第 5 章 信息隐藏和信息泄露

第 4 章认为模块应该是深的。本章及随后的几个章节讨论了创建深模块的技术。

5.1 信息隐藏

实现深模块最重要的技术之一是 *信息隐藏*。该想法最早是由 David Parnas 在一篇经典的论文 [1] 中提出的。基本思想是每个模块应封装一些知识 [2]，这些知识代表设计决策。该知识嵌入在模块的实现中，但不会出现在其接口中，因此其他模块不可见。

隐藏在模块中的信息通常包含有关如何实现某种机制的详细信息。以下是一些信息可能隐藏在模块中的示例：

- 如何在 B 树中存储信息，以及如何高效地访问它。
- 如何识别文件中每个逻辑块相对应的物理磁盘块。
- 如何实现 TCP 网络协议。
- 如何在多核处理器上调度线程。
- 如何解析 JSON 文档。

隐藏的信息包括与该机制相关的数据结构和算法。它也可以包含较低层级的详细信息（例如页面大小），还可以包含更抽象的较高层级的概念，例如假设大多数文件是较小的。

信息隐藏从两个方面降低了复杂性。首先，它简化了模块的接口。接口以更简单、更抽象的方式反映了模块的功能，并隐藏了细节。这减少了使用该模块的开发人员的认知负荷。例如，使用 B 树类的开发人员不需要考虑树节点的理想扇出（fanout: 指的是每个节点允许的最大子节点数），也不需要考虑如何保持树的平衡。其次，信息隐藏使系统更容易扩展。如果隐藏了一段信息，那么在包含该信息的模块之外就不存在对该信息的依赖，因此与该信息相关的设计变更将只影响一个模块。例如，如果 TCP 协议发生了变化（例如引入一种新的拥塞控制机制），协议的实现就必须进行修改，但是在使用 TCP 发送和接收数据的更高层级的代码中不需要进行任何修改。

设计新模块时，应仔细考虑可以在该模块中隐藏哪些信息。如果可以隐藏更多信息，您就应该能够简化模块的接口，这会使模块更深。

注意：通过声明私有变量和私有方法来隐藏类中的变量和方法与信息隐藏不是同一回事。私有元素可以帮助隐藏信息，因为它们无法从类外部直接被访问。但是，私有属性仍可以通过公共方法（如 getter 和 setter 方法）公开。在这种情况下，私有属性的性质和用法就如同公有属性一样是公开的。

信息隐藏的最佳形式是将信息完全隐藏在模块中，从而使该信息对模块的用户无关且不可见。但是，隐藏部分信息也是有价值的。例如，如果某特性或信息只被少数类使用，并且只通过单独的方法访问，那么在最常见的场景中这些信息是不可见的，所以它们在大部分情况下也是隐藏的。这与将信息暴露给所有类使用者相比，这种方式产生的依赖更少。

5.2 信息泄露

信息隐藏的反面是信息泄露。当一个设计决策反映在多个模块中时，就会发生 *信息泄露*。这在模块之间创建了依赖关系：对该设计决策的任何更改都将要求对所有涉及的模块进行更改。如果一条信息反映在模块的接口中，根据定义，该信息已经泄露；因此，更简单的接口往往会隐藏更多的信息。但是，即使信息未出现在模块的接口中，也可能会泄露信息。假设两个类都具有特定文件格式的知识（也许一个类读取该格式的文件，而另一个类写入它们），即使两个类都不在其接口中公开该信息，但它们都依赖于文件格式：如果文件格式被更改，则两个类都将需要修改。像这样的后门泄露比通过接口泄露更为严重，因为它的隐蔽性更强。

信息泄露是软件设计中最重要的危险信号之一。作为一个软件设计师，你能学到的最好的技能之一就是对信息泄露的高度敏感性。如果您在类之间发现信息泄露，请自问"我如何才能重新组织这些类，使这些特定的知识只包含在一个类中呢？"如果受影响的类相对较少，并且它们与泄露的信息紧密相关，那么将它们合并到一个类中可能是有意义的；另一种方法是将信息从所有受影响的类中提出来，并创建一个新类来封装这些信息。但是，这种方法只有在你能找到一个能够抽象掉所有细节的简单接口时才有效。如果新类通过其接口公开了大部分知识，那么这么做的价值也不大（您只不过是用接口泄露取代了后门泄露）。

