



# 第十二章 迭代器模式

任课教师：武永亮

wuyongliang@edu2act.org



## ▣ 上节回顾

## ■ 课程内容

- 环境及问题
- 迭代器模式详解
- 迭代器模式实现
- 扩展练习

## ■ 课程内容

- 环境及问题
- 迭代器模式详解
- 迭代器模式实现
- 扩展练习

## ■ 环境

- 在软件构建过程中，集合对象内部结构常常变化各异。但对于这些集合对象，我们希望在暴露其内部结构的同时，可以让外部客户端代码透明地访问其中包含的元素；同时这种“透明遍历”也为“同一种算法在多种集合对象上进行操作”提供了可能。

## ■ 环境

■ 站起来换频道.....



## ■环境

■坐下来换频道.....



## ■环境

■好多遥控。。。。





## ■环境

### ■先进的时代——万能遥控器



## ■ 应用场景

- 访问一个集合对象的内容，而无需暴露它的内部表示。
- 支持对集合对象的多种遍历。
- 为遍历不同的集合结构提供一个统一的接口(即, 支持多态迭代)。

迭代器模式 ( Iterator )

## ■ 课程内容

- 环境及问题
- 迭代器模式详解
- 迭代器模式实现
- 扩展练习

## ■ 迭代器模式(Iterator)。

- 又叫做游标 ( Cursor ) 模式
- 提供一种方法访问一集合对象中各个元素，而又不需暴露该对象的内部细节。
- 迭代器模式是为容器 ( 或者集合 ) 而生。

