深圳大学
Shenzhen University

第13-2讲

迭代器模式

软件体系结构与设计模式

Software Architecture & Design Pattern

深圳大学计算机与软件学院

主要内容

- 迭代器模式动机与定义
- 迭代器模式结构与分析
- 迭代器模式实例与解析
- 迭代器模式效果与应用

迭代器模式动机

- **电视机遥控器与电视机示意图**

遍历频道

电视机遥控器

电视机

（电视频道的集合）

迭代器模式动机

- 电视机 ← → 存储电视频道的集合 ← → 聚合类(Aggregate Classes)
- 电视机遥控器 ← → 操作电视频道 ← → 迭代器(Iterator)
- 访问一个聚合对象中的元素但又不需要暴露它的内部结构 ← → 迭代器模式

迭代器模式定义

- 对象行为型模式

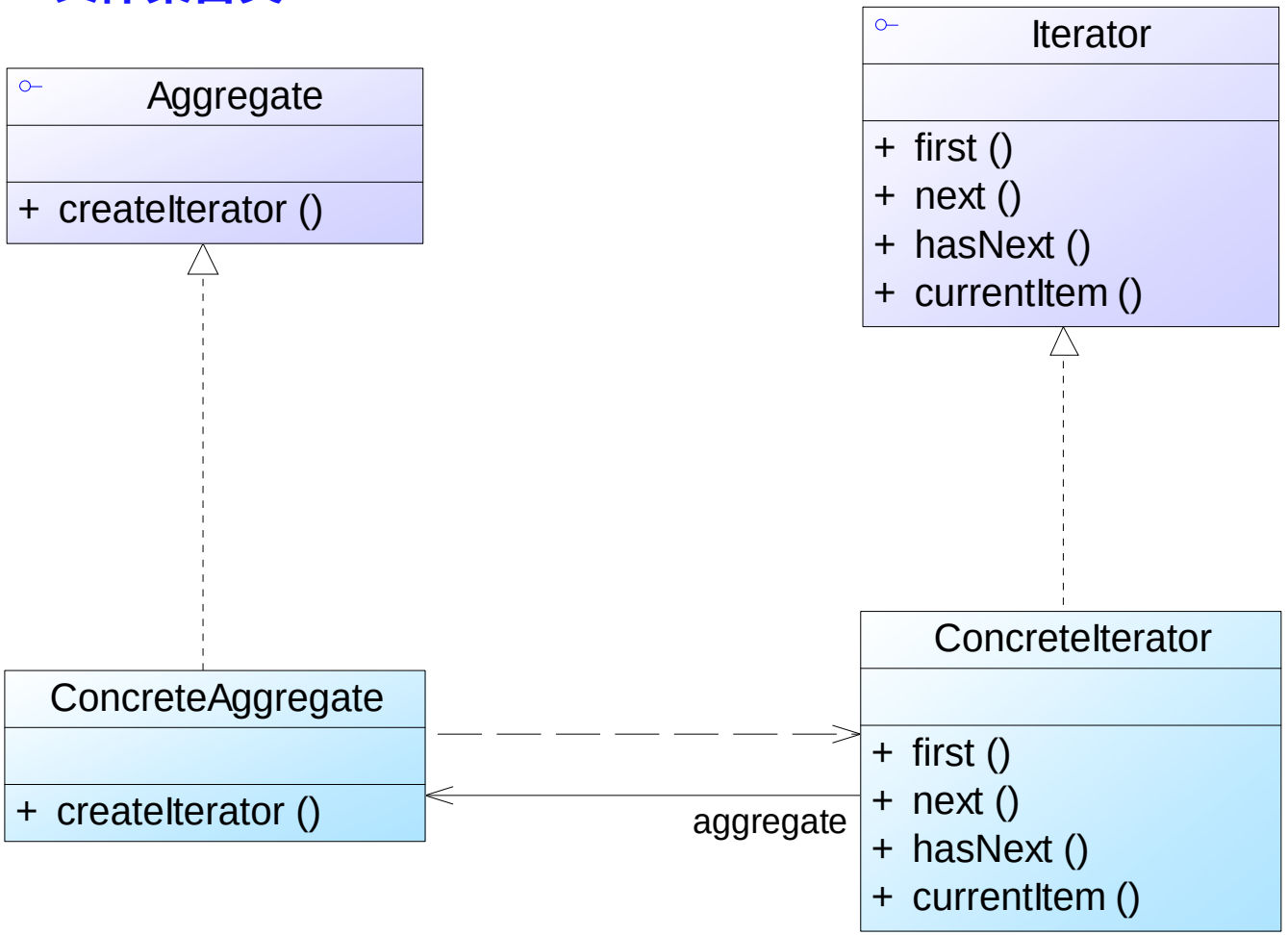
迭代器模式：提供一种方法顺序访问一个聚合对象中各个元素且不用暴露该对象的内部表示。