 危险信号：信息泄露 

> 当在多个地方使用相同的知识时，就会发生信息泄露，例如上文中提到的两个都依赖特定文件格式类型的类。

5.3 时间顺序分解

一种我称之为 *时间顺序分解* 的设计风格是导致信息泄露的常见原因。在时间顺序分解中，系统的结构对应于操作发生的时间顺序。考虑一个应用程序：它读取特定格式的文件，修改文件内容，然后再次将文件写入。通过时间顺序分解，该应用程序可能被分解为三个类：一个类用于读取文件，另一个类用于执行修改操作，还有一个类用于写入新版本的文件。文件读取和文件写入步骤都有文件格式相关的知识，这会导致信息泄露。解决方案是将用于读写文件的核心机制合并到一个类中，该类将在应用程序的读取和写入阶段使用。因为在编写代码时通常会想到操作的执行顺序，所以很容易陷入时间顺序分解的陷阱。然而，大多数设计决策会在应用程序的生命周期内的多个不同时间点显现；因此，时间顺序分解经常导致信息泄露。

顺序固然很重要，所以它会在应用程序中有所体现。但是，除非该结构与信息隐藏保持一致（不同执行阶段使用完全不同的信息），否则不应将其反映在模块结构中。**在设计模块时，应专注于执行每个任务所需的知识，而不是任务的执行顺序。**

 危险信号：时间分解 

> 在时间顺序分解中，执行顺序反映在代码结构中：在不同时间发生的操作在不同的方法或类中。如果相同的知识在不同的执行点使用，它会在多个位置被编码，从而导致信息泄露。

5.4 示例：HTTP 服务器

为了阐述信息隐藏中的问题，我们可以参考在软件设计课程中，学生在实现HTTP协议时所做出的设计决策。分析他们做得好的方面以及遇到困难的地方是非常有帮助的。

HTTP 是 Web 浏览器用来与 Web 服务器通信的机制。当用户单击 Web 浏览器中的链接或提交表单时，浏览器使用 HTTP 通过网络将请求发送到 Web 服务器。服务器处理完请求后，会将响应发送回浏览器。该响应通常包含要显示的新网页。HTTP 协议指定了请求和响应的格式，两者均以文本形式表示。图 5.1 显示了描述表单提交的 HTTP 请求示例。在该课程中，学生们被要求实现一个或多个类，以使 Web 服务器可以轻松地接收传入的 HTTP 请求并发送响应。

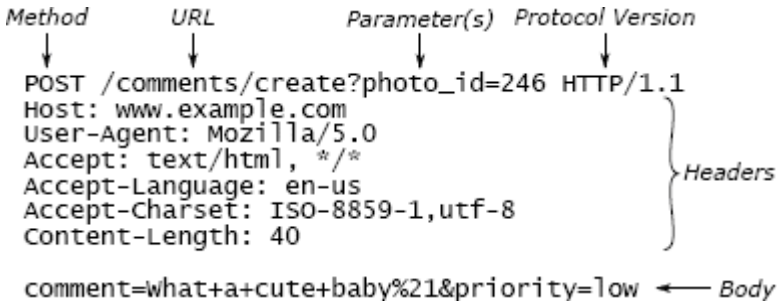

图 5.1：HTTP 协议中的 POST 请求包含通过 TCP 套接字发送的文本。每个请求都包含一个初始行、一个由空行终止的标头（Header）集合以及一个可选的请求体（Body）。初始行包含请求类型（POST 用于提交表单数据），指示操作（`/comments/create`）和可选参数（`photo_id` 的值为 246）的 URL，以及发送方使用的 HTTP 协议版本。每个标头行由一个名称（例如 `Content-Length`）及其后的值组成。对于此请求，请求体包含了其他的参数（备注和优先级）。

