# 11个热门物联网开发平台的比较

2016-03-07 [CSDN](javascript:void(0);)

本文基于我们对物联网（IoT）供应商进行的详细分析，针对现有IoT软件平台做了一份综合调查。首先，我们制作了一个清单，列出了IoT软件平台的重要功能。然后我们对比了当前11个较为热门的IoT软件平台上在这些关键功能的开发程度。最后根据我们的观察对列表进行缩减，总结需要改进的功能。

**简介**

从1999年Kevin Ashton第一次提出这个概念以来，物联网已经经历了迅速的转变。随着近年来连接到物联网的设备在多样性和数量方面出现指数式的增长，物联网已经成为了一种主流技术，在推动现在社会的生活方式方面有着极大的潜力。

在物联网的技术与工程上，硬件与软件平台之间目前仍有明确的界限，其中大多数供应商都将精力放在硬件方面。只有极少数供应商提供物联网软件服务：例如，Mattermark根据所获总投资排名的前100名物联网创业公司中，只有13家提供物联网软件服务。

本文针对现有物联网软件平台，基于我们对IoT供应商进行的详细分析做了一份综合调查。而本文最后选择的物联网供应商，完全是基于这样的标准：这些供应商是否提供软件解决方案，来处理从物联网设备/传感器获取的信息。**注意：虽然我们希望尽可能全面，但本文中仍有可能漏掉了一些这些平台的最新改进。**

**物联网软件平台想要的重要功能**

基于最近的几份调查，我们选出了物联网软件平台最关键的功能：设备管理、集成、安全性、数据收集协议、分析类型以及支持可视化，以便对样本功能进行比较。本文的后半段中会对这些特性进行简单介绍。

**设备管理与支持集成**

设备管理是物联网软件平台所需的重要功能之一。物联网平台应当维护着一堆与之连接的设备，并跟踪这些设备的运行状态；还应当能够处理配置、固件（或其他软件）更新问题，并提供设备级的错误报告和处理方案。每天结束前，设备用户应当能够获得个人设备级的统计。

支持集成是物联网软件平台需要的另一个重要功能。需要从物联网平台上公布的重要操作和数据应当能通过API访问，REST API常用于这一目的。

**信息安全**

运营物联网软件平台所需的信息安全手段，比普通软件应用和服务所需的要求更高。数百万台设备与物联网平台连接，代表着我们需要处理的漏洞也是相应比例的。一般来讲，为了避免被窃听，物联网设备与物联网软件平台之间的网络连接需要通过强大的加密机制来保障。

然而，在现代的物联网软件平台上，大多低成本、低功率的设备都无法支持这样的高级访问控制措施。因此，物联网软件平台自身需要采取替代措施，以解决这类设备级的问题。例如：将物联网流量划分为专用网络，依靠云应用级的强大安全性，要求定期更新密码并支持验证更新固件，还有签名才能更新软件等等，这些手段都能加强物联网软件平台的安全级别。

**数据收集协议**

需要注意的另一个重要方面，是物联网软件平台的各个组件之间用于数据通信的协议类型。物联网平台可能需要扩展到数百万甚至数十亿设备（节点）上。应当使用轻量级通信协议，以实现低能耗以及低带宽功能。

注意：虽然我们在本文中将协议作为概述性词汇，不过用以收集数据的协议可分为下面几类：比如应用、负载容器、信息传递和遗留协议。

**数据分析**

从连接到物联网平台的传感器中所收集的数据需要通过智能化手段进行分析，以获得有意义的见解。

物联网数据分析有四种主要类型：实时分析、批处理分析、预测分析与交互式分析。实时分析：对数据流执行在线（动态）分析。样本操作包括基于窗口的集成、筛选、转换等。

批处理分析：对积累的数据集进行操作。这样，批处理操作会在预定时间段内运行，也许持续数小时或数日。预测分析：基于各类统计与机器学习技术，集中进行预测。交互式分析：对数据流和批数据执行多个探索性分析。最后一个就是实时分析，在任何软件平台都占据较重的份量。

**当前的物联网软件平台**

对当前的物联网软件平台进行仔细调查后，我们发现上面提到的每个功能都已实现，只是程度不同而已。我们在下面列出了相关的平台，并进行了功能总结对比：

表1 相关平台功能总结对比(标着“未知”的栏目代表相关信息在可用文件中无法找到)

很明显，上面列举的物联网初创公司，其中很多可能还不具备设备管理功能。在这方面，还需要物联网软件平台供应商提供解决方案。

此外在分析生成的物联网数据时，在计算及可视化方面提供的支持相对较少。它们大多支持实时分析——这是任何物联网框架的必备功能。然而，只有极少数物联网软件平台为其他3种分析类型提供支持。而可视化界面大多表现为门户网站这样的简单模式，允许对物联网生态环境进行管理，不过很少提供可视化的数据分析功能。

在不同的物联网软件平台中，还有几个常见功能，包括基于集成的REST API，支持用MQTT协议来收集数据，以及使用SSL进行链路加密。尽管在表一中没有提到，不过单ParStream公司就能达到300万到400万行/秒的吞吐量。

这表明大多数物联网软件平台设计时并未太多考虑物联网部署的系统性能，而在真实情况下这是非常关键的。

**需要改进的功能**

很明显有若干地方需要改进。在本节中，我们首先提供了一张改进功能列表。在物联网软件平台供应商的努力下，其中一些项目已经实现，还有一些性能等待实现。之后我们提供了一张列表，包括现在尚未实现的这些新功能。

**现有功能**

**数据分析**

现在物联网软件平台大多支持实时分析，不过批处理分析和交互式数据分析也许同样重要。

在这一点上，有人可能会争辩：在其他知名的处理平台中包括这类分析功能，想要配置用于分析场景的软件系统也很简单。不过，这谈何容易。用于实时分析（Storm、Samza等）、用于批处理分析（Hadoop、Spark等）、用于预测分析（Spark MLLIB等）、用于交互式分析（Apache Drill等）的知名数据处理系统，并不能直接用在物联网案例中。

**基准**

物联网软件平台需要有扩展性，还应包含描述和评估系统性能的设备。定义良好的性能指标需要：能够塑造与测量物联网系统的性能，并考虑到网络特性、能耗特点、系统吞吐率、计算资源消耗以及其他运行特征。

**边缘分析**

需要采取措施以减少传感器设备与物联网服务器之间的大量网络带宽损耗。解决方案之一是使用轻量级的通讯协议。另一个办法就是使用边缘分析法，以减少传输到物联网服务器上的原始数据总量。即便是在简单的硬件嵌入系统中（如Arduino），也可以实现边缘分析法。

**其他问题**

应当注意：有多个与物联网软件平台相关的其他问题，比如伦理、道德和法律问题，在本文中并未涉及。尽管这些问题也很重要，但在本文中不作讨论。

**需要添加的功能**

**处理无序进程**

在任何物联网应用中都有可能碰到无序事件，在传感器所发出的事件流中，元组顺序混乱可能是网络延迟、时钟偏移等原因所导致的。处理无序的物联网事件可能会导致系统故障。处理无序事件时，需要在结果准确性与延迟之间做出权衡。

有四项主要的处理技术：基于缓存（Buffer-based）、基于标点（Punctuation-based）、基于推测（Speculation-based）以及基于近似（Approximation-based techniques）。在物联网解决方案中，应当使用其中的一项或多项来解决无序事件的问题。

**支持物联网背景**

背景主要由个体、其偏好或过去的行为构成。例如：在移动电话案例中，由于现代移动电话中有很多不同类型的传感器，因此我们能够获得丰富的背景信息。在物联网分析中，这些背景数据应当被纳入考虑。

**结论**

物联网模式的快速发展需要强大的物联网软件平台，能通过物联网用例满足出现的需求。本文中，我们调查了现有最先进的物联网软件平台的功能，调查集中在这些方面：设备管理、集成、安全性、数据收集协议、分析类型、可视化支持。从这项研究中，像设备管理、物联网数据分析、物联网软件系统可扩展性以及性能这样的领域明显需要物联网平台社区投入特别的关注。