Iterator Pattern: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

- 又名游标(Cursor)模式
- 通过引入迭代器，客户端无须了解聚合对象的内部结构即可实现对聚合对象中成员的遍历，还可以根据需要很方便地增加新的遍历方式

迭代器模式结构

- **迭代器模式包含如下角色：** **Iterator: 抽象迭代器**
- **ConcreteIterator: 具体迭代器 ， Aggregate: 抽象聚合类**
- **ConcreteAggregate: 具体聚合类**



迭代器模式分析

- 聚合对象的两个职责：
  - 存储数据，聚合对象的基本职责
  - 遍历数据，既是可变化的，又是可分离的
- 将遍历数据的行为从聚合对象中分离出来，封装在迭代器对象中
- 由迭代器来提供遍历聚合对象内部数据的行为，简化聚合对象的设计，更符合单一职责原则

迭代器模式分析

- 抽象迭代器示例代码：

```
public interface Iterator {
    public void first();                //将游标指向第一个元素
    public void next();                 //将游标指向下一个元素
    public boolean hasNext();           //判断是否存在下一个元素
    public Object currentItem();        //获取游标指向的当前元素
}
```

迭代器模式分析

- 具体迭代器示例代码：

```
public class ConcreteIterator implements Iterator {
    private ConcreteAggregate objects; //维持一个对具体聚合对象的引用，以便于访问存储在聚合对象中的数据
    private int cursor;  //定义一个游标，用于记录当前访问位置
    public ConcreteIterator(ConcreteAggregate objects) {
        this.objects=objects;
    }

    public void first() {  ......  }

    public void next() {  ......  }

    public boolean hasNext() {  ......  }

    public Object currentItem() {  ......  }
}
```

迭代器模式分析

- **抽象聚合类示例代码：**

```
public interface Aggregate {
    Iterator createIterator();
}
```

迭代器模式分析

- **具体聚合类示例代码：**

```
public class ConcreteAggregate implements Aggregate {
    ......
    public Iterator createIterator() {
        return new ConcreteIterator(this);
    }
    ......
}
```

迭代器模式分析

- JDK内置迭代器：
  - java.util.Collection
  - java.util.Iterator

```
package java.util;

public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove();
}
```

```
package java.util;

public interface Collection<E> extends Iterable<E> {
    ……
    boolean add(Object c);
    boolean addAll(Collection c);
    boolean remove(Object o);
    boolean removeAll(Collection c);
    boolean remainAll(Collection c);
    Iterator iterator();
    ……
}
```

