**Same-threading**

* [Why Single-threaded Systems?](http://tutorials.jenkov.com/java-concurrency/same-threading.html#why-single-threaded-systems)
* [Same-threading, Single-threading Scaled Out](http://tutorials.jenkov.com/java-concurrency/same-threading.html#same-threading-single-threading-scaled-out)
  + [One Thread Per CPU](http://tutorials.jenkov.com/java-concurrency/same-threading.html#one-thread-per-cpu)
* [No Shared State](http://tutorials.jenkov.com/java-concurrency/same-threading.html#no-shared-state)
* [Load Distribution](http://tutorials.jenkov.com/java-concurrency/same-threading.html#load-distribution)
  + [Single-threaded Microservices](http://tutorials.jenkov.com/java-concurrency/same-threading.html#single-threaded-microservices)
  + [Services With Sharded Data](http://tutorials.jenkov.com/java-concurrency/same-threading.html#services-with-sharded-data)
* [Thread Communication](http://tutorials.jenkov.com/java-concurrency/same-threading.html#thread-communication)
* [Simpler Concurrency Model](http://tutorials.jenkov.com/java-concurrency/same-threading.html#simpler-concurrency-model)
* [Illustrations](http://tutorials.jenkov.com/java-concurrency/same-threading.html#illustrations)

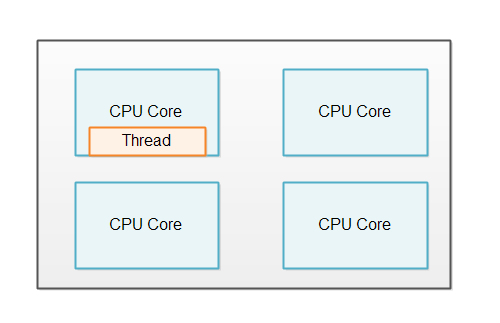
相同线程是一种并发模型，其中一个单线程系统扩充到N个单线程系统。结果是N个单线程系统并行运行。

一个相同线程系统不是一个纯粹的单线程系统，因为它包含多个线程。但是每一个线程运行类似一个单线程系统

**Why Single-threaded Systems?**

你可能好奇为什么有人会设计单线程系统如今。但单线程系统获得流行度因为它们的并发模型比多线程系统简单得多。单线程系统不共享任何数据跟其他线程。这使得单个线程可以使用非阻塞数据结构，利用CPU和CPU缓存更充分。

不幸的是，单线程系统不能充分利用现代的CPU。一个现代的CPU通常带有2、4或者更多核心。每个核心作为一个独立的CPU行使功能。单线程系统仅仅利用多个核心中的一个，说明如下：

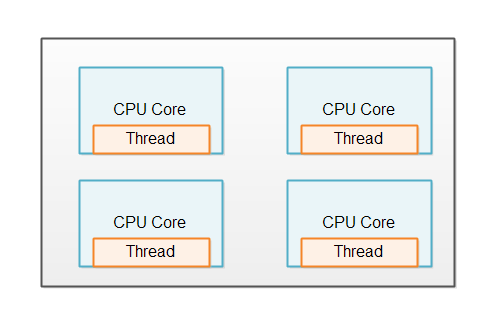


**Same-threading, Single-threading Scaled Out**

为了充分利用CPU的全部核心，单线程系统可能被扩大到利用全部计算机。

**One Thread Per CPU**

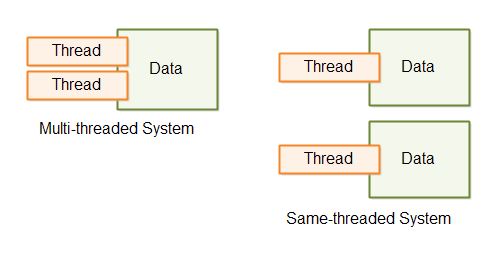
相同线程系统通常计算机中每个CPU有一个线程运行。如果一台计算机包含4个CPU组，或者一个CPU包含4个核心，那么运行相同线程系统(4个单线程系统)的4个实例是正常的。下面的图解展示了这一原理：



**No Shared State**

一个相同线程系统看起来近似于一个多线程系统，因为一个相同线程系统内部有多个线程运行。但是仍有微妙区别。

相同线程和多线程系统区别在于相同线程系统中的线程不共享数据。不具有共享的线程并发访问的内存。没有并发数据结构等用于线程共享数据。区别图解如下：



共享态的缺少使得每个线程表现得像是单线程系统。然而，因为相同线程系统可能包含超过一个单线程，因此它实际并不是一个“单线程系统”。缺少一个更好的名字，我发现称呼这样的一个系统为一个相同线程系统更精准，相比一个“多线程系统使用一个单线程设计”。相同线程更易于说和理解。

相同线程基本意味着数据处理保持在相同线程中，一个相同线程系统中没有线程并发共享数据。

**Load Distribution**

显然，相同线程系统需要在运行中的单线程实例间共享工作负载。否则，只有一个单实例将会得到全部工作，系统结果将变成单线程。

具体如何分布负载在不同的实例之间，依赖于你的系统设计。我将会在接下来的章节提到一些。

**Single-threaded Microservices**

如果你的系统由多个微服务组成，每个微服务可以在单线程模式运行。当你部署多个单线程微服务到同一台机器，每个微服务可以在一个单独CPU上运行一个单独的线程。

微服务不天然共享任何数据，因此微服务对于相同线程系统是一个好的用例。

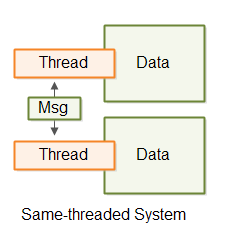
**Services With Sharded Data**

如果你的系统确实需要共享数据，或者至少一个数据库，你可能能够给数据库分片。分片意味着数据划分到多个数据库中。数据典型划分以便于相互关联的全部数据共同坐落于同一数据库。举个实例，全部数据某个“所有者”实体的数据将被插入同一个数据库。不过分片超出了这篇教程的范围，所以你可能需要搜索关于那个话题的教程。

**Thread Communication**

如果相同线程系统中的线程需要交流，它们通过信息传递完成。一个线程想要发送信息给线程A可以通过生成一条信息(一个字节序列)。线程B然后可以复制那条信息(字节序列)并读取它。通过复制信息，线程B确保在读取期间线程A不能修改信息。一旦被复制，它对于线程A是不可改动的。

线程通过消息机制交流图解：



线程交流能发生通过队列、管道、unix套接字、TCP套接字等。无论什么只要适合你的系统。

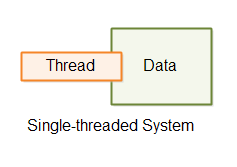
**Simpler Concurrency Model**

相同线程系统中每个在自身线程中运行的系统可以被实现类似单线程。这意味着内部并发模型变得比共享态线程简单。你不必担心并发数据结构和所有这种数据结构可能导致的并发问题。

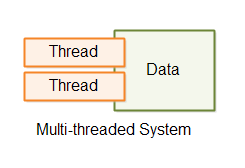
**Illustrations**

这是单线程、多线程和相同线程系统的图解，以便更容易得到他们之间不同点的一个概览。

第一个图解展示了一个单线程系统



第二个图解展示了一个多线程系统，线程之间共享数据。



第三个图解展示一个具有两个独立数据的线程的相同线程系统，交流通过互相传递消息。

