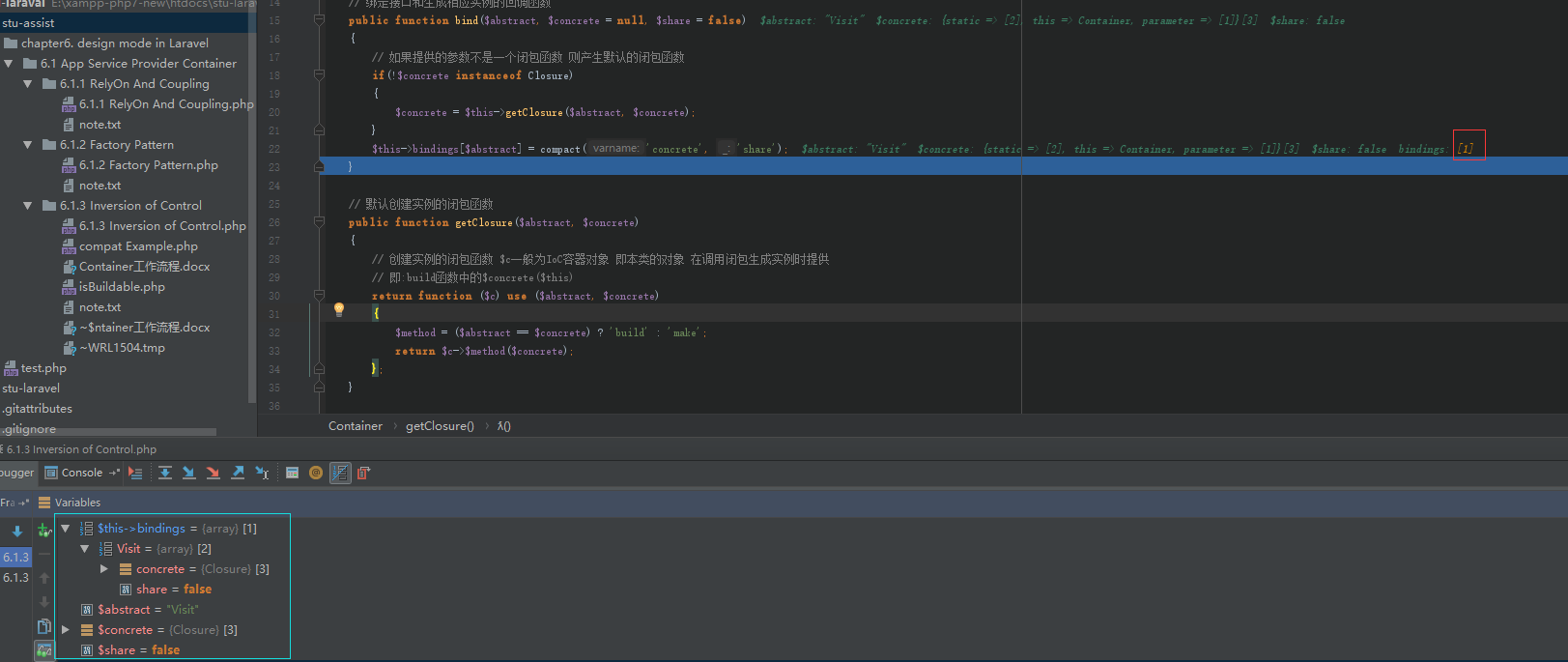
“concrete” => 闭包函数,

“share” => false

];

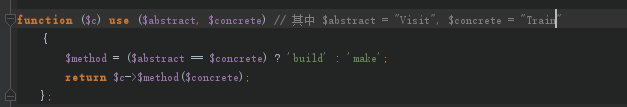


$this->bindings此时的情况如下图示:

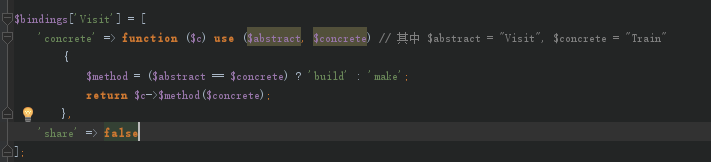


此处,无法通过Xdebug追踪出闭包函数的内容.

但是我猜测,应该是这样的:



此时,我猜测,$this->bindings的模样应该是这样的:



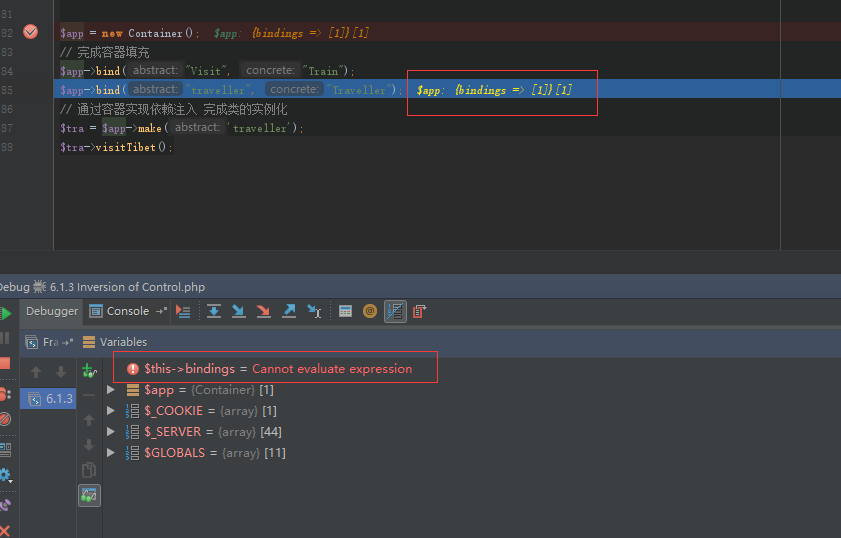
**至此,$app->bind(“Visit”, “Train”)这一行 执行完毕**