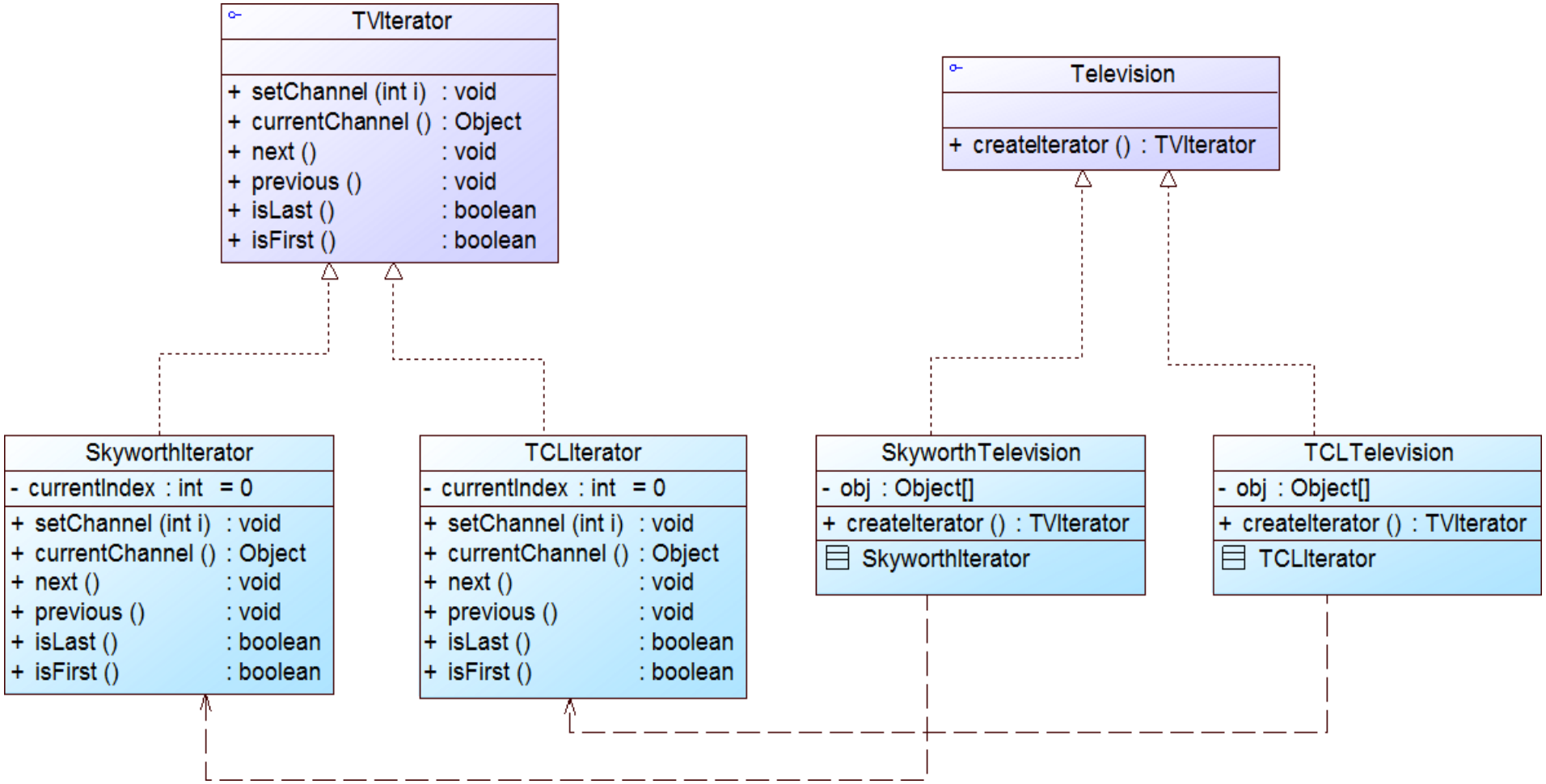
```
//测试代码
……
public static void process(Collection c) {
    Iterator i = c.iterator(); //创建迭代器对象

    //通过迭代器遍历聚合对象
    while(i.hasNext()) {
        System.out.println(i.next().toString());
    }
}
……
```

迭代器模式实例

- **电视机遥控器：实例说明**
  - 电视机遥控器就是一个迭代器的实例，通过它可以实现对电视机频道集合的遍历操作，本实例我们将模拟电视机遥控器的实现。

迭代器模式实例与解析

- **电视机遥控器：参考类图**



迭代器模式实例与解析

- **迭代器模式实例**
  - **电视机遥控器：参考代码**
- DesignPatterns之iterator包

Television.java

```
1  package iterator;
2
3  public interface Television
4  {
5      TVIterator createIterator();
6  }
```

TVIterator.java

```
1  package iterator;
2
3  public interface TVIterator
4  {
5      void setChannel(int i);
6      void next();
7      void previous();
8      boolean isLast();
9      Object currentChannel();
10     boolean isFirst();
11 }
```

TCLTelevision.java

```
3  public class TCLTelevision implements Television
4  {
5      private Object[] obj={"湖南卫视","北京卫视","上海卫视","湖北卫视","黑龙江卫视"};
6      public TVIterator createIterator()
7      {
8          return new TCLIterator();
9      }
10
11     class TCLIterator implements TVIterator
12     {
13         private int currentIndex=0;
14
15         public void next()
16         {
17             if(currentIndex<obj.length)
18             {
19                 currentIndex++;
20             }
21         }
22
23         public void previous()
24         {
25             if(currentIndex>0)
26             {
27                 currentIndex--;
28             }
```