# 8大排序算法图文讲解

2016-03-08 [51CTO](javascript:void(0);)

常见的内部排序算法有：插入排序、希尔排序、选择排序、冒泡排序、归并排序、快速排序、堆排序、基数排序等。

本文将依次介绍上述八大排序算法。

## 算法一：插入排序



插入排序示意图

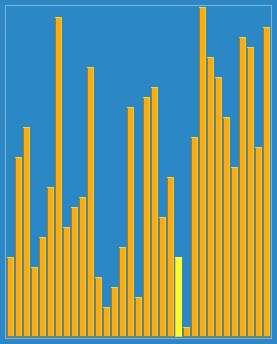
插入排序是一种最简单直观的排序算法，它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。

**算法步骤：**

1）将第一待排序序列第一个元素看做一个有序序列，把第二个元素到最后一个元素当成是未排序序列。

2）从头到尾依次扫描未排序序列，将扫描到的每个元素插入有序序列的适当位置。（如果待插入的元素与有序序列中的某个元素相等，则将待插入元素插入到相等元素的后面。）

## 算法二：希尔排序

希尔排序示意图

**希尔排序**，也称**递减增量排序算法**，是插入排序的一种更高效的改进版本。但希尔排序是非稳定排序算法。

希尔排序是基于插入排序的以下两点性质而提出改进方法的：

* 插入排序在对几乎已经排好序的数据操作时， 效率高， 即可以达到线性排序的效率
* 但插入排序一般来说是低效的， 因为插入排序每次只能将数据移动一位

希尔排序的基本思想是：先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对全体记录进行依次直接插入排序。

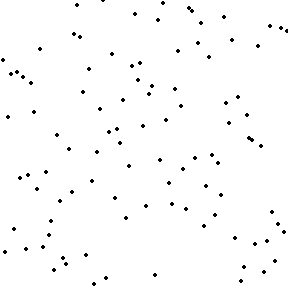
**算法步骤**：

1）选择一个增量序列t1，t2，…，tk，其中ti>tj，tk=1；

2）按增量序列个数k，对序列进行k 趟排序；

3）每趟排序，根据对应的增量ti，将待排序列分割成若干长度为m 的子序列，分别对各子表进行直接插入排序。仅增量因子为1 时，整个序列作为一个表来处理，表长度即为整个序列的长度。

## 算法三：选择排序



选择排序示意图

**选择排序**(Selection sort)也是一种简单直观的排序算法。

**算法步骤**：

1）首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置

2）再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。

3）重复第二步，直到所有元素均排序完毕。

## 算法四：冒泡排序

冒泡排序示意图

**冒泡排序**（**Bubble Sort**）也是一种简单直观的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端。

**算法步骤**：

1）比较相邻的元素。如果第一个比第二个大，就交换他们两个。

2）对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。

3）针对所有的元素重复以上的步骤，除了最后一个。

4）持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

## 算法五：归并排序

归并排序示意图

**归并排序（Merge sort）**是建立在归并操作上的一种有效的排序算法。该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。

**算法步骤：**

1. 申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列

2. 设定两个指针，最初位置分别为两个已经排序序列的起始位置

3. 比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置

4. 重复步骤3直到某一指针达到序列尾

5. 将另一序列剩下的所有元素直接复制到合并序列尾

**详细介绍：归并排序**

## 算法六：快速排序

快速排序示意图

**快速排序**是由东尼·霍尔所发展的一种排序算法。在平均状况下，排序 n 个项目要**Ο**(n log n)次比较。在最坏状况下则需要**Ο**(n2)次比较，但这种状况并不常见。事实上，快速排序通常明显比其他**Ο**(n log n) 算法更快，因为它的内部循环（inner loop）可以在大部分的架构上很有效率地被实现出来。

快速排序使用分治法（Divide and conquer）策略来把一个串行（list）分为两个子串行（sub-lists）。

**算法步骤：**

1 从数列中挑出一个元素，称为 “基准”（pivot），

2 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为**分区（partition）**操作。

3 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

递归的最底部情形，是数列的大小是零或一，也就是永远都已经被排序好了。虽然一直递归下去，但是这个算法总会退出，因为在每次的迭代（iteration）中，它至少会把一个元素摆到它最后的位置去。

**详细介绍：快速排序**

## 算法七：堆排序

堆排序示意图

**堆排序**（Heapsort）是指利用堆这种数据结构所设计的一种排序算法。堆积是一个近似完全二叉树的结构，并同时满足堆积的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。

堆排序的平均时间复杂度为**Ο**(nlogn) 。

**算法步骤：**

1）创建一个堆H[0..n-1]

2）把堆首（最大值）和堆尾互换

3）把堆的尺寸缩小1，并调用shift\_down(0),目的是把新的数组顶端数据调整到相应位置

4） 重复步骤2，直到堆的尺寸为1

**详细介绍：堆排序**

## 算法八：基数排序

**基数排序**是一种非比较型整数排序算法，其原理是将整数按位数切割成不同的数字，然后按每个位数分别比较。由于整数也可以表达字符串（比如名字或日期）和特定格式的浮点数，所以基数排序也不是只能使用于整数。

说基数排序之前，我们简单介绍桶排序：

**算法思想：**是将阵列分到有限数量的桶子里。每个桶子再个别排序（有可能再使用别的排序算法或是以递回方式继续使用桶排序进行排序）。桶排序是鸽巢排序的一种归纳结果。当要被排序的阵列内的数值是均匀分配的时候，桶排序使用线性时间（Θ（n））。但桶排序并不是 比较排序，他不受到 O(n log n) 下限的影响。  
简单来说，就是把数据分组，放在一个个的桶中，然后对每个桶里面的在进行排序。

例如要对大小为[1..1000]范围内的n个整数A[1..n]排序

首先，可以把桶设为大小为10的范围，具体而言，设集合B[1]存储[1..10]的整数，集合B[2]存储   (10..20]的整数，……集合B[i]存储(   (i-1)\*10,   i\*10]的整数，i   =   1,2,..100。总共有  100个桶。

然后，对A[1..n]从头到尾扫描一遍，把每个A[i]放入对应的桶B[j]中。  再对这100个桶中每个桶里的数字排序，这时可用冒泡，选择，乃至快排，一般来说任  何排序法都可以。

最后，依次输出每个桶里面的数字，且每个桶中的数字从小到大输出，这  样就得到所有数字排好序的一个序列了。

假设有n个数字，有m个桶，如果数字是平均分布的，则每个桶里面平均有n/m个数字。如果

对每个桶中的数字采用快速排序，那么整个算法的复杂度是

O(n   +   m   \*   n/m\*log(n/m))   =   O(n   +   nlogn   –   nlogm)

从上式看出，当m接近n的时候，桶排序复杂度接近O(n)

当然，以上复杂度的计算是基于输入的n个数字是平均分布这个假设的。这个假设是很强的  ，实际应用中效果并没有这么好。如果所有的数字都落在同一个桶中，那就退化成一般的排序了。

前面说的几大排序算法 ，大部分时间复杂度都是O（n2），也有部分排序算法时间复杂度是O(nlogn)。而桶式排序却能实现O（n）的时间复杂度。但桶排序的缺点是：

1）首先是空间复杂度比较高，需要的额外开销大。排序有两个数组的空间开销，一个存放待排序数组，一个就是所谓的桶，比如待排序值是从0到m-1，那就需要m个桶，这个桶数组就要至少m个空间。

