# 本文适合所有简单了解过Java的人观看

1.1

一段程序实际上就是多个对象通过发送消息来通知彼此要干什么

每一个对象都有类型：每一个对象都是通过某个类生成的实例

同一类型的对象可以接收相同的消息

对象具有状态（内部数据）、行为（方法体）、标识

1.2

一旦创建了一个类，就可以用它穿件任意多个对象，然后在操作这些对象时，可以把它们视为存在于问题空间的元素。我们需要在问题空间的元素和解决方案空间的对象之间建立一一对应。

我们利用类生成对象，然后再对对象发出请求

Light lt = new Light(); lt.on();

用.将对象名和请求（方法）连接

在 Java 中，当你创建一个“对象”，你实际上并不是直接得到那个对象本身，而是：得到一个 **指向该对象的引用**

Person p = new Person("Alice");等号后面在**堆内存**中创建了一个 Person 对象。

p在 **栈内存** 中创建了一个变量 p，  
这个变量 **保存的是堆中对象的地址（引用）**

1.3

对象是服务的提供者，编程的任务即创建一些提供对应服务以解决问题的对象

对象的功能不能太多，比如负责打印的对象无需负责排版

1.4

作为类的创建者，我们只需要暴露接口（用户需要的标准）即可，而具体的实现不需要。

Private是墙，无法从外部访问，protected是在墙上开了一个子类通道的墙

1.5

组合：利用已有的类组合为一个新类

类的内部的private对象，外部本就无法调用，我们便可以在不影响外部已有代码的情况下修改这些对象（甚至是运行的时候，从而动态调整程序行为）

相比之下，创建新类的时候考虑组合，清晰、简洁、灵活

1.6

我们创建完一个类以后，我们需要一个功能相近的类，而且只需要在原类的基础上添改一些东西，我们便使用继承

我们可以通过基类呈现核心思想，从中派生的众多子类则为其核心思想的不同实现

比如基类Shape，可以派生出Circle，Triangle……

通过相同的接口调用该方法，但是希望其可以在新的类中实现不同的效果——重写

1.7

面向对象语言使用“后期绑定”：当向某一个对象发送消息时，直到运行时才会确定哪一段代码会被调用。

Void doSomething(Shape shape){ //这个方法对Shape类的所有对象都适用

shape.erase();

……

shape.draw();

}

Circle c = new Circle();

Triangle t = new Triangle();

doSomething(c);

doSomething(t);

不管对象是哪个类的，doSomething都可以运行

参数列表里要求是Shape类的对象，但是我们实际传入了Circle类的对象c。因为Circle也是一个Shape。这便是多态的本质与意义

即，这是一个Shape，我知道他可以进行erase，draw的操作，那就这么办，具体的细节交给Shape自己处理就好

1.8

……（往后看就懂了，此处只是介绍了一下Java及动态面向对象语言的一种特性）

1.9

（集合的简介）

单根层次结构使得Java的所有对象都输入Object类型

我们在集合中加入元素后，这个元素会向上转型为Object（Circle向上转型为Shape），那么当这个元素被取出的时候，会导致放入的元素被取出后不是原来的类型了，这时候我们需要下面的机制（向上——通用，向下——具体）

参数化类型（泛型）：一个被参数化的类型是一种特殊的类，可以让编译器自动适配特定的类型（e.g.编译器可以只接受特定的Shape的对象，如此集合也只能取出Shape对象）

ArrayList<Shape> s = new ArrayList<>();

