**[C++设计模式——适配器模式Bridge-Pattern](https://www.cnblogs.com/wkfvawl/p/12608911.html)**

**动机(Motivation)**

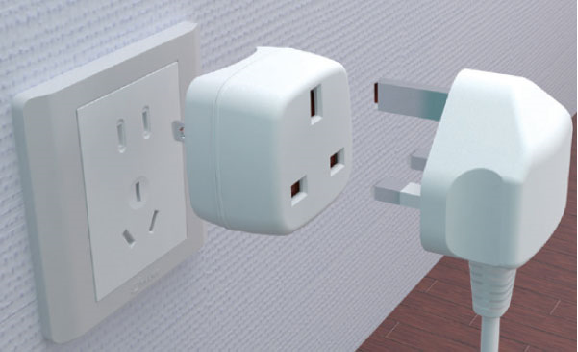
* 由于应用环境的变化，常常需要将”一些现存的对象“放在新的环境中应用，但是新环境要求的接口是这些现存对象所不满足。
* 如何应对这些”迁移的变化“？

**模式定义**

将一个类的接口转换成客户希望的另一个接口。Adapter模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。 ——《设计模式》GoF

**模式举例**

人们出国旅行经常会带着电源转换器，因为很多国家之间的电器插座标准是不同的。



适配器模式就是一种从生活实例中演化而来的设计模式，可将一个类的接口转换成用户希望的另外一个接口，即在新接口和老接口之间进行适配。

 这里举一个例子

[复制代码](javascript:void(0);)

#include <iostream>

using namespace std;

class CCnOutlet//中式插座

{

public:

void useCnplug(){ cout<<"use Cn plug."<<endl; }

};

int main()

{

CCnOutlet cn;

cn.useCnplug();

return 0;

}

[复制代码](javascript:void(0);)

上例中，有个CCnOutlet类（中式插座类），它有个功能是useCnplug（使用中式插头）。现在新买了个港版手机，它使用的是英式插头，必须使用英式插座，它对应的类是：

class CEnOutlet/英式插座

{

public:

void useEnplug() { cout << "use En plug" << endl; }

};

   买回来的英式插头必须插在家里的中式插座上，如果还想使用原来的client代码，即main函数的代码（人们的使用插座的方式不变），就只好修改CCnOutlet类了。

class CCnOutlet/英式插座

{

public:

void useCnplug(){ cout << "use En plug" << endl; }

};

这样就能在家中插座上使用英式插头了，但是这种需求变化后修改已有类的做法是不可取的。新手机刚买没多久，你朋友从美国回来，送了你一台他在美国买的手机。现在你手头上有2个手机了，你都非常喜欢。这时要想在家中使用美式的插头，就不得不再去修改CCnOutlet类了，该如何改呢？将useCnplug函数改为：

void useCnplug(){ cout << "use Us plug" << endl; }

这样改后，原先的手机就不能用了！怎么办呢？突然，发现家里的插座不只1个呀，有很多个插座呀，有办法了，可以这样改：

[复制代码](javascript:void(0);)

class CCnOutlet\_1

{

public:

void useCnplug()

{

cout << "use En plug" << endl;

}

};

class CCnOutlet\_2

{

public:

void useCnplug()

{

cout << "use Us plug" << endl;

}

};

[复制代码](javascript:void(0);)

这样显然也是不可取的，除了增加了一些功能相似的类，还得修改主程序。

由此引出了适配器模式，我们可以这样设计代码：

[复制代码](javascript:void(0);)

#include <iostream>

using namespace std;

class CCnOutlet

{

public:

virtual void useCnplug()

{

cout<<"use Cn plug."<<endl;

}

};

class IAdaptee

{

public:

void virtual useplug() = 0;

};

class CEnOutlet : public IAdaptee

{

public:

void useplug()

{

cout<<"use En plug."<<endl;

}

};

class Adapter : public CCnOutlet//公有继承目标

{

private:

IAdaptee \*pA;//适配对象

public:

Adapter(IAdaptee \*p)

{

pA = p;

}

void useCnplug()

{

pA->useplug();

}

};

int main()

{

CEnOutlet \*pEn = new CEnOutlet;//英式插座

CCnOutlet \*pCn = new Adapter(pEn);

pCn->useCnplug();

delete pEn;

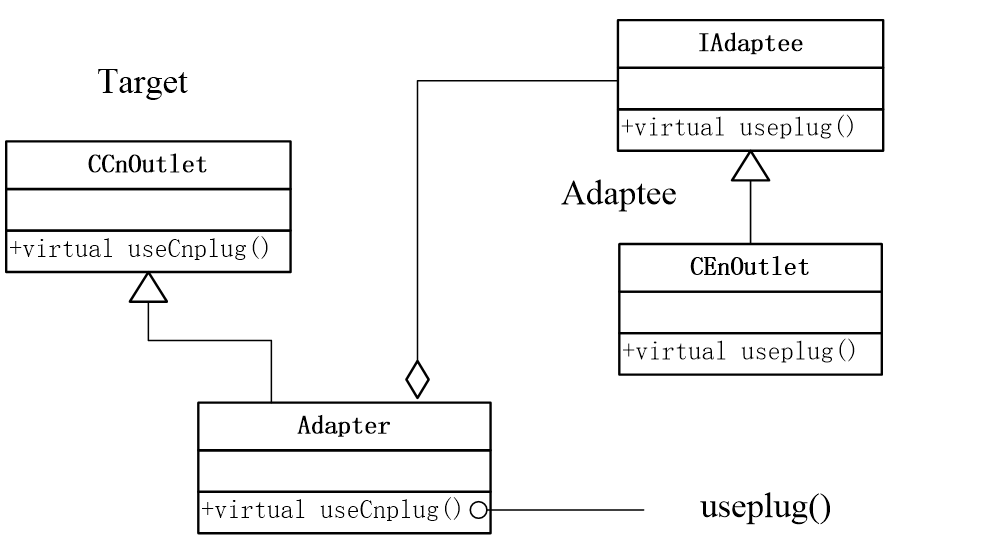
delete pCn;