2）其次待排序的元素都要在一定的范围内等等。

## 总结

各种排序的稳定性，时间复杂度、空间复杂度、稳定性总结如下图：

**关于时间复杂度：**

(1)平方阶(O(n2))排序  
各类简单排序:直接插入、直接选择和冒泡排序；

(2)线性对数阶(O(nlog2n))排序  
　　快速排序、堆排序和归并排序；  
(3)O(n1+§))排序,§是介于0和1之间的常数。

       希尔排序  
(4)线性阶(O(n))排序  
基数排序，此外还有桶、箱排序。

**关于稳定性：**

稳定的排序算法：冒泡排序、插入排序、归并排序和基数排序

不是稳定的排序算法：选择排序、快速排序、希尔排序、堆排序

# AlphaGo这个系统主要由几个部分组成：

1、走棋网络（Policy Network），给定当前局面，预测/采样下一步的走棋。/2

2、快速走子（Fast rollout），目标和1一样，但在适当牺牲走棋质量的条件下，速度要比1快1000倍。

3、估值网络（Value Network），给定当前局面，估计是白胜还是黑胜。

4、蒙特卡罗树搜索（Monte Carlo Tree Search，MCTS)，把以上这三个部分连起来，形成一个完整的系统。

我们的DarkForest和AlphaGo同样是用4搭建的系统。DarkForest较AlphaGo而言，在训练时加强了1，而少了2和3，然后以开源软件Pachi的缺省策略 (default policy)部分替代了2的功能。以下介绍下各部分。

**1、走棋网络：**

走棋网络把当前局面作为输入，预测/采样下一步的走棋。它的预测不只给出最强的一手，而是对棋盘上所有可能的下一着给一个分数。棋盘上有361个点，它就给出361个数，好招的分数比坏招要高。DarkForest在这部分有创新，通过在训练时预测三步而非一步，提高了策略输出的质量，和他们在使用增强学习进行自我对局后得到的走棋网络（RL network）的效果相当。当然，他们并没有在最后的系统中使用增强学习后的网络，而是用了直接通过训练学习到的网络（SL network）,理由是RL network输出的走棋缺乏变化，对搜索不利。

有意思的是在AlphaGo为了速度上的考虑，只用了宽度为192的网络，而并没有使用最好的宽度为384的网络（见图2(a))，所以要是GPU更快一点（或者更多一点），AlphaGo肯定是会变得更强的。

所谓的0.1秒走一步，就是纯粹用这样的网络，下出有最高置信度的合法着法。这种做法一点也没有做搜索，但是大局观非常强，不会陷入局部战斗中，说它建模了“棋感”一点也没有错。我们把DarkForest的走棋网络直接放上KGS就有3d的水平，让所有人都惊叹了下。可以说，这一波围棋AI的突破，主要得益于走棋网络的突破。这个在以前是不可想像的，以前用的是基于规则，或者基于局部形状再加上简单线性分类器训练的走子生成法，需要慢慢调参数年，才有进步。

当然，只用走棋网络问题也很多，就我们在DarkForest上看到的来说，会不顾大小无谓争劫，会无谓脱先，不顾局部死活，对杀出错，等等。有点像高手不经认真思考的随手棋。因为走棋网络没有价值判断功能，只是凭“直觉”在下棋，只有在加了搜索之后，电脑才有价值判断的能力。

**2、快速走子**

那有了走棋网络，为什么还要做快速走子呢？有两个原因，首先走棋网络的运行速度是比较慢的，AlphaGo说是3毫秒，我们这里也差不多，而快速走子能做到几微秒级别，差了1000倍。所以在走棋网络没有返回的时候让CPU不闲着先搜索起来是很重要的，等到网络返回更好的着法后，再更新对应的着法信息。

其次，快速走子可以用来评估盘面。由于天文数字般的可能局面数，围棋的搜索是毫无希望走到底的，搜索到一定程度就要对现有局面做个估分。在没有估值网络的时候，不像国象可以通过算棋子的分数来对盘面做比较精确的估值，围棋盘面的估计得要通过模拟走子来进行，从当前盘面一路走到底，不考虑岔路地算出胜负，然后把胜负值作为当前盘面价值的一个估计。这里有个需要权衡的地方：在同等时间下，模拟走子的质量高，单次估值精度高但走子速度慢；模拟走子速度快乃至使用随机走子，虽然单次估值精度低，但可以多模拟几次算平均值，效果未必不好。所以说，如果有一个质量高又速度快的走子策略，那对于棋力的提高是非常有帮助的。

为了达到这个目标，神经网络的模型就显得太慢，还是要用传统的局部特征匹配（local pattern matching）加线性回归（logistic regression）的方法，这办法虽然不新但非常好使，几乎所有的广告推荐，竞价排名，新闻排序，都是用的它。与更为传统的基于规则的方案相比，它在吸纳了众多高手对局之后就具备了用梯度下降法自动调参的能力，所以性能提高起来会更快更省心。AlphaGo用这个办法达到了2微秒的走子速度和24.2%的走子准确率。24.2%的意思是说它的最好预测和围棋高手的下子有0.242的概率是重合的，相比之下，走棋网络在GPU上用2毫秒能达到57%的准确率。这里，我们就看到了走子速度和精度的权衡。

和训练深度学习模型不同，快速走子用到了局部特征匹配，自然需要一些围棋的领域知识来选择局部特征。对此AlphaGo只提供了局部特征的数目（见Extended Table 4)，而没有说明特征的具体细节。我最近也实验了他们的办法，达到了25.1%的准确率和4-5微秒的走子速度，然而全系统整合下来并没有复现他们的水平。我感觉上24.2%并不能完全概括他们快速走子的棋力，因为只要走错关键的一步，局面判断就完全错误了；而图2(b)更能体现他们快速走子对盘面形势估计的精确度，要能达到他们图2(b)这样的水准，比简单地匹配24.2%要做更多的工作，而他们并未在文章中强调这一点。

在AlphaGo有了快速走子之后，不需要走棋网络和估值网络，不借助任何深度学习和GPU的帮助，不使用增强学习，在单机上就已经达到了3d的水平（见Extended Table 7倒数第二行)，这是相当厉害的了。任何使用传统方法在单机上达到这个水平的围棋程序，都需要花费数年的时间。在AlphaGo之前，Aja Huang曾经自己写过非常不错的围棋程序，在这方面相信是有很多的积累的。

**3、估值网络**

AlphaGo的估值网络可以说是锦上添花的部分，从Fig 2(b)和Extended Table 7来看，没有它AlphaGo也不会变得太弱，至少还是会在7d-8d的水平。少了估值网络，等级分少了480分，但是少了走棋网络，等级分就会少掉800至1000分。特别有意思的是，如果只用估值网络来评估局面（2177），那其效果还不及只用快速走子（2416），只有将两个合起来才有更大的提高。我的猜测是，估值网络和快速走子对盘面估计是互补的，在棋局一开始时，大家下得比较和气，估值网络会比较重要；但在有复杂的死活或是对杀时，通过快速走子来估计盘面就变得更重要了。考虑到估值网络是整个系统中最难训练的部分（需要三千万局自我对局），我猜测它是最晚做出来并且最有可能能进一步提高的。

关于估值网络训练数据的生成，值得注意的是文章中的附录小字部分。与走棋网络不同，每一盘棋只取一个样本来训练以避免过拟合，不然对同一对局而言输入稍有不同而输出都相同，对训练是非常不利的。这就是为什么需要三千万局，而非三千万个盘面的原因。对于每局自我对局，取样本是很有讲究的，先用SL network保证走棋的多样性，然后随机走子，取盘面，然后用更精确的RL network走到底以得到最正确的胜负估计。当然这样做的效果比用单一网络相比好多少，我不好说。

一个让我吃惊的地方是，他们完全没有做任何局部死活/对杀分析，纯粹是用暴力训练法训练出一个相当不错的估值网络。这在一定程度上说明深度卷积网络（DCNN）有自动将问题分解成子问题，并分别解决的能力。

