C语言中“.”与“->”有什么区别？

從 知乎 轉載

轉載幾篇知乎上我自己的回答，因爲不喜歡知乎的排版，所以在博客裏重新排版一遍。

原問題：C语言中“.”与“->”有什么区别？

除了表达形式有些不同，功能可以说完全一样阿。那为何又要构造两个功能一样的运算符？ 效率有差异？可是现在编译器优化都那么强了，如果真是这样岂不是有些多此一举

刚刚翻了下书，说早期的C实现无法用结构直接当作参数在函数间传递，只能用指向结构的指针在函数间进行传递！我想这应该也是最直观的原因吧。

我的回答

首先 `a->b` 的含義是 `(*a).b` ，所以他們是不同的，不過的確 `->` 可以用 `*` 和 `.` 實現，不需要單獨一個運算符。 嗯，我這是說現代的標準化的 C 語義上來說， `->` 可以用 `*` 和 `.` 的組合實現。

早期的 C 有一段時間的語義和現代的 C 的語義不太一樣。

稍微有點彙編的基礎的同學可能知道，在機器碼和彙編的角度來看，不存在變量，不存在 struct 這種東西，只存在寄存器和一個叫做內存的大數組。

所以變量，是 C 對內存地址的一個抽象，它代表了一個位置。舉個例子，C 裏面我們寫：

```
1  a = b
```

其實在彙編的角度來看更像是

```
1  *A = *B
```

其中 A 和 B 各是兩個內存地址，是指針。

好，以上是基本背景。

基於這個背景我們討論一下 struct 是什麼，以及 struct 的成員是什麼。 假設我們有

```
1  struct Point {
2          int x;
3          int y;
4  };
5  struct Point p;
6  struct Point *pp = &p;
```

從現代語義上講 p 就是一個結構體對象， x 和 y 各是其成員，嗯。

從彙編的語義上講， p 是一個不完整的地址，或者說，半個地址，再或者說，一個指向的東西是虛構出來的地址。而 x 和 y 各是在 Point 結構中的地址偏移量。也就是說，必須有 p 和 x 或者 p 和 y 同時出現，才形成一個完整的地址，單獨的一個 p 沒有意義。

早期的 C 就是在這樣的模型上建立的。所以對早期的 C 而言， *pp 沒有意義，你取得了一個 struct ，而這個 struct 不能塞在任何一個寄存器裏，編譯器和 CPU 都無法表達這個東西。

這時候只有 p.x 和 p.y 有意義，它們有真實的地址。

早期的 C 就是這樣一個看起來怪異的語義，而它更貼近機器的表達。 所以對早期的 C 而言，以下的代碼是對的：

```
1  p.x = 1;
2  int *a;
3  a = &(p.x);
```

而以下代碼是錯的：

```
1  (*pp).x = 1;
```

因爲作爲這個賦值的目標地址表達式的一部分， *pp ，這個中間結果沒法直譯到機器碼。

所以對早期的 C 而言，對 pp 解引用的操作，必須和取成員的偏移的操作，這兩者緊密結合起來變成一個單獨的操作，其結果纔有意義。

所以早期的 C 就發明了 -> ，表示這兩個操作緊密結合的操作。於是纔能寫：

```
1  pp->x = 1;
```

嗯，這就是它存在的歷史原因。而這個歷史原因現在已經不重要了，現代的符合標準的 C 編譯器都知道 `(*pp).x` 和 `pp->x` 是等價的了。

說句題外話，C++ 裏面還發明了 `.*` 和 `->*` 這兩個運算符（注意 `->*` 不是單獨的 `->` 和 `*` 並排放的意思），關於爲什麼要發明這兩個運算符，而不能直接說 `a ->* b` 的意思就是 `a ->(*b)`，這個就作爲課堂作業吧。