转载请注明链接：https://blog.csdn.net/feather_wch/article/details/78559517

> 本文以面试题的形式归纳总结，可以直接学习和背诵。
> 鸣谢: 本文基础部分归纳总结自《Head First 设计模式》

如果有帮助，请点个赞，万分感谢！

适配器模式

版本：2018/8/24-1(23:24)

介绍

1、适配器模式是什么?

> 1. 适配器模式-Adapter Pattern
> 2. 适配器模式：包装对象并且给他们新的职责，来使得它们的接口看起来像那些它们根本不是的东西。
> 3. 我们可以采用一种设计，可以将一个interface看成一个实现了很多接口(interface)的类(class)。

2、现实生活中的适配器模式

> 1. 欧洲交流电插座(三口)，美国电脑两口，这就需要一个适配器，将一个接口转为另一个接口。
> 2. 一般家庭，墙壁上有一个三口的插槽，机顶盒是两口插槽，这就需要一个拖线板来提供三口和两口的插槽给这些设备使用

定义

3、适配器模式的定义

> Adapter将一个类的接口转换为另一个客户需要的接口，这样能使得原本接口不兼容的类能够一起工作。

作用

4、适配器模式有什么用?

> 1. 将一个接口转换为客户需要的接口。
> 2. 使得客户和供应商各自的代码都不需要变换，只要一个中间作用的适配器即可。

设计原则

5、适配器模式是如何遵循原则的?

> 1. 提供适配器类，将所有的改变封装在一个类中。
> 2. 用可改变的接口将被适配者包裹起来---这样好处就是，任何被适配者的子类(subclass)都可以配合adapter使用。

实例

6、适配器模式实例: 火鸡变鸭子

> 1-鸭子

```
public interface Duke {
        public void fly();
        public void quack();
}
public class BlackDuke implements Duke{
        public void fly() {
                System.out.println("flying a long distance");
        }
        public void quack() {
                System.out.println("quack");
        }
}
```

> 2-火鸡

```
public interface Turkey {
        public void fly();
        public void gobble();
}
public class WildTurkey implements Turkey{
        public void fly() {
                System.out.println("fly a short distance");
        }
        public void gobble() {
                System.out.println("gobble");
        }
}
```

> 3-适配器: 实现鸭子接口，将火鸡行为转换为鸭子的行为。

```
// 火鸡飞行的很短，鸭子要长一些，就可以让火鸡多飞几次，就和鸭子飞行的效果一致。
public class DukeAdapter implements Duke{
        Turkey turkey;
        public DukeAdapter(Turkey turkey) {
                this.turkey = turkey;
        }
        public void fly() {
                for(int i = 0; i < 5; i++) {
                        turkey.fly();
                }
        }
        public void quack() {
                turkey.gobble();
        }
}
```

> 4-测试

```
public class DukeTest {
        public static void main(String[] args) {
                Duke blackDuke = new BlackDuke(); //黑鸭子
                Duke dukeFromTurkey = new DukeAdapter(new WildTurkey());//真实鸭子

                System.out.println("This is Duke:");
                testDuke(blackDuke);
                System.out.println("This is Duke from turkey:");
                testDuke(dukeFromTurkey);
        }
        public static void testDuke(Duke duke) {
                duke.fly();
                duke.quack();
        }
}
```

两种适配器

object适配器

7、对象适配器是什么？如何实现?

> 1. 使用组合composition实现，上文实现的就是对象适配器。
> 2. Adapter继承自Duke，实现了Duke的方法，且参数传入火鸡，保存到Adapter内部。

class适配器

8、class适配器是什么?

> 1. 类适配器需要多重继承(multiple inheritance)。
> 2. adapter是Target(目标)和Adaptee(被适配者)的子类。
> 3. 因此这在java中是无法实现的。所以本文不过多介绍。

适配器模式的应用场景

9、适配器应用于Java中的场景

> 集合类的迭代器

10、Enumerators是什么?

> 1. Java中的集合类(Vector、Stack等)在旧时代的时候提供了枚举器(Enumerators<接口>)
> 2. 可以不管元素在集合中如何安排的，直接获取到所有元素。

```
hasMoreElements()
nextElement()
```

11、Iterators是什么?

> 1. 新版的集合提供了迭代器，去和枚举器一样获取所有元素，还增加了删除功能。

```
hasNext()
next()
remove()
```

12、适配器模式在集合的迭代器中的使用?

> 1. 需要将 `Enumeration` 通过适配器，适配成 `Iterator`
> 2. 使得对于新代码来说，通过适配，实现外在是Iterator接口，内在还是Enumeration。
> 3. Enumerator就是adaptee(被适配者)。

```
/**
 * 适配器：将Enumeration接口转换为Iterator接口
 * */
public class EnumerationIterator implements Iterator{
        Enumeration enumeration;
        public EnumerationIterator(Enumeration enumeration) {
                this.enumeration = enumeration;
        }
        public boolean hasNext() {
                return enumeration.hasMoreElements();
        }

        public Object next() {
                return enumeration.nextElement();
        }
        //无法实现就直接抛出异常
        public void remove() {
                throw new UnsupportedOperationException();
        }
}
```

适配器和装饰者

13、Adapter适配器和Decorator装饰者的区别?

> 1. 装饰者致力于给对象加上新的职责和行为。
> 2. 当你需要一系列类一起工作，并且给客户提供需要的接口。这就需要适配器。