What's all this fuss about Erlang?

原文：http://www.pragprog.com/articles/erlang
作者：Joe Armstrong
译者：许式伟 朱照远

没人可以预言未来，但我却打算做一些有依据的推测。

让我们假设 Intel 是正确的，而且 Keifer 项目会获得成功。如果是这样，那么 32 核的处理器在 2009/2010 年就将会出现在市场上。

这毫不奇怪，Sun 已经制造出了 Niagara，它拥有 8 个核，每个核运行 4 个超线程（这相当于 32 个核）。

这是一个令 Erlang 程序员欢呼雀跃的进展。为此，他们已经等待了 20 年，现在，是获得回报的时候了。

对于 Erlang 程序员来说，好消息是：

> **你的 Erlang 程序在 N 核的处理器上运行将快 N 倍。**

这是真的吗？

差不多吧。尽管为时尚早，但我们仍很乐观（相当地乐观，在过去的 20 年里，我从来没见过如此的乐观！）。

有时我们需要对我们的程序作点小调整 —— 当我在一台 Sun Niagara 机器（拥有相当于 32 个核）上生成 Erlang 文档时，我把我的程序改了一行（我把一个 map 换成了 pmap —— 不好意思，我在这里提一点技术细节，pmap 只是“并行的 map”(parallel map）而已）。

这个程序（它根据 wiki 标记生成 63 篇文档）的运行速度提高了 7 倍。不是 32 倍，这一点我承认，但是已经显著的加快了。（后来的工作使我们意识到我们是在写盘时遇到了 I/O 瓶颈，所以除非能够让磁盘的 I/O 也并行了，否则我们会停留在这个 7 倍上 :-)

在 Ericsson，这个我工作和 Erlang 被开发出来的地方，我们正在把一些应用程序移植到 4 核处理器上 —— 你猜怎么着？在作了一些小调整后，它们运行几乎都快了 4 倍。呵呵，对 Intel 在实验中的 80 个核的处理器，我们有点等不及了…

为什么我们的程序运行得更快了？这全跟可变状态（mutable state）和并发（concurrency）有关。

可变状态和并发

回首过去（20 多年以前），有两种并发模型：

- 共享状态并发（Shared state concurrency）
- 消息传递并发（Message passing concurrency）

现在，整个世界都走了一条路线（朝着共享状态的方向），而我们选择了另一条。

几乎没有语言沿“消息传递并发之路”而行，例外的则有 Oz 和 Occam。

在消息传递并发模型中，我们宣称其中没有共享状态。所有的计算都在进程中完成，并且交换数据的**唯**一途径是通过异步消息传递。

那为什么这是有益的呢？

共享状态并发模型被“可变状态”（顾名思义就是可被修改的内存）的思想所拖累。所有像 C，Java， C++等等的语言，都认为有一种东西叫“内存”，我们可以去修改它。

只要你仅有**一个**进程对内存进行修改，那它就没什么问题。但是如果你有多个进程共享并且修改**同**一内存的话，则后患无穷——这是很愚蠢的。

为了防止对共享内存的同时修改，我们需要一种锁机制。你称它为互斥体（mutex）也好，同步方法（synchronised method）也好，或者其他你愿意的名字，它仍然是锁。

如果程序在临界区（即当它们持有锁的时候）崩溃，灾难就来了。所有其他的程序都将不知所措。

程序员怎么修正这些问题呢？他们不修正。他们只有祈祷。在单核处理器上，他们的程序或许还可以工作，但是在多核上——灾难。

有各种各样解决这个问题的方案（事务型内存（*transaction memories*）可能是最好的）。但是，最好的情况下，它们也仅仅是“杂烩”，最坏情况下，它们就是噩梦。

Erlang 没有可变的数据结构

（虽然不是百分百的准确，但也足够准确了）

没有可变的数据结构 = 没有锁

没有可变的数据结构 = 容易并行化

我们如何做并行呢？很简单，程序员将问题的解决方案分解成许多并行的进程。

这种编程风格有它自己的术语——它叫

面向并发编程（Concurrency Oriented Programming）

Erlang 并不是面向对象的 ——它有它自己的表示方法。

对象下台，并发上场。

世界是并发、并行的，同一时刻发生着很多事情。 如果我没有很直觉地理解并发的思想，那么我不可能在公路上开车。我们一直在进行着纯粹的消息传递并发。

想象有一群人，他们没有共享的状态。

我有我的私有的记忆（在我的脑海里面），你有你的，二者并**不**共有。我们通过传递信息的方式（声波和光波）进行交流。基于这些信息的接收，我们更新我们的私有状态。

简而言之，这就是**面向并发编程**。

就把可变状态隐藏在对象里面（译者按：这是面向对象的核心理念之一——封装变化）而言：**正是**这个特性使得并行成为一个几乎不可能解决的难题。

它有效吗？

是的。 Erlang 在那些对可靠性很看重的高科技项目中被广泛使用着。 Erlang 的旗舰项目（由瑞典电信公司 Ericsson 创立）是 AXD301 ，它有超过 2 百万行的 Erlang 代码。

