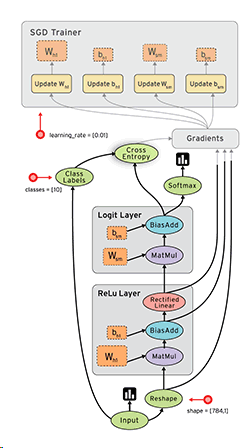
# Tensorflow代码分析

TensorFlow基于数据流图，用于大规模分布式数值计算的开源框架。节点表示某种抽象的计算，边表示节点之间相互联系的张量。



TensorFlow支持各种异构的平台，支持多CPU/GPU，服务器，移动设备，具有良好的跨平台的特性；TensorFlow架构灵活，能够支持各种网络模型，具有良好的通用性；此外，TensorFlow架构具有良好的可扩展性，对OP的扩展支持，Kernel特化方面表现出众。

TensorFlow最初由Google大脑的研究员和工程师开发出来，用于机器学习和神经网络方面的研究，于2015.10宣布开源，在众多深度学习框架中脱颖而出，在Github上获得了最多的Star量。

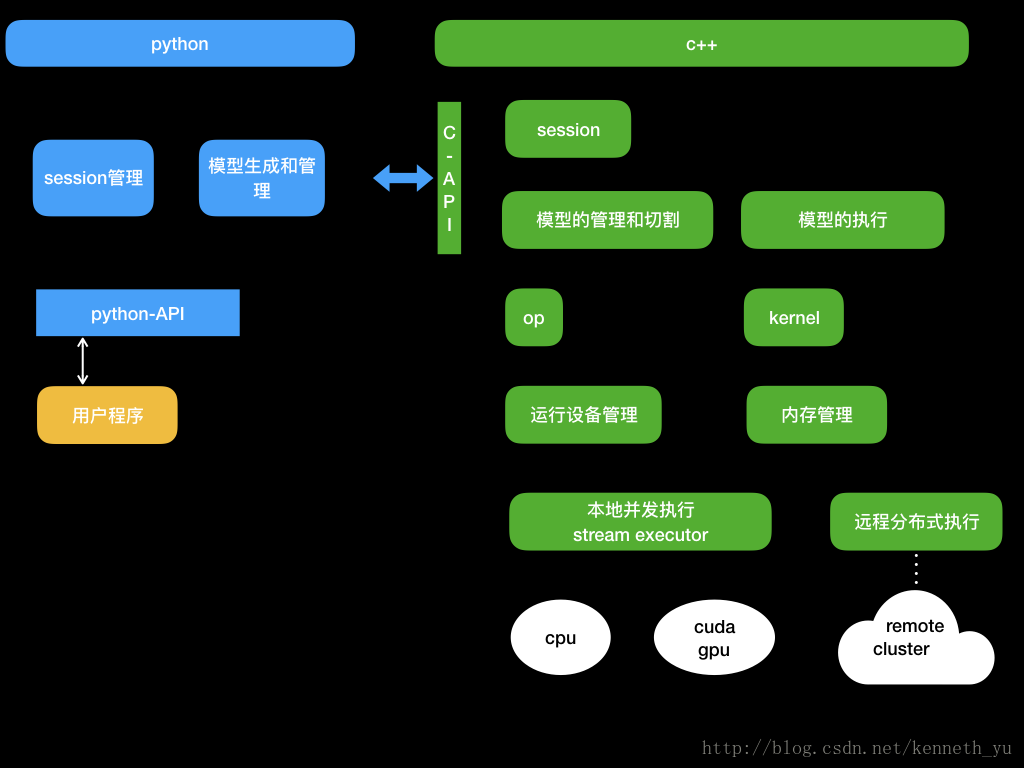
本文将阐述TensorFlow的系统架构，帮助读者加深理解TensorFlow的工作机理。