**第3行代码:**

**$app->bind(“traveller”, “Traveller”);**

**// step3.1 $app->bind(“traveller”, “Traveller”);**

注意:此时$this->bindings里面是有内容的,见下图.



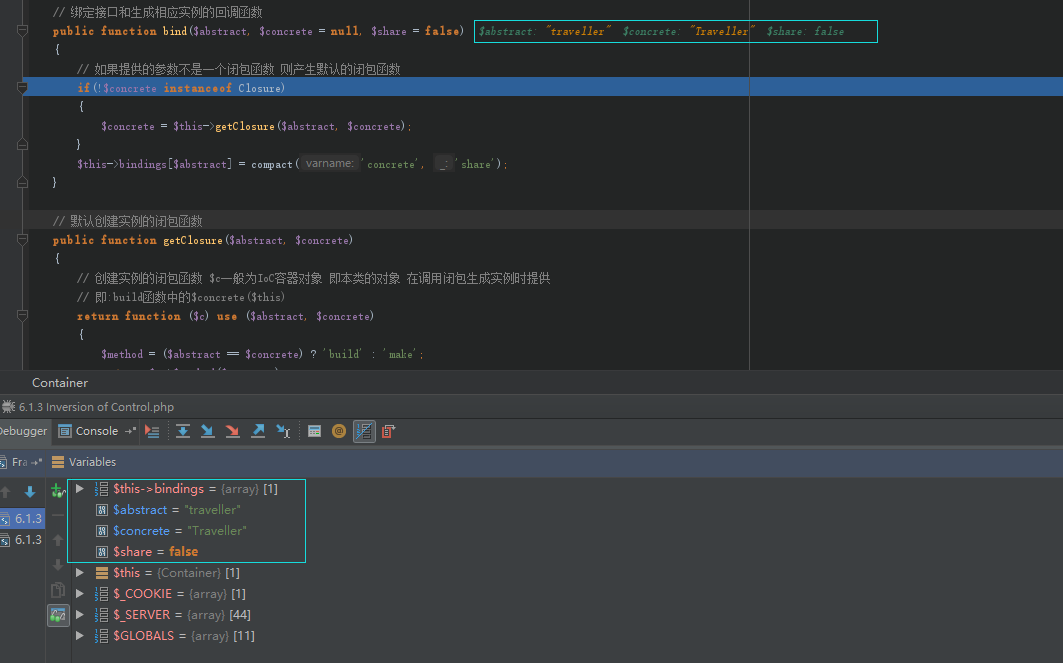
**// step3.2 走进bind()方法的if{}代码块中**

此时:

$abstract = “traveller”

$concrete = “Traveller”

$share = false;



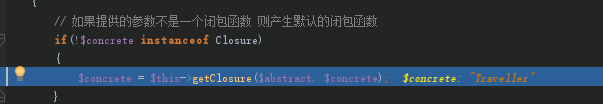
**// step3.3 调用本类的getClosure()方法**

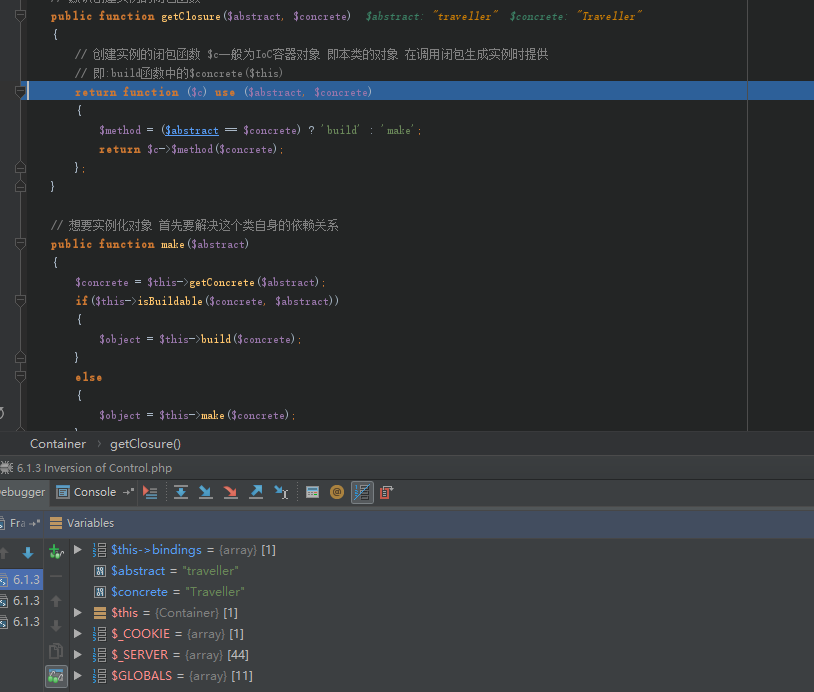
此时:

$abstract = “traveller”

$concrete = “Traveller”

返回默认的创建实例用闭包函数 注意:此时闭包函数尚未执行!