return 0;

}

[复制代码](javascript:void(0);)

客户端想用像CCnOutlet 一样的方法来用CEnOutlet，即像使用中式插头那样使用英式插头，但CEnOutlet中没有useCnplug方法，只有useEnplug方法。这时就需要一个包装（Wrapper）类Adapter，这个包装类包装了一个CEnOutlet的实例，从而将客户端与CEnOutlet衔接起来。



使用适配器的对象模式的好处是：适配器可以有多个适配者，如果需要兼容美式插头，不需要更改原有代码，只需要新增加个类CUsOutlet即可。

class CUsOutlet: public IAdaptee

{

public:

void useplug() { cout<<"use Us plug."<<endl; }

};

使用该手机代码为：

CUsOutlet \*pUs = new CUsOutlet;

CCnOutlet \*pCn = new Adapter(pUs);

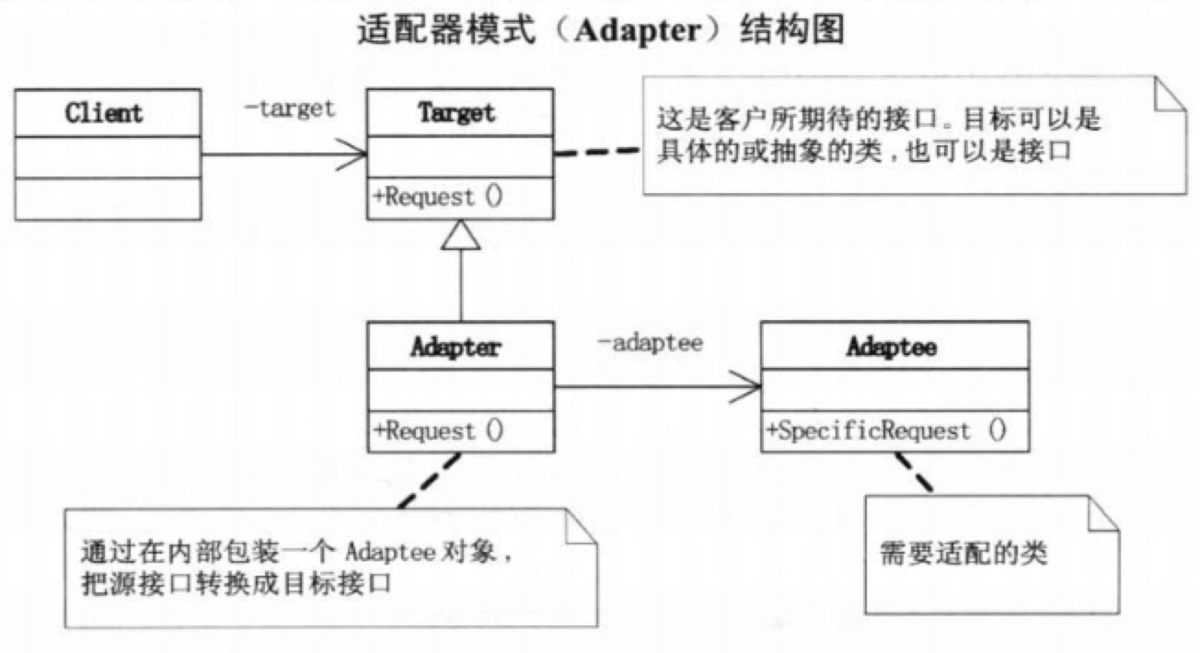
pCn->useCnplug();

**要点总结**

适配器模式，是将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

在遗留代码复用、类库迁移等方面有用。

**结构（Structure）**



**基本代码**

[复制代码](javascript:void(0);)

#include <iostream>

using namespace std;

class Target { // Target，客户期望的接口，可以使具体或抽象的类，也可以是接口

public:

virtual void Request() = 0;

virtual ~Target(){};

};

class Adaptee { // 需适配的类

public:

void SpecificRequest() { cout << "Adaptee" << endl; }

};

class Adapter : public Target { // 通过内部包装一个Adaptee对象，把源接口转换为目标接口：

private:

Adaptee\* adaptee;

public:

Adapter() { adaptee = new Adaptee(); }

void Request() { adaptee->SpecificRequest(); } // 调用Request()方法会转换成调用adaptee.SpecificRequest()

~Adapter() { delete adaptee; }

};

int main() {

Target\* target = new Adapter();

target->Request();

delete target;

return 0;

}