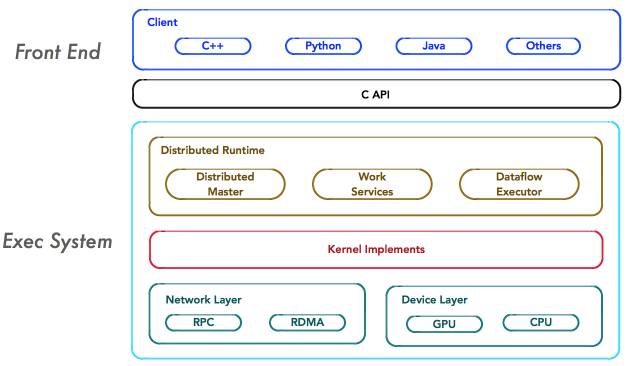
## 1.系统概述

TensorFlow的系统结构以C API为界，将整个系统分为「前端」和「后端」两个子系统：

前端系统：提供编程模型，负责构造计算图；

后端系统：提供运行时环境，负责执行计算图。





如上图所示，重点关注系统中如下4个基本组件，它们是系统分布式运行机制的核心。

（1）Client

Client是前端系统的主要组成部分，它是一个支持多语言的编程环境。它提供基于计算图的编程模型，方便用户构造各种复杂的计算图，实现各种形式的模型设计。Client通过Session为桥梁，连接TensorFlow后端的「运行时」，并启动计算图的执行过程。

（2）Distributed Master

在分布式的运行时环境中，Distributed Master根据Session.run的Fetching参数，从计算图中反向遍历，找到所依赖的「最小子图」。然后，Distributed Master负责将该「子图」再次分裂为多个「子图片段」，以便在不同的进程和设备上运行这些「子图片段」。最后，Distributed Master将这些「子图片段」派发给Work Service；随后Work Service启动「子图片段」的执行过程。

（3）Worker Service

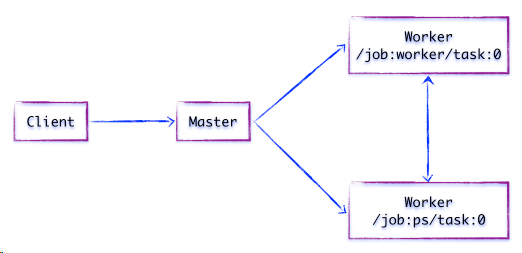
对于每以个任务，TensorFlow都将启动一个Worker Service。Worker Service将按照计算图中节点之间的依赖关系，根据当前的可用的硬件环境(GPU/CPU)，调用OP的Kernel实现完成OP的运算(一种典型的多态实现技术)。

另外，Worker Service还要负责将OP运算的结果发送到其他的Work Service；或者接受来自其他Worker Service发送给它的OP运算的结果。

（4）Kernel Implements

Kernel是OP在某种硬件设备的特定实现，它负责执行OP的运算。

### 1.1.组件交互



如上图所示，假设存在两个任务：

/job:ps/task:0: 负责模型参数的存储和更新

/job:worker/task:0: 负责模型的训练或推理

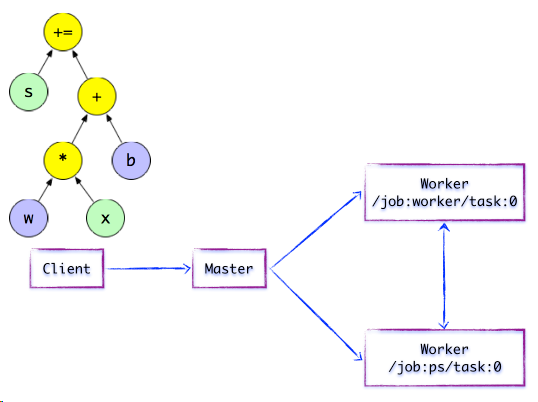
接下来，我们将进一步抽丝剥茧，逐渐挖掘出TensorFlow计算图的运行机制。

（1）客户端

Client基于TensorFlow的编程接口，构造计算图。目前，TensorFlow主流支持Python和C++的编程接口，并对其他编程语言接口的支持日益完善。

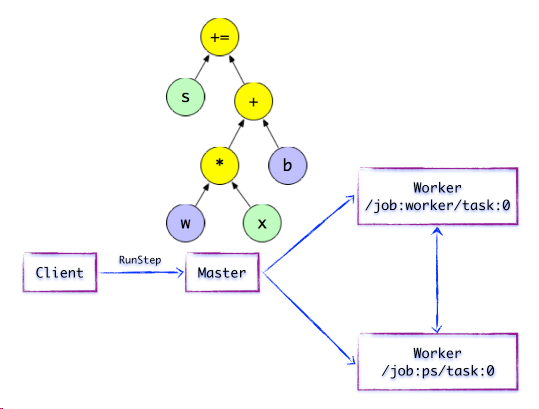
此时，TensorFlow并未执行任何计算。直至建立Session会话，并以Session为桥梁，建立Client与后端运行时的通道，将Protobuf格式的GraphDef发送至Distributed Master。也就是说，当Client对OP结果进行求值时，将触发Distributed Master的计算图的执行过程。