1.10

Java特有的垃圾收集器，自动清除无用的对象

Java只允许动态分配内存。创建对象时都需要使用new来创建一个对象的动态实例

1.11

（异常处理，后面会讲，了解即可）

1.12

过程式程序：包含了数据定义和函数调用（想要理解他，你必须查看他的函数调用与底层代码）

我们会发现，面向对象的编程语言就像故事

2.（自己看）

3.1&3.2（名字有什么意义呢，就算用其他名字称呼玫瑰，也依旧不减它的芬芳）

实际操作程序，是对对象的引用

new关键字的意思是：我要一个这种类型的对象

String s = new String(“asdf”); 1

String s = “asdf”; 2

注意一下上面两种的区别

1：JVM将asdf放入String Pool中，再在heap中创建一个新的String对象，内容复制String Pool的asdf，s是heap中的新对象

2：JVM将asdf放入String Pool中，在运行时，检测其内是否有asdf字符，有就复用

s是指向String Pool的那一个唯一对象

数据保存：1.寄存器（存在CPU中）：有限，需按需分配

2.栈（存在RAM中）：处理器通过栈指针直接操作数据

3.堆（heap）：用于存放Java的所有对象（new）

4.常量存储：DNA

5.非RAM存储

new关键字在Heap中创建对象，所以我们创建了无须使用new来创建基本类型的变量，而是直接创建一个“自动变量”

包装类

创建一个用于防止对象的数组时，实际上数组里包含的是[引用](#对象，引用)，其会被自动初始化为null（一个没有指向任何对象的）

如果是基本类型的数组，会自动初始化为0

3.3 注释

3.4

作用域（{……}）

对象的作用域

{String s = new String(“awa”)}引用s会在作用域结束后消失，但指向的String对象还会继续占用内存，该String对象在作用域结束后就无法再获取了，Java会负责将它们销毁

3.5

局部变量就算是基本类型也不会有默认值（方法内部）

3.6

3.7

Java的名称可见性：将互联网域名反转使用(e.g. com.bilibili.库名)反转后的几个”.”描述了子目录的结构

有时会出现两种情况：1.我们需要一小块共享空间来保存特定字段，而不关心创建多少对象，甚至是否创建对象2.需要一个类的某个方法，而方法和具体对象无关（希望即便没有生成任何该类的对象，也可以调用其方法）

static字段是基于类创建的，非static字段是基于对象创建的

**静态字段**：  
“类的所有对象共用的一块记事本。”

T t1 = new T();

T t2 = new T();

