Single Assignment 谣言

Author: Tony Arcieri
Link：http://tonyarcieri.org/articles/2008/09/30/single-assignment-myths
译者：于曦鹤[KrzyCube] krzycube AT gmail.com
译于：2008-12-04 晚

首先请允许我澄清一下这篇文章所呈现的腔调。对于 Single Assignment 的编程语言，我没有任何的反对意见。如果正在阅读这篇文章的你是一个 Single Assignment 的拥护者，请相信我绝不是在打击你对 Erlang 的好感和信心，因为我也是 Erlang 的 Fans。Single Assignment 本质上没有任何不对的地方，并且，在希望提供无类型 Lambda 表达式的编程语言中，它毫无疑问拥有一席之地。

不得不说，很多人出于对 Single Assignment 的完全不了解（原文的词更激荡：无知）而错误地夸大了它的神奇作用。我发现普遍的现象，是将一些 Erlang 本身所具有的特性，归功于 Single Assignment，而实际上它们根本没什么关系。我不知道这些现象的本源是什么，更不知道它为什么被不断地重复。只能做个猜测：一群相对而言的初学者，在没有正确理解 Erlang 的情况下，告诉另外一群初学者为什么 Erlang 使用 Single Assignment，而因为老手们其实甚至并不了解这两者中的任何一个，就将 Erlang 本身的特性与 Single Assignment 的性质混淆在一起了。

一切都是关于绑定

首先，来搞清楚什么是 Single Assignment：它是一个关于你可以如何修改函数绑定关系的约束。一个绑定是局部变量和它所代表的值的映射，并且绑定是受函数的作用域限制的。在 Erlang 中，绑定发生在比 Erlang 的并发原语粒度更小的底层。更有甚者，绑定本身也是可变的，这也是为什么 Erlang 中“变量”被称为“变量”。（ "variables" are called "variables"）
你可以随意为一个未被绑定的变量绑定一个值，但是，你无法为一个已经绑定的变量再绑定另外一个值。

这里，先再次**强制灌输**一下绑定的观点：即使是在 Single Assignment 的情况下，绑定仍然是可变的。

Multiple assignment 去掉了上述的再次绑定的限制，不过你也只是在以不同的方式改变绑定的状态，正如你在 Single Assignment 中所做的那样（指从 unbound->bound）。

现在，我将通过仔细推敲 Single Assignment 的一些谣言，来深化我的观点。

谣言 #1:你不喜欢 Single Assignment 只因为你是个菜鸟

这是很多个比较令人讨厌的谣言之一，因为它甚至算不上一个理由（Non-Argument）。它没有揭示任何 Single Assignment 的真相，相反，它直接将对 Single Assignment 的反对意见比作无知。好吧，让我先证明一下我并不是个菜鸟。我正在实现一个自己的编程言－－

Reia，它基于 Erlang Virtual Machine（BEAM）。Reia 通过实现静态单赋值转换(Static Single Assignment Transform)来做到支持 Multiple assignment。在 Reia 中，SSA 是用 Erlang 实现的，并且用上了几乎所有为人所称道的东西来规避一些 SSA 带来的问题，如：map，fold，lambdas，patter matching，以及 recursion。下面是一段来自 Reia 的代码片段，属于 reia 的 visitor 功能模块，而 Visitor 是 Reia 中 SSA 机制的根基：

```
[Type, Line | Elements] = tuple_to_list(Node),
{Elements2, {State2, _Fun}} = lists:mapfoldl(fun transform_node/2, {State, Fun}, Elements),
{ok, State2, list_to_tuple([Type, Line | Elements2])};
```

在这里我们看到了什么呢？

1.不仅仅是 fold，而是 mapfold，它组合了 map 和 fold。

2.返回值被深陷递归模式匹配的桎梏。

3.提供 transformation 的是一个 lambda。

这些都还好，最不能忍受的是：

4.**这里仍然用了土老冒的版本化变量**。（译者注：下标标注的不同变量版本，请参见 SSA 的概念）

如果仍然有人认为我在这里使用版本化变量是由于我对函数式编程的神圣符咒（juju）的无知，我恳求你帮我重写一个不用版本化变量实现的 Reia Visitor 功能模块。它在这里：http://github.com/tarcieri/reia/tree/master/src/reia/compiler/reia_visitor.erl。我可以向你保证的是，之所以使用它们，并不是因为我对函数式编程理念的匮乏。(译者注：这里原文是 paradigms，通常我们将 paradigm 译为“范式”，不过我觉得这里用“理念”更通顺)。

不过多问一句：你真的理解上面那几行代码吗？的确，它看起来非常简洁，因为它是用函数式语言的神符紧凑地组合起来的，所以它根本不用那么多的语句。可与此同时，我却觉得这令它非常难以理解，如果有人能真正理解了 Reia 的 Visitor 模块到底干了什么，我起立为他鼓掌（译者注：本来我想译“我 orz 他”），因为我自己也觉得很难理解。

说了那么多，来看一些我收到的与这个谣言一脉相承的评论，评论它们时，请参照刚才我在上面讨论的内容：

> “I think you just haven’t been exposed to enough functional languages.
> The problems you’re describing can be solved with folds.”