**// step3.4 getClosure()方法返回默认的创建实例用闭包函数给bind()方法,然后**

**$this0>bindings[$abstract] = compat(‘concrete’, ‘share’);**

此时的$this->bindings[]数组的情况

$this->bindings[“Visit”] = [

“concrete” => 闭包函数,

“share” => false

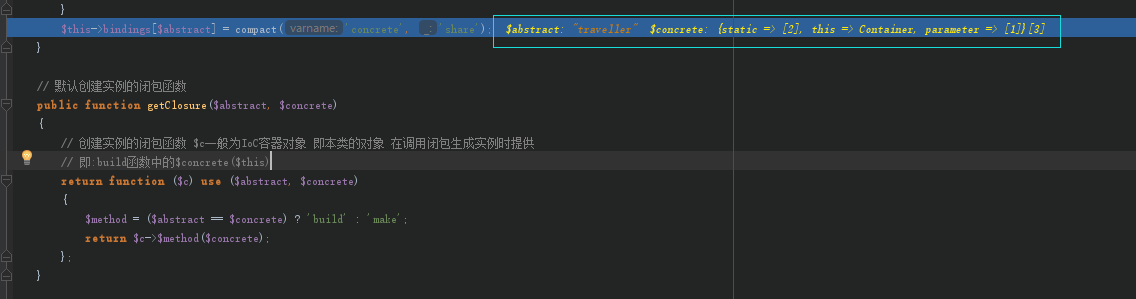
];

$this->bindings[“traveller”] = [

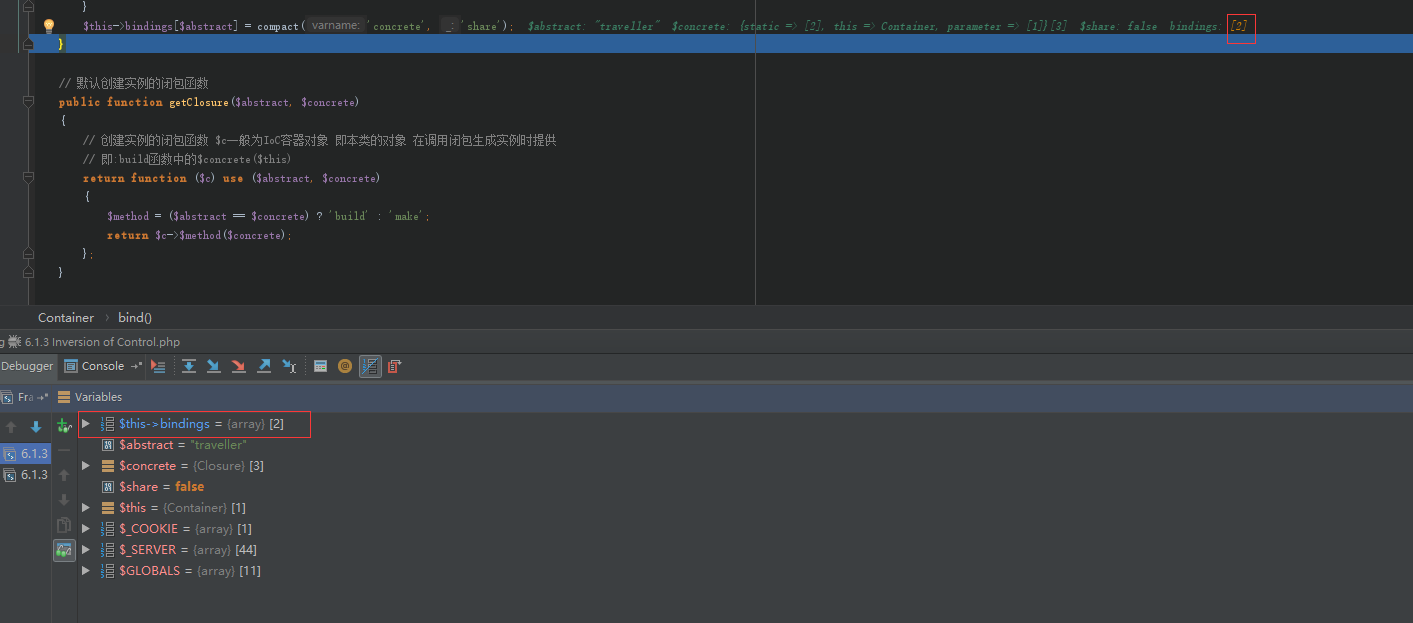
“concrete” => 闭包函数,

“share” => false

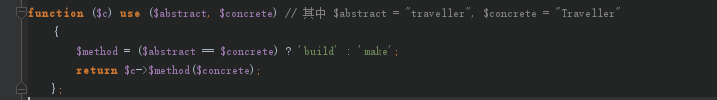
];



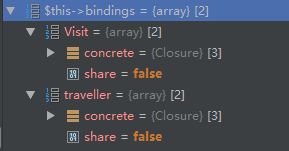
可以看到,$this->bindings的长度为2了现在.