t1.i 和 t2.i 都是0

**静态方法**：  
“属于类本身的工具函数。”

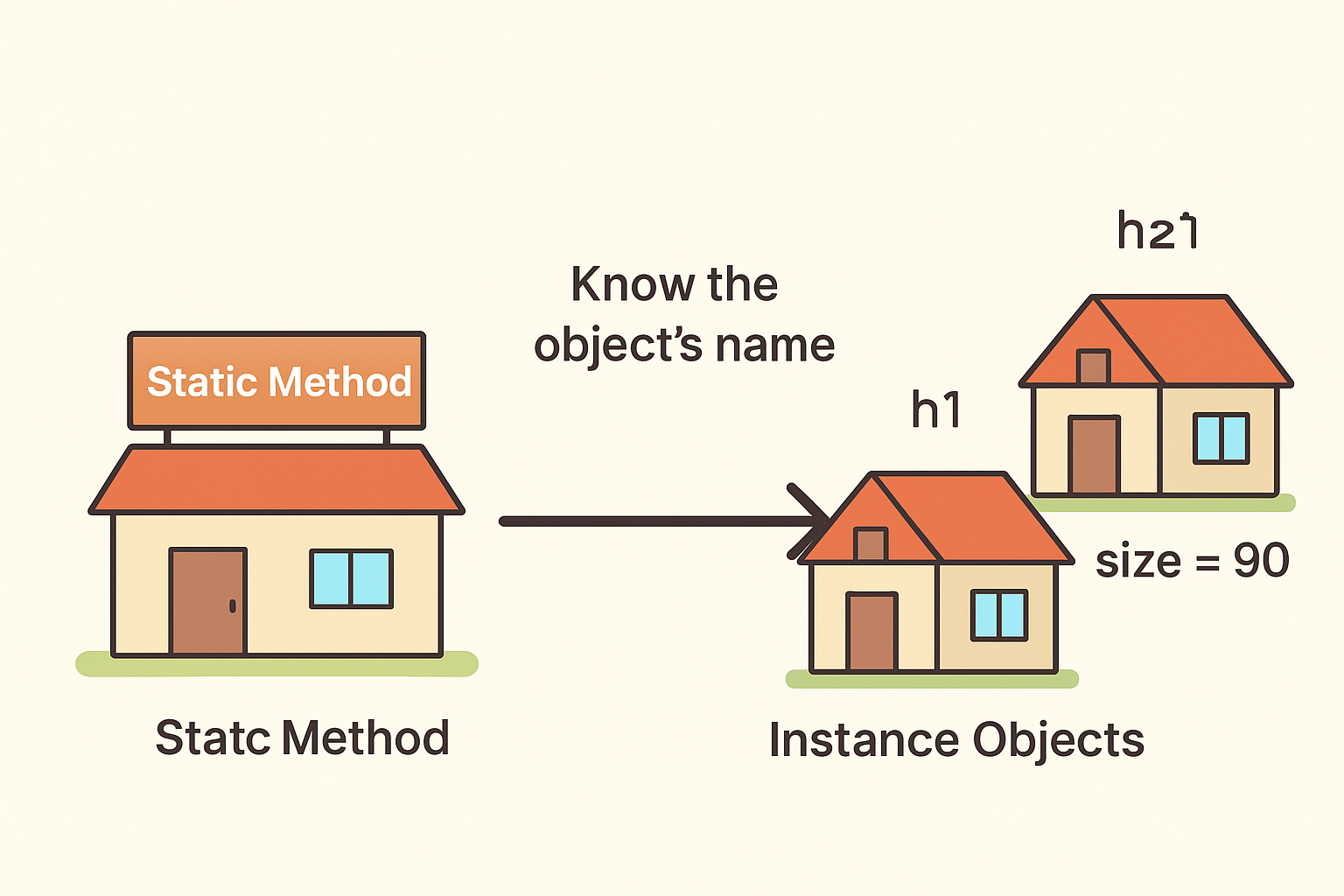
Class T{

static int i = 0;

}这也是为什么你无法在静态方法中不用类的对象访问成员变量

因为static方法和类一样，类本身不依赖对象，所以没有对象他不知道成员对象在哪

调用static变量:1.类名.变量名 2.通过对象调用 :(

类似的逻辑适用于静态方法

| **类型** | **位置** | **比喻** |
| --- | --- | --- |
| 实例变量 | 在对象的房间里 | **每个对象有自己的储物柜** |
| 静态变量 | 在类的大堂 | **整个类共用的公告栏** |
| 局部变量 | 在方法的桌子上 | **你当前工作的笔记本** |

