瘦身前后兼谈C 语言进化(2) 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_469828.htm 进化两个例子 先举一个平易近人的例子（Walter BrightD语言发明者曾在他的一个presentation中使用这个例子），如果我们想要遍历一个数组，在C里面我们是这么做（或者用指针，不过指针有指针自己的问题）：int arr[10].… // initialize arrfor(int i = 0. i {int value = arr[i].…printf} 这个貌似简单的循环其实有几个主要的问题： 1. 下标索引不应该是int，而应该是size_t，int未必能足够存放一个数组的下标。 2. value的类型依赖于arr内元素的类型，违反DRY，如果arr的类型改变为long或unsigned，就可能发生截断。 3. 这种for只能对数组工作，如果是另一个自定义容器就不行了。 在现代C 里面，则是这么做：for(std::vector::iterator iter = v.begin(). iter != v.end(). iter) {…} 其实最大的问题就是一天三遍的写，麻烦。for循环的这个问题上篇讲auto的时候也提到。 Walter Bright然后就把D里面支持的foreach拿出来对比（当然，支持foreach的语言太多了，这也说明了这个结构的高效性）。foreach(i. v) {…} 不多不少，刚好表达了意思：对v中的每个元素i做某某事情。 这个例子有人说太Na?ve了，其实我也赞成，的确，每天不知道有多少程序员写下一个个的循环结构，究竟有多少出了上面提到的三个问题呢？最大的问题恐怕还是数组越界。此外大家也都亲身体验过违反DRY原则的后果：改了一处地方的类型，编译，发现到处都是类型错误，结果一通“ 查找替换 ”是免不了的了，谁说程序员的时间是宝贵的来着？ 既然这个例子

太Nave，那就说一个不那么Nave的。Java为什么要加入closure？以C STL为例，如果我们要：transform(v1.begin(), v1.end(), v2.begin(), v3.begin(), _1 _2). 也就是说将v1和v2里面的元素对应相加然后放到v3当中去。这里用了boost.lambda，但大家都知道boost.lambda又是一个经典的鸡肋。_1 _2还算凑活，一旦表达式复杂了，或者其中牵涉到对其它函数的调用了，简直就是一场噩梦，比如说我们想把v1和v2中相应元素这样相加：f(_1) f(_2)，其中f是一个函数或仿函数，可以做加权或者其它处理，那么我们可以像下面这样写吗： transform(…, f(_1) f(_2)). 答案是不行，你得这样写：transform(…, boost::bind(std::plus(), boost::bind(f, _1), boost::bind(f, _1))).