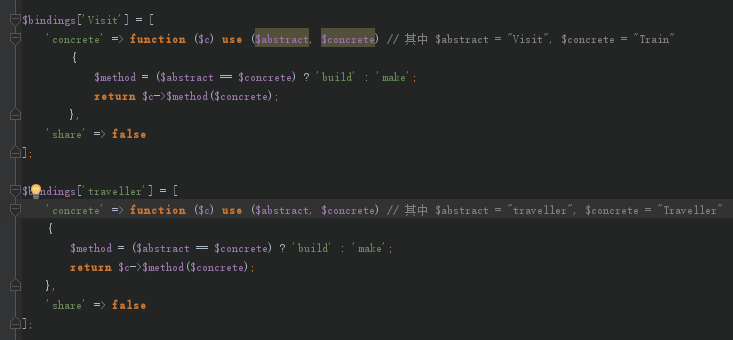


此处,我猜测闭包函数的模样:



猜测$this->bindings[]函数的模样:





**至此,$app->bind(“traveller”, “Traveller”)这一行 执行完毕**

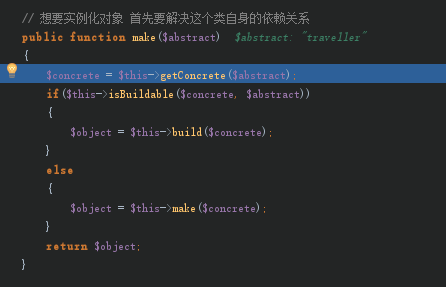
**第4行代码:**

**$tra = $app->make(‘traveller’);**

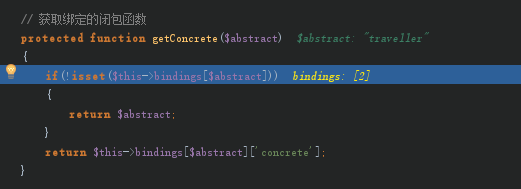
**// step4.1. 调用make()方法**

此时:

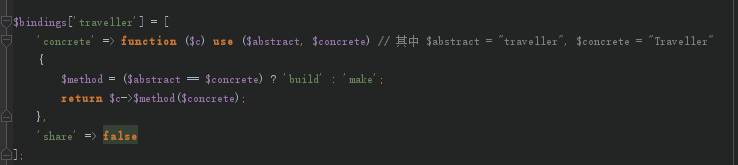
$abstract = “traveller”;



**// step4.2 调用本类的getConcrete(‘traveller’)方法**



也就是说,将:



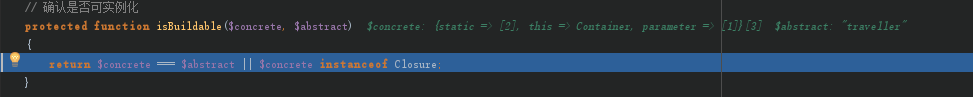
这个闭包函数返回给了make()方法

**// step4.3 调用本类的isBuildable()方法**

此时:

$concrete = 上图所示的闭包函数

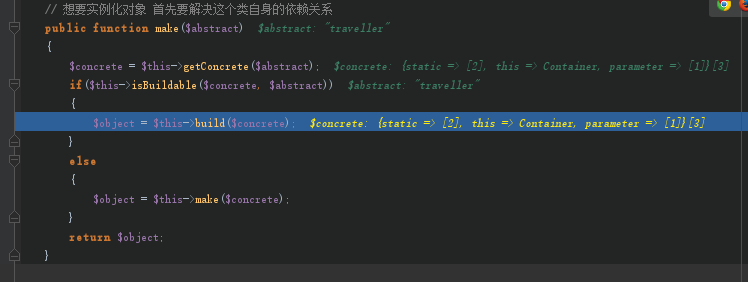
$abstract = ‘traveller’



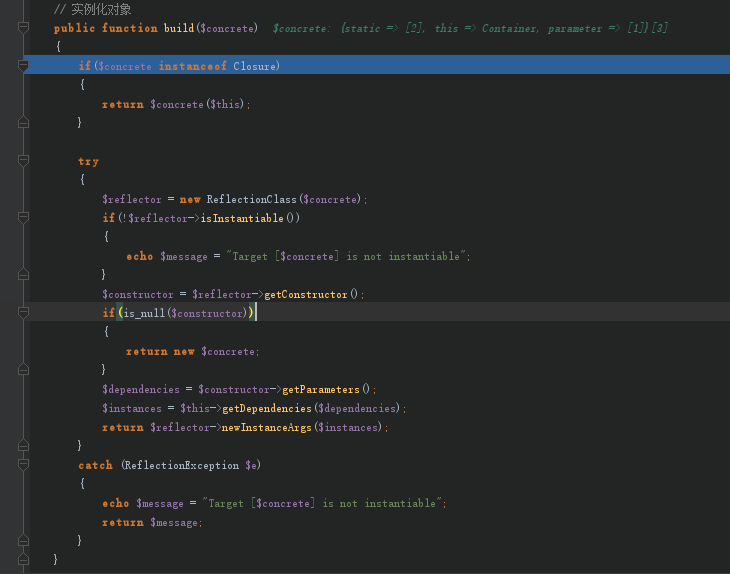
∵ $concrete是闭包函数

∴ 返回了true

**// step4.4 受到$this->isBuildable()方法的影响,进入if{}else{}代码块的if部分**



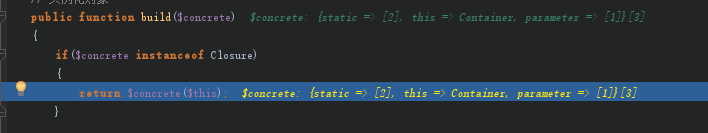
那么来看build()方法:



