1 适配器模式 (Adapter)

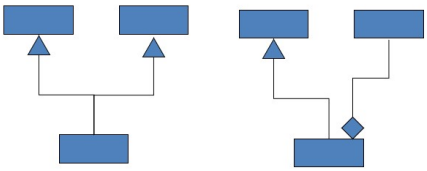
结构模式描述如何将类或者对象组织在一起形成特定的结构，以实现更为复杂、强大的功能。结构模式可以分为：

- 类的结构模式：使用继承来把类、接口等组合起来，以形成更大的结构。类的结构模式是静态的。典型例子是类形式的适配器模式。
- 对象的结构模式：描述怎样把各种不同类型的对象组合在一起，以实现新的功能的方法。对象的结构模式是动态的。典型的结构模式有代理模式，其它包括合成模式、享元模式、装饰模式、对象形式的适配器模式。

适配器模式 (Adapter Pattern) 把一个类的接口变换成客户端所需要的另一种接口，从而使原本因接口不匹配而无法在一起工作的两个类能够在一起工作。

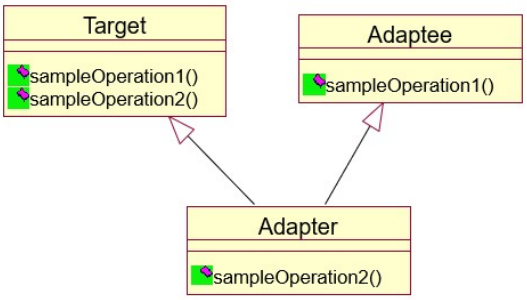
名称的由来: 这很像变压器，变压器把一种电压变换成另一种电压。把美国的电器拿回中国大陆来用的时候，用户就面临电压不同的问题。美国的生活用电电压是 110v，而中国的电压是 220v。如果要在中国使用美国的电器，就必须有一个能把 220v 电压转换成 110v 电压的变压器。而这就像是本模式所做的事，因此此模式也常常被称为变压器模式.

适配器模式的两种形式：适配器模式有类的适配器模式和对象的适配器模式两种不同的形式。如下图所示，左边是类的适配器模式，右边是对象的适配器模式。



1.1 类的适配器模式的结构

类的适配器模式把被适配的类的 API 转换成为目标类的 API，其静态结构图如下图所示。



Adaptee 类没有提供 sampleOperation2() 方法，而客户端则需要使用这个方法。为使客户端能够使用 Adaptee 类，提供一个中间环节，即类 Adapter，把 Adaptee 的 API 与 Target 类的 API 衔接起来。

模式所涉及的角色有：

- 目标 (Target) 角色：这就是所期待得到的接口。
- 源 (Adaptee) 角色：现有需要适配的接口。

- 适配器 (Adapter) 角色：适配器类是本模式的核心。适配器把源接口转换成目标接口。显然，这一角色不可以是接口，而必须是具体类。

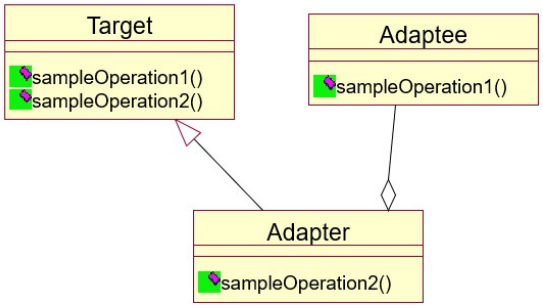
```
public interface Target {
  void sampleOperation1(); //源类的方法
  void sampleOperation2(); //源类没有的方法
}

public class Adaptee {
  public void sampleOperation1() { }
}

public class Adapter extends Adaptee implements Target {
  /* 由于源类没有方法sampleOperation2,因此适配器类补上这个方法 */
  public void sampleOperation2() {
      // Write your code here
  }
}
```

1.2 对象的适配器模式的结构

与类的适配器模式一样，对象的适配器模式把被适配的类的 API 转换成为目标类的 API，与类的适配器模式不同的是，对象的适配器模式不是使用继承关系连接到 adaptee 类，而是使用委派关系连接到 Adaptee 类。对象的适配器模式的静态结构如下图所示。




```
public interface Target {
  void sampleOperation1(); // 源类有的方法
  void sampleOperation2(); // 源类没有的方法
}

public class Adaptee {
  public void sampleOperation1() { }
}

public class Adapter implements Target {
```

```
  private Adaptee adaptee;
  public Adapter(Adaptee adaptee) {
    super();
    this.adaptee = adaptee;
  }
  public void sampleOperation1() {
    adaptee.sampleOperation1();
  }
  public void sampleOperation2() {
    // Write your code here
  }
}
```

适配器模式的目的是将接口不同而功能相同或相近的接口加以转换，这里面包括适配器角色补充了一个源角色没有的方法。

对象型的适配器模式的效果:

- 一个适配器可以把多种不同的源适配到同一个目标。即同一个适配器可以把源类和它的子类都适配到目标接口。
- 与类的适配器模式相比，要想替换源类的方法就不容易。如果一定要置换源类的一个或多个方法，就要先做一个源类的子类，将源类的方法替换，然后把源类的子类当作真正的源来适配。
- 虽然要想替换源类的方法不容易，但要想增加新的方法则很方便，而新增加的方法可适用于所有的源。

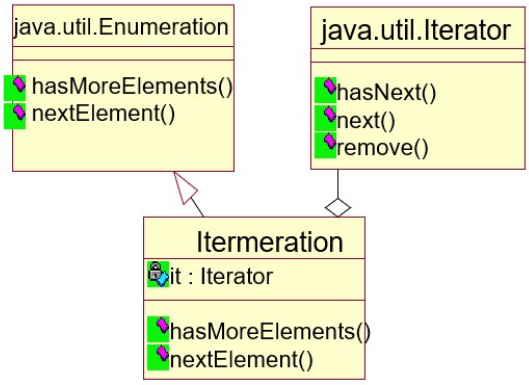
什么情况下使用适配器模式:

1. 系统需要使用现有的类，而此类的接口不符合系统的需要。
2. 想要建立一个可以重复使用的类．用于一些彼此之间没有太大关联的一些类，包括一些可能在将来引进的类一起工作，这些源类不一定有很复杂的接口。

1.3 Iterator 和 Enumeration

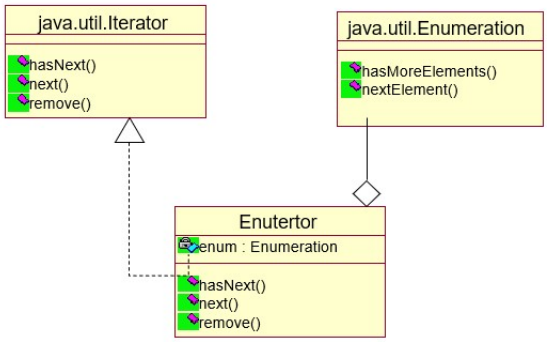
在 JDK 1.0 和 1.1 版本里没有 java 聚集 (collection) 的框架，这一框架是在 JDK l．2 版本中给出的。与此相对应，JDK1.0 和 1.1 版本提供了 enumeration 接口，而 JDK 1.2 版本给出了 iterator 接口。如果有很多的 java 代码是为老版本 Java 编译器写的，使用的是 Enumeration，现在想使用新版本编译器和新的 Java 聚集库包的话，需要将已有代码的 Iterator 接口换成 Enumeration 接口．因为 Java 聚集要求 Iterator 接口，这样才能使已有的代码可以使用新版本的聚集对象。

从 Iterator 到 Enumeration 的适配: 适配器 Itermeration 使用了对象的适配器模式，将一个 Iterator 对象封装在一个 Enumeration 类的具体类里。如下图所示:




```
1  import java.util.Enumeration;
2  import java.util.Iterator;
3  import java.util.NoSuchElementException;
4  public class Itermeration implements Enumeration {
5    private Iterator it;
6    public Itermeration(Iterator it) {
7      this.it = it;
8    }
9    public boolean hasMoreElements() {
10     return it.hasNext();
11   }
12   public Object nextElement() throws NoSuchElementException {
13     return it.next();
14   }
15 }
```

从 Enumeration 到 Iterator 的适配: Enuterator 使用了对象的适配器模式将 Enumeration 接口适配到 Iterator 接口。如下图所示:




```
1  import java.util.Enumeration;
2  import java.util.Iterator;
3  import java.util.NoSuchElementException;
4  public class Enuterator implements Iterator {
5    private Enumeration enum;
```

```
6   public Enuterator(Enumeration enum) { this.enum = enum; }
7   public boolean hasNext() { return enum.hasMoreElements(); }
8   public Object next() throws NoSuchElementException {
9     return enum.nextElement();
10  }
11  public void remove() {
12    throw new UnsupportedOperationException();
13  }
14 }
```

1.4 适配器模式在架构层次上的应用

- WINE 是一个开源的免费软件，允许在 linux 环境里运行 windows 程序。WINE 提供了从 Linux 到 Window 图形界面的适配器。
- MKS Toolkit 软件是一个为 Windows 系统设计的商业软件，它提供了 C shell、Korn shell、Perl 以及多种 UNIX 命令的解释器。在安装了此软件之后，Windows 的用户可以在自己的系统上使用 Unix 的 C Shell 和 Korn shell 指令集合，以及如 ls、vi、grep 等的 unix 命令。也就是说 MKS Toolkit 提供了 UNIX 命令集从 UNIX 到 windows 的适配，是一种命令集层次上的适配器模式。
- JDBC 驱动软件与适配器模式：JDBC 给出一个客户端的通用界面。每一个数据库引擎的 JDBC 驱动软件都是一个介于 JDBC 接口和数据库引擎之间的适配器软件。
- 抽象的 JDBC 接口和各个数据库引擎的 API 之间都需要相应的适配器软件，即为各个数据库引擎准备的驱动软件。