## ■ 迭代器模式(Iterator)包括**四种角色**。

### ■ **抽象集合**

- 一个接口，规定了具体集合需要实现的操作。

### ■ **具体集合**

- 具体的集合按照一定的结构存储对象。
- 具体集合应该有一个方法，该方法返回一个针对该集合的具体迭代器。

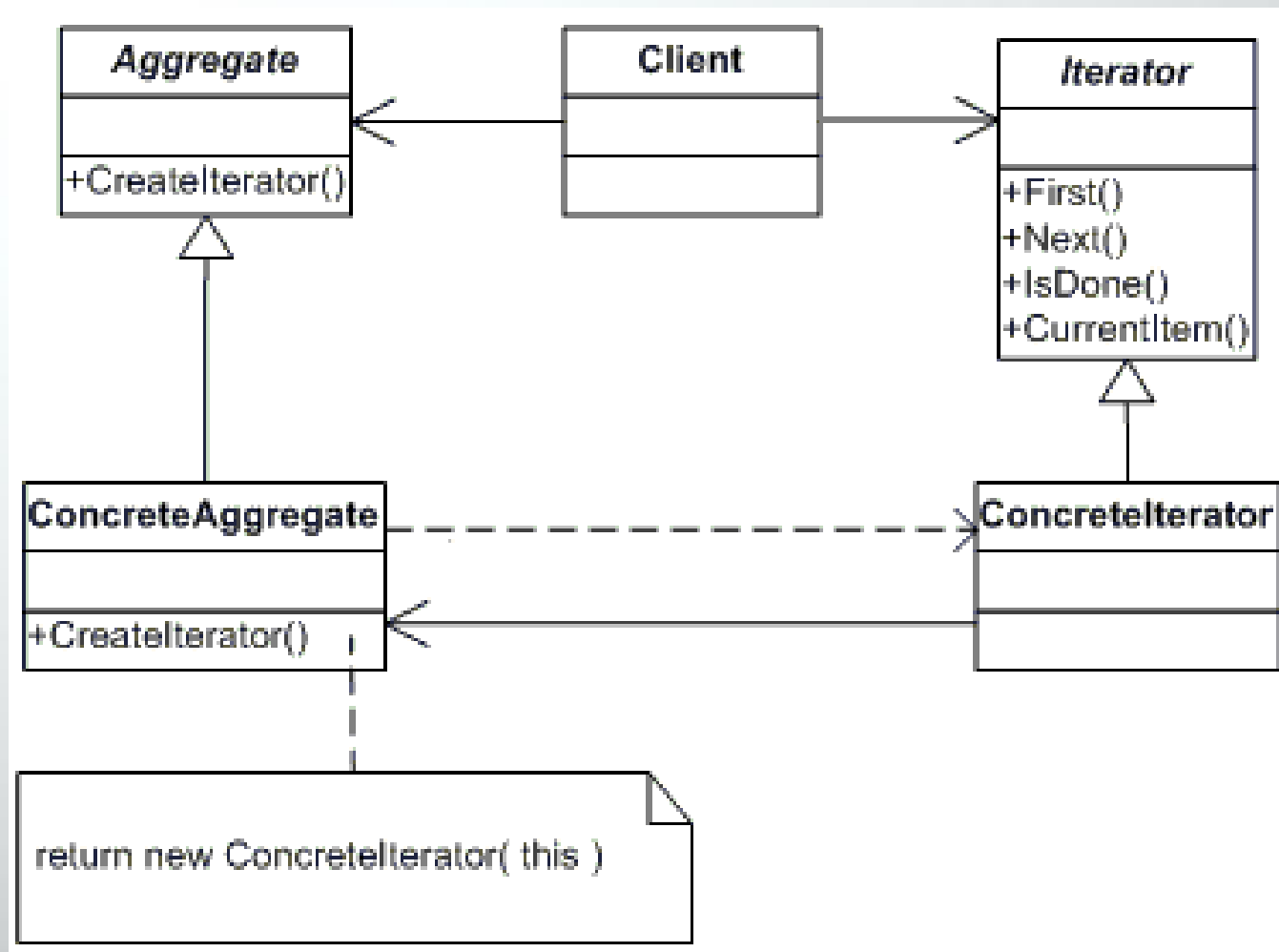
### ■ **抽象迭代器**

- 一个接口，规定了遍历具体集合的方法，比如 next() 方法。

### ■ **具体迭代器**

- 实现了迭代器接口的类的实例。

## ■ 迭代器模式设计类图详解



## ■ 课程内容

- 环境及问题
- 迭代器模式详解
- 迭代器模式实现
- 扩展练习

## ■ 迭代器模式实现





## ■ 迭代器模式实现

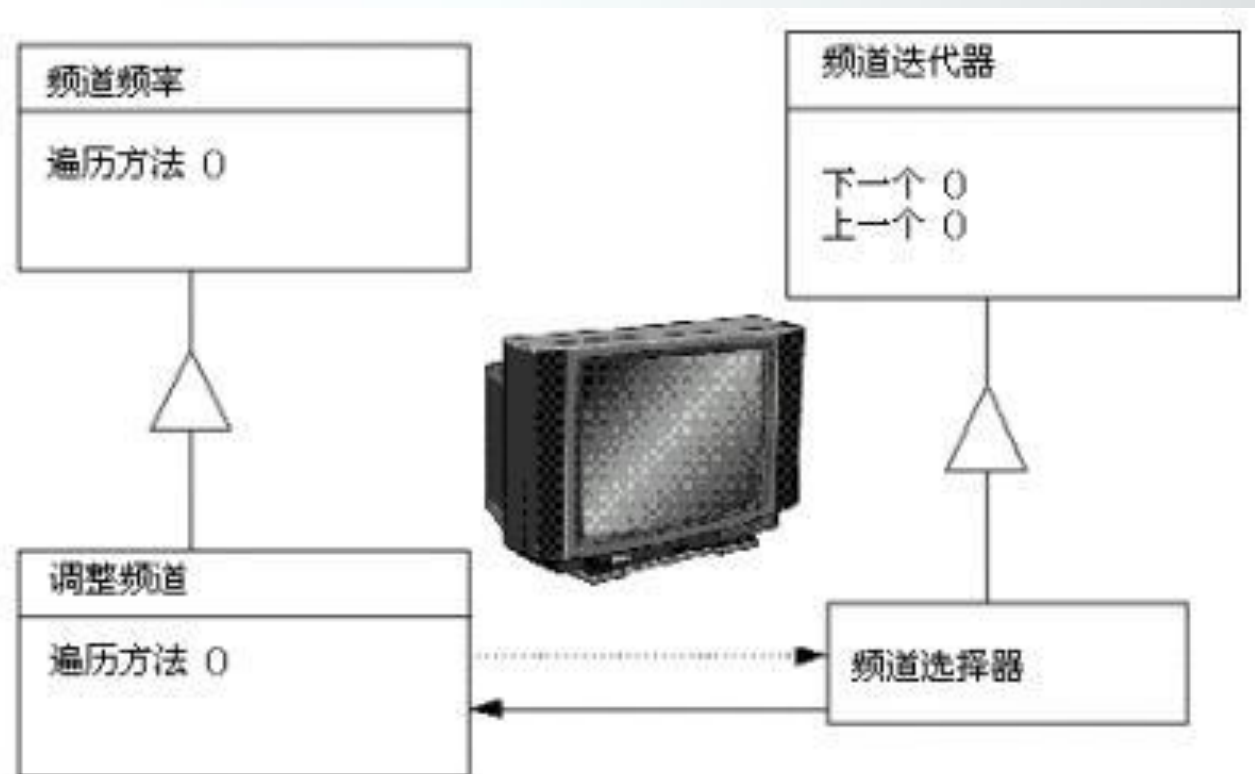


图 16: 使用选频器作例子的迭代模式对象图

## ■ 迭代器模式实现代码

```
public class Item{  
    private String name;  
    public Item(String aName){  
        name = aName;  
    }  
    public String getName(){  
        return name;  
    }  
}
```

```
public interface Iterator{  
    public Item first();  
    public Item next();  
    public boolean isDone();  
    public Item currentItem();  
}
```

```
public interface Television{  
    public Iterator createIterator();  
    public Vector getChannel();  
}
```