如下图所示，Client构建了一个简单计算图。它首先将w与x进行矩阵相乘，再与截距b按位相加，最后更新至s。

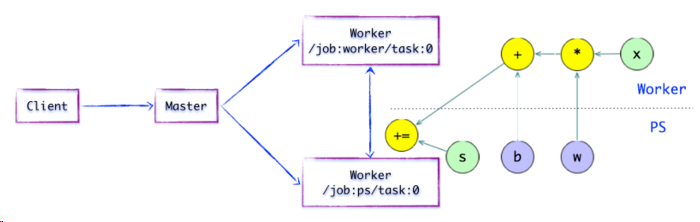


（2）Distributed Master

在分布式的运行时环境中，Distributed Master根据Session.run的Fetching参数，从计算图中反向遍历，找到所依赖的最小子图。然后Distributed Master负责将该子图再次分裂为多个「子图片段」，以便在不同的进程和设备上运行这些「子图片段」。最后，Distributed Master将这些图片段派发给Work Service。随后Work Service启动「本地子图」的执行过程。Distributed Master将会缓存「子图片段」，以便后续执行过程重复使用这些「子图片段」，避免重复计算。

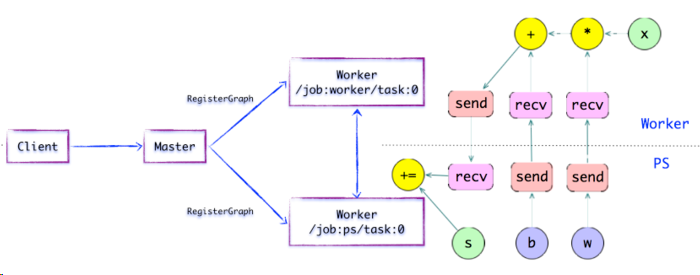


（3）子图片段



如上图所示，存在一种合理的「子图片段」划分算法。Distributed Master将模型参数相关的OP进行分组，并放置在PS任务上。其他OP则划分为另外一组，放置在Worker任务上执行。

（4）SEND/RECV节点



如上图所示，如果计算图的边被任务节点分割，Distributed Master将负责将该边进行分裂，采用Rendezvous机制在两个分布式任务之间插入SEND和RECV节点，实现数据的传递。

