适配器模式 - 托尼托尼 · 乔巴

发表于 2022-09-03 | 更新于 2023-04-06 | 设计模式

| 字数总计: 2.9k | 阅读时长: 9 分钟 | 阅读量: 799 | 评论数: 2

配套视频课程已更新完毕，大家可通过以下两种方式观看视频讲解：

关注公众号： 爱编程的大丙 ，或者进入 大丙课堂 学习。

苏丙榅

合抱之木，生于毫末；九层之台，起于垒土；千里之行，始于足下。

1. 翻译家

在海贼王中，托尼托尼·乔巴（Tony Tony Chopper）是草帽海贼团的船医，它本来是一头驯鹿，但是误食了动物系·人人果实之后可以变成人的形态。

目录

乔巴吃了恶魔果实之后的战斗力暂且抛开不谈，说说它掌握的第二技能： 语言 ，此时的他 既能听懂人的语言，又能听懂动物语言 ，妥妥的语言学家。

人和动物本来无法直接交流，但是有了乔巴的存在，就相当于了有了一条纽带，一座桥梁，使得二者之间能够顺畅的沟通。在这里边，乔巴充当的就是一个适配器， 将一个类的接口转换成用户希望的另一个接口，使不兼容的对象能够相互配合并一起工作，这种模式就叫适配器模式。 说白了，适配器模式就相当于找了一个翻译。

需要适配器的例子或者场景很多，随便列举几个：

1. STL 标准模板库有六大组件，其中之一的就是适配器。
    - 六大组件分别是：容器、算法、迭代器、仿函数、适配器、空间适配器。
    - 适配器又可以分为：容器适配器、函数适配器、迭代器适配器
2. 台湾省的电压是 110V，大陆是 220V，如果他们把大陆的电器带回台湾就需要适配器进行电压的转换。
3. 香港的插座插孔是欧式的，从大陆去香港旅游，就需要带转换头（适配器）
4. 儿媳妇儿和婆婆打架，就需要儿子从中调解，此时儿子就适配器。
5. 手机、平板、电脑等需要的电压并不是 220V，也需要适配器进行转换。