很抱歉，请回过头去试试我说的，重新实现一个不用版本化变量的 Reia Visitor 模块。

> "…if you are going to be adding functions to that pipeline,
> perhaps what you are actually doing is applying a list of functions to an initial value,
> in which case folding the value over the list of functions is what you want to do."

可惜，对一个命名函数做一次应用 lambda 的 fold，还是不足以消除那“不必要的手工变量版本化”。

> "I honestly think that once you get past stage of running around wailing ‘help help,
> here is something new which I have never seen before, woe is me’
> it all goeas away and you never consider again."

这个评论来自 Robert Virding，Erlang 的原创者之一（译者注：就是这个人创造了 LFE）。在这里，我要说的是，Robert 在我尝试理解 Erlang 的 group leader system 和 I/O servers 时给予了非常大的帮助，而这两个东西，在此之前，是我在实现自己的基于 BEAM 的 Shell 时碰到的主要绊脚石，非常感激，Robert！这至少说明了，我在此对于 Single Assignment 的不待见，并不是因为我在函数式编程方面很白痴，而是基于我在实践过程中真正碰到的问题。

谣言 #2: Single assignment 是 Erlang 高容错性的基础

这也是比较令人讨厌的谣言之一，因为它源自对 Erlang 的容错性的重大误解。直接来看一下一个对于我的编程语言 Reia 的评论：

"It sounds as if the actual language design has given up on fault tolerance" (context)

很明显发表这个评论的人，甚至都没有经过仔细的思考，不过这只是比较普通的。在这之后才是幽默的开始：Single assignment 是 Erlang 的容错性的基础。好吧！好吧！让我来说明一下，Single assignment 到底是分管哪片儿的：它规避了一种特定的人为错误（Programmer Error），那就是，通常程序员会对某些变量做再次绑定，而这经常无意识地覆盖了这个变量上原来绑定的值而带来问题。所以说，它只是个片警。这个郁闷的问题基本上在所有支持 Multiple assignment 的语言中都会碰到，但它真的跟 Erlang 的容错性八竿子打不着。

在容错性问题上，Erlang 实践了很多非常关键的设计哲学和决策，其中比较突出的是：

"fail early, fail often" policy,
linked processes,
supervision trees,
以及更重要的是：
distribution and failover.

这些能力并不是全都烙印在 OTP 里头的，相反，OTP 通过提供这些组件，让你可以将这些能力添加到你的应用中。它们之中，没有哪个是依附于 Single assignment 实现的。即便是事务语义（Transactional semantics）这个 Joe Armstang 在他的书里大肆褒奖的东西，也并不是必须依赖于 Single assignment 的，很显然 Multiple assignment 也可以轻松实现它。

Single assignment 只是规避了一种特定类型的错误。可是他真的规避了吗？没有，同类型的错误还是会以不同的方式发生，比如返回了一个错误版本的变量。所以说，Single assignment 并没有对 Erlang 的容错性做出什么重大贡献，而将 Erlang 的容错性归功于 Single assignment 的人，是因为对那些真正让 Erlang 拥有这个能力的特性一无所知。

谣言 #3:Single assignment 是 Erlang 出色的并发能力的基础

这是一个更加荒谬的谣言，因为它完全错了，并且再次体现了无知，对 Erlang 如何实现并发的无知。Single assignment 影响的是局部变量绑定，变量又是在函数作用域约束下的，

而函数是粒度比进程小得多的原语。但恰恰进程才是真正的 Erlang 并发的基础。Single assignment 在并发方面作出的贡献，只是让变量大写了(&_&)。

这里有个对于我上一次的 Single assignment 讨论的评论：

> "Breaking single assignment is not good idea.In mine opinion it would be messy.
> References (aliasing) will work inside one process but doesn't outside by message passing."

HOHO！这简直是足以写进教科书的"混淆 Single assignment 和 Erlang 的并发能力"的典型。Erlang 的 Processs 只能通过传递消息来通讯，而局部变量明显不会影响到 Message 的内容，这可是 Erlang 的基础设计目标之一。

> "I have the impression, though,that the single assignment restriction has to do with Erlang's great safety in parallel processing. Isn't that the case?"