4

5.4 for-in语法

for(float x :arr) 会逐一输出arr中的内容（将x=arr[i]）

for(;;) and while(true)均为死循环

5.7

标签是以冒号结尾的标志符，其唯一的放置地方恰好就是迭代语句之前

6.1 构造器需要初始化

Java自动调用构造器以满足对象的初始化

（再次强调，创建对象这里十分重要，关系到你对Java的理解，务必熟悉）

new ClassName();给对象分配存储空间，然后调用这个类的构造器，其会保证这个对象在可用前就被初始化了

还有一点需要注意的是，类名虽然和构造器名相同，但是构造器是一种特殊的方法

6.2（多态中重要的一节）:注意方法重载的意义是在实践上

方法重载有两个原因：1.对于同属一个类型的操作（wash，drive……），省略唯一标识符

2.你的类需要不同方式（注意这里不是对象的本质不同，而是在实践时的不同）创建对象

e.g.我们的类需要两种方法构造对象：1.默认方法构造2.利用传入的文件里面读取信息初始化

方法的重载有且仅有参数列表不同（参数的顺序不同也可以，但这种方法不好）

基本类型可以从较小类型自动提升到较大类型（程序很好理解，看完程序就能理解），其中char类型在没有精确匹配时编程int类型

如果传入的数据类型的方法参数类型更宽，(基本数据类型)x 强制转换

6.4

我们在调用同一个方法的时候可能会用不同的对象a，b，Java中方法里面有一个隐藏参数peel(a,1)（这里“a,”是隐藏参数，但是你写代码的时候不能这么写）

想要在方法体内调用的对象，this表示对该对象的引用

a.makefriends(b); 如果在makefriends方法体内有this.name 它指向的就是a对象

同理，那么后面的return this返回的就是调用这个方法的对象的引用

将自身传递给外部方法，就需要使用this

注意这一段代码： 我们来一步步解释它

new Person().eat(new Apple());

我们先创建了一个Person对象（具体语法细节不需要管），然后调用eat方法，传入一个新建的Apple，其传入到eat方法中，然后用apple对象调用getPeeled方法，那这个时候，this就是调用这个方法的对象（apple），然后其被传入Peel里面。

所以说this因对象调用自己的实例方法而生

在构造器内调用构造器：this()

它会调用()中相同参数列表的构造器。 注意它只能在构造器内且必须是第一行

this.参数名是代表方法体内部的参数的（与成员属性重名的时候）

this只能在非静态方法中使用，你不能从静态方法内部调用非静态方法（反过来可以）。你可以在没有创建对象时，直接通过类本身调用一个静态方法。

6.5

Java的Garbage collector只能清理new创建的对象，防止特殊情况，我们有finalize()

6.6

6.7

无论创建了多少对象，静态数据都只有一份储存空间

我们注意输出里面的Cupboard()上下。它一共出现了三次，第一次上面有bowl(4&5)但后面就没有了，这就是静态字段和类一样，如果不创建某类的对象并且也不引用这些静态字段，那么它们永远不会创建。而且它们仅会在第一个类对象被创建（或第一次访问静态数据）时被初始化，然后便再也不会初始化了。

static Table table = new Table();->加载Table类->static Bowl bowl1/2 = new Bowl(1/2);

&Table构造器->new Cupboard();->执行实例初始化：Bowl(3)&执行构造器：Cupboard()，f1(2)