2. 斜杠型人才

所谓的斜杠型人才就是多才多艺，适配器也一样，如果它能给多个不相干的对象进行相互之间的适配，这个适配器就是斜杠适配器。

CMake 保姆级教程（下）
2023-03-15

CMake 保姆级教程（上）
2023-03-06

访问者模式 - 再见，香波地群岛
2022-09-22

模板方法模式 - 和平主义者
2022-09-21

状态模式 - 文斯莫克 · 山治
2022-09-20

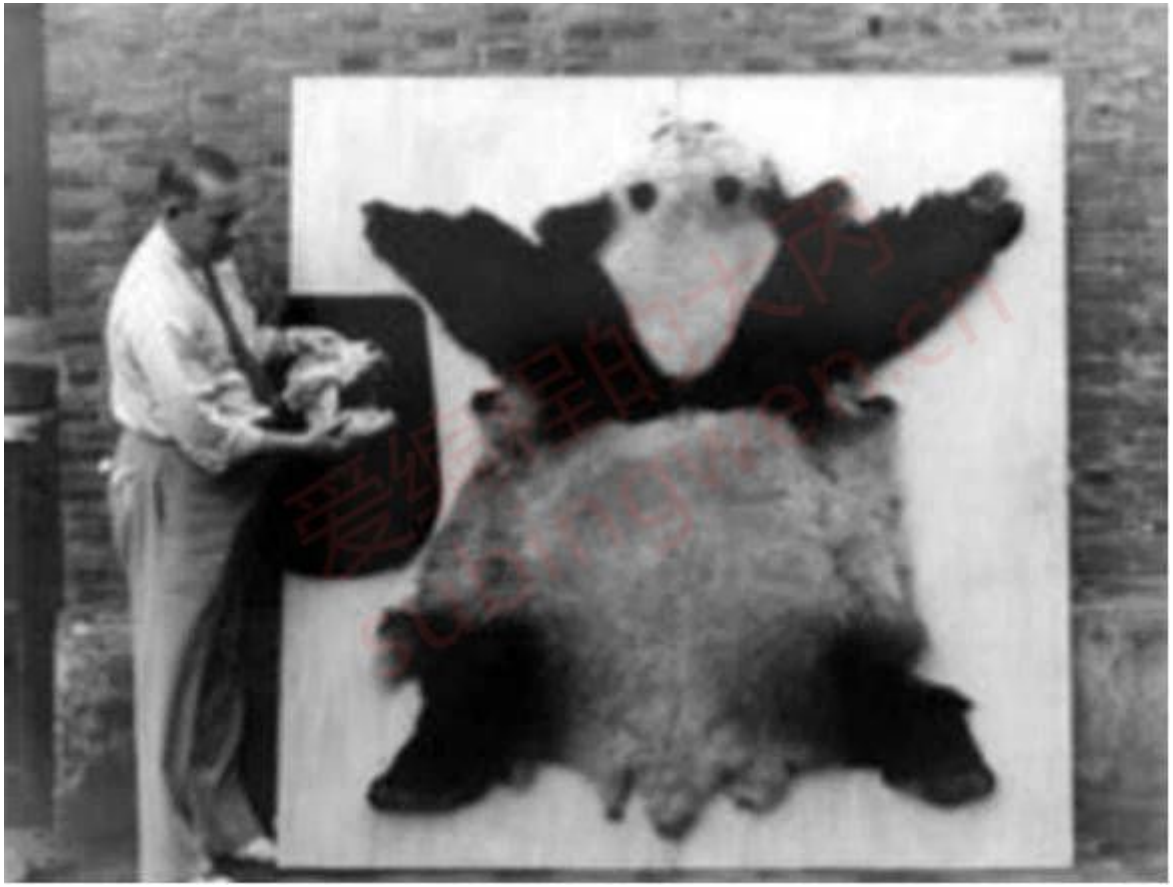
还是拿乔巴举例子，他既能把人的语言翻译给动物，又能把动物的语言翻译给人，那么此时的乔巴就是一个双向适配器。

2.1 国宝的血泪史

覆巢之下无完卵，清朝末年，作为曾经蚩尤的坐骑而今呆萌可爱的国宝大熊猫惨遭西方国家围猎和杀戮，其中以美利坚尤甚。

1869 年春天，法国传教士阿尔芒 · 戴维在四川宝兴寻找珍稀物种的时候发现了大熊猫，国宝的厄运就此开始。

这是美国总统罗斯福两个儿子制作的大熊猫标本：



那个时候的旧中国对熊猫还没有一个确切的认识，并没有采取任何措施，于是外国人对熊猫的猎杀活动更变本加厉起来。据统计，在 1936 年到 1946 年之间，超过 16 只活体大熊猫从中国运出，而熊猫标本更是多达 70 余具！

一切终成过去，作为一个程序猿我也不能改变什么，但是我决定要写个程序让这群混蛋给被他们杀死的国宝道歉！

2.2 抽丝剥茧

对于杀害大熊猫的这群西方的混蛋，不论他们怎么忏悔大熊猫肯定也是听不懂的，所以需要适配器来完成这二者之间的交流。但是这帮人来自不同的国家，它们都有自己的语言，所以这个适配器翻译的不是一种人类语言而是多种，因此我们要做如下的处理：

1. 忏悔的人说的是不同的语言，所以需要一个抽象类。
2. 适配器需要翻译不同国家的语言，所以适配器也需要一个抽象类。
3. 西方人需要给大熊猫道歉，因此需要给大熊猫定义一个类。

西方罪人

先把西方这群烂人对应的类定义出来：

C++

```cpp
class Foreigner
{
public:
    virtual string confession() = 0;
    void setResult(string msg)
    {
        cout << "Panda Say: " << msg << endl;
    }
    virtual ~Foreigner() {}
};

// 美国人
class American : public Foreigner
{
public:
    string confession() override
    {
        return string("我是畜生, 我有罪!!!");
    }
};

// 法国人
class French : public Foreigner
{
public:
```

```
    string confession()
    {
        return string("我是强盗, 我该死!!!");
    }
};
```

