瘦身前后兼谈C 语言进化(4) PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/469/2021_2022__E7_98_A6_E8_BA_AB_E5_89_8D_E5_c67_469826.htm (3)boost采用的办法也是C 98唯一的办法，就是为不同参数个数的Signature进行特化：templateclass function{R operator()(T1 a1).}.templateclass function{R operator()(T1 a1, T2 a2).}.templateclass function{R operator()(T1 a1, T2 a2, T3 a3).}.… // 再写下去页宽不够了，打住… 如此一共N（N由一个宏控制）个版本。 这种做法有两个问题：一，函数的参数个数始终还是受限的，你作出N个特化版本，那么对N 1个参数的函数就没辙了。boost::tuple也是这个问题。二，代码重复。每个特化版本里面除了参数个数不同之外基本其它都是相同的；boost解决这个问题的办法是利用宏，宏本身的一大堆问题就不说了，你只要打开boost.function的主体实现代码就知道有多糟糕了，近一千行代码，其中涉及元编程和宏技巧无数，可读性可以说基本为0。好在这是个标准库（boost.function将加入tr1）不用你维护，如果是你自己写了用的库，恐怕除了你谁也别想动了。所以第二个问题其实就是可读性可维护性问题，用Matthew Wilson的说法就是可发现性和透明性的问题，这是一个很严重的问题，许多C 现代库因为这个问题而遭到诟病。 现在，让我们来看一看加入了variadic templates之后的C 09实现：templatestruct invoker_base { virtual R invoke(Args...) = 0.virtual ~invoker_base() { } }.templatestruct functor_invoker : public invoker_base { explicit functor_invoker(F f) : f(f) { }R invoke(Args... args) { return f(args...). }private:F f.}.templateclass

function.templateclass function { public:templatefunction(F f) : invoker(0) {invoker = new functor_invoker(f).}R operator()(Args... args) const {return invoker->invoke(args...).}private:invoker_base* invoker.}. 