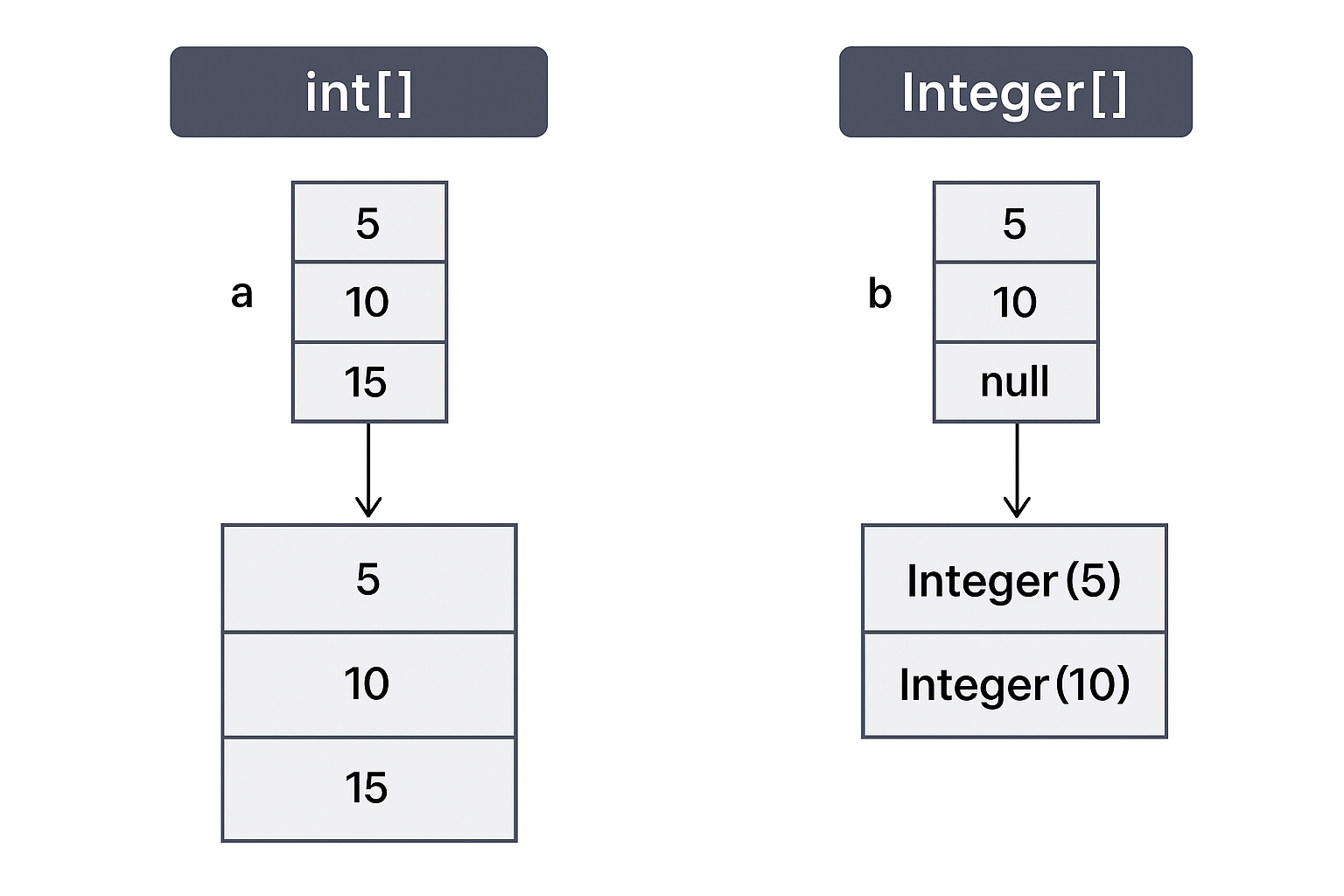
初始化的顺序是从静态字段开始的，首先StaticInitialization类加载，然后初始化它的静态字段，最后加载main方法

假设一个T类：1.由于构造器是静态方法，第一次创建类型为T的对象时，或第一次访问类T的静态方法\字段时，Java解释器会定位T.class文件 2.T.class被加载后，它的所有静态初始化都会执行（唯一一次） 3.使用new T()创建对象时，首先在Heap上为T对象分配空间 4.这块空间被清空，然后自动将该T对象中所有基本类型设为默认值，引用则为null 5.执行所有出现在字段定义处的初始化

静态子句（静态块）：放置一个类中多个静态初始化语句的地方

6.8

也许你会对引用数组这个名字很陌生：引用数组（Integer[]）顾名思义就是存放对象的引用的（即指针），它只负责指向对象而不是存储对象



此外，指针取出对应的值应该用valueOf()

至于print()为什么不会输出指针的地址而是数，是因为方法重载

可变（数量）参数列表：由于所有类最终都继承自公共根类Object，所以可以创建一个接受Object数组的方法

6.9枚举类型enum

pubilc enum S{AWA,QWQ,TOT};因为是常量所以惯例大写

S face = S.AWA;创建一个引用，然后分配给某个实例

System.out.println(face);就会输出AWA

6.10

类型推断:在一个局部定义中（即在方法体内），编译器可以自动发现类型

var不能用于字段，方法

7.1

一个Java源代码文件就是一个编译单元（转译单元），在编译单元内只能有一个public类，如果该编译单元有其他类，包之外是看不到的

JAR:Java的一个可运行程序就是一堆.class文件，可以使用jar归档期将它们打包并压缩为JAR

通过import可以引用该package内部的的public类，或者正常创建对象，然后包名.类名调用

防止package重名主要是两种方法：1.拥有自己的域名or用你的名字 2.将package名解析为电脑上的一个目录 （e.g.CLASSPATH（环境变量）\com\bilibili\zeqwq）