不同国家的西方罪人需要使用不同的语言向大熊猫忏悔，所以 美国人 和 法国人 作为子类需要重写从父类继承的用于忏悔的虚函数 `confession()` 。

当乔巴这个适配器翻译了熊猫的语言之后，需要通过 `void setResult(string msg)` 函数将信息传递给西方罪人对象。

大熊猫

再把国宝对应的类定义出来

C++

```
// 大熊猫
class Panda
{
public:
    void recvMessage(string msg)
    {
        cout << msg << endl;
    }
    string sendMessage()
    {
        return string("强盗、凶手、罪人是不可能被宽恕和原谅的！");
```

```
    }
};
```

大熊猫类有两个方法：

- `recvMessage(string msg)`：接收忏悔信息。
- `string sendMessage()`：告诉西方人是否原谅他们。

乔巴登场

同时能听懂人类和动物语言非乔巴莫属，由于要翻译两种不同的人类语言，所以需要一个抽象的乔巴适配器类，在其子类中完成 英语 ⟺ 熊猫语 、 法语 ⟺ 熊猫语 之间的翻译。

C++

```
// 抽象乔巴适配器类
class AbstractChopper
{
public:
    AbstractChopper(Foreigner* foreigner) : m_foreigner(foreigner) {}
    virtual void translateToPanda() = 0;
    virtual void translateToHuman() = 0;
    virtual ~AbstractChopper() {}
protected:
    Panda m_panda;
    Foreigner* m_foreigner = nullptr;
};

// 英语乔巴适配器
class EnglishChopper : public AbstractChopper
```

```
{
public:
    // 继承构造函数
    using AbstractChopper::AbstractChopper;
    void translateToPanda() override
    {
        string msg = m_foreigner->confession();
        // 翻译并将信息传递给熊猫对象
        m_panda.recvMessage("美国人说: " + msg);
    }
    void translateToHuman() override
    {
        // 接收熊猫的信息
        string msg = m_panda.sendMessage();
        // 翻译并将熊猫的话转发给美国人
        m_foreigner->setResult("美国佬, " + msg);
    }
};

// 法语乔巴适配器
class FrenchChopper : public AbstractChopper
{
public:
    using AbstractChopper::AbstractChopper;
    void translateToPanda() override
    {
        string msg = m_foreigner->confession();
        // 翻译并将信息传递给熊猫对象
```

在上面的适配器类中，同时访问了 `Foreigner 类`和 `Panda 类`，这样适配器类就可以拿到这两个类对象中的数据进行转译，最后再将其分别发送给对方，这样这两个不相干的没有交集的类对象之间就可以正常的沟通了。

不可原谅

最后编写程序进行测试，这部分程序其实是通过客户端的操作并被执行的，此处就将其直接写到 `main()` 函数中了:

C++

```cpp
int main()
{
    Foreigner* human = new American;
    EnglishChopper* american = new EnglishChopper(human);
    american->translateToPanda();
    american->translateToHuman();
    delete human;
    delete american;

    human = new French;
    FrenchChopper* french = new FrenchChopper(human);
    french->translateToPanda();
    french->translateToHuman();
    delete human;
    delete french;

    return 0;
}
```