5.5 示例：太多的类

学生们最常犯的错误是将他们的代码分成大量的浅类，这导致了类之间的信息泄露。有一个小组使用了两个不同的类来接收 HTTP 请求：第一个类将来自网络连接的请求读取为字符串，第二个类解析该字符串。这是时间顺序分解的一个示例（“首先读取请求，然后解析它”）。发生信息泄露是因为不解析消息就无法读取 HTTP 请求。例如，`Content-Length` 标头指定了请求体的长度，因此必须对标头进行解析才能计算总的请求长度。结果，这两个类都需要了解 HTTP 请求的大部分结构，并且解析代码在两个类中都是重复的。这种方法也给调用方带来了额外的复杂性，他们在接收请求时必须以特定的顺序调用不同类中的两个方法。

由于这些类共享大量信息，因此最好将它们合并为一个同时处理请求读取和解析的类。这样便将请求格式的所有知识隔离在一个类中，提供了更好的信息隐藏，并且还为调用者提供了一个更简单的接口（只需要调用一个方法）。

此示例说明了一个软件设计中的通用主题：**通常可以通过使类稍大一些来改善信息隐藏。**这样做的一个原因是将与特定功能相关的所有代码（例如解析 HTTP 请求）组合在一起，以便生成的类包含与该功能相关的所有内容。增加类大小的第二个原因是提高接口的级别。例如，与其为计算的三个步骤中的每一个步骤使用单独的方法，不如使用一个方法来执行整个计算。这样可以简化接口。这两个好处都适用于上一段的示例：组合类将与解析

HTTP 请求相关的所有代码组合在一起，并且用一个方法替换了原来的两个外部可见的方法。组合后的类比原有的类都更深。

当然，过度扩大类的范围也是可能的（例如整个应用程序都包在一个类里）。第 9 章将讨论把代码分成多个较小的类的合理条件。

5.6 示例：HTTP 参数处理

服务器收到 HTTP 请求后，服务器需要访问该请求中的某些信息。处理图 5.1 中的请求的代码可能需要知道 `photo_id` 参数的值。参数可以在请求的第一行中指定（图 5.1 中的 `photo_id`），有时也可以在请求体中指定（图 5.1 中的 `comment` 和 `priority`）。每个参数都有一个名称和一个值。参数的值使用一种称为 URL 编码的特殊编码。例如，在图 5.1 中的备注值中，`+` 代表空格字符，`%21` 代表 `!`。为了处理请求，服务器需要某些参数值的解码形式。

关于参数处理，大多数学生项目都做出了两个不错的选择。首先，他们认识到服务器应用程序不在乎是否在标头行或请求体指定了参数，因此他们对调用者隐藏了这种区别，并将两个位置的参数合并在一起。其次，他们隐藏了 URL 编码的知识：HTTP 解析器在将参数值返回到 Web 服务器之前先对其进行解码，以便图 5.1 中的 `comment` 参数的值将返回 `What a cute baby!`，而不是 `What+a+cute+baby%21`。在这两种情况下，信息隐藏都使得 HTTP 模块的 API 更加简单。

但是，大多数学生使用的返回参数的接口太浅，这导致丢失了 *信息隐藏* 的机会。大多数项目使用 `HTTPRequest` 类型的对象来保存已解析的 HTTP 请求，并且 `HTTPRequest` 类提供了类似如下的方法来返回参数：

```
public Map<String, String> getParams() {
    return this.params;
}
```

该方法不是返回单个参数，而是返回了内部用于存储所有参数的映射（Map）的引用。这个方法是浅的，它公开了 `HTTPRequest` 类用来存储参数的内部实现。对该实现的任何更改都将导致接口的更改，这将需要对所有调用者进行修改。在修改实现时，更改通常涉及关键数据结构表示的更改（例如为了提高性能）。因此，尽量避免暴露内部数据结构是很重要的。这种方法还导致了调用者的更多工作：调用者必须首先调用 `getParams`，然后必须调用另一个方法来从映射中检索特定的参数。最后，调用者必须意识到他们不应该修改 `getParams` 返回的映射，因为这会影响 `HTTPRequest` 的内部状态。

这是一个用于检索参数值的更好的接口：

```
public String getParameter(String name) { ... }

public int getIntParameter(String name) { ... }
```