环境变量是D:\Jvav；C:\JAVA\Lib，包名com.bilibili.zeqwq

JAR文件的类路径需要包含JAR的文件名D:\Java\awa.jar

你的 .class 文件不一定非得放在当前目录；

JVM 只要知道“去哪儿找它们”，程序就能运行；

这个“去哪儿找”的信息，就是 **类路径（classpath）**；

.jar 文件是一种“把很多类装进一个包”的方式；

因此只要类路径正确，JVM 能找到类，**它们放哪儿都行**。

你只需要告诉Java类路径是什么就行了（具体的方式后面会说）

如果导入的两个包有相同的名称出现编译错误时，需要包名.类名

自制工具库：如果我们创建了一个Range类，如果我们要调用他，需要把Range.java放到CLASSPATH的子目录的package下，如果用import static语句，则可以在系统的任意位置使用

7.2

：包访问，包内所有其他类都可以访问，但是包外的所有类都不可以

Cookie.java生成的类文件必须置于名为d的子目录中，该子目录位于hiding目录下，hiding目录必须位于CLASSPAtH指定的路径之一

package hiding.d;

public class Cookie(){

System.out.print(“awa”);void bite(){System.out.print(“yummy”);}

}

在别的包里面，你写了import hiding.d.\*; Cookie x = new Cookie(); x.bite();

这时候你会发现x.bite();报错，因为注意到Cookie类里面bite()是包访问的

默认包：如果几个文件在同一个目录下，而且没有明确包名，Java会将这些文件看做属于该目录的“默认包”的隐含部分

private：只有包含该成员的类自己可以访问，（注意P153的代码，return new Sundae();返回的是对象，这里就可以看出，无参构造器被private修饰，而我们又写了一个返回Sundae对象的方法体，（注意到正是因为它们在同一个类里面所以可以调用）于是我们就可以控制对象的创建方式，防止特定的构造器被调用）

我们会发现，如果一个方法是仅仅为了完成这个类自己的某个操作而设计的话，我们最好将它设置为private，以防止意外的调用

protected：在包访问的基础上，仅添加上继承的子类可以访问

7.3 Interface

访问控制常常被称为实现隐藏，将数据和方法包装在类中，并与实现隐藏结合，称为封装，其结果就是具有特征和行为的数据类型（这也是笔者之前说过的写Java就是写小说，舞台剧）

访问边界的设置是为了使用户知道自己能用什么不能用什么，也可以允许你在不破坏客户端的代码时候，自由更改非public代码

我们规定一种好习惯：public、protected、private依次从上到下写

7.4

编译单元里可以没有public类，这个时候就可以随意的明明编译单元了

对于非内部类（后面会说）只能public或者“空”修饰，如果想防止该类被访问，将构造器设为private，从而防止该类对象被访问，但是[你可以在这个类的static方法中创建对象](#private构造器)，而且如此操作的好处就是，利用static创建的对象是单例模式，（任何由他创建的对象都是连接到同一个引用上面）而且仅能用public方法获取对象

7.5

JDK9以前，Java程序依赖整个Java库，这意味着最简单的程序也会带有大量未使用的库代码。还有部分程序员使用“reflect”的方法访问底层Java的代码，这导致对Java库的更新很困难，于是在这之后发布了moudle

他可以将代码清晰的划模块，然后申明它们所依赖的模块，并定义导出哪些组件不可用

Jigsaw项目将JDK库分拆为多个平台模块，现在使用库组件时，仅仅会得到该组件的模块以及依赖项。如有必要，使用EscapeHatch来访问隐藏库组件（风险较高）

最后，你的访问控制要取决于应用的场景