幸运的是，总算有人在尝试理清楚这些东西。

> "No, it isn't. Since different processes never have access to each other's local variable bindings,rebinding these variables can not result in concurrency problems."

Bingo！更赞的是有人已经抓住了重点。

谣言 #4:SSA 不可能快过手工 Single assignment

这可真是个奇怪的论断，看起来是来自一个函数式编程的程序员。

> "Your thesis is that single assignment is a dinosaur that still exists because of lack of a smart enough compiler. […] Single assignment offers speed-ups because […] a program with Multiple assignment transformed to SSA form will not be equivalent it's handwritten Single Assignment form."

这还激起了一片汇编语言拥护者的回应，说编译器能做到的自动优化，怎么也赶不上那些做过理想的手工优化的汇编代码。实际上呢？有多少机会编程人员对代码的手工优化，比编译器做的优化还好？大多数情况下的事实是，差得多。如果你不是个一丝不苟的微处理器优化者，也不是很喜欢仔细地审查你写的每一行代码，那是什么让你那么自信地论断说你的手工优化比编译器还生猛？我实在是无法理解。

不错，人类善于推理问题的缘由，但却不善于将一个形式化的优化集合应用于代码，而这正是编译器擅长的。即便是相信人能够和编译器做得一样好，无论何时，我还是选择使用编译器，因为人总是想偷懒，编译器则不会。

谣言 #5: 你想废除 Single assignment? 可你正在推销 MUTABLE STATE!

这个是最恼人的论点了。它是彻底的，完完全全错误的。Single assignment 和 Mutata state 是完全不同的两个东西。风马牛不相及！你这种论断等同于说布什能当上美国总统是因为天空是蓝的，地球是会转的。

Reia，我之前提到的那个我正在实现的语言，没有 Single assignment，但 Reia 中的状态是不可变的，跟 Erlang 一样。Reia 提供对变量 rebind 的能力，所以你可以改变 function，即局部绑定。Erlang 也一样。Erlang 允许你将变量从 unbound 变成 bound，Reia 则是允许你将一个绑定了的变量指向一个新的值。这两者都没改变状态。状态的改变是要求状态的值变了，而在 Reia 中，值是不允许改变的。

Reia 将被编译到 Erlang，一个状态不可变的语言。得益于 Erlang VM 的优点，值不会变也不能变。地球转到哪它也还是圆的。

乍听起来为了达到这个目的，我不得不深入到 Erlang VM 的 C 代码里头，操刀为 Eralng VM 整容来达到 Reia 的多次赋值的目的。幸运的是，我不用这么干，Reia 是被编译到 Erlang 的抽象形式，不用改变 VM 的行为。

Reia 这种将一个允许变量 rebound 的命令式语言，转换到一个 Single assignment 的形式的行为，在多种多样的编译器中是很常见的行为，包括牛 x 得要死的 GCC。这种转换既没有引进可变状态，也没有要求在与 Erlang 不同的语言中引入改变绑定的机制，它发生在 AST 层，在抽象语法被求值之前。转换出来的代码与 Erlang 程序员根据版本化变量机制手工写出来的代码极其相似。

我还是奇怪为什么人们会将 single assignment 和 immutable state 混淆在一起。我猜他们对这两个概念都没有正确的理解。由于这两者明显没有什么相互影响，所以我不打算继续探寻为什么有些人会将他们结合得这么紧密了。然而还是有些声称是 Single assignment 拥护者的人，将 mutable state 作为他们的论据：

> "Do you actually want to add Mutable State to Erlang?[…] Immutable state is one of the most basic features of Erlang-style concurrency."

噢，不！不！不！我不想要 Mutable State。Single Assignment 也算不上是 Erlang 的重要基石.Actor model 才是,Process linking 也是，Supervision trees 亦然.Single assignment 却不是.

结论

已经有太多关于 Single Assignment 的争辩了，所以请你在为它辩解，或者批评 Reia 在这方面的不足时，不要再使用那些基于以上谣传的火星论据了。我自信对 Erlang 和函数式编程理念的理解非常不错，可我还是不喜欢 Single Assignment，或许在不久的将来，有什么将促使我改变这个想法。不过在此之前，要是有人愿意重写 reia_visitor.erl 而不使用 hand-versioning 变量，那就太棒了。

最后：

Erlang 卓越的容错性跟 Single Assignment 没什么关系！
Erlang 卓越的并发能力也跟 Single Assignment 没什么关系！
手工劳动也不是代码优化的必备材料。
最重要的是:

"Single assignment has nothing to do with immutable state."