看完《Effective C++》条款 31 有感。。

假设有一个 Date 类

Date.h

[cpp] view plain copy

```cpp
class Date {
private:
    int year, month, day;
};
```

如果有个 Task 类的定义要用到 Date 类，有两种写法

其一

Task1.h

[cpp] view plain copy

```cpp
class Date;
class Task1 {
public:
    Date getData();
};
```

其二

Task2.h

[cpp] view plain copy

```cpp
#include "Date.h"
class Task2 {
public:
    Date getData();
};
```

一个采用前置声明，一个采用#include<Date.h>加入了 Date 的定义。两种方法都能通过编译。但是 Task1.h 这种写法更好。**如果 Date.h 的 private 成员变量改变，比如变成 double year, month, day; ，Task1.h 不需要重新编译，**

而 Task2.h 就要重新编译，更糟的是如果 Task2.h 还与其他很多头文件有依赖关系，就会引发一连串的重新编译，花费极大的时间。可是事实上改变一下写法就可以省去很多功夫。

所以能用前置声明代替#include 的时候，尽量用前置声明

有些情况不能用前置声明代替#include

比如 Task1.h 改成

[cpp] view plain copy

```cpp
class Date;
class Task1 {
public:
    Date d;
};
```

会编译错误，因为 Date d 定义了一个 Date 类型变量，编译器为 d 分配内存空间的时候必须知道 d 的大小，必须包含定义 Date 类的 Date.h 文件。

这是可以采用指针来代替

[cpp] view plain copy

```cpp
class Date;
class Task1 {
public:
    Date *d;
};
```

指针的大小是固定的。在 32 位机上是 4 字节，64 位机上是 8 字节。这时编译 Task1 的时候不需要 Date 的大小，所以和 Date 的定义无关。

何时可以用前置声明代替#include

（http://blog.csdn.net/rogeryi/archive/2006/12/12/1439597.aspx）

上述例子可以说明

如果使用 object reference 或 object point 可以完成任务，就不要用 object

这样可以尽最大可能避免#include

为声明式和定义是提供不同的头文件

在函数库的设计过程中，**接口的设计就要遵循上述准则。**

一个接口的头文件是这样的

interface.h

```cpp
[cpp] view plain copy
```

```cpp
class Date;
class Address;
class Email;
Date getDate();
```

如果客户只用到 Date 类，编译器就只会去编译 Date.h，而不去编译 Address.h，Email.h 等等文件。