另外，我猜测他们在取训练样本时，判定最终胜负用的是中国规则。所以说三月和李世石对局的时候也要求用中国规则，不然如果换成别的规则，就需要重新训练估值网络（虽然我估计结果差距不会太大）。至于为什么一开始就用的中国规则，我的猜测是编程非常方便（我在写DarkForest的时候也是这样觉得的）。

**4、 蒙特卡罗树搜索**

这部分基本用的是传统方法，没有太多可以评论的，他们用的是带先验的UCT，即先考虑DCNN认为比较好的着法，然后等到每个着法探索次数多了，选择更相信探索得来的胜率值。而DarkForest则直接选了DCNN推荐的前3或是前5的着法进行搜索。我初步试验下来效果差不多，当然他们的办法更灵活些，在允许使用大量搜索次数的情况下，他们的办法可以找到一些DCNN认为不好但却对局面至关重要的着法。

一个有趣的地方是在每次搜索到叶子节点时，没有立即展开叶子节点，而是等到访问次数到达一定数目(40)才展开，这样避免产生太多的分支，分散搜索的注意力，也能节省GPU的宝贵资源，同时在展开时，对叶节点的盘面估值会更准确些。除此之外，他们也用了一些技巧，以在搜索一开始时，避免多个线程同时搜索一路变化，这部分我们在DarkForest中也注意到了，并且做了改进。

**5、总结**

总的来说，这整篇文章是一个系统性的工作，而不是一两个小点有了突破就能达到的胜利。在成功背后，是作者们，特别是两位第一作者David Silver和Aja Huang，在博士阶段及毕业以后五年以上的积累，非一朝一夕所能完成的。他们能做出AlphaGo并享有现在的荣誉，是实至名归的。

从以上分析也可以看出，与之前的围棋系统相比，AlphaGo较少依赖围棋的领域知识，但还远未达到通用系统的程度。职业棋手可以在看过了寥寥几局之后明白对手的风格并采取相应策略，一位资深游戏玩家也可以在玩一个新游戏几次后很快上手，但到目前为止，人工智能系统要达到人类水平，还是需要大量样本的训练的。可以说，没有千年来众多棋手在围棋上的积累，就没有围棋AI的今天。

在AlphaGo中，增强学习（Reinforcement Learning）所扮演的角色并没有想像中那么大。在理想情况下，我们希望人工智能系统能在对局中动态地适应环境和对手的招式并且找到办法反制之，但是在AlphaGo中增强学习更多地是用于提供更多质量更好的样本，给有监督学习（Supervised Learning）以训练出更好的模型。在这方面增强学习还有很长的路要走。

另外，据他们的文章所言，AlphaGo整个系统在单机上已具有了职业水平，若是谷歌愿意开几万台机器和李世石对决（这对它来说再容易不过了，改个参数就行），相信比赛会非常精彩。

===========================

一些更新。

问题1：“Alphago的MCTS做rollout的时候，除了使用快速走子，还用了搜索树的已有部分，看起来像是AMAF/RAVE反过来：AMAF是把快速走子的信息传导到树的其它无关部分，Alphago是把树的其它无关部分拿来增强快速走子。我怀疑这是不是它棋力比其它DCNN+MCTS强的原因之一。”

这个办法在解死活题的文章中出现过，会在一定程度上提高搜索效率，但是提高多少还不知道。

问题2：“rollout的走法质量变好可能会导致棋力下降。”

这里要分两种情况，tree policy和default policy。在AlphaGo的文章里面已经说过了，tree policy的分布不能太尖，不然在搜索时太过重视一些看起来的好着，可能使得棋力下降。但是除了这种原因，一般来说tree policy变好棋力还是会变强的。

default policy这边，即（半）随机走子到最后然后判分，就很复杂了，质量变好未必对局面能估得更准。default policy需要保证的是每块棋的死活大体正确，不要把死的棋下成活的或者反之，而对大局观的要求反而没有那么高。双方完全可以配合着把每块棋下完，然后转战另一块，而不是说抢在对方前去别处占先手。

# RabbitMQ的几种典型使用场景

2016-02-22 ifeve [并发编程网](javascript:void(0);)

**AMQP**

AMQP协议是一个高级抽象层消息通信协议，RabbitMQ是AMQP协议的实现。它主要包括以下组件：

1.Server(broker): 接受客户端连接，实现AMQP消息队列和路由功能的进程。

2.Virtual Host:其实是一个虚拟概念，类似于权限控制组，一个Virtual Host里面可以有若干个Exchange和Queue，但是权限控制的最小粒度是Virtual Host

3.Exchange:接受生产者发送的消息，并根据Binding规则将消息路由给服务器中的队列。ExchangeType决定了Exchange路由消息的行为，例如，在RabbitMQ中，ExchangeType有direct、Fanout和Topic三种，不同类型的Exchange路由的行为是不一样的。

4.Message Queue：消息队列，用于存储还未被消费者消费的消息。

5.Message: 由Header和Body组成，Header是由生产者添加的各种属性的集合，包括Message是否被持久化、由哪个Message Queue接受、优先级是多少等。而Body是真正需要传输的APP数据。

6.Binding:Binding联系了Exchange与Message Queue。Exchange在与多个Message Queue发生Binding后会生成一张路由表，路由表中存储着Message Queue所需消息的限制条件即Binding Key。当Exchange收到Message时会解析其Header得到Routing Key，Exchange根据Routing Key与Exchange Type将Message路由到Message Queue。Binding Key由Consumer在Binding Exchange与Message Queue时指定，而Routing Key由Producer发送Message时指定，两者的匹配方式由Exchange Type决定。

7.Connection:连接，对于RabbitMQ而言，其实就是一个位于客户端和Broker之间的TCP连接。

8.Channel:信道，仅仅创建了客户端到Broker之间的连接后，客户端还是不能发送消息的。需要为每一个Connection创建Channel，AMQP协议规定只有通过Channel才能执行AMQP的命令。一个Connection可以包含多个Channel。之所以需要Channel，是因为TCP连接的建立和释放都是十分昂贵的，如果一个客户端每一个线程都需要与Broker交互，如果每一个线程都建立一个TCP连接，暂且不考虑TCP连接是否浪费，就算操作系统也无法承受每秒建立如此多的TCP连接。RabbitMQ建议客户端线程之间不要共用Channel，至少要保证共用Channel的线程发送消息必须是串行的，但是建议尽量共用Connection。

9.Command:AMQP的命令，客户端通过Command完成与AMQP服务器的交互来实现自身的逻辑。例如在RabbitMQ中，客户端可以通过publish命令发送消息，txSelect开启一个事务，txCommit提交一个事务。

在了解了AMQP模型以后，需要简单介绍一下AMQP的协议栈，AMQP协议本身包括三层：

1.Module Layer，位于协议最高层，主要定义了一些供客户端调用的命令，客户端可以利用这些命令实现自己的业务逻辑，例如，客户端可以通过queue.declare声明一个队列，利用consume命令获取一个队列中的消息。

2.Session Layer，主要负责将客户端的命令发送给服务器，在将服务器端的应答返回给客户端，主要为客户端与服务器之间通信提供可靠性、同步机制和错误处理。

3.Transport Layer，主要传输二进制数据流，提供帧的处理、信道复用、错误检测和数据表示。

**RabbitMQ使用场景**

场景1：单发送单接收

使用场景：简单的发送与接收，没有特别的处理。

http://mmbiz.qpic.cn/mmbiz/Bf4u9qKuXWvE4icVnEhO06rsvrwbEEpHuwdsicibbNLbVfwsewg9IPRIXIZZuxeY5SoGYzUFzF67JtcKjUr04sUBw/640?wx_fmt=png&tp=webp&wxfrom=5&wx_lazy=1