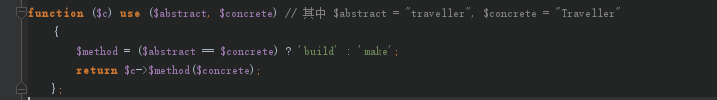
∵ $concrete是闭包

∴ 执行的代码块中的if部分

即:将本类当做形参$c的实参,传递给了闭包函数.并执行闭包函数



闭包函数如下图示:



我分析:

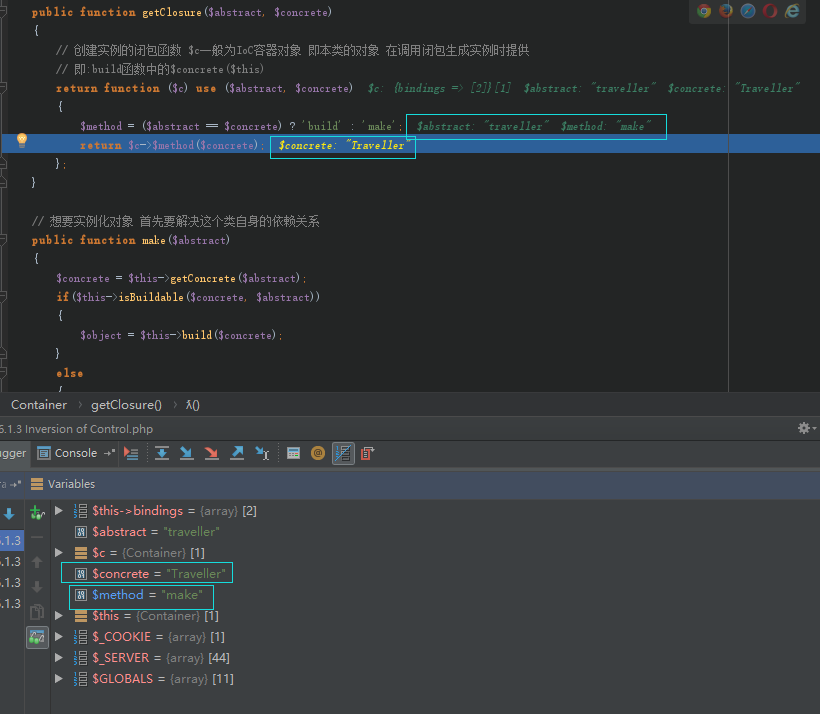
∵ $abstract = “traveller”

且 $concrete = “Traveller”

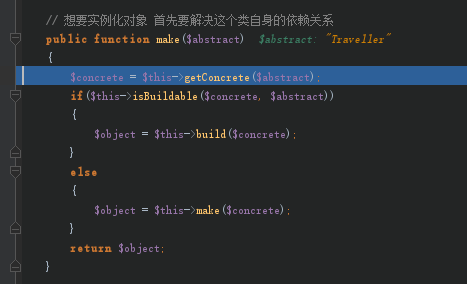
∴ $method = ‘make’ 不等于 ‘build’

即:

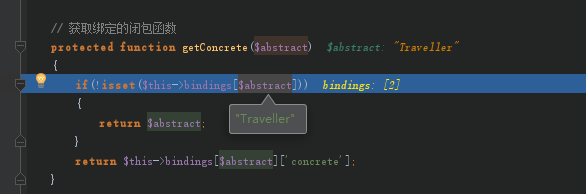
执行本类的make(‘Traveller’)方法



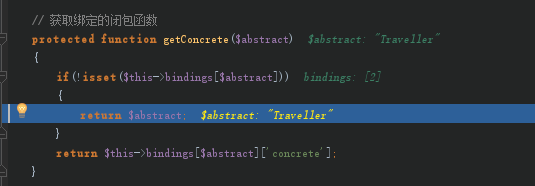
**// step4.4 执行本类的make(“Traveller”)方法**



调用本类的getConcrete(“Traveller”)方法



此时,由于$bindings中不存在key = “Traveller”的一项,所以返回”Traveller”



