插件化修改策略-引用函数指针

---孙春风

1. 问题分析

var (  
 mapp=make( map[string]testInterface)  
)  
func init(){  
 mapp["test1"]=NewTest1()  
 mapp["test2"]=NewTest2()  
}  
type testInterface interface {  
 add(int,int)int  
 del(int,int)int  
}  
func NewTest1()\*test1{  
 return &test1{}  
}  
func NewTest2()\*test2 {  
 return &test2{}  
}

修改前的map存放的value实际上是一个结构体，带来的弊端

* + 1. 第一方面：一般情况下只需要一个实体即可，则另外的对象实体是内存的浪费
    2. 第二方面：一般New函数里会有go函数，无用的New会带来一些意向不到的问题

1. 修改后的代码

var (  
 mapp=make( map[string]func()testInterface)  
 DefaultConfig="test1"  
)  
func init(){  
 mapp["test1"]=NewTest1  
 mapp["test2"]=NewTest2  
}  
type testInterface interface {  
 add(int,int)int  
 del(int,int)int  
}  
func NewTest1()testInterface{  
 return &test1{}  
}  
func NewTest2()testInterface {  
 return &test2{}  
}  
func New()testInterface{*//对外提供的New函数* return mapp[DefaultConfig]()//此时才真正调用插件的New函数  
}

修改后的代码对于插件的初始化和新建做出限制

1. mapp维护的不是各个插件，而是插件的New方法
2. 根据配置读到那个插件，再调用那个插件对应的New方法进行插件的创建