**Producer**：

import com.rabbitmq.client.ConnectionFactory; import com.rabbitmq.client.Connection; import com.rabbitmq.client.Channel; public class Send {        private final static String QUEUE\_NAME = "hello";   public static void main(String[] argv) throws Exception {                      ConnectionFactory factory = new ConnectionFactory();     factory.setHost("localhost");     Connection connection = factory.newConnection();     Channel channel = connection.createChannel();     channel.queueDeclare(QUEUE\_NAME, false, false, false, null);     String message = "Hello World!";     channel.basicPublish("", QUEUE\_NAME, null, message.getBytes());     System.out.println(" [x] Sent '" + message + "'");          channel.close();     connection.close();   } }

**Consumer**：

import com.rabbitmq.client.ConnectionFactory; import com.rabbitmq.client.Connection; import com.rabbitmq.client.Channel; import com.rabbitmq.client.QueueingConsumer; public class Recv {          private final static String QUEUE\_NAME = "hello";     public static void main(String[] argv) throws Exception {     ConnectionFactory factory = new ConnectionFactory();     factory.setHost("localhost");     Connection connection = factory.newConnection();     Channel channel = connection.createChannel();     channel.queueDeclare(QUEUE\_NAME, false, false, false, null);     System.out.println(" [\*] Waiting for messages. To exit press CTRL+C");          QueueingConsumer consumer = new QueueingConsumer(channel);     channel.basicConsume(QUEUE\_NAME, true, consumer);          while (true) {       QueueingConsumer.Delivery delivery = consumer.nextDelivery();       String message = new String(delivery.getBody());       System.out.println(" [x] Received '" + message + "'");     }   } }

场景2：单发送多接收

使用场景：一个发送端，多个接收端，如分布式的任务派发。为了保证消息发送的可靠性，不丢失消息，使消息持久化了。同时为了防止接收端在处理消息时down掉，只有在消息处理完成后才发送ack消息。

发送端和场景1不同点：

1、使用“task\_queue”声明了另一个Queue，因为RabbitMQ不容许声明2个相同名称、配置不同的Queue

2、使"task\_queue"的Queue的durable的属性为true，即使消息队列durable

3、使用MessageProperties.PERSISTENT\_TEXT\_PLAIN使消息durable

When RabbitMQ quits or crashes it will forget the queues and messages unless you tell it not to. Two things are required to make sure that messages aren't lost: we need to mark both the queue and messages as durable.

接收端和场景1不同点：

1、使用“task\_queue”声明消息队列，并使消息队列durable

2、在使用channel.basicConsume接收消息时使autoAck为false，即不自动会发ack，由channel.basicAck()在消息处理完成后发送消息。

3、使用了channel.basicQos(1)保证在接收端一个消息没有处理完时不会接收另一个消息，即接收端发送了ack后才会接收下一个消息。在这种情况下发送端会尝试把消息发送给下一个not busy的接收端。

注意点：

1）It's a common mistake to miss the basicAck. It's an easy error, but the consequences are serious. Messages will be redelivered when your client quits (which may look like random redelivery), but RabbitMQ will eat more and more memory as it won't be able to release any unacked messages.

2）Note on message persistence

Marking messages as persistent doesn't fully guarantee that a message won't be lost. Although it tells RabbitMQ to save the message to disk, there is still a short time window when RabbitMQ has accepted a message and hasn't saved it yet. Also, RabbitMQ doesn't do fsync(2) for every message -- it may be just saved to cache and not really written to the disk. The persistence guarantees aren't strong, but it's more than enough for our simple task queue. If you need a stronger guarantee you can wrap the publishing code in atransaction.

3）Note about queue size

If all the workers are busy, your queue can fill up. You will want to keep an eye on that, and maybe add more workers, or have some other strategy.

场景3：Publish/Subscribe

使用场景：发布、订阅模式，发送端发送广播消息，多个接收端接收。

**Producer**：

import com.rabbitmq.client.ConnectionFactory; import com.rabbitmq.client.Connection; import com.rabbitmq.client.Channel; public class EmitLog {   private static final String EXCHANGE\_NAME = "logs";   public static void main(String[] argv) throws Exception {     ConnectionFactory factory = new ConnectionFactory();     factory.setHost("localhost");     Connection connection = factory.newConnection();     Channel channel = connection.createChannel();     channel.exchangeDeclare(EXCHANGE\_NAME, "fanout");     String message = getMessage(argv);     channel.basicPublish(EXCHANGE\_NAME, "", null, message.getBytes());     System.out.println(" [x] Sent '" + message + "'");     channel.close();     connection.close();   }      private static String getMessage(String[] strings){     if (strings.length < 1)             return "info: Hello World!";     return joinStrings(strings, " ");   }      private static String joinStrings(String[] strings, String delimiter) {     int length = strings.length;     if (length == 0) return "";     StringBuilder words = new StringBuilder(strings[0]);     for (int i = 1; i < length; i++) {         words.append(delimiter).append(strings[i]);     }     return words.toString();   } }

发送端：

发送消息到一个名为“logs”的exchange上，使用“fanout”方式发送，即广播消息，不需要使用queue，发送端不需要关心谁接收。

**Consumer**：

import com.rabbitmq.client.ConnectionFactory; import com.rabbitmq.client.Connection; import com.rabbitmq.client.Channel; import com.rabbitmq.client.QueueingConsumer; public class ReceiveLogs {   private static final String EXCHANGE\_NAME = "logs";   public static void main(String[] argv) throws Exception {     ConnectionFactory factory = new ConnectionFactory();     factory.setHost("localhost");     Connection connection = factory.newConnection();     Channel channel = connection.createChannel();     channel.exchangeDeclare(EXCHANGE\_NAME, "fanout");     String queueName = channel.queueDeclare().getQueue();     channel.queueBind(queueName, EXCHANGE\_NAME, "");          System.out.println(" [\*] Waiting for messages. To exit press CTRL+C");     QueueingConsumer consumer = new QueueingConsumer(channel);     channel.basicConsume(queueName, true, consumer);     while (true) {       QueueingConsumer.Delivery delivery = consumer.nextDelivery();       String message = new String(delivery.getBody());       System.out.println(" [x] Received '" + message + "'");        }   } }

接收端：

1、声明名为“logs”的exchange的，方式为"fanout"，和发送端一样。

2、channel.queueDeclare().getQueue();该语句得到一个随机名称的Queue，该queue的类型为non-durable、exclusive、auto-delete的，将该queue绑定到上面的exchange上接收消息。

3、注意binding queue的时候，channel.queueBind()的第三个参数Routing key为空，即所有的消息都接收。如果这个值不为空，在exchange type为“fanout”方式下该值被忽略！

场景4：Routing (按路线发送接收)

使用场景：发送端按routing key发送消息，不同的接收端按不同的routing key接收消息。

**Producer**：

import com.rabbitmq.client.ConnectionFactory; import com.rabbitmq.client.Connection; import com.rabbitmq.client.Channel; public class EmitLogDirect {   private static final String EXCHANGE\_NAME = "direct\_logs";   public static void main(String[] argv) throws Exception {     ConnectionFactory factory = new ConnectionFactory();     factory.setHost("localhost");     Connection connection = factory.newConnection();     Channel channel = connection.createChannel();     channel.exchangeDeclare(EXCHANGE\_NAME, "direct");     String severity = getSeverity(argv);     String message = getMessage(argv);     channel.basicPublish(EXCHANGE\_NAME, severity, null, message.getBytes());     System.out.println(" [x] Sent '" + severity + "':'" + message + "'");     channel.close();     connection.close();   }      private static String getSeverity(String[] strings){     if (strings.length < 1)             return "info";     return strings[0];   }   private static String getMessage(String[] strings){      if (strings.length < 2)             return "Hello World!";     return joinStrings(strings, " ", 1);   }      private static String joinStrings(String[] strings, String delimiter, int startIndex) {     int length = strings.length;     if (length == 0 ) return "";     if (length < startIndex ) return "";     StringBuilder words = new StringBuilder(strings[startIndex]);     for (int i = startIndex + 1; i < length; i++) {         words.append(delimiter).append(strings[i]);     }     return words.toString();   } }

