```
    param2 = tmp
}
var p1 = "15"
var p2 = "40"
exchange(param1: &p1, param2: &p2)
print(p1,p2)//输出 40 15
```

如上面的代码所示，泛型可以作为函数的参数。其在语法上的表现为：在函数参数列表前使用尖括号，将要定义的泛型类型列出，其作用域为函数的参数列表与整个函数实现部分，如果要在一个函数中定义多个泛型，使用逗号进行分隔即可。

泛型除了可以用于定义函数的参数类型外，在定义数据类型时，也起着十分重要的作用。分析一下Array和Dictionary结构体的实现，读者可以发现：在声明这类集合类型时，开发者可以同时设置这些集合类型中所要存放的元素的类型。我们也可以通过泛型来实现。

熟悉程序开发的都知道，栈是一种先进后出的数据结构，其对数据的操作分为入栈和出栈两种，可以通过图11-1来理解。

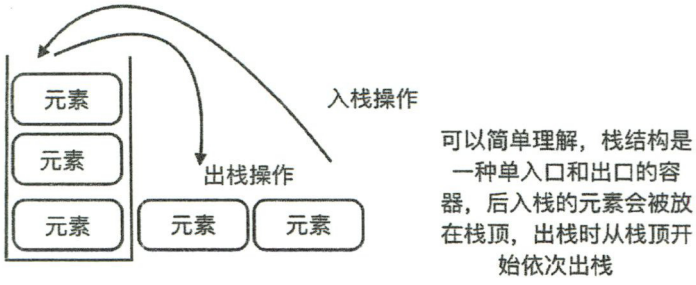

图 11-1　栈结构示意图

模仿系统集合类型的实现思路，实现一个自定义结构体类型：栈结构体，示例代码如下：

```
//定义一个栈结构体，ItemType 为定义栈中元素类型的泛型
struct Stack<ItemType> {
    //内部有关元素类型的操作都使用 ItemType
    var items:[ItemType] = []
    mutating func push(param:ItemType)  {
        self.items.append(param)
    }
    mutating func pop()->ItemType{
        return self.items.removeLast()
    }
}
//整型栈
var obj7 = Stack<Int>()
//进行入栈和出栈操作
obj7.push(param: 1)
obj7.pop()
//字符串栈
var obj8 = Stack<String>()
//进行入栈和出栈操作
```