C 中的struct专题研究 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/133/2021_2022_C___E4_B8_AD_E7_9A_84s_c97_133856.htm C 之父Bjarn Stroustrup对C 语言概括的第一条就是"a better C"。struct这个关键字就来源于C 。而在C 中，struct的含义已经和C中的struct不尽相同了。在C 中，用户定义类型，也就是class，拥有和内建类型一样的地位。这可以从C 中struct定义的类型在声明变量时不必再写出struct关键字看出。如 struct Foo { // ... }. Foo f. 在C 程序中，人们似乎更热衷于使用class，而几乎忽略了struct的存在。实际上，struct就是成员默认为public的class（在class中，成员默认为private）。事实上以下两端代码完全等价： 代码一： struct Foo { // ... }. 代码二： class Foo { public: // ... }. 那么为什么要有struct的存在呢？首先的原因自然是保持对C的兼容。原先的C代码可以不必修改就成为合法的C 代码。第二个原因（个人愚见），是为了让struct来表示抽象的数据类型以及抽象接口，而与class所表示的类的概念相区别。 struct在C中的使用方式转移到C 以后仍然是非常重要，作为一组相关的数据而存在于一个struct，说明了他们在逻辑上是相互关联的数据，同时他们被保存在struct里而不是全局变量，也是对数据的一种管理。个人认为一种很朴素的用法要取代当前很热门的get-set用法。比如 class Foo { int bar. public: int get_bar() { return bar. } void set_bar(const int&amp. b) { bar = b. } }. 这里Foo::bar是一个需要频繁存取的数据对象，它作为类Foo的私有成员存在，而通过公共接口存取。这是面向对象思想中数据封装的体现。而考虑一下这个Foo::bar是否有必要成为私

有成员？没有，因为它就是一个数据，没有必要用私有类成员的思想来封装它，似乎可以看成是面向对象思想的过度滥用。等效的可以写成： struct Foo { int bar. }. 然后通过普通的赋值操作来完成。这样做似乎是回到了原始时代，但没有必要用的就不要用，否则还会影响效率。 其次一个struct的应用就是来描述纯虚类，也就是后来Java语言中类似接口的东西：struct Foo { virtual void Bar1() = 0. virtual void Bar2() = 0. // ... }. 使用struct可以些许节省编译器的语法分析时间：），而且能在语义上表达的更为清楚。 一般来讲，当一个类中有必要进行数据隐藏时，请用class声明，并将私有数据标记为private，公共接口标记为public；而当所有成员都有必要成为公有成员的时候，请用struct来声明它。 