发送端和场景3的区别：

1、exchange的type为direct

2、发送消息的时候加入了routing key

**Consumer**：

import com.rabbitmq.client.ConnectionFactory; import com.rabbitmq.client.Connection; import com.rabbitmq.client.Channel; import com.rabbitmq.client.QueueingConsumer; public class ReceiveLogsDirect {   private static final String EXCHANGE\_NAME = "direct\_logs";   public static void main(String[] argv) throws Exception {     ConnectionFactory factory = new ConnectionFactory();     factory.setHost("localhost");     Connection connection = factory.newConnection();     Channel channel = connection.createChannel();     channel.exchangeDeclare(EXCHANGE\_NAME, "direct");     String queueName = channel.queueDeclare().getQueue();          if (argv.length < 1){       System.err.println("Usage: ReceiveLogsDirect [info] [warning] [error]");       System.exit(1);     }          for(String severity : argv){           channel.queueBind(queueName, EXCHANGE\_NAME, severity);     }          System.out.println(" [\*] Waiting for messages. To exit press CTRL+C");     QueueingConsumer consumer = new QueueingConsumer(channel);     channel.basicConsume(queueName, true, consumer);     while (true) {       QueueingConsumer.Delivery delivery = consumer.nextDelivery();       String message = new String(delivery.getBody());       String routingKey = delivery.getEnvelope().getRoutingKey();       System.out.println(" [x] Received '" + routingKey + "':'" + message + "'");        }   } }

接收端和场景3的区别：

在绑定queue和exchange的时候使用了routing key，即从该exchange上只接收routing key指定的消息。

场景5：Topics (按topic发送接收)

使用场景：发送端不只按固定的routing key发送消息，而是按字符串“匹配”发送，接收端同样如此。

发送端和场景4的区别：

1、exchange的type为topic

2、发送消息的routing key不是固定的单词，而是匹配字符串，如"\*.lu.#"，\*匹配一个单词，#匹配0个或多个单词。

接收端和场景4的区别：

1、exchange的type为topic

2、接收消息的routing key不是固定的单词，而是匹配字符串。

注意点：

Topic exchange

Topic exchange is powerful and can behave like other exchanges. When a queue is bound with "#" (hash) binding key - it will receive all the messages, regardless of the routing key - like in fanout exchange. When special characters "\*" (star) and "#" (hash) aren't used in bindings, the topic exchange will behave just like a direct one.

# 每个程序员应该阅读的10本经典书籍

2016-03-07 ifeve [并发编程网](javascript:void(0);)

如果你是一个程序员，除了编码之外，你还需要大量的阅读。今天我要为大家介绍几本值得一读的书，包括《The Pragmatic Programmer》，《The Mythical Man-month: Essays on Software Engineering》和《Clean Code: A Handbook of Agile Software Craftsmanship》。

书籍是知识和智慧的重要来源。但不幸的是，现在很多人已经不愿意看书了。程序员更是罕见地会去读书，最常见的依靠互联网搜索结果来找寻答案。

技术向前的步伐比人类历史上的任何时候都要走得更快。用不了几个月，就会有新的编程语言和工具问世，弥补现有语言、工具和方法的缺陷。

事实上，许多伟人都已经遇到过差不多的问题，并指出了解决这些问题的最佳途径。而这些方法和解决方案都收录在一些超棒的书籍中。

下面就是在这个行业中开发人员应该阅读的一系列伟大的书籍。

**《The Pragmatic Programmer》**

绝对是书籍中的瑰宝！这不是常规地建议你编码，编码还是编码的编程书。事实上，它并不限定于某种特定的编程语言：在这本书中的智慧适用于所有编程语言。

这本书对许多有趣的领域都提出了真知灼见，如各种探索性编程，在代码中编码，从模型中分离的观点，昂贵的工具并不产生更好的设计，开发一个伟大的团队，管理预期，避免知识的重复等。

这本书不仅可以帮助改变编码的习惯，还可以改变你作为一个程序员的性格。它充满了关于如何改进自己和代码的实用建议。

还有一个总结了提示和检查清单的小册子。

**《The Mythical Man-month》: 关于软件工程的散文集**

非常经典，被奉为软件行业的圣经。第一次出版于 1970 年，但是里面的内容比起以前，可能更适用于现在！

有听说过这些话吗？它们均摘自于这本书！

“所有的程序员都是乐观的：一切都会顺利。”

“添加人手到一个延迟的项目中只会导致完成得更慢。”

“生一个孩子总是需要九个月的时间，不管安排多少个女性。”

“一个煎蛋，承诺在两分钟内完成，但如果两分钟后还是没有准备好，那么客户有两种选择——等待或吃半熟品，软件客户也只能这样选择。”

不幸的是，一年又一年地过去，而我们总是在软件开发中犯着相同的错误。这本书是每一个项目经理和开发人员都必须阅读的。

正从标题中所说的那样，这是一本散文集。文辞优美。这本书唯一的缺点就是引用了年迈 25 的古老技术。但是，这并不影响这本书的魅力。

**《Clean Code》: 敏捷软件工艺的手册**

有没有在看他人代码的过程中，不由自主地发出“哦，天哪，这是什么？”的经历，那么这个人肯定没有阅读过《Clean Code》。

这是一本关于软件工艺史诗般的书籍。这本书不仅会告诉你如何编写好的代码，而且还提供了软件开发的高效途径。照着去做的话，必将改变你的工作前景。

书中描述了编写干净代码的原则、模式和做法。里面一些关于整洁代码的几个案例都是开发人员宝贵的经验教训。

请注意，虽然在这本书中的所有实例都是关于 Java 的，但是从中学到的经验教训可以应用于任意的编程语言。

这篇文章所列出的这些书籍中，这本书出版得比较晚，所以可能更能引起年轻开发人员的共鸣。

**《The Clean Coder》：专业程序员的行为守则**

此列表中 Rob Martin 的第二本书。建议你在读了《Clean Code》后，再读这本书。《Clean Code》讲的是代码，而这本书是关于“Coder”。

该书探讨了一些程序员经常忽视的主题。

1. 成为专业的程序员意味着什么？
2. 如何打磨自己成为一个真正的软件工匠
3. 冲突和紧张的日程处理
4. 如何管理你的时间？如何扩张技能？
5. 何时说“不”
6. 避免倦怠
7. ..以及更多。

你可能并不总是同意作者的观点，但它提供了良好的精神食粮。这可能并非你所期望的，但可能正是你所需要的。

**《Refactoring》：改善已有代码的设计**

不管你怎么努力，除非改进它，否则，你交付的代码不会是最优化的。有时即使工作正常，也会实施重构。

这本书从重构的通用原则说起：为什么以及什么时候重构，如何处理有关重构的管理等等。然后讲述了如何实现改进的过程。

1. 代码的设计缺陷指标是什么？
2. 如何构建类、方法和其他的逻辑块？
3. 单元测试
4. 如何将功能从一个对象移动到另一个？
5. 重构工具
6. ..以及更多

这是改进现有代码必读的书。请注意，所有的代码示例用的都是 Java，但现在复制起来也很方便！

**《Working Effectively With Legacy Code》**

我们都必须工作于一些我们痛恨的东西——对于大多数人而言，遗留代码真是令人头痛无比。

如何修改遗留代码？如何识别需要重构的代码部分？如何破坏重构代码之间的依赖关系？如何确保新的代码能完成预期的工作？如何一次一小步地重构遗留代码？

在这本杰出的书中，只是回答了一些关键的问题。如果非要用一句话总结这本书的精华，那就是“写单元测试，重构代码，确保测试都通过。”

