GoF（四人帮）那本《设计模式 可复用面向对象软件的基础》可谓是设计模式方面的经典之作，其中介绍的23种设计模式，

也可谓是经典中的经典。但是，设计模式的种类绝不仅仅是这23种，除此之外还有很多巧妙可爱的设计模式值得我们学习。这些

被遗忘的设计模式，也可以堪称经典之作。今天我们来一起学习被遗忘的设计模式——空对象模式（Null Object Pattern）。

一起看看这个模式会带给我们怎样的惊喜？

### ****一、Pattern name****

Provide an object as a surrogate for the lack of an object of a given type. The Null Object provides intelligent do nothing behavior, hiding the details from its collaborators.

### ****二、Problem****

任何没有实际应用场景的设计模式，都是在耍流氓。学习设计模式，不仅仅是为了领悟其精髓，更为了在实践设计当中去运用，去变通，下面我们来看看，什么情况下，这个Null Object Pattern会派上用场呢？

假设这样一个场景：

在一个图书信息查询系统中，你调用一个方法，传过去你要查找图书的ID，然后它返回给你，你要查找的图书对象，这样你就可以调用对象的方法来输出图书的信息。

我想这种场景在程序设计中还是比较常见的。下面，我们来实现以下具体的代码。

首先，我们来看一下ConcreteBook类的代码（提供构造函数和展示图书信息的show()方法。）：

|  |
| --- |
| public class ConcreteBook {  private int ID;  private String name;  private String author;  // 构造函数  public ConcreteBook(int ID, String name, String author) {  this.ID = ID;  this.name = name;  this.author = author;  }  /\*\*  \*  \* Description About show: <br>  \* 展示图书的相关信息  \*  \* @version V1.0  \*/  public void show() {  System.out.println(ID + "\*\*" + name + "\*\*" + author);  }  } |

我们再来看看创建图书对象的图书工厂的代码(主要提供一个获得ConcreteBook的方法)：

public class BookFactory {

/\*\*

\*

\* Description About getBook: <br>

\* 根据ConcreteBook的ID，获取图书对象。

\* @param ID 图书的ID

\* @return 图书对象

\* @version V1.0

\*/

public ConcreteBook getBook(int ID) {

ConcreteBook book = null;

switch (ID) {

case 1:

book = new ConcreteBook(ID, "设计模式", "GoF");

break;

case 2:

book = new ConcreteBook(ID, "被遗忘的设计模式", "Null Object Pattern");

break;

default:

book = null;// 其实这个可以省略，因为初始化已经赋值为null。

break;

}

return book;

}

}

最后，来看一下客户端的代码：

public class Client {

static void main(String[] args) {

BookFactory bookFactory = new BookFactory();

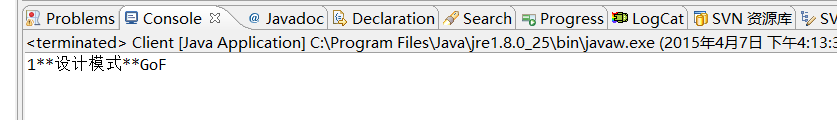
ConcreteBook book = bookFactory.getBook(1);

book.show();

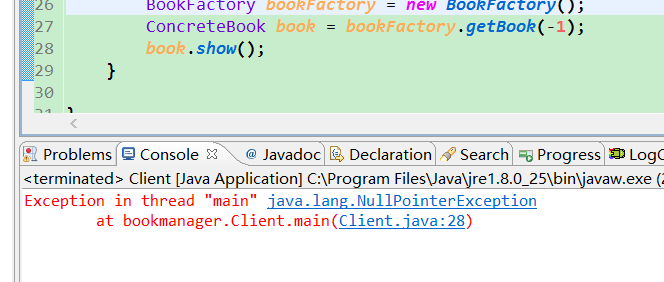
}

* }

上面三段代码很简单，我就不做详细解释了。下面，我们来运行一下，结果如下：



很好，运行很顺利，这时，我们把ConcreteBook book = bookFactory.getBook(1);中的1改为2，恩，也运行成功。这时候，我们改成-1。再来运行一下，发现如下报错：



空指针报错，是的，这应该是Java初学者见到最多的报错了。它提示我们第28行book.show()报错。这是为什么呢？因为我们通过bookFactory.getBook()方法获取ConcreteBook对象的时候，如果我们传入的参数，即图书的ID，属于非法值（如-1）或者不存在（如3）的话（其实这种情况是经常遇到的。），就会返回null，表示我们查找的图书信息并不存在。这时，book为null.你再调用book.show()。当然要报空指针的错误了。那怎么解决呢？

我们比较常规的做法就是在客户端加一个判断，判断是否为null。如果为null的话，就不再调用show()方法。如果不为null再调用show()方法。更改如下：

public static void main(String[] args) {

BookFactory bookFactory = new BookFactory();

ConcreteBook book = bookFactory.getBook(-1);

//判断book对象是否为null。

if (book == null) {

System.out.println("book对象为 null。");

} else {

book.show();

