Java 设计模式之迭代器模式（十五）

一、前言

本篇主题为行为型模式中的第三个模式–迭代器模式。上篇 Java 设计模式主题为《Java 设计模式之命令模式（十四）》。

二、简单介绍

2.1 定义

迭代器模式是行为模式之一，它把对容器中包含的内部对象的访问委让给外部类（此外部类是指非自身的类），使用 Iterator（遍历）按顺序进行遍历访问的设计模式。

2.2 参与角色

1. 迭代器接口（Iterator）：该接口必须定义实现迭代功能的最小定义方法集。比如提供 hasNext() 和 next() 方法。
2. 迭代器实现类（ConcreteIterator）：迭代器接口 Iterator 的实现类。可以根据具体情况加以实现。
3. 容器接口（Aggregate）：定义基本功能以及提供类似 Iterator iterator() 的方法。

2.3 应用场景

1. 访问一个聚合对象的内容而无需暴露它的内部表示。
2. 支持对聚合对象的多种遍历。
3. 为遍历不同的聚合结构提供一个统一的接口。

三、实现方式

我们以坐车买票为例，乘客上车后，由售票员对每个乘客进行售票操作（相当于遍历操作）。

售票员接口（Iterator）：

```
public interface Conductor {

    /**
     * 将游标指向第一个元素
     */
    void first();

    /**
     * 将游标指向下一个元素
     */
    void next();

    /**
     * 判断是否存在下一个元素
     * @return
     */
    boolean hasNext();

    /**
     * 判断是否是第一个元素
     * @return
     */
    boolean isFirst();

    /**
     * 判断是否是最后一个元素
     * @return
     */
    boolean isLast();

    /**
     * 获取当前游标指向的对象
     * @return
     */
    Object getCurrentObj();
}
```

巴士接口（Aggregate）：

```
public interface Bus {

    public void getOn(Object obj);

    public void getOff(Object obj);

    public Conductor conductor();
}
```

机场大巴（Aggregate 实现类）：

```
public class SkyBus implements Bus{

    private List<Object> list = new ArrayList<Object>();

    @Override
    public void getOn(Object obj) {
        this.list.add(obj);
    }

    @Override
    public void getOff(Object obj) {
        this.list.remove(obj);
    }

    @Override
    public Conductor conductor() {
        return new BusConductor();
    }

    public List<Object> getList() {
        return list;
    }

    public void setList(List<Object> list) {
        this.list = list;
    }

    // 使用内部类定义迭代器，可以直接使用外部类的属性
    private class BusConductor implements Conductor {

        /**
         * 定义游标用于记录遍历时的位置
         */
        private int cursor;

        @Override
        public void first() {
            cursor = 0;
        }
```

```
        @Override
        public Object getCurrentObj() {
            return list.get(cursor);
        }

        @Override
        public boolean hasNext() {
            return cursor < list.size();
        }

        @Override
        public boolean isFirst() {
            return cursor == 0;
        }

        @Override
        public boolean isLast() {
            return cursor == (list.size() - 1);
        }

        @Override
        public void next() {
            if (cursor < list.size()) {
                cursor++;
            }
        }

    }
}
```

在 Aggregate 实现类中定义内部类实现 Iterator 接口。

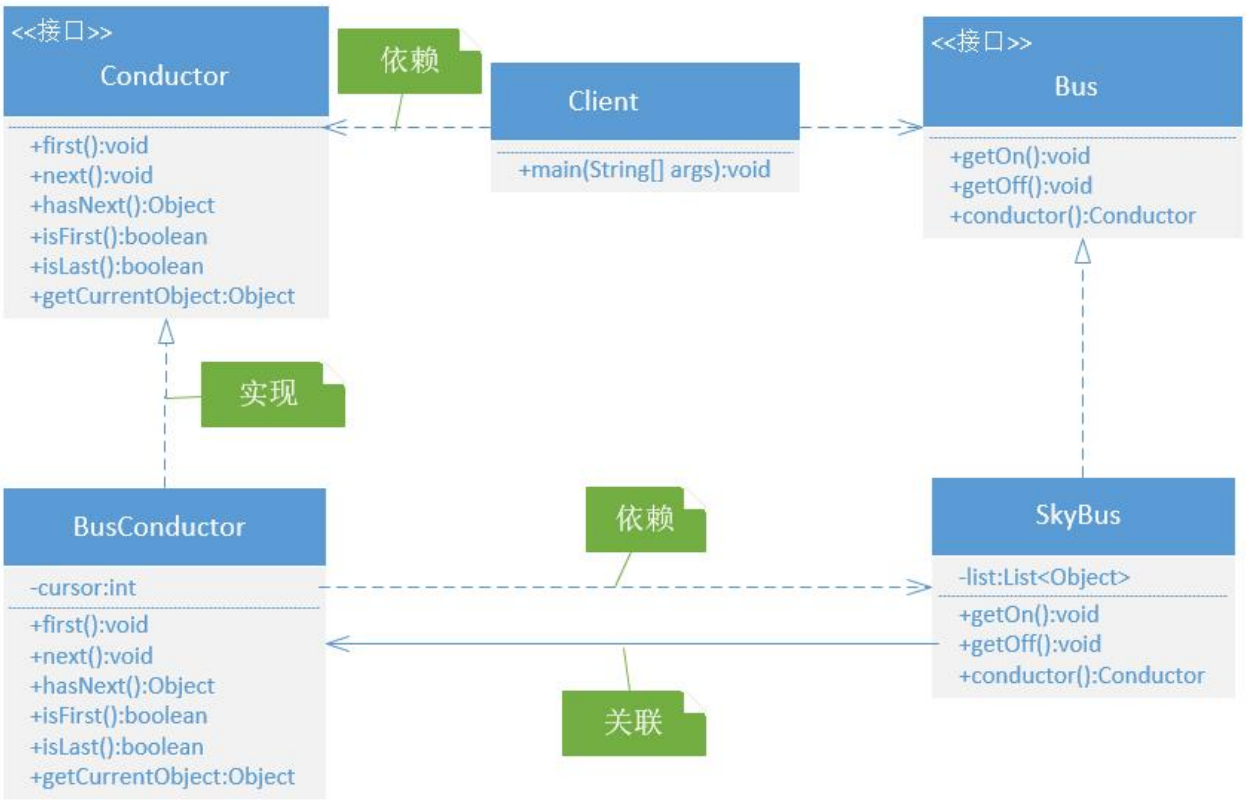
客户端：

```
public class Client {

    public static void main(String[] args) {
        Bus bus = new SkyBus();
        bus.getOn("张三");
        bus.getOn("李四");
        bus.getOn("王五");

        Conductor iter = bus.conductor();
        while(iter.hasNext()){
            System.out.println(iter.getCurrentObj() + "乘客，请买票！");
            iter.next();
        }
    }
}
```

打印结果：

```
张三乘客，请买票！
李四乘客，请买票！
王五乘客，请买票！
```

UML 类图表示如下：


<<接口>>
Conductor
+first():void
+next():void
+hasNext():Object
+isFirst():boolean
+isLast():boolean
+getCurrentObject:Object
依赖
Client
+main(String[] args):void
<<接口>>
Bus
+getOn():void
+getOff():void
+conductor():Conductor
实现
BusConductor
-cursor:int
+first():void
+next():void
+hasNext():Object
+isFirst():boolean
+isLast():boolean
+getCurrentObject:Object
依赖
SkyBus
-list:List<Object>
+getOn():void
+getOff():void
+conductor():Conductor
关联