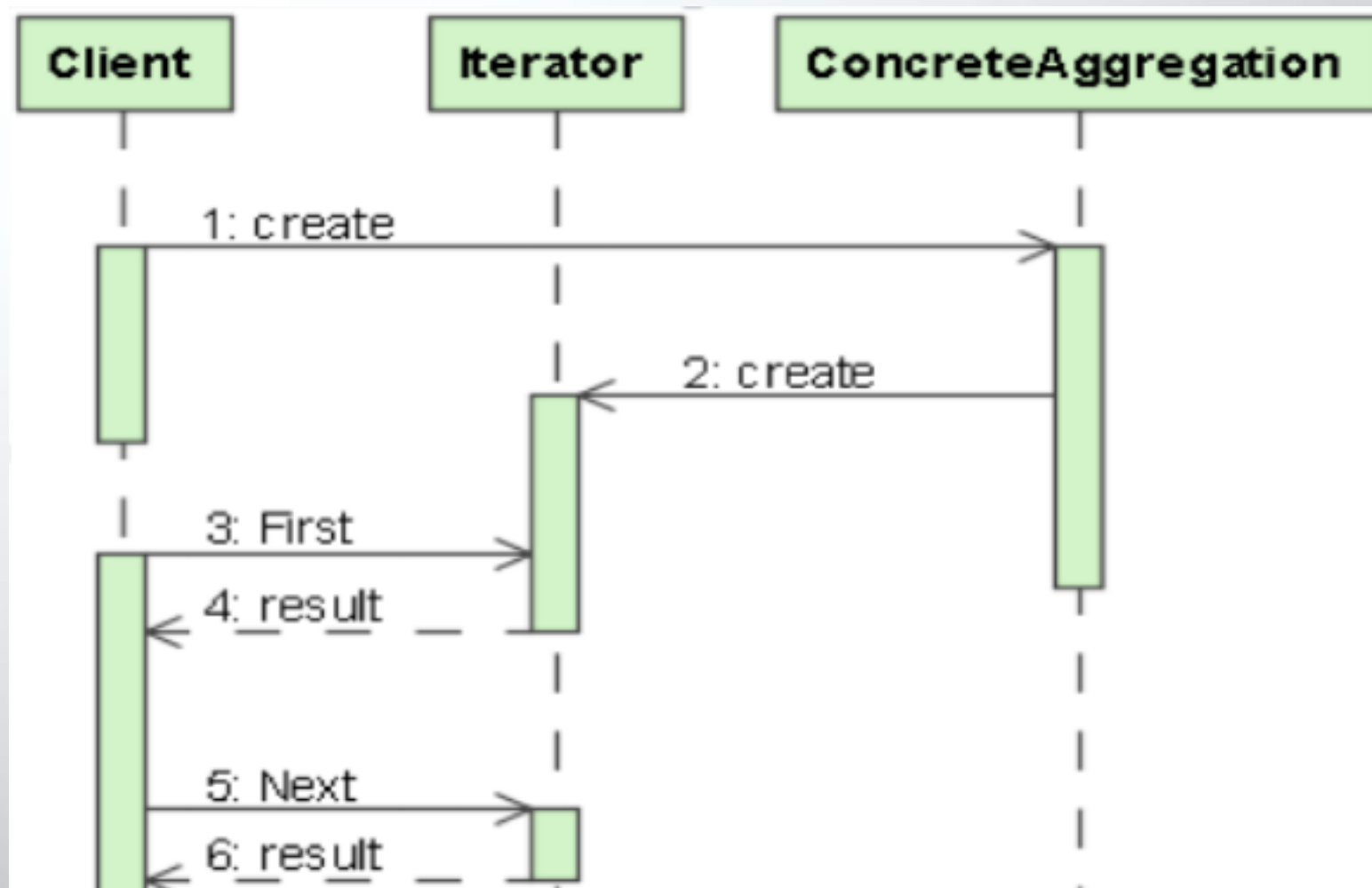
```
public class HaierTV implements Television{  
    private Vector channel;  
  
    public HaierTV(){  
        channel = new Vector();  
        channel.addElement(new Item("channel 1"));  
        channel.addElement(new Item("channel 2"));  
        channel.addElement(new Item("channel 3"));  
        channel.addElement(new Item("channel 4"));  
        channel.addElement(new Item("channel 5"));  
        channel.addElement(new Item("channel 6"));  
        channel.addElement(new Item("channel 7"));  
    }  
  
    public Vector getChannel(){  
        return channel;  
    }  
  
    public Iterator createIterator(){  
        return new Controller(channel);  
    }  
}
```

```
public class Controller implements Iterator{  
  
    private int current =0;  
    Vector channel;  
  
    public Controller(Vector v){  
        channel = v;  
    }  
  
    public Item first(){  
        current = 0;  
        return (Item)channel.get(current);  
    }  
  
    public Item next(){  
        current ++;  
        return (Item)channel.get(current);  
    }  
  
    public Item currentItem(){  
        return (Item)channel.get(current);  
    }  
  
    public boolean isDone(){  
        return current>= channel.size()-1;  
    }  
}
```

## ■ 迭代器模式实现代码

```
public class Client{  
    public static void main(String[] args){  
        Television tv = new HaierTV();  
        Iterator it = tv.createIterator();  
        System.out.println(it.first().getName());  
        while(!it.isDone()){  
            System.out.println(it.next().getName());  
        }  
    }  
}
```

## ■ 迭代器模式实现



## ■ 课程内容

- 环境及问题
- 迭代器模式详解
- 迭代器模式实现
- 扩展练习

## ■ 扩展案例

■ 超市顾客排队结账，收银员一个个顺序结账





## 案例





## 扩展说明

- 其实.Net框架已经准备好了迭代器接口，只需要实现接口就行了。
  - IEnumerator 支持对非泛型集合的简单迭代
- Java中的Collection，List、Set、Map等，这些集合都有自己的迭代器，通用的迭代器接口Iterator。

## 扩展说明

### ■ 优点：

- 简化了遍历方式
- 可以提供多种遍历方式（正序遍历，倒序遍历.....）
- 封装性良好

### ■ 缺点：

- 对于比较简单的遍历（像数组或者有序列表），使用迭代器方式遍历较为繁琐，大家可能都有感觉，像ArrayList，我们宁可愿意使用for循环和get方法来遍历集合

## ■ 小结

- 迭代器模式是**与集合共生共死**的
- 大多数语言在实现容器的时候都给提供了迭代器，假如我们要实现一个这样的新的容器，当然也需要引入对应迭代器的实现。

**Thank You , 谢谢 !**