}

}

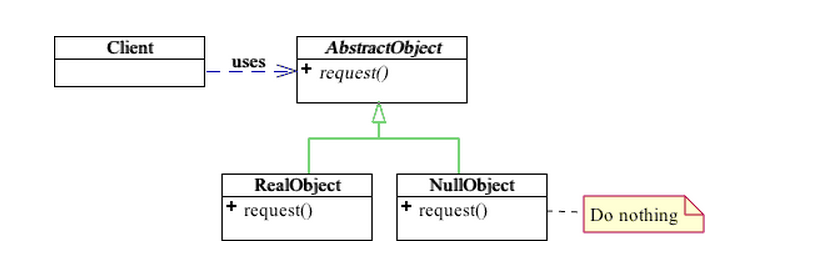
此时，再运行，就不会报错了。而是，输出了：book对象为null。

****但是，你有没有考虑过？这样做，确实消除了报错，但是这样做真的好吗？你想如果在一段程序中有很多处调用getBook()方法或者有很多个客户端的话（比如图书馆的查询终端肯定不止一个啊），岂不是很多处都要判断book对象是否为null？这还不算坏，如果哪一处没有判断，然后报错了，很有可能导致程序没法继续运行甚至崩溃。而且，你要记住，永远都不要太相信客户端（Client），不要把整个程序的稳定性寄托在客户端身上。还有，像上面的处理方法，当获取对象为null的时候，输出的提示信息是有客户端来定制的，这样岂不是把主动权交给了客户端，而不是我们系统本身？****

那究竟应该如何实现才会更加合适呢？那就要用到我们今天要将的Null Object Pattern——一种被遗忘的设计模式

### ****三、Solution****

首先，我们来看一下Null Object Pattern的UML类图结构：



这个类图结构其实还是很简单的，这里面的RealObject其实就相当于我们的ConcreteBook类，而NullObject就是我们将要增加的空对象类，而AbstractObject类就是我们要提出来的父类。我们只是在Client和AbstractObject之间增加了一个BookFactory而已。

下面，我们来改一下我们的代码：

新增的抽象接口Book类的代码：

interface Book {

// 判断Book对象是否为空对象（Null Object）

public boolean isNull();

// 展示Book对象的信息内容。

public void show();

}

新增的空对象类NullBook类的代码（继承Book类）：

public class NullBook implements Book {

public boolean isNull() {

return true;

}

public void show() {

}

}

原有的ConcreteBook类修改后的代码（增加对Book接口的实现，实现isNull方法）：

public class ConcreteBook implements Book{

private int ID;

private String name;

private String author;

// 构造函数

public ConcreteBook(int ID, String name, String author) {

this.ID = ID;

this.name = name;

this.author = author;

}

/\*\*

\*

\* Description About show: <br>

\* 展示图书的相关信息

\*

\* @version V1.0

\*/

public void show() {

System.out.println(ID + "\*\*" + name + "\*\*" + author);

}

public boolean isNull(){

return false;

}

}

工厂类（BookFactory）修改后的代码（返回对象从ConcreteBook改为Book，并当ID属于非法值或者不存在时，返回NullBook对象。）：

public class BookFactory {

/\*\*

\* Description About getBook: <br>

\* 根据ConcreteBook的ID，获取图书对象。

\* @param ID 图书的ID

\* @return 图书对象

\* @version V1.0

\*/

public Book getBook(int ID) {

Book book;//将原来的ConcreteBook改为Book

switch (ID) {

case 1:

book = new ConcreteBook(ID, "设计模式", "GoF");

break;

case 2:

book = new ConcreteBook(ID, "被遗忘的设计模式", "Null Object Pattern");

break;

default:

book = new NullBook();//创建一个NullBook对象

break;

}

return book;

}

* }

客户端的代码为：

public static void main(String[] args) {

BookFactory bookFactory = new BookFactory();

Book book = bookFactory.getBook(-1);

book.show();

}

****运行一下，我们发现，即使传入的参数是非法值或者不存在的值时，也不会报错了，这是Null Object Pattern的第一个好处。但是现在不报错，也没有任何输出，肯定不够友好，不够人性化。此时，在NullBook类的show方法中，我们可以定制我们的输出提醒，当用户调用空对象的show方法时，就会输出我们定制的提醒。这回我们可以实现，一处定制，处处输出，主动权在我们手里，而不是在客户端的手里。这是Null Object Pattern的第二个好处。****

比如我们进行如下修改，修改后的NullBook类代码：

public class NullBook implements Book {

public boolean isNull() {

return true;

}

public void show() {

System.out.println("Sorry，未找到符合您输入的ID的图书信息，请确认您输入的不是非法值。");

}

}

此时，在执行一下Client，你会发现控制台输出为：Sorry，未找到符合您输入的ID的图书信息，请确认您输入的不是非法值。

****其实，虽然在客户端我们不进行检测也可以保证程序不报错，但是最好的方式，还是进行相应的检测，如下：****

public static void main(String[] args) {

BookFactory bookFactory = new BookFactory();

Book book = bookFactory.getBook(-1);

if (book.isNull()) {

//这里由客户端定制提醒代码

System.out.println("兄弟，你输入的ID不符合规范吧。");

}else{

book.show();

}

}

我们看到相比之下，book.isNull()比book == null更加优雅一点。到这里，Null Object Pattern大概就介绍完了。我们可以看到，其实Null Object Pattern还是有点意思的，可以说使整个系统更加坚固了。

### ****四、Consequences****

Null Object Pattern，作为一种被遗忘的设计模式，却有着不能被遗忘的作用。

（1）它可以加强系统的稳固性，能有有效地防止空指针报错对整个系统的影响，使系统更加稳定。   
（2）它能够实现对空对象情况的定制化的控制，能够掌握处理空对象的主动权。   
（3）它并不依靠Client来保证整个系统的稳定运行。   
（4）它通过isNull对==null的替换，显得更加优雅，更加易懂。