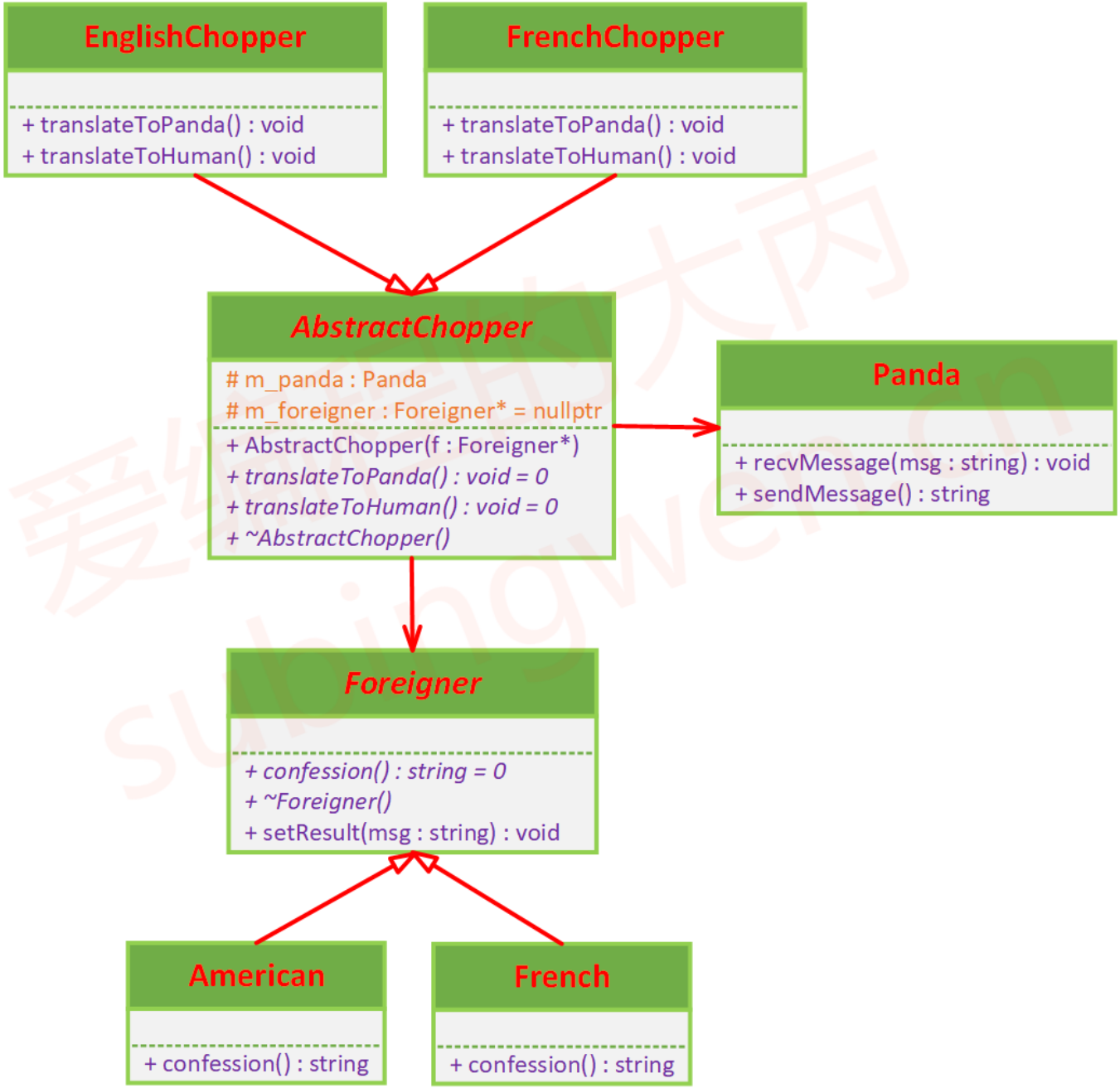
程序输出的结果:

C++

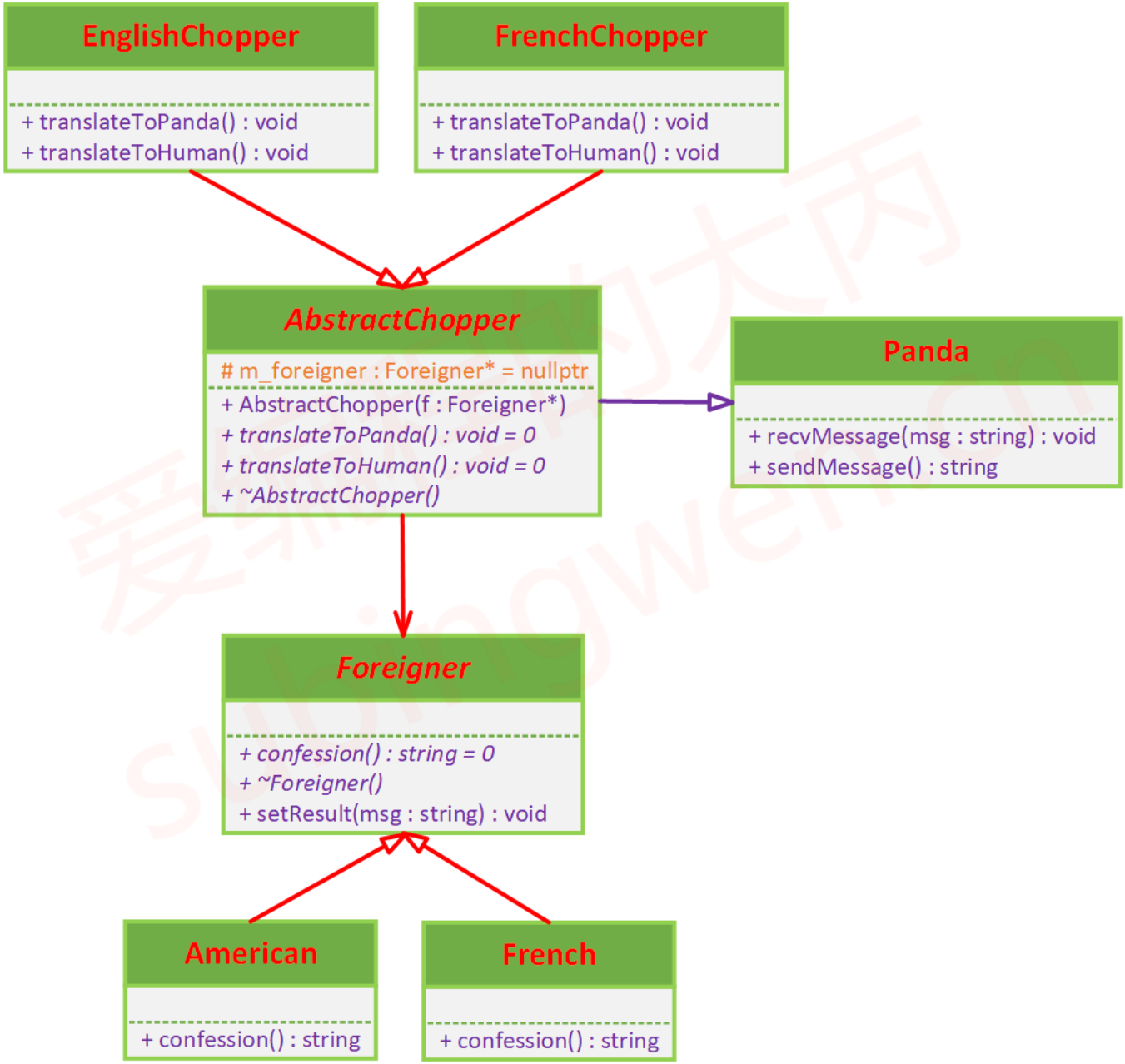
```
美国人说：我是畜生，我有罪!!!
Panda Say：美国佬，强盗、凶手、罪人是不可能被宽恕和原谅的!
============================
法国人说：我是强盗，我该死!!!
Panda Say：法国佬，强盗、凶手、罪人是不可能被宽恕和原谅的!
```

3. 结构图

最后根据上面的代码，把对应的 UML 类图画一下（再次强调，UML 类图是在在写程序之前画的，用来梳理程序的设计思路。学会了设计模式之后，就需要在写程序之前画类图了。）


EnglishChopper
+ translateToPanda() : void
+ translateToHuman() : void
FrenchChopper
+ translateToPanda() : void
+ translateToHuman() : void
AbstractChopper
m_panda : Panda
m_foreigner : Foreigner* = nullptr
+ AbstractChopper(f : Foreigner*)
+ translateToPanda() : void = 0
+ translateToHuman() : void = 0
+ ~AbstractChopper()
Panda
+ recvMessage(msg : string) : void
+ sendMessage() : string
Foreigner
+ confession() : string = 0
+ ~Foreigner()
+ setResult(msg : string) : void
American
+ confession() : string
French
+ confession() : string