遗留代码不是一个神话，它是一个活着的传奇！在软件行业中没有什么比遗留代码更能经受测试的考验了——Deepak Karanth

**《Code Complete》：软件构建的实用手册**

在一个庞大的作品中，如果你想要阅读所有关于编程结构和最佳实践的内容，那么这是本必读书。真正的百科全书式书籍——其最新版本有多达 960 页！不要被这本书的厚度吓倒，你可以按照自己的节奏阅读。最后，你会庆幸你阅读了这本书。

书中解释了软件开发的每个方面。从代码结构，代码格式化，到变量、方法和类的命名，再一路说到管理一个团队，对所有一切都提出了实用的建议。

提供了覆盖特定主题的丰富参考和补充材料，这些也非常值得一读。

只有一小部分的软件开发人员会读这本书，所以如果你也是他们中的一个，那么你就有了优势。通过阅读这本书，你就可以获得许多年宝贵经验。

**《Head First Design Patterns》**

看上去最不像技术的编程书籍！每个页面都包含涂鸦、图片以及其他一些吸引眼球的东西。可能给人的印象是一本阅读起来很轻松的书，但事实上它会讨论编程的一些核心主题——设计模式。

这本书虽然没有覆盖所有存在于这个世界的模式，但是会涵盖所有你可能需要用于解决现实问题的模式。它将帮助你创建功能性的，优雅的，可重用的和灵活的软件。每个模式的优劣也被明确指出。大多数关于设计模式的书籍谈论的是如何实现模式，但这本书的作者同时还解释了为什么以及怎么样。

最新版本包括针对 Java 8 的更新——主要是 Lambda。

**《Peopleware: Productive Projects and Teams》**

很棒的一部作品。这本书并非关于编程。这是一本有关管理和激励程序员的书籍。开发人员也应该阅读。很多时候，开发人员，尤其是那些没有经验的开发人员，不理解管理的思维过程。

软件开发是一个创造性的过程。但是，大多数管理人员把它当作是流水线。开发人员被视为是机器上可替换的齿轮零部件。管理人员普遍性地会给予一个紧迫的时间期限，当作促进积极性唯一途径。他们对开发人员的工作不感兴趣，甚至可悲的是，他们也不会试着去理解开发人员或他们自己的工艺。

如果你想成为一个想要的是质量，而不仅仅是数量的管理人员，那么请立刻阅读这本书！

作者解释了管理者应该如何以一种可持续的方式使他们的软件开发团队认识到他们的潜力。

**《Soft Skills: The Software Developer’s Life Manual》**

同样的，这也不是一本关于编程的书。但是，却是每一个程序员都应该阅读的书。

这本伟大的书着重于管理开发人员生活的“其他”方面。可以是你生活的每一个方面——事业、生活、身体、头脑，以及不管你相信与否——还有灵魂。

作者他自己也遵循这些技术，并且获得了成功。他的网站上说，他能够在他 30 出头的时候放弃他的日常工作。该作者将他的生活经验整理成整齐的，主题内容为 Career、Marketing yourself、Learning、Productivity、Finances, Fitness 和 Spirit 的短章。每个篇章都很短，可以在休息时间阅读，非常方便。

你会是一个更加满意和快乐的人，如果你按照这本书的建议去做的话，那么你将成为一个更令人满意和幸福的人和程序员。

# 行业人的读白，预测物联网技术趋势走向

2015-07-31 [粤嵌IT研发编程技术宅](javascript:void(0);)

信息世界的核心产品，我认为就三种：芯片、操作系统、应用。芯片是作为硬件系统的核心，操作系统是软件系统的核心，应用是用户体验的核心。

PC时代，Intel， 微软，google可以分别代表这三大核心力量。手机时代，高通、安卓、微信可以分别代表这几个力量。

这里说的时代，都是以交互方式来区分的。互联网如果从信息的角度来讲，它只是信息交换的一种形态，跟PC时代和手机时代的意义不太一样，因此，这里没有将它称为一个时代参照。

物联网会是继手机之后的下一个时代，这已经是产业共同的判断。如我之前所说，物联网时代会发两个显着的应用形态变化：

　　1.信息中心和交互中心的完全分离；

　　2.交互方式的多样化、交互融入生活场景，使用更自然。

**基于这两点应用形态的变化，以前的PC和手机的技术形态会完全不适用，物联网行业的技术趋势也会彻底变化。**

　　物联网的应用技术需求特点

**1.终端与运算的完全分离**

　　这里的终端等同于PC时代的外设，可以包括传感、控制器、交互设备等等。典型的应用就是远程控制。

**2.无线通信终端百倍增长**

　　人均电子终端数量会爆炸性增长，生活的每一个场景都有可能会形成一个无线应用终端。

**3.终端应用场景的碎片化**

　　终端会直接融合到生活的各种场景之中。每一个场景都是一个细分应用，吃饭、喝水、卧室听音乐、客厅玩游戏都会形成一个终端。

**4.终端处理能力的多样化**

　　应用场景的多样化，出于成本、功耗的考虑，终端处理能力的要求也会各种各样。

**5.云端数据量兆倍以上增长**

　　手机时代，移动互联网的数据还仅仅只是个人生活信息存储到了云端。当设备量倍增，个人周边所有相关的信息汇集到云端的时候，这个数据量级会是惊人的增长。

**物联网时代的芯片行业**

**1.群雄并起，绝对是长尾市场。**

　　根据不同的应用场景，单片机、PLC、ARM、X86都可以形成最适合自己的应用市场。

　　每个细分应用场景都有可能会跑出一家不同的CPU企业。而不会像PC时代一样，有一家可以占有市场50%以上的市场份额。

**2.SoC会是大趋势**

　　物联网应用终端的处理会更专业化，对于终端处理芯片的综合性能不会要求太高，而更要求低功耗、小型。

　　因此，结合应用场景的一体化芯片是更优的解决方案。目前越来越多的传感SoC，通信SoC类芯片已经在彰显这一趋势。

**3.中国会出现一波物联网时代的知名芯片企业**

　　目前的中国的产业积累，以及后续广大的应用市场，都能在中国催生出一波有影响力的芯片企业，特别是传感器、射频通信、处理器。

**物联网操作系统：**

**1.云端运算、终端处理**

　　这种架构已经在移动互联网中已经得到了充分的验证。

　　在物联网时代，这种方式会更加走向极致。

**2.操作系统的应用部分全部位于云端**

　　移动互联网已经看到的数据管理、各种业务应用会扩展到物联相关的其它个人关注数据。

　　而更关键的是，硬件开发的IDE、硬件管理的设备管理工具等也都会云端化。

**3.物联网操作系统由终端、网络、云端三个部分共同组成**

　　PC和手机操作系统的重心，依然是在于本地硬件设备的管理。

　　但是物联网应用非常依赖于通信和云端。因此，在物联网操作系统中，网络通信和云端的管理比重会非常大。

**4.终端嵌入式系统内核轻量级；终端系统功能要求组件化、模块化、可裁剪**

　　这是由终端的多样化决定的。

　　嵌入式操作系统要满足硬件配置极低场景，比如MCU。同时，应用终端的功能会非常多样化。组件化、积木式的嵌入式操作系统才能满足要求。

　　LINUX已经做了很好的示范，但是物联网操作系统的终端部分要求更进一步。

**5.远程通信要求安全、高效、动态**

　　个人隐私数据、海量通信数据、无线通信网络环境，这些特征要求物联网操作系统在通信上结合业务，做更多的优化。

**6.通信同时要求支持基本的P2P通信**

　　物联，同时也指物与物之间的联结。

　　远程通信解决的是远距离信息交换、或者需要复杂计算的应用场景。

　　在近距离、同时基本计算与逻辑处理场景，物与物之间直接通信效率更高。

**7.中国会出现一个真正意义上的操作系统级的产品**

　　这依然是由产业发展，应用需求催生的。

# 如何阅读Android系统源码－收藏必备

字数1465 阅读7652 评论20 喜欢375



对于任何一个对Android开发感兴趣的人而言，对于android系统的学习必不可少。而学习系统最佳的方法就如linus所言:"RTFSC"(Read The Fucking Source Code)。  
下面从知乎整理了一些优质回答，以飨读者。

