设计模式-迭代器模式

参考

- Go设计模式(23)-迭代器模式_程序员麻辣烫的博客-CSDN博客
- https://www.runoob.com/design-pattern/iterator-pattern.html

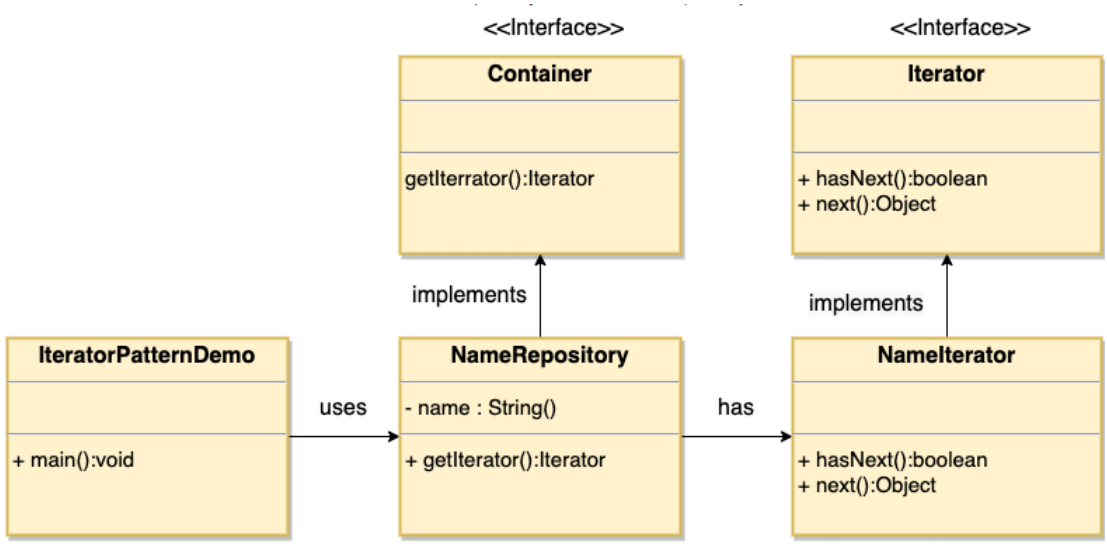
---

迭代器模式是JAVA和.Net编程环境中非常常用的设计模式。这种模式用于顺序访问集合对象的元素，不需要知道集合对象的底层原理。

使用场景

- 访问一个聚合对象的内容而无需暴露它的内部表示
- 需要为聚合对象提供多种遍历方式
- 为遍历不同的聚合结构提供一个统一的接口

Demo分析



> 分析：
>
> 我们将创建一个虚数导航方法的Iterator接口和一个返回迭代器的Container接口。实现了Container接口的实体类将负责实现Iterator接口，我们使用NamesRepository来打印出NamesRepository中存储为集合的Names。

Go实现

```go
package main

import (
    "errors"
)

/**
* 迭代器接口
 */
type Iterator interface {
    hasNext() bool
    next() (string, error)
}

type Container interface {
    getIterator() Iterator
}

// 内部实现迭代器
type NameRepository struct {
    Names []string
}

func (r *NameRepository) getIterator() Iterator {
    return &NameIterator{
        Index: -1,
        Names: r.Names,
    }
}

type NameIterator struct {
    Index int
    Names []string
}

func (n *NameIterator) hasNext() bool {
    return n.Index < len(n.Names)-1
}

func (n *NameIterator) next() (string, error) {
    if n.hasNext() {
        n.Index++
        return n.Names[n.Index], nil
    }
    return "", errors.New("no next")
}

func IteratorPatternDemo() {
    names := []string{"yyy", "cy", "ly", "cc"}
    nameRepository := &NameRepository{Names: names}
    iterator := nameRepository.getIterator()
    for {
        if name, err := iterator.next(); err == nil {
            println(name)
        } else {
```

```
            break
        }
    }
}

func main() {
    IteratorPatternDemo()
}
```

输出

```
yyy
cy
ly
cc
```

Python实现

```
class Iterator:
  def hasNext(self):
    pass
  def next(self):
    pass

class Container:
  def getIterator(self):
    pass

class NameRepository(Container):
  def __init__(self, names):
    self.names = names
  def getIterator(self):
    return NameIterator(self.names)

class NameIterator(Iterator):
  def __init__(self, names):
    self.names = names
    self.index = -1
  def hasNext(self):
    return self.index < len(self.names) - 1
  def next(self):
    if self.hasNext():
      self.index += 1
      return self.names[self.index]
    return None

def IteratorPatternDemo():
  nameRepository = NameRepository(['John', 'Jack', 'Camila', 'Ingrid'])
  iter = nameRepository.getIterator()
  while True:
    if not iter.hasNext():
      break
    name = iter.next()
```

```
    print(name)

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

输出

```
John
Jack
Camila
Ingrid
```

小结

迭代器模式包含了迭代对象还有一个容器对象（原始的对象其中包含了可迭代的数据）。通过封装`getIterator`以及`next`和`hasNext`方法可以方便我们完成对一个可迭代对象进行遍历。