设计模式-备忘录模式

参考

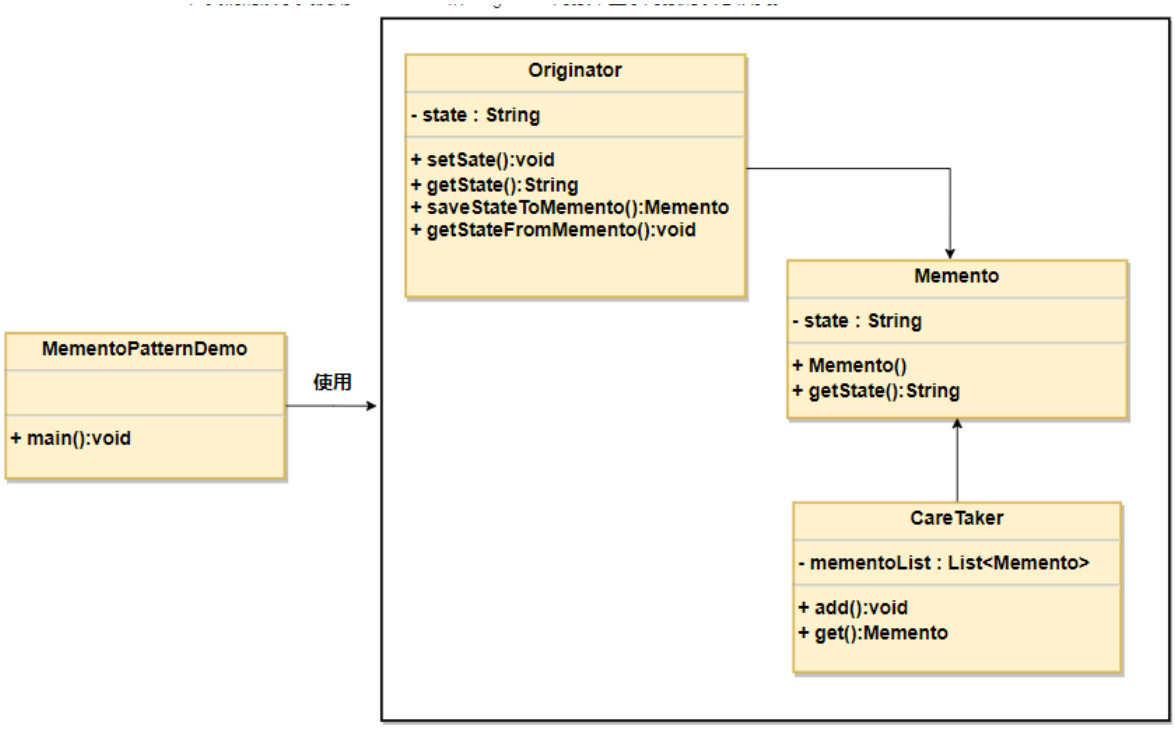
- Go设计模式(25)-备忘录模式_程序员麻辣烫的博客-CSDN博客
- (https://www.runoob.com/design-pattern/memento-pattern.html)[https://www.runoob.com/design-pattern/memento-pattern.html]

备忘录模式保存一个对象的某个状态，以便在适当的时候恢复对象。备忘录模式属于行为模式。

使用场景

- 需要保存/恢复数据的相关状态场景
- 提供一个可回滚的操作

Demo分析



> 分析：

> 备忘录模式使用三个类 *Memento*、*Originator* 和 *CareTaker*。 *Memento*包含了要被恢复的对象的状态。*Originator*创建并在*Memento*对象中存储状态。*CareTaker*对象负责从*Memento*中恢复对象的状态。

Go实现

```
package main

import "fmt"

/**
* 数据存储的对象
 */
type Memento struct {
    state string
}

func (m *Memento) GetState() string { return m.state }

type Originator struct {
    state string
}

func (m *Originator) GetState() string { return m.state }

func (m *Originator) SetState(state string) { m.state = state }

/**
* 保存状态
 */
func (m *Originator) saveStateToMemento() *Memento {
    return &Memento{m.state}
}

/**
* 加载状态
 */
func (m *Originator) getStateFromMemento(mem *Memento) {
    m.state = mem.GetState()
}

/**
* 备忘录模式核心，记录状态
 */
type CareTaker struct {
    mementoList []*Memento
}

func (c *CareTaker) add(m *Memento) {
    c.mementoList = append(c.mementoList, m)
}

func (c *CareTaker) get(index int) *Memento {
    return c.mementoList[index]
}
```

```
func test() {
    originator := &Originator{state: "State"}
    careTaker := &CareTaker{mementoList: make([]*Memento, 0)}
    originator.SetState("State == 1")
    careTaker.add(originator.saveStateToMemento())
    originator.SetState("State == 2")
    careTaker.add(originator.saveStateToMemento())

    fmt.Printf("current state: %v\n", originator.state)
    originator.getStateFromMemento(careTaker.get(0))
    fmt.Printf("current state: %v\n", originator.state)
    originator.getStateFromMemento(careTaker.get(1))
    fmt.Printf("current state: %v\n", originator.state)
}

func main() {
    test()
}
```

输出

```
current state: State == 2
current state: State == 1
current state: State == 2
```

Python实现

```
class Memento:
  """
    ## 要记录的状态
  """
  def __init__(self, state):
    self.state = state
  def GetState(self):
    return self.state

class Originator:
  """
    ## 行为对象，可加载存储的状态
  """
  def __init__(self):
    self.state = None
  def SetState(self, state):
    self.state = state
  def GetState(self):
    return self.state
  def saveStateToMemento(self):
    return Memento(self.state)
  def loadStateFromMemento(self, mem):
    self.state = mem.GetState()

class CareTaker:
  """
```

```
    ## 用于记录状态
  """
  def __init__(self):
    self.mementoList = []
  def addMemento(self, memento):
    self.mementoList.append(memento)
  def getMemento(self, index):
    if index > len(self.mementoList):
      raise Exception("index out of range")
    return self.mementoList[index]

def test():
  original = Originator()
  caretaker = CareTaker()
  original.SetState("State 1")
  caretaker.addMemento(original.saveStateToMemento())
  original.SetState("State 2")
  caretaker.addMemento(original.saveStateToMemento())
  original.SetState("State 3")

  print(f"current state: {original.GetState()}")
  original.loadStateFromMemento(caretaker.getMemento(0))
  print(f"first state: {original.GetState()}")
  original.loadStateFromMemento(caretaker.getMemento(1))
  print(f"second state: {original.GetState()}")

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

输出

```
current state: State 3
first state: State 1
second state: State 2
```

小结

从上述的案例可以看出备忘录模式，一般用于数据的缓存，恢复。针对这些功能，备忘录模式也大致被分为三块：

1. `行为对象`：用于执行起作用的对象，也就是说恢复的状态是给这个用的。
2. `状态对象`：包含被记录状态的具体形式。
3. `记录对象`：提供了状态的历史记录以及恢复到指定的状态。