## 巨人的肩膀

* AOSP项目官方: <https://source.android.com/source/index.html>  
  **这个一定要先读**. 项目介绍, 代码下载, 环境搭建, 刷机方法, Eclipse配置都在这里. 这是一切的基础.
* Android官方手册: <https://developer.android.com/training/index.html>  
  这个其实是给App开发者看的. 但是里面也有不少关于系统机制的介绍, 值得细读.
* 老罗的Android之旅: <http://blog.csdn.net/luoshengyang>  
  此老罗非彼老罗. 罗升阳老师的博客非常有营养, 基本可以作为**指引你开始阅读AOSP源码的教程**. 你可以按照博客的时间顺序一篇篇挑需要的看.但这个系列的博客有些问题:  
  早期的博客是基于旧版本的Android;  
  大量的代码流程追踪. 读文章时你一定要清楚你在看的东西在整个系统处于什么样的位置.
* Innost的专栏: <http://blog.csdn.net/innost>  
  邓凡平老师也是为Android大牛, 博客同样很有营养. 但是不像罗升阳老师的那么系统. 更多的是一些技术点的深入探讨.
* Android Issues: <http://code.google.com/p/android/issues/list>  
  Android官方Issue列表. 我在开发过程中发现过一些奇怪的bug, 最后发现这里基本都有记录. 当然你可以提一些新的, 有没有人改就是另外一回事了.
* Google: [https://www.google.com](https://www.google.com/)  
  一定要能流畅的使用这个工具. 大量的相关知识是没有人系统的总结的, 你需要自己搞定.

## 阅读方法

假设我想研究Android的UI系统，首先要找什么和UI有亲戚关系吧！  
View大神跳出来了，沿着它往下找找看，发现它在贴图在画各种形状，但是它在哪里画呢，马良也要纸吧？  
开发Android的同学逃不掉Activity吧！它有个setcontentview()的方法，从这个名字看好像它是把view和activity结合的地方。赶紧看它的实现和被调用，然后我们就发现了Window，ViewRoot和WindowManager的身影，沿着WM和WMS我们就惊喜会发现了Surface，以及draw的函数，它居然在一个DeCorView上画东西哈。借助Source Insight， UIJava层的横向静态图呼之欲出了。  
完成这个静态UML，我觉得我可以开始功能实现上追踪了，这部分主要是C++的代码（这也是我坚定劝阻的放弃Eclipse的原因），我沿着draw函数，看到了各个层级的关系，SurfaceSession的控制和事务处理，SharedBuffer读写控制，彪悍的SurfaceFlinger主宰一切，OpenGL ES的神笔马良。FrameBuffer和FrameBufferDevice的图像输出，LCD设备打开后，开始接收FBD发过来的一帧帧图像，神奇吧。  
好吧，就这样，再往底层我爱莫能助了！

## 软件

当我决定要阅读源码，要具备一款好用的阅读器、下载源码等

* Windows阅读器: [**Source Insight**](http://www.sourceinsight.com/)

在这个工具帮助下，你才可以驾驭巨大数量的Android 源码，你可以从容在Java，C++,C代码间遨游，你可以很快找到你需要的继承和调用关系。

* Mac OS阅读器: [**Understand**](http://www.scitools.com/),

参考：[OS X 下真怀念 Source Insight](http://www.v2ex.com/t/103051" \t "_blank)

* 源码下载：  
  如果你有梯子：  
  [官方下载](https://source.android.com/source/building.html)  
  [git路径](https://android.googlesource.com/)  
  如果没有:  
  [github路径](https://github.com/android/platform_frameworks_base) （可以直接download zip或者使用git clone）

**欢迎读者提供优质下载路径(镜像等)来共享**

## 相关知识

* **Java**  
  Java是AOSP的主要语言之一. 没得说, 必需熟练掌握.  
  熟练的Android App开发
* **Linux**  
  Android基于Linux的, 并且AOSP的推荐编译环境是Ubuntu 12.04. 所以熟练的使用并了解Linux这个系统是必不可少的. 如果你想了解偏底层的代码, 那么必需了解基本的Linux环境下的程序开发. 如果再深入到驱动层, 那么Kernel相关的知识也要具备.
* **设计模式**  
  去学习一下，android系统里的代码很多地方都闪烁着设计模式的光芒，这也是你成为大牛的必经之路.当然你只要先了解一下，在阅读中慢慢感受就行。
* **Make**  
  AOSP使用Make系统进行编译. 了解基本的Makefile编写会让你更清晰了解AOSP这个庞大的项目是如何构建起来的.
* **Git**  
  AOSP使用git+repo进行源码管理. 这应该是程序员必备技能吧.
* **C++**  
  Android系统的一些性能敏感模块及第三方库是用C++实现的, 比如: Input系统, Chromium项目(WebView的底层实现).

## 感谢知乎及知乎er

[大牛们是怎么阅读 Android 系统源码的？](http://www.zhihu.com/question/19759722)

## 彩蛋

最后，很多优秀资源来源于国外，如果android学习者连android官网都打不开的话，那就有点。。。  
关于vpn要**委婉吐槽**一句：

与其像本人以前一样花一大堆时间去搜索各种不稳定的**免费vpn**，还真心不如花个十几块钱省一个月心来的实在。学习技术总要支付学费，但购买优质vpn这点个人觉得是性价比极高也极机智的做法。

## 关于作者

欢迎关注本人的[Github](https://github.com/wingjay) <https://github.com/wingjay>, 及阅读本人相关文章  
[《如何在一天之内完成一款具备cool属性的Android产品<简诗>》](http://www.jianshu.com/p/cf496fc408b2)

# 请大家记住这道题，这不是巧合！

2015-03-04 [大学生励志网](javascript:void(0);)

http://mmbiz.qpic.cn/mmbiz/bMyuXcCdTEHTPKKXOAw3K1J5YVvg2QLNWZfS82vDqVYv8Be0uWnQ0tC9JibysWaw7ibyF3ppyoAtZMkUgVZT4D9A/0?tp=webp&wxfrom=5&wx_lazy=1

http://mmbiz.qpic.cn/mmbiz/gBJLhicQA6iaTBuvDb7DBkEcLObYF1nczyFN6hQ8qvdQvgueBFyBTwx2VovOILQ9JJYVUAyTGK2k9zUZia4oFdy8g/0?tp=webp&wxfrom=5&wx_lazy=1回复“PPT”免费下载1200套绝美PPT模版

如果令 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 分别等于

　　1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26

　　那么

**Hard work （努力工作）**

　　H+A+R+D+W+O+R+K＝8+1+18+4+23+15+18+11=98%

**Knowledge（知识）**

　　K+N+O+W+L+E+D+G+E＝11+14+15+23+12+5+4+7+5=96%

**Love（爱情）**

　　L+O+V+E＝12+15+22+5=54%

**Luck（好运）**

　　L+U+C+K＝12+21+3+11=47%

　　看出来没？这些我们通常认为重要的东西往往并不是最重要的。

　　那么，什么可以生活得圆满？

**是Money（金钱）吗？**

　　不! M+O+N+E+Y=13+15+14+5+25=72%

**是Leadership（领导能力）吗？**

　　不! L+E+A+D+E+R+S+H+I+P=12+5+1+4+5+18+19+9+16=89%

　　金钱，权力都不能使圆满，什么能使我们的生活变成100%的圆满呢？

　　每个问题都有其解决之道，只要你把目光放得远一点！

　　其实，真正能使我们生活圆满的东西就在我们自己身上！

**ATTITUDE（心态）**

　　A+T+T+I+T+U+D+E＝1+20+20+9+20+21+4+5=100%

　　我们对待工作、学习的态度能够使我们的生活达到100%的圆满！

　　你用什么态度去看待世界，你就会得到什么样的世界！