AXD301 已经获得了九个 9 的可靠性（是的，你读的没错， 99.9999999% ）。让我们把它放到这样的背景：五个 9 已经被认为是优秀了（宕机时间 5.2 分钟/ 年），七个 9 几乎就达不到 … 但是我们做到了九个 9 。

为什么呢？因为没有共享状态，加上还有一个精妙的错误恢复模型。你可以在我的博士论文中了解全部的细节。

谁在使用 **Erlang?**

- “了解内幕”的人们
- 初创企业
- Ericsson
- wings, 一个 3D 建模程序 http://www.wings3d.com/
- ejabberd, 一个即时消息服务器（jabber/XMPP）
- tsung, 一个多协议分布的负载测试工具
- yaws,一个性能非常高的 web 服务器
- 数以千计的（“我希望我可以在上班时间做这个”）的爱好者

Erlang 难吗？

不——不过它有点与众不同。

Erlang 没有“一种类似于 C 的语法以使之容易被学习”，它既不是“面向对象的”，也没有“可变状态”，它是一门“函数式编程语言（Functional Programming Language）”。

这就是所有让人感到害怕——并让新的使用者望而却步的东西。然而有意思的是， Erlang 其实是一门非常小而且简单的语言。

可能你正在对 Erlang 代码看上去的样子感到迷惑不解。 Erlang 大量使用了模式匹配语法；这里有一个 Erlang 代码的小例子（摘自新版的 Programming Erlang 一书）：

```
-module(geometry).

-export([area/1]).

area({rectangle, Width, Ht}) -> Width * Ht;

area({square, X}) -> X * X;

area({circle, R}) -> 3.14159 * R * R.
```

现在，让我们在 Erlang shell 中编译并运行它：

```
1> c(geometry).

{ok,geometry}

2> geometry:area({rectangle, 10, 5}).

50

3> geometry:area({circle, 1.4}).

6.15752
```

相当的简单 … 以下是做类似事情的 Java 代码：

```
abstract class Shape {
    abstract double area();
}
class Circle extends Shape {
    final double radius;
    Circle(double radius) { this.radius = radius; }
    double area() { return Math.PI * radius*radius; }
```

```
}
class Rectangle extends Shape {
    final double ht;
    final double width;
    Rectangle(double width, double height) {
       this.ht = height;
      this.width = width;
    }
    double area() { return width * ht; }
}
class Square extends Shape {
    final double side;
    Square(double side) {
      this.side = side;
    }
    double area() { return side * side; }
}
```

我从哪里下载 Erlang？

你可以从 erlang.org 处下载 Erlang。

我如何进一步了解 Erlang？

哦，我刚写完了 Programming Erlang: Software for a Concurrent World (Pragmatic Bookshelf, US$36.95, 978-1-934356-00-5) 一书。

这是一本 Erlang 的教程导引。它覆盖了整个语言，并且包含了许多演示程序及其完整的源代码，其中包括：

- 一个类似 irc/chat 的系统
- 一个流媒体 SHOUTcast 服务器

- 一个用来构建全文检索系统的 map-reduce 实现

博客

在博客圈子里面气氛非常活跃，你可以访问 Google 搜索引擎搜索 Erlang 博客。此外，这里还有一些你可能喜欢的文章：

- More Erlang
- Concurrency is Easy
- Web 2.0 is Shifting

网络文档

现在去看看这个目录吧。你将在这里找到 50 多个 PDF 文件。我已经为你挑选了其中的一些，以供你查阅：

- asn1-1.4.4.12.pdf
- design_principles-5.5.3.pdf
- dialyzer-1.5.1.pdf
- efficiency_guide-5.5.3.pdf
- getting_started-5.5.3.pdf
- inets-4.7.8.pdf
- mnesia-4.3.3.pdf
- parsetools-1.4.1.pdf
- programming_examples-5.5.3.pdf
- reference_manual-5.5.3.pdf
- sasl-2.1.4.pdf
- snmp-4.8.4.pdf
- tutorial-5.5.3.pdf

它们中的每一个都代表了一本有待编纂的书 :-)

祝你阅读愉快！