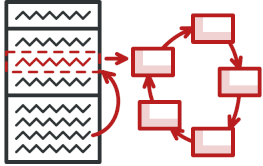




[🏠](#) / [Паттерны проектирования](#) / [Состояние](#) / [Go](#)



# Состояние на Go

**Состояние** — это поведенческий паттерн, позволяющий динамически изменять поведение объекта при смене его состояния.

Поведения, зависящие от состояния, переезжают в отдельные классы. Первоначальный класс хранит ссылку на один из таких объектов-состояний и делегирует ему работу.

 [Подробнее о паттерне Состояние →](#)

## Навигация

 [Интро](#)

 [Концептуальный пример](#)

 [vendingMachine](#)

 [state](#)


 [noItemState](#)

[Главная](#) [Рефакторинг](#) [Паттерны](#) [Премиум контент](#)  
[Форум](#) [Связаться](#)



© 2014-2023 Refactoring.Guru.

Все права защищены.

 Иллюстрации нарисовал Дмитрий Жарт

[Условия использования](#)

[Политика конфиденциальности](#)

[Использование контента](#) [About us](#)

# Концептуальный пример

Давайте применим паттерн проектирования Состояние в контексте торговых автоматов. Для упрощения задачи представим, что торговый автомат может выдавать только один товар. Также представим, что автомат может пребывать только в одном из четырех состояний:

- hasItem (имеетПредмет)
- noItem (неИмеетПредмет)
- itemRequested (выдаётПредмет)
- hasMoney (получилДеньги)

Торговый автомат может иметь различные действия. Опять-таки, для простоты оставим только четыре из них:

- Выбрать предмет
- Добавить предмет
- Ввести деньги
- Выдать предмет

Паттерн Состояние нужно использовать в случаях, когда объект может иметь много различных состояний, которые он должен менять в зависимости от конкретного поступившего запроса.

В нашем примере, автомат может быть в одном из множества состояний, которые непрерывно меняются. Припустим, что торговый автомат находится в режиме `itemRequested`. Как только произойдет действие «Ввести деньги», он сразу же перейдет в состояние `hasMoney`.

В зависимости от состояния торгового автомата, в котором он находится на данный момент, он может по-разному отвечать на одни и те же запросы. Например, если пользователь хочет купить предмет, машина выполнит действие, если она находится в режиме `hasItemState`, и отклонит запрос в режиме `noItemState`.

Программа торгового автомата не захламлена этой логикой; весь режимозависимый код обитает в реализациях соответствующих состояний.

## vendingMachine.go: Контекст

```
package main

import "fmt"

type VendingMachine struct {
    hasItem      State
    itemRequested State
    hasMoney     State
    noItem       State

    currentState State

    itemCount int
    itemPrice int
}

func newVendingMachine(itemCount, itemPrice int) *VendingMachine {
    v := &VendingMachine{
        itemCount: itemCount,
        itemPrice: itemPrice,
    }
    hasItemState := &HasItemState{
        vendingMachine: v,
    }
    itemRequestedState := &ItemRequestedState{
        vendingMachine: v,
    }
    hasMoneyState := &HasMoneyState{
        vendingMachine: v,
    }
    noItemState := &NoItemState{
        vendingMachine: v,
    }

    v.setState(hasItemState)
    v.hasItem = hasItemState
    v.itemRequested = itemRequestedState
    v.hasMoney = hasMoneyState
    v.noItem = noItemState
    return v
}

func (v *VendingMachine) requestItem() error {
    return v.currentState.requestItem()
}
```

```

}

func (v *VendingMachine) addItem(count int) error {
    return v.currentState.addItem(count)
}

func (v *VendingMachine) insertMoney(money int) error {
    return v.currentState.insertMoney(money)
}

func (v *VendingMachine) dispenseItem() error {
    return v.currentState.dispenseItem()
}

func (v *VendingMachine) setState(s State) {
    v.currentState = s
}

func (v *VendingMachine) incrementItemCount(count int) {
    fmt.Printf("Adding %d items\n", count)
    v.itemCount = v.itemCount + count
}

```

## state.go: Интерфейс состояния

```

package main

type State interface {
    addItem(int) error
    requestItem() error
    insertMoney(money int) error
    dispenseItem() error
}

