C ：最强大的.NET语言之内存与资源 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/253/2021_2022_C___EF_BC_9A_E6_9C_80_E5_c97_253832.htm 内存管理 本地C 为程序员提供了超越内存管理的直接控制能力，在堆栈上分配一个对象，意味着只有在进入特定函数时，才会为对象分配内存，而当函数返回或堆栈展开时，内存被释放。可使用操作符new来动态地为对象分配内存，此时内存分配在CRT堆中，并且需要程序员显存地对对象指针使用操作符0delete，才能释放它。这种对内存的精确控制，也是C 可用于编写极度高效的程序的原因之一，但如果程序员不小心，这也是内存泄漏的原因。另一方面，你不需要求助于垃圾回收器来避免内存泄漏--实际上这是CLR所采取的方法，而且是一个非常有效的方法，当然，对于垃圾回收堆，也有其他一些好处，如改进的分配效率及引用位置相关的优势。所有这一切，都可以在C 中通过库支持来实现，但除此之处，CLR还提供了一个单一的内存管理编程模型，其对所有的编程语言都是通用的，想一想与C 中COM自动化对象相交互和调度数据类型所需做的一切工作，就会发现其重要意义所在--横跨数种编程语言的垃圾回收器，作用是非常巨大的。 为了效率，CLR也保留了堆栈的概念，以便值类型可在其上分配，但CLR也提供了一个newobj中间语言指令，以在托管堆中分配一个对象，但此指令只在C#中对引用对象使用操作符new时提供。在CLR中，没有与C 中的0delete操作符对应的函数，当应用程序不再引用某对象时，分配的内存最后将由垃圾回收器回收。 当操作符new应用于引用类型时，托管C 也会生成newobj指令，当然

，对此使用0delete操作符是不合法的。这确实是一个矛盾，但同时也证明了为什么用C 指针概念来表示一个引用类型不是一个好的做法。 在内存管理方面，除了上述在对象构造一节讨论过的内容，C /CLI没有提供任何新的东西；资源管理，才是C /CLI的拿手好戏。 资源管理 CLR只有在资源管理方面，才能胜过本地C 。Bjarne Stroustrup的"资源获取即初始化"的技术观点，基本定义了资源类型的模式，即类的构造函数获取资源，析构函数释放资源。这些类型是被当作堆栈上的局部对象，或复杂类型中的成员，其析构函数自动释放先前分配的资源。一如Stroustrup所言"对垃圾回收机制来说，C 是最好的语言，主要是因为它生成很少的垃圾。" 也许有一点令人惊讶，CLR并没有对资源管理提供任何显式运行时支持，CLR不支持类似析构函数的C 概念，而是在 .NET Framework中，把资源管理这种模式，提升到一个IDisposable核心接口类型的中心位置。这种想法源自包装资源的类型，理应实现此接口的单一Dispose方法，以便调用者在不再使用资源时，可调用该方法。不必说，C 程序员会认为这是时代的倒退，因为他们习惯于编写那些缺省状态下清理就是正确的代码。 因为必须要调用一个方法来释放资源，由此带来的问题是，现在更难编写"全无异常"的代码了。因为异常随时都可能发生，你不可能只是简单地在一段代码后，放置一个对对象的Dispose方法的调用，这样做的话，就必须要冒资源泄漏的风险。在C#中解决这个问题的办法是，使用try-finally块和using语句，在面对异常时，可提供一个更可靠的办法来调用Dispose方法。有时，构造函数也会使用这种方法，但一般的情况是，你必须要记住手工编写它们，如果忘记了，生

成的代码可能会存在一个悄无声息的错误。对缺乏真正析构函数的语言来说，是否需要try-finally块和using语句，还有待论证。using (SqlConnection connection = new SqlConnection("Database=master. Integrated Security=sspi")){ SqlCommand command = connection.CreateCommand(). command.CommandText = "sp_databases". command.CommandType = CommandType.StoredProcedure. connection.Open(). using (SqlDataReader reader = command.ExecuteReader()) { while (reader.Read()) { Console.WriteLine(reader.GetString(0)). } }} 