853769620

（1）大白话讲设计模式：先用大白话，纯纯的大白话，非常的接地气，不能有任何一丁点高深莫测的感觉。快速领会各个模式的一个要领，每个模式长什么样子？如果不用这个模式会有什么问题？如果用了这个模式会有什么好处？一般在企业开发中这个模式是怎么用的？

（2）在电商系统里，将我们讲解的所有这些设计模式，全部在真实复杂的电商业务中去深入的使用，结合业务来设计这个模式如何运用到业务中，让我们基于设计模式开发出高内聚、低耦合、高度可扩展、维护成本低、逻辑清晰、易读易懂的代码

（3）如果你未来要将设计模式转换成你自己的本领，在你做项目的时候，可以随心所欲的使用设计模式，必须得依靠咱们的第二点，你只有在一个真实的复杂业务系统中，实践过设计模式的运用，体会到这个东西是怎么玩儿的，这么玩儿以后好处是什么。设计模式才能转化为你自己的东西，然后以后你在自己开发的时候，可以随心所欲的使用设计模式

（4）如果你出去面试，人家考察你设计模式，此时，如果你干背设计模式的理论、定义、举个例子，这个是肯定不行的，你有没有设计模式运用和实践的功底，是否透彻理解了，是否融会贯通。你要结合你自己做过的复杂业务系统，给人家先讲一讲业务，然后说一下，如果不用模式来设计这个业务的代码实现，可能会导致代码成一坨屎，但是你当时合理的运用了设计模式，将代码设计的结构如何优良，可扩展性如何的好，维护性如何的好。

（5）设计模式，网上有很多的视频，还有很多的书，大话设计模式。。。但是他们都有一个问题，就是你学习的过程中，哪怕是给你举个例子，说这个模式是怎么怎么用的，但是那都是一些demo级的例子。就我所知，正儿八经的设计模式在纯复杂业务系统的项目中，综合性的设计模式项目实战，我一个都没看到。综合运用设计模式来开发了一个自动代码生成框架，但是那个有一个问题，模式的运用很多地方是生搬硬涛，不自然，业务，但是对于大部分同学来说，平时经历的还是那种有一定复杂度业务的系统。

iterator模式【迭代器模式：遍历相关的需求】

需求：给大家举个例子，现在假设，有一个教室类，里面包含了一堆学生，我现在要遍历教室里的学生，怎么来玩儿

【如果不用任何设计模式，直接去遍历一个类中的集合；一旦这个类中对集合的使用改版了，比如从数组 -> map，还有别的可能，你迭代的这块代码，就要改动】

public interface **Aggregate** {

   public abstract Iterator iterator();

}

public interface **Iterator** {

   public abstract boolean hasNext();

   public abstract Object next();

}

public class **Student** {

   private String name;

   public Student (String name) {

       this.name = name;

  }

   public String getName() {

       return name;

  }

}

public class **Classroom implements Aggregate** {

   private Student[] students;

   private int last = 0;

   public Classroom(int size) {

       this.students = new Student[size];

  }

   public Student getStudent(int index) {

       return students[index];

  }

   public void addStudent(Student student) {

       this.students[last] = student;

       last++;

  }

   public int getLength() {

       return last;

  }

   public Iterator iterator() {

       return new ClassroomIterator(this);

  }

}

public class **ClassroomIterator implements Iterator** {

   private Classroom classroom;

   private int index;

   public ClassroomIterator(Classroom classroom) {

       this.classroom = classroom;

       this.index = 0;

  }

   public boolean hasNext() {

       if(index < classroom.getLength()) {

           return true;

      } else {

           return false;

      }

  }

   public Object next() {

       Student student = classroom.getStudent(index);

       index++;

       return student;

  }

}

面向Iterator接口编程，无论底层的数据结构和迭代算法如何变化，调用者都不用修改代码

高内聚，低耦合，漂亮

其实一般很少自己写这个iterator模式的，一般都是在集合编程中使用，尤其是如果要对集合元素遍历过程中做插入删除操作，那就用iterator，这个我们会在本次项目阶段一来实践，体验JDK已经封装好的iterator模式，加深印象，如果要对某个类中的集合进行遍历，由那个集合类返回一个iterator回来，我们统一面向iterator迭代器接口来编程遍历，提高系统整体的可维护性，可扩展性

如果自己写iterator模式，一般是研发底层的框架，比如**提供某个数据给外部遍历，那么可以使用iterator模式自己封装迭代器**，以后阶段如果有机会，我们也会去实践