TCLTelevision.java

```
29          }
30
31          public void setChannel(int i)
32          {
33              currentIndex=i;
34          }
35
36
37          public Object currentChannel()
38          {
39              return obj[currentIndex];
40          }
41
42          public boolean isLast()
43          {
44              return currentIndex==obj.length;
45          }
46
47          public boolean isFirst()
48          {
49              return currentIndex==0;
50          }
51      }
52  }
```

SkyworthTelevision.java

```
3  public class SkyworthTelevision implements Television
4  {
5      private Object[] obj={"CCTV-1","CCTV-2","CCTV-3","CCTV-4","CCTV-5",
6              "CCTV-6","CCTV-7","CCTV-8"};
7      public TVIterator createIterator()
8      {
9          return new SkyworthIterator();
10     }
11
12     private class SkyworthIterator implements TVIterator
13     {
14         private int currentIndex=0;
15
16         public void next()
17         {
18             if(currentIndex<obj.length)
19             {
20                 currentIndex++;
21             }
22         }
23
24         public void previous()
25         {
26             if(currentIndex>0)
27             {
28                 currentIndex--;
29             }
```

SkyworthTelevision.java

```
30          }
31
32          public void setChannel(int i)
33          {
34              currentIndex=i;
35          }
36
37
38          public Object currentChannel()
39          {
40              return obj[currentIndex];
41          }
42
43          public boolean isLast()
44          {
45              return currentIndex==obj.length;
46          }
47
48          public boolean isFirst()
49          {
50              return currentIndex==0;
51          }
52      }
53  }
```

XMLUtil.java

```java
public class XMLUtil
{
//该方法用于从XML配置文件中提取具体类类名，并返回一个实例对象
    public static Object getBean()
    {
        try
        {
            //创建文档对象
            DocumentBuilderFactory dFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = dFactory.newDocumentBuilder();
            Document doc;
            doc = builder.parse(new File("Iterator"
                    + "config.xml"));

            //获取包含类名的文本节点
            NodeList nl = doc.getElementsByTagName("className");
            Node classNode=nl.item(0).getFirstChild();
            String cName=classNode.getNodeValue();

            //通过类名生成实例对象并将其返回
            Class c=Class.forName(cName);
            Object obj=c.newInstance();
            return obj;
           }
            catch(Exception e)
            {
                e.printStackTrace();
```

XMLUtil.java

```java
                return null;
            }
        }
}
```

Client.java

```java
public class Client
{
    public static void display(Television tv)
    {
        TVIterator i=tv.createIterator();
        System.out.println("电视机频道: ");
        while(!i.isLast())
        {
            System.out.println(i.currentChannel().toString());
            i.next();
        }
    }

    public static void reverseDisplay(Television tv)
    {
        TVIterator i=tv.createIterator();
        i.setChannel(5);
        System.out.println("逆向遍历电视机频道: ");
        while(!i.isFirst())
        {
            i.previous();
            System.out.println(i.currentChannel().toString());
        }
    }
```

迭代器模式实例与解析

Client.java

```
27
28      public static void main(String a[])
29      {
30          Television tv;
31          tv=(Television)XMLUtil.getBean();
32          display(tv);
33          System.out.println("--------------------------");
34          reverseDisplay(tv);
35      }
36  }
```

迭代器模式效果与应用

- **迭代器模式优点：**
  - 支持以不同的方式遍历一个聚合对象，在同一个聚合对象上可以定义多种遍历方式
  - 简化了聚合类
  - 由于引入了抽象层，增加新的聚合类和迭代器类都很方便，无须修改原有代码，符合开闭原则

迭代器模式效果与应用

- **迭代器模式缺点：**
  - 在增加新的聚合类时需要对应地增加新的迭代器类，类的个数成对增加，这在一定程度上增加了系统的复杂性
  - 抽象迭代器的设计难度较大，需要充分考虑到系统将来的扩展。在自定义迭代器时，创建一个考虑全面的抽象迭代器并不是一件很容易的事情

迭代器模式效果与应用

- **在以下情况下可以使用迭代器模式：**
  - 访问一个聚合对象的内容而无须暴露它的内部表示
  - 需要为一个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，在该接口的实现类中为不同的聚合结构提供不同的遍历方式，而客户端可以一致性地操作该接口

谢谢