设计模式-状态模式

参考

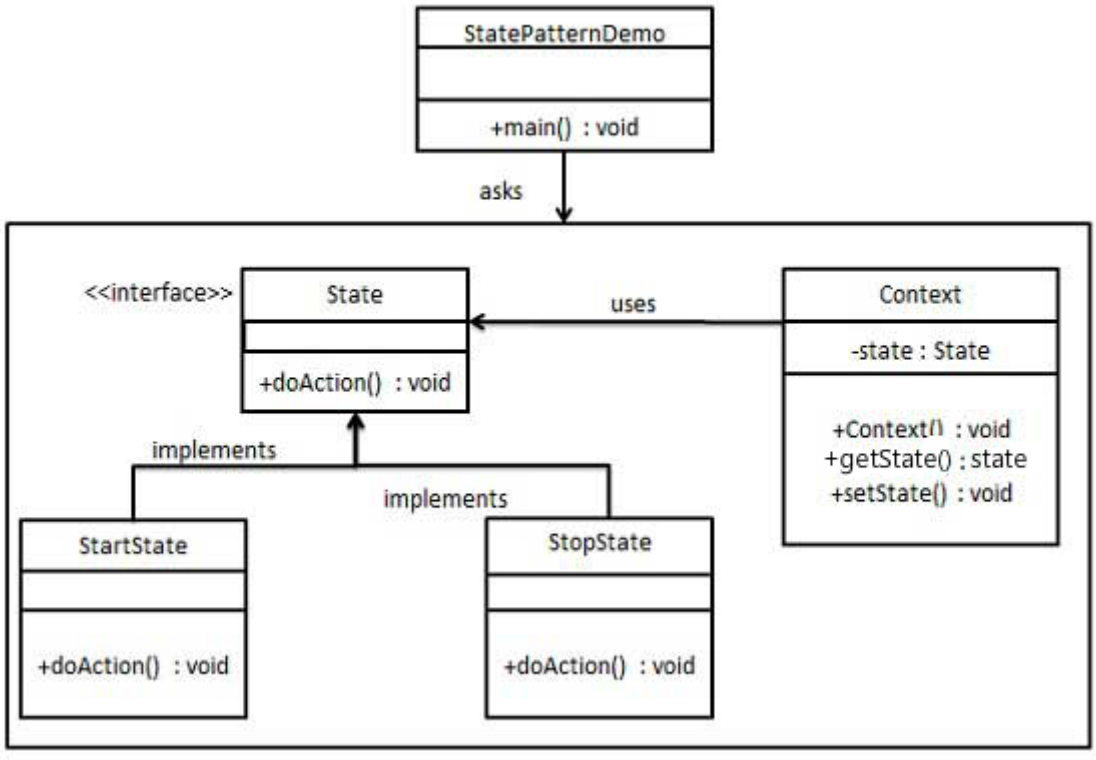
- Go设计模式(22)-状态模式*程序员麻辣烫的博客-CSDN博客*go 状态模式
- https://www.runoob.com/design-pattern/state-pattern.html

在状态模式中，类的行为是基于它的状态改变的。这种类型的设计模式属于行为型模式。在状态模式中，我们创建表示各种状态的对象和一个行为随着状态对象改变而改变的context对象。

使用场景

- 行为随状态改变而改变的场景
- 条件、分支语句的代替者

Demo分析



> 分析：
>
> 我们将创建一个State接口和实现了State接口的实体状态类。Context是一个带有某个状态的类。

Go实现

```
package main

import "fmt"

/**
* 状态接口
 */
type State interface {
    doAction(c Context)
}

type StartState struct{}

func (s *StartState) doAction(c Context) {
    fmt.Println("StartState")
    c.setState(s)
}

type StopState struct{}

func (s *StopState) doAction(c Context) {
    fmt.Println("StopState")
    c.setState(s)
}

/**
 * Context类
 */
type Context struct {
    state State
}

func (c *Context) setState(state State) {
    c.state = state
}

func (c *Context) getState() *State {
    return &c.state
}

func statePatternDemo() {
    context := &Context{}
    startState := &StartState{}
    stopState := &StopState{}
    startState.doAction(*context)
    stopState.doAction(*context)
}

func main() {
    statePatternDemo()
}
```

输出

```
StartState
StopState
```

Python实现

```
class State(object):
  """
    ## 状态的抽象类
  """
  def adAction(self, action):
    pass

class Context(object):
  def __init__(self):
    self.state = None
  def setState(self, state):
    self.state = state
  def getState(self):
    return self.state

class StartState(State):
  def adAction(self, action):
    print("Start state")
    action.setState(self )

class StopState(State):
  def adAction(self, action):
    print("Stop state")
    action.setState(self)

def test():
  context = Context()
  startState = StartState()
  stopState = StopState()
  startState.adAction(context)
  stopState.adAction(context)

if __name__ == '__main__':
  test()
```

输出

```
Start state
Stop state
```

小结

通过状态模式，将状态对象独立于Contex对象方便了状体的管理。