第 17 章 一致性

一致性是一个强大的工具，可以降低系统复杂性并使其行为更容易理解。如果系统是一致的，则意味着相似的事情以相似的方式完成，而不同的事情则以不同的方式完成。一致性创造了认知杠杆：一旦您了解了某个地方的工作方式，就可以使用该知识立即了解其他使用相同方式的地方。如果系统没有以一致的风格实现，则开发人员必须分别了解每种情况，这将花费更多的时间。

一致性减少了失误。如果系统不是一致的，两种实际上不相同的情况可能看起来是一样的。开发人员可能会看到一个看起来很熟悉的模式，并根据以前遇到的模式做出错误的假设。另一方面，如果系统是一致的，则基于看起来很熟悉的情况所做的假设就会很安全。一致性允许开发人员更快速的工作，还能减少失误。

17.1 一致性示例

一致性可以应用于系统中的许多层级，这里是一些例子。

名称。 第 14 章已经讨论了以一致的方式使用名称的好处。

编码风格。 如今，开发组织通常会制定编码风格指南，在编译器所强制执行的规则之外对程序进行额外的结构限制。现代的风格指南解决了一系列问题，例如缩进、花括号放置、声明顺序、命名、注释以及对认为危险的语言特性的限制。风格指南使代码更易于阅读，并且可以减少某些类型的错误。

接口。 具有多种实现的接口是一致性的另一个示例。一旦了解了接口的其中一种实现，其他的实现都将变得更易于理解，因为您已经知道它将会提供的特性。

设计模式。 设计模式是某些常见问题的普遍接受的解决方案，例如用于用户界面设计的模型-视图-控制器（MVC）方法。如果您可以使用现有的设计模式来解决问题，则实现会更快地进行，也更有可能奏效，并且您的代码对读者来说也会可读性更好。设计模式将在第 19.5 节中详细讨论。

不变量。 不变量是参数值或结构的属性，它总是不变的。例如，存储文本行的数据结构可能会强制要求每行始终以换行符终止。不变量减少了代码中必须考虑的特殊情况，并且更容易推断代码的行为。

17.2 确保一致性

一致性很难保持，尤其是当许多人长时间从事一个项目时。一个小组的人可能不了解另一小组中建立的约定。新来的人不了解约定，因此他们无意间违反了约定，并创建了与现有约定冲突的新约定。以下是建立和保持一致性的一些技巧：

文档。 创建一个文档，并列出最重要的总体约定，例如编码风格指南。将文档放置在开发人员容易会看到的位置，例如项目维基（Wiki）的显要位置。鼓励新加入小组的成员来阅读这些文档，并鼓励现有的团队人员时不时对文档进行审查。一些来自不同组织的风格指南已经在网上发布；考虑从其中之一开始。

对于那些更加局部的约定，例如不变量，请在代码中找到合适的位置进行记录。如果您不把这些约定写下来，那么其他人也不太可能会遵循它们。

强制执行。 即使有好的文档，开发人员也很难记住所有约定。强制执行约定的最佳方法是编写一个检查违规的工具，并确保代码在通过检查之前不能提交到存储库。自动检查对于低层级的语法约定特别有用。

我最近的一个项目有行终止符的问题。一些开发人员在 Unix 上工作，行由换行符终止；其他的人工作在 Windows 上，行通常由回车符加上换行符来终止。如果某个系统上的开发人员对先前在另一个系统上编辑过的文件进行了编辑，那么编辑器有时会将所有行终止符替换为适合该系统的行终止符。这给人的感觉是文件的每一行都被修改了，也就让人很难追踪有意义的变化。我们建立了一个约定，即文件应该只包含换行符，但是很难确保每个开发人员使用的每个工具都遵循这个约定。每当一个新的开发人员加入这个项目，我们都会遇到大量的行终止符问题，直到开发人员适应了这个约定。

我们最终通过编写了一个简短的脚本解决了这个问题，该脚本会在变更提交到源代码存储库之前自动执行。该脚本检查所有已修改的文件，如果其中任何一行包含回车符，则中止提交。该脚本也可以手动运行，通过用换行符替换回车符加换行符来修复损坏的文件。这一下子就消除了问题，并且还有助于培训新的开发人员。

代码审查为强制执行约定和向新的开发人员培训这些约定提供了另一个契机。代码审阅者越挑剔，团队中的每个人学习约定的速度就越快，并且代码越清晰。

入乡随俗。 所有约定中最重要的约定是每个开发人员都应遵循古老的格言：“到了罗马，就按罗马人那样做。” 在编写新代码时，请先看看现有的代码是如何组织的。是否在私有变量和方法之前声明了所有的公有变量和方法？方法是否按字母顺序排列？变量是像 `firstServerName` 那样使用驼峰命名法，还是像 `first_server_name` 中那样使用蛇形命名法？当您看到任何看起来可能是约定的内容时，请遵循该约定。在做出设计决策时，请问自己是否有可能在项目的其它地方做出了类似的决策；如果是这样，请找到一个现有示例，并在新代码中使用相同的方式。

不要改变现有约定。 抵制“改善”现有约定的冲动。**有一个“更好的想法”并不是引入不一致的充分借口。** 您的新想法可能确实更好，但是一致性胜过不一致的价值几乎总是大于一种方法胜过另一种方法的价值。在引入不一致的行为之前，请问自己两个问题。首先，您是否拥有在建立旧约定时还不存在的全新信息来支持使用新的方法？其次，新的方法是否好到值得花时间去更新所有旧的用法？如果您的组织同意这两个问题的回答均为“是”，那么就去进行更新；当您完成后，应该不会有任何旧约定的痕迹留下来。然而，您仍然面临着其他开发人员不了解新约定的风险，因此他们将来可能会重新引入旧的方法。总体而言，重新考虑已建立的约定很少会是对开发人员时间的良好利用。

17.3 做过头了

一致性不仅意味着相似的事情应该以相似的方式完成，而且不同的事情也应该以不同的方式完成。如果您对一致性过于热衷，并试图对不同的事物强制采用相同的方法，例如对确实不同的事物使用相同的变量名，或者对不适合该模式的任务使用现有的设计模式，那只会造成复杂性和混乱。一致性只有在开发人员确信“如果它看起来像 `x`，它确实是 `x`”时才

会带来好处。

17.4 结论

一致性是投资思维的另一个例子。确保一致性的工作将需要一些额外的工作：决定使用什么约定、创建自动检查程序、寻找旧代码中的类似情况以在新代码中模仿、在代码审查中培训团队成员。这项投资的回报是您的代码可读性更好。开发人员将能够更快和更准确地了解代码的行为，这将使他们能够更快地工作，并引入更少的缺陷。