在这个 UML 类图中，将抽象的 乔巴类（抽象适配器类） 和 熊猫类 设置为了 关联关系 ， 除了使用这种方式我们 还可以让抽象的适配器类继承熊猫类，这样在适配器类中就可以直接使用熊猫类中定义的方法了 ， 如下图：


EnglishChopper
+ translateToPanda() : void
+ translateToHuman() : void
FrenchChopper
+ translateToPanda() : void
+ translateToHuman() : void
AbstractChopper
m_foreigner : Foreigner* = nullptr
+ AbstractChopper(f : Foreigner*)
+ translateToPanda() : void = 0
+ translateToHuman() : void = 0
+ ~AbstractChopper()
Panda
+ recvMessage(msg : string) : void
+ sendMessage() : string
Foreigner
+ confession() : string = 0
+ ~Foreigner()
+ setResult(msg : string) : void
American
+ confession() : string
French
+ confession() : string

上图对应的代码如下：

C++

```cpp
class Foreigner
{
public:
    virtual string confession() = 0;
    void setResult(string msg)
    {
        cout << "Panda Say: " << msg << endl;
    }
    virtual ~Foreigner() {}
};

// 美国人
class American : public Foreigner
{
public:
    string confession() override
    {
        return string("我是畜生, 我有罪!!!");
    }
};

// 法国人
class French : public Foreigner
{
public:
    string confession()
    {
        return string("我是强盗, 我该死!!!");
    }
};
```

```
31
32  // 大熊猫
33  class Panda
34  {
35  public:
36      void recvMessage(string msg)
37      {
38          cout << msg << endl;
39      }
40      string sendMessage()
41      {
42          return string("强盗、凶手、罪人是不可能被宽恕和原谅的！");
```

上面的代码和第一个版本的代码其实是没有太大区别，如果仔细观察会发现，在适配器类中使用熊猫类中的方法的时候就无需通过熊猫类的对象来调用了，因为适配器类变成了熊猫类的子类，把这些方法继承下来了。

使用这样的模型结构，有一点需要注意：如果熊猫类有子类，那么还是建议将熊猫类和适配器类设置为关联关系。

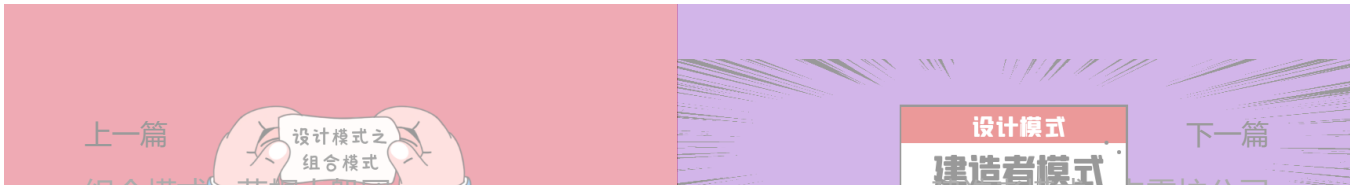
其实关于适配器模式还有另外的一种实现方式：就是让适配器类继承它要为之提供服务器的类，也就是这个例子中的外国人类和熊猫类（如果外国人来没有子类可以使用这种方式），这种解决方案要求使用的面向对象的语言支持多继承，对于这一点 C++ 是满足要求的，但是很多其它面向对象的语言不支持多继承。

> **再次强调，在使用适配器类为相关的类提供适配服务的时候，如果这个类没有子类就可以让适配器类继承这个类，如果这个类有子类，此时使用继承就不太合适了，建议将适配器类和要被适配的类设置为关联关系。**

在画 UML 类图的时候，需要具体问题具体分析，使用相同的设计模式处理不同的业务场景，绘制出的类和类之间的关系也是有些许差别的，不要死读书，读死书，头脑要活泛！

设计模式

打赏



相关推荐

评论

昵称 邮箱 网址(http://)

来都来了, 说点什么吧...

提交

2 评论

菜狗涛 Firefox 109.0 Windows 11

2023-02-05 回复

写得好，点个赞

Anonymous Firefox 109.0 Windows 11

2023-02-05 回复

666

Powered By Valine

v1.5.1