`getParameter` 以字符串形式返回参数值。它提供了一个比上面的 `getParams` 更深的接口。更重要的是，它隐藏了参数的内部实现。`getIntParameter` 将参数的值从 HTTP 请求中的字符串形式转换为整数（例如，图 5.1 中的 `photo_id` 参数）。这使调用者不必单独请求字符串到整数的转换，并且对调用者隐藏了该机制。如果需要，可以定义更多其他数据类型的方法，例如 `getDoubleParameter`。（如果所需的参数不存在，或者无法将其转换为所请求的类型，则所有这些方法都将抛出异常；上面的代码中省略了异常声明）。

5.7 示例：HTTP 响应中的默认值

HTTP 项目还必须提供对生成 HTTP 响应的支持。学生在该领域中最常见的错误是默认值不足。每个 HTTP 响应必须指定一个 HTTP 协议版本。有一个小组要求调用者在创建响应对象时明确指定此版本。但是，响应版本必须与请求对象中的版本相对应，并且在发送响应时一定已经将请求作为参数传递（它指示将响应发送到何处）。因此，HTTP 类自动提供响应版本更有意义。调用者不太可能知道要指定哪个版本，并且如果调用者确实指定了一个值，则可能导致 HTTP 库和调用者之间的信息泄露。HTTP 响应还包括一个日期标头，用于指定发送响应的时间，HTTP 库也应该为此提供一个合理的默认值。

默认值体现了设计接口时应使常见情况尽可能简单的原则。它们还是隐藏部分信息的一个示例：在正常情况下，调用者无需知道默认值的存在。在极少数情况下，调用方需要覆盖默认值，它才需要知道该值，并且可以调用特殊方法来对其进行修改。

只要有可能，类就应该“做正确的事”，而无需明确要求。默认值就是一个例子。第 4.7 节的 Java I/O 示例以负面方式说明了这一点。大家都希望在文件 I/O 中缓冲，以至于没有人需要明确要求它，甚至不知道它的存在。I/O 类应该做正确的事情并自动提供它。最好的功能是那些您甚至不知道它们存在的功能。

 危险信号：过度暴露 

> 如果一个常用特性的 API 迫使用户了解其他很少使用的特性，这将增加不需要使用这些特性的用户的认知负荷。

5.8 类内部的信息隐藏

本章中的信息隐藏示例着重于类的外部可见 API，但是信息隐藏也可以应用于系统中的其他层级，比如类的内部。可以尝试在类中设计私有方法，使得每个方法都封装一些信息或能力，并将其对类的其余部分隐藏。此外，请尽量减少每个实例变量的使用位置数量。有些变量可能需要在整个类中广泛使用，但是其他变量可能只需要在少数地方使用；如果可以减少使用变量的位置数量，则将消除类内的依赖关系并降低其复杂性。

5.9 做过头了

信息隐藏只有在被隐藏的信息在模块外部不需要时才有意义。如果模块外部需要该信息，则不得隐藏它。假设模块的性能受某些配置参数的影响，并且模块的不同用途将需要对参数进行不同的设置。在这种情况下，将参数暴露在模块的接口中很重要，以便可以对其进行适当的调整。作为软件设计师，您的目标应该是最大程度地减少模块外部所需的信息

量。例如，如果模块可以自动调整其配置，那将比公开配置参数更好。但是，重要的是要识别模块外部需要哪些信息，并确保将其公开。

5.10 结论

信息隐藏和深模块密切相关。如果模块隐藏了很多信息，则往往会增加模块提供的功能，同时还会减少其对外接口数量。这使得模块更深。相反，如果一个模块没有隐藏太多信息，则它要么功能不多，要么接口复杂。无论哪种方式，模块都是浅的。

将系统分解为模块时，请尽量不要受运行时操作顺序的影响，否则您将沿着时间顺序分解的错误道路前进，这将导致信息泄露和浅模块。相反，请考虑执行应用程序的任务所需的不同知识，并在设计每个模块时封装这些知识中的一个或几个。这样将产生一个整洁和简单的深模块设计。

[1] David Parnas，“关于将系统分解为模块的标准”，ACM 通讯，1972 年 12 月。

[2] 译者注：关于知识（knowledge）可以了解“最少知识原则”的设计原则。