```

## noItemState.go: Конкретный интерфейс

```

package main

import "fmt"

type NoItemState struct {

```

```

        vendingMachine *VendingMachine
    }

    func (i *NoItemState) requestItem() error {
        return fmt.Errorf("Item out of stock")
    }

    func (i *NoItemState) addItem(count int) error {
        i.vendingMachine.incrementItemCount(count)
        i.vendingMachine.setState(i.vendingMachine.hasItem)
        return nil
    }

    func (i *NoItemState) insertMoney(money int) error {
        return fmt.Errorf("Item out of stock")
    }

    func (i *NoItemState) dispenseItem() error {
        return fmt.Errorf("Item out of stock")
    }
}

```

## hasItemState.go: Конкретный интерфейс

```

package main

import "fmt"

type HasItemState struct {
    vendingMachine *VendingMachine
}

func (i *HasItemState) requestItem() error {
    if i.vendingMachine.itemCount == 0 {
        i.vendingMachine.setState(i.vendingMachine.noItem)
        return fmt.Errorf("No item present")
    }
    fmt.Printf("Item requestd\n")
    i.vendingMachine.setState(i.vendingMachine.itemRequested)
    return nil
}

func (i *HasItemState) addItem(count int) error {
    fmt.Printf("%d items added\n", count)
    i.vendingMachine.incrementItemCount(count)
    return nil
}

```

```

}

func (i *HasItemState) insertMoney(money int) error {
    return fmt.Errorf("Please select item first")
}

func (i *HasItemState) dispenseItem() error {
    return fmt.Errorf("Please select item first")
}

```

## itemRequestedState.go: Конкретный интерфейс

```

package main

import "fmt"

type ItemRequestedState struct {
    vendingMachine *VendingMachine
}

func (i *ItemRequestedState) requestItem() error {
    return fmt.Errorf("Item already requested")
}

func (i *ItemRequestedState) addItem(count int) error {
    return fmt.Errorf("Item Dispense in progress")
}

func (i *ItemRequestedState) insertMoney(money int) error {
    if money < i.vendingMachine.itemPrice {
        return fmt.Errorf("Inserted money is less. Please insert %d", i.vendingMa
    }
    fmt.Println("Money entered is ok")
    i.vendingMachine.setState(i.vendingMachine.hasMoney)
    return nil
}

func (i *ItemRequestedState) dispenseItem() error {
    return fmt.Errorf("Please insert money first")
}

```

## hasMoneyState.go: Конкретный интерфейс

```

package main

import "fmt"

type HasMoneyState struct {
    vendingMachine *VendingMachine
}

func (i *HasMoneyState) requestItem() error {
    return fmt.Errorf("Item dispense in progress")
}

func (i *HasMoneyState) addItem(count int) error {
    return fmt.Errorf("Item dispense in progress")
}

func (i *HasMoneyState) insertMoney(money int) error {
    return fmt.Errorf("Item out of stock")
}

func (i *HasMoneyState) dispenseItem() error {
    fmt.Println("Dispensing Item")
    i.vendingMachine.itemCount = i.vendingMachine.itemCount - 1
    if i.vendingMachine.itemCount == 0 {
        i.vendingMachine.setState(i.vendingMachine.noItem)
    } else {
        i.vendingMachine.setState(i.vendingMachine.hasItem)
    }
    return nil
}

```

## main.go: Клиентский код

```

package main

import (
    "fmt"
    "log"
)

func main() {
    vendingMachine := newVendingMachine(1, 10)

    err := vendingMachine.requestItem()
}

```

```

    if err != nil {
        log.Fatalf(err.Error())
    }

    err = vendingMachine.insertMoney(10)
    if err != nil {
        log.Fatalf(err.Error())
    }

    err = vendingMachine.dispenseItem()
    if err != nil {
        log.Fatalf(err.Error())
    }

    fmt.Println()

    err = vendingMachine.addItem(2)
    if err != nil {
        log.Fatalf(err.Error())
    }

    fmt.Println()

    err = vendingMachine.requestItem()
    if err != nil {
        log.Fatalf(err.Error())
    }

    err = vendingMachine.insertMoney(10)
    if err != nil {
        log.Fatalf(err.Error())
    }

    err = vendingMachine.dispenseItem()
    if err != nil {
        log.Fatalf(err.Error())
    }
}

```

## output.txt: Результат выполнения

```

Item requestd
Money entered is ok
Dispensing Item

```



Adding 2 items

Item requestd

Money entered is ok

Dispensing Item

По материалам: *Golang By Example*

ВЕРНУТЬСЯ НАЗАД

ЧИТАЕМ ДАЛЬШЕ

← Наблюдатель на Go

Стратегия на Go →

## Состояние на других языках программирования