**// step4.5 make()方法判定isBuildable(“Traveller”, “Traveller”)**

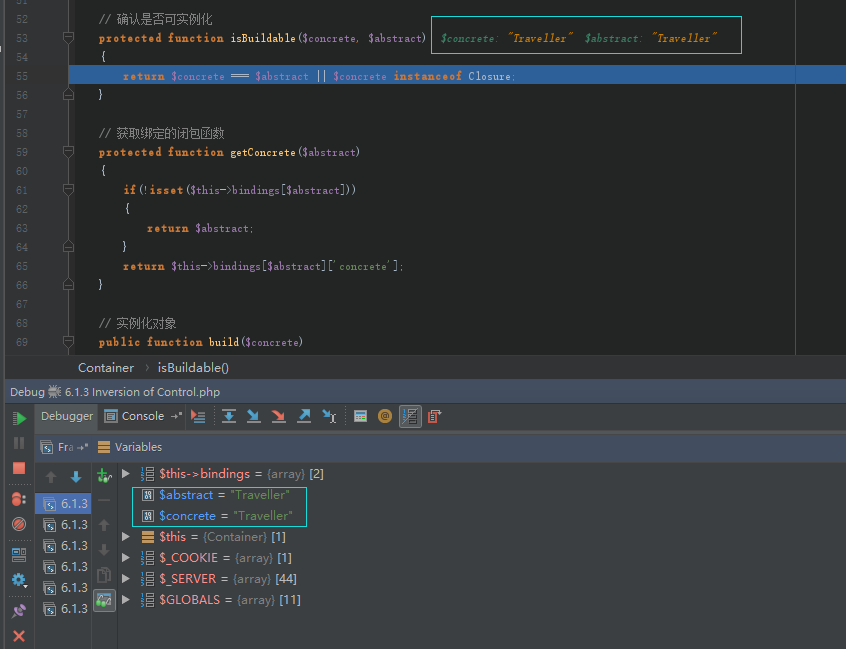


即:

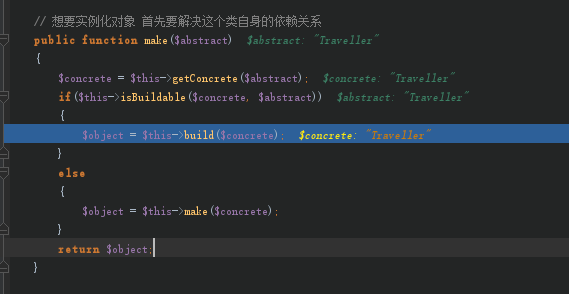
判定$concrete === $abstract 或 $concrete是否为闭包

∵ $concrete === $abstract

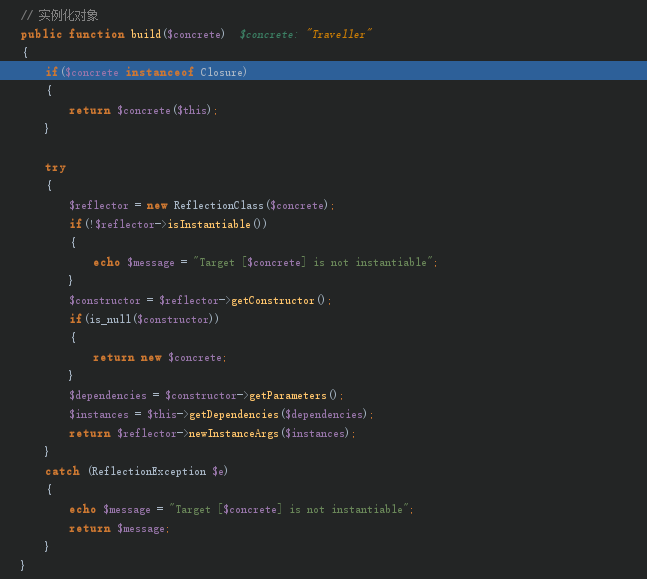
∴ 返回true



即:调用本类的build(“Traveller”)方法



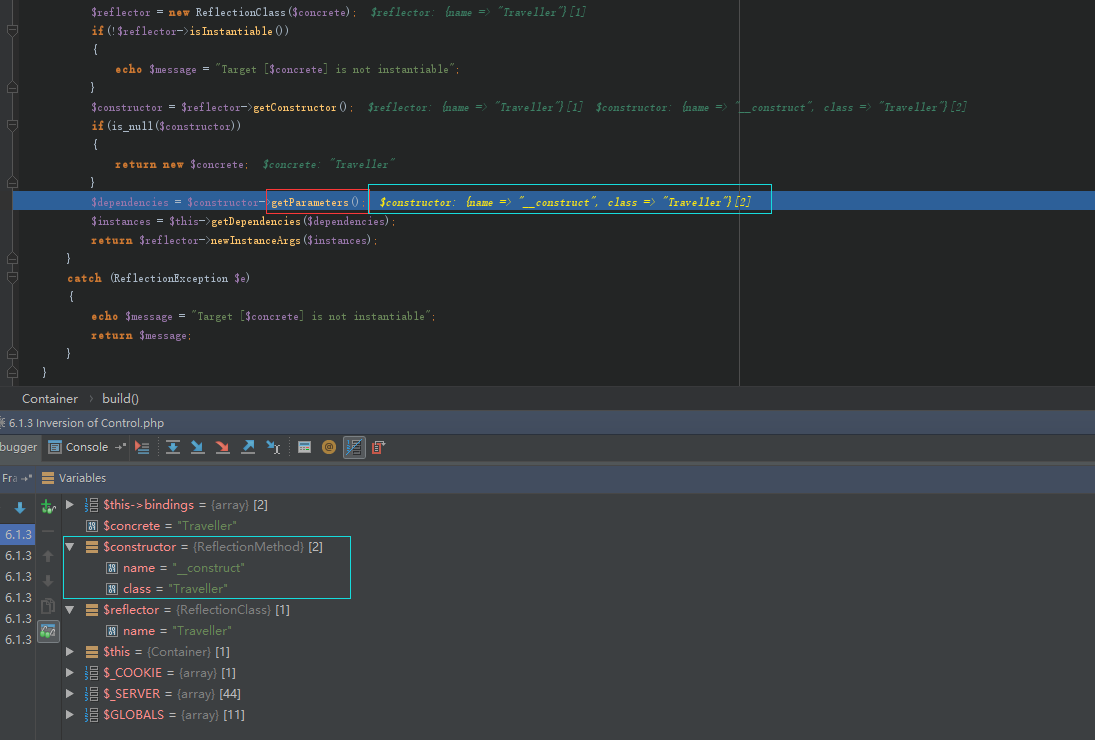
**// step4.6 build(“Traveller”)**



此时:

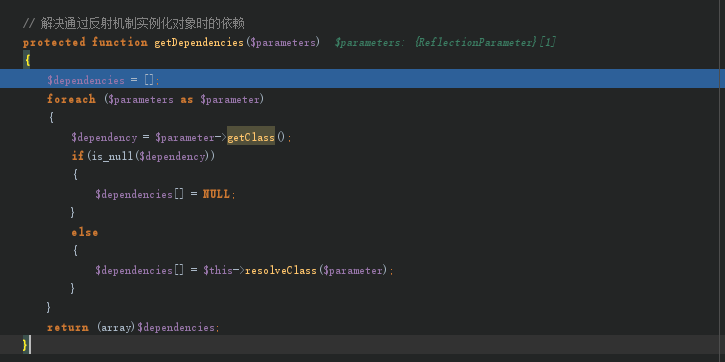
∵ $concrete = “Traveller” 不是闭包

所以 执行try代码块



通过反射进行依赖注入

调用本类的getDependencies(反射参数数组)

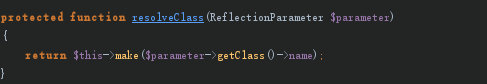


如下图:

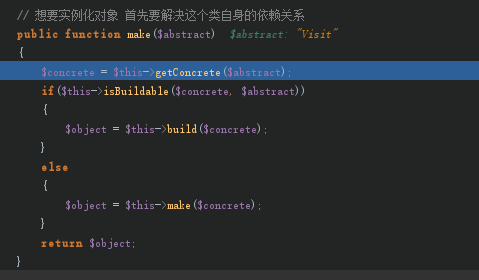
反射API解析出了Traveller类想要实例化,需要Visit类的实例作为形参$trafficTool的实参,才能实例化.所以调用了本类的resolveClass( [“name” => “trafficTool”])



**// step4.7 调用本类的resolveClass( [“name” => “trafficTool”])**



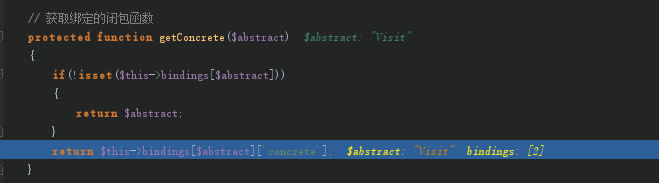
再次调用本类的make(“Visit”)方法

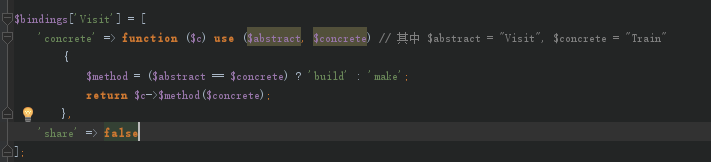


调用本类的getConcrete(“Visit”)方法

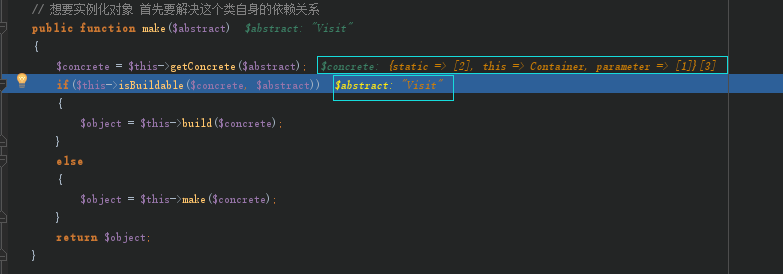
此时:

∵ $this->bindings[“Visit”]是存在的 所以返回闭包函数$this->bindings[“Visit”][‘concrete’]





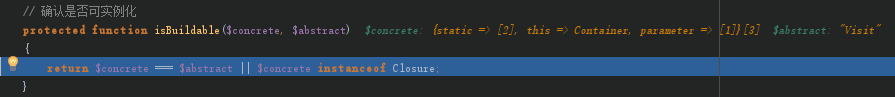
make()方法继续让isBuildable(闭包函数,”Visit”)判断

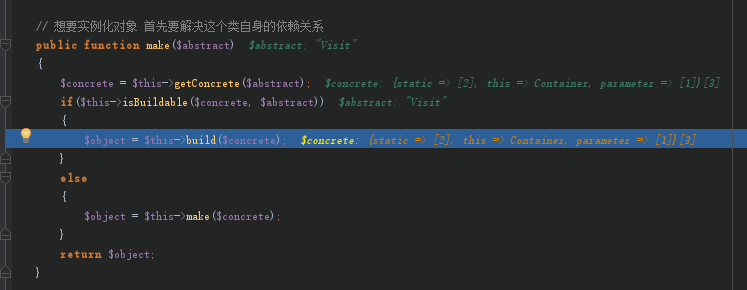


此时:

∵ $concrete是闭包函数

∴ 返回true



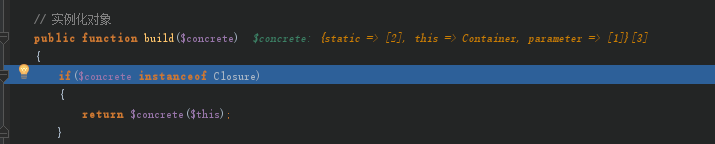


继续调用本类的build(闭包函数)

此时:

∵ $concrete是闭包函数

∴ 将本类的实例作为闭包函数形参$c的实参传入,执行这个闭包函数的返回结果



执行闭包函数:

此时:

闭包函数中:

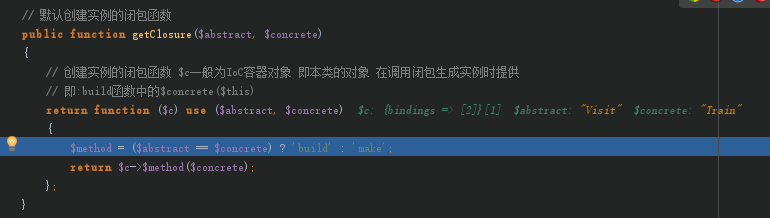
$abstract = “Visit”

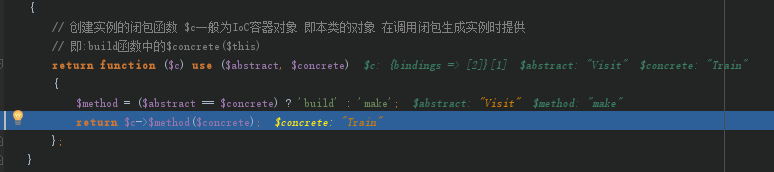
$concrete = “Train”

∵ $abstract === $concrete 不成立

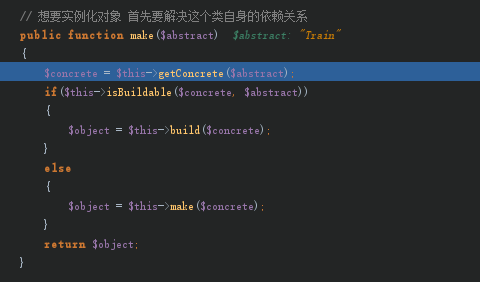
∴ $method = make

∴ 执行了 $c->make(‘Train’);



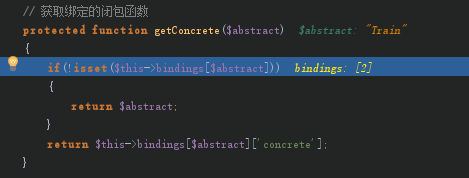


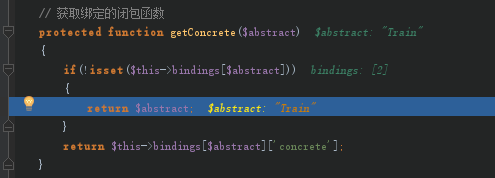
继续执行make(“Train”)



调用本类getConcrete(“Train”)方法

此时,$this->bindings[‘Train’]是不存在的,所以返回”Train”



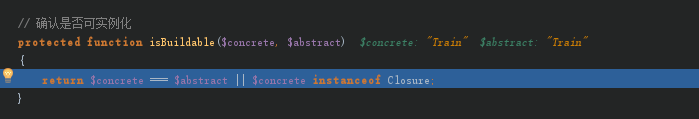


make(‘Train’)中,继续调用本类的isBuildable(“Train”,”Train”)方法

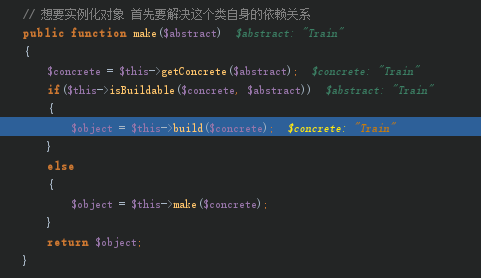
此时:

∵ “Train” === ”Train” 为true

∴ 返回true



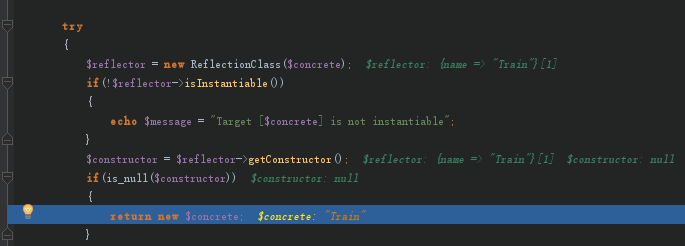
make(“Train”)方法中受isBuildable(“Train”, “Train”)方法的影响,调用了本类的build(“Train”)方法



build(“Train”)



此时,由于Train类没有依赖,build方法直接实例化Train类的对象并返回了就



则:make(“Train”)返回$object = new Train()给闭包函数

问题:怎么到的getDependencies(ReflectionParameter)?

到了getDependencies(ReflectionParameter)之后 因为拿到了Train类的实例了,所以可以顺利的实例化Traveller类了.

然后又到了make(“Traveller”)方法 make(“Traveller”)方法返回了Traveller类的实例

第5行代码

调用$tra->visitTibet() 这个没什么可说的