设计模式-适配器模式

参考

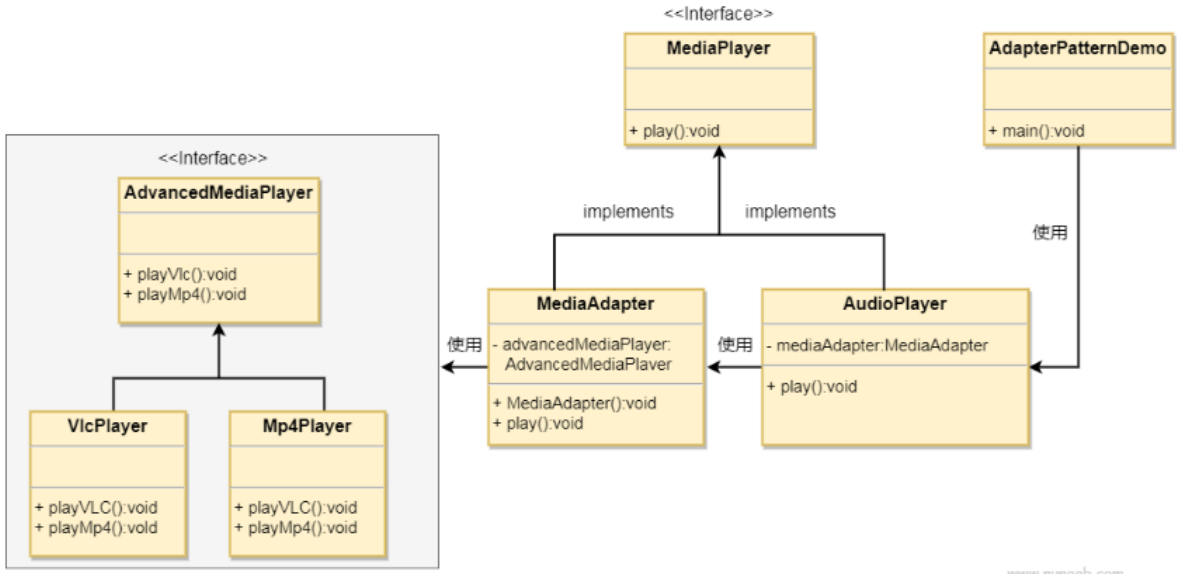
- Go设计模式(14)-适配器模式*程序员麻辣烫的博客-CSDN博客*go 适配器模式
- https://www.runoob.com/design-pattern/adapter-pattern.html

适配器模式是作为两个不兼容的接口之间的桥梁。这种类型的设计模式属于结构型模式，它结合了两个独立接口的功能。这种模式设计到一个单一的类，该类负责加入独立的或不兼容的接口功能。我们通过下面的实例来掩饰适配器模式的使用。其中，音频播放器设备只能播放MP3文件，通过使用一个更高级的音频播放器来播放vlc和MP4。

使用场景

有机动地修改一个正常运行的系统的接口，应该考虑使用适配器模式。

Demo分析



> 分析：
>
> 我们有一个 `MediaPlayer` 接口和一个实现了对应接口的实体类 `AudioPlayer`,默认情况下 `AudioPlayer` 可以播放MP3格式的音频文件。我们同时有一个接口 `AdvancedMediaPlayer` 和实现了对应接口的实体类，该类可以播放vlc和MP4格式的文件。

Go实现

```go
package main

import "fmt"

//集成多种方法
type AdvanceMediaPlayer interface {
    PlayMp4(fileName string)
    PlayVlc(fileName string)
}

type VlcPlayer struct {
}

func (p *VlcPlayer) PlayMp4(fileName string) {
}
func (p *VlcPlayer) PlayVlc(fileName string) {
    fmt.Printf("Playing vlc file: %s\n", fileName)
}

type Mp4Player struct {
}

func (p *Mp4Player) PlayVlc(fileName string) {
}
func (p *Mp4Player) PlayMp4(fileName string) {
    fmt.Printf("Playing mp4 file: %s\n", fileName)
}

type MediaPlayer interface {
    Play(audioType string, fileName string)
}

type MediaAdapter struct {
    player AdvanceMediaPlayer
}

func (p *MediaAdapter) initPlayer(audioType string) {
    if audioType == "vlc" {
        p.player = &VlcPlayer{}
    } else if audioType == "mp4" {
        p.player = &Mp4Player{}
    }
}

func (p *MediaAdapter) Play(audioType string, fileName string) {
    if audioType == "vlc" {
        p.player.PlayVlc(fileName)
    } else if audioType == "mp4" {
        p.player.PlayMp4(fileName)
    }
}

type AudioPlayer struct {
}
```

```go
func (p *AudioPlayer) Play(audioType string, fileName string) {
    if audioType == "mp3" {
        fmt.Printf("Playing mp3 file: %s\n", fileName)
    } else if audioType == "mp4" || audioType == "vlc" {
        var mediaAdapter MediaAdapter
        mediaAdapter.initPlayer(audioType)
        mediaAdapter.Play(audioType, fileName)
    } else {
        fmt.Printf("Invalid media type: %s\n", audioType)
    }
}

func test() {
    audioPlayer := &AudioPlayer{}
    audioPlayer.Play("mp3", "beyond the horizon.mp3")
    audioPlayer.Play("mp4", "alone")
    audioPlayer.Play("vlc", "I want it that way")
}

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

输出

```
Playing mp3 file: beyond the horizon.mp3
Playing mp4 file: alone
Playing vlc file: I want it that way
```

Python实现

```python
class AdvanceMediaPlayer:
  """
    ## 播放器抽象类
  """
  def play(self, fileName):
    pass

class VlcPlayer(AdvanceMediaPlayer):
  """
    ## Vlc播放器具体实现
  """
  def play(self, fileName):
    print(f"Playing vlc {fileName}")

class Mp4Player(AdvanceMediaPlayer):
  """
    ## MP4播放器具体实现
  """
  def play(self, fileName):
    print(f"Playing mp4 {fileName}")

class MediaPlayer(object):
  def Play(self):
```

```
    pass
class MediaAdapter(MediaPlayer):
  def __init__(self):
    self.player = None
  def MediaAdapter(self, audioType):
    if audioType == "vlc":
      self.player = VlcPlayer()
    elif audioType == "mp4":
      self.player = Mp4Player()
  def play(self, fileName):
    self.player.play(fileName)

class AudioPlayer(MediaPlayer):
  def play(self, audioType, fileName):
    if audioType == "mp3":
      print(f"Playing mp3 {fileName}")
    elif audioType == "mp4" or audioType == "vlc":
      player = MediaAdapter()
      player.MediaAdapter(audioType)
      player.play(fileName)

def test():
  player = AudioPlayer()
  player.play("mp3", "beyond the horizon.mp3")
  player.play("mp4", "alone.mp4")
  player.play("vlc", "far far away.vlc")

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

小结

适配器模式简单好用，用对了场景能够极大提高扩展和优雅性。