随后，Distributed Master将「子图片段」派发给相应的任务中执行，在Worker Service成为「本地子图」，它负责执行该子图的上的OP。

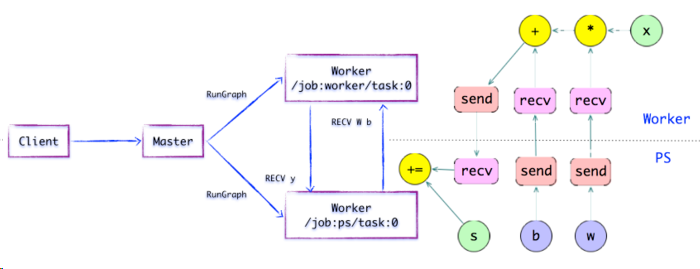
（5）Worker Service

对于每个任务，都将存在相应的Worker Service，它主要负责如下3个方面的职责：

处理来自Master的请求；

调度OP的Kernel实现，执行本地子图；

协同任务之间的数据通信。



Worker Service派发OP到本地设备，执行Kernel的特定实现。它将尽最大可能地利用多CPU/GPU的处理能力，并发地执行Kernel实现。

另外，TensorFlow根据设备类型，对于设备间的SEND/RECV节点进行特化实现：

使用cudaMemcpyAsync的API实现本地CPU与GPU设备的数据传输；

对于本地的GPU之间则使用端到端的DMA，避免了跨host CPU昂贵的拷贝过程。

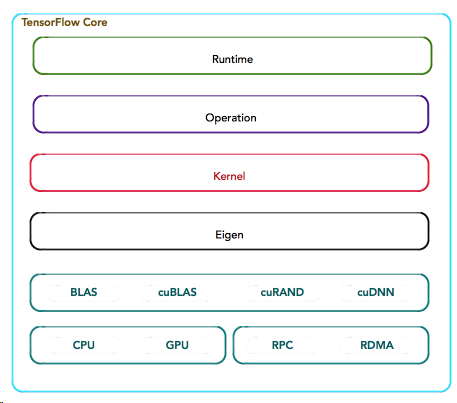
对于任务之间的数据传递，TensorFlow支持多协议，主要包括：

gRPC over TCP

RDMA over Converged Ethernet

（6）Kernel Implements

TensorFlow的运行时包含200多个标准的OP，包括数值计算，多维数组操作，控制流，状态管理等。每一个OP根据设备类型都会存在一个优化了的Kernel实现。在运行时，运行时根据本地设备的类型，为OP选择特定的Kernel实现，完成该OP的计算。



其中，大多数Kernel基于Eigen::Tensor实现。Eigen::Tensor是一个使用C++模板技术，为多核CPU/GPU生成高效的并发代码。但是，TensorFlow也可以灵活地直接使用cuDNN实现更高效的Kernel。

此外，TensorFlow实现了矢量化技术，使得在移动设备，及其满足高吞吐量，以数据为中心的应用需求，实现更高效的推理。

如果对于复合OP的子计算过程很难表示，或执行效率低下，TensorFlow甚至支持更高效的Kernle实现的注册，其扩展性表现相当优越。

### 1.2.技术栈

最后，按照TensorFlow的软件层次，通过一张表格罗列TensorFlow的技术栈，以便更清晰地对上述内容做一个简单回顾。



作者：\_RayCloud\_

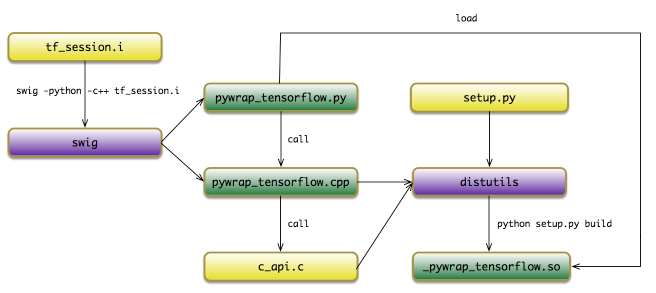
链接：https://www.jianshu.com/p/a5574ebcdeab

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 2.实现细节

### 2.1.Python到C++接口--swig使用



TensorFlow架构与设计：会话生命周期: https://www.jianshu.com/p/667cbb20d802

### 2.2.graph表示

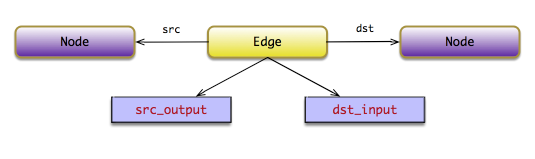
（1）普通边：Edge持有前驱节点与后驱节点，从而实现了计算图的连接，也是计算图前向遍历，后向遍历的衔接点。

边上的数据以Tensor的形式传递，Tensor的标识由源节点的名称，及其所在边的src\_output唯一确定。也就是说，tensor\_id = op\_name:src\_output

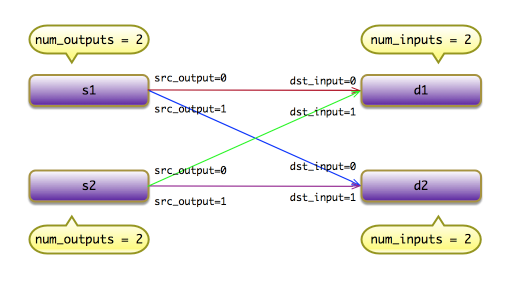
Edge持有两个重要的属性：

src\_output：表示该边为前驱节点的第src\_output条输出边；

dst\_input：表示该边为后驱节点的第dst\_input条输入边。



例如，存在两个前驱节点s1, s2，都存在两条输出边；存在两个后驱节点d1, d2，都存在两条输入边。



（2）控制依赖边

计算图中存在两类边：

普通边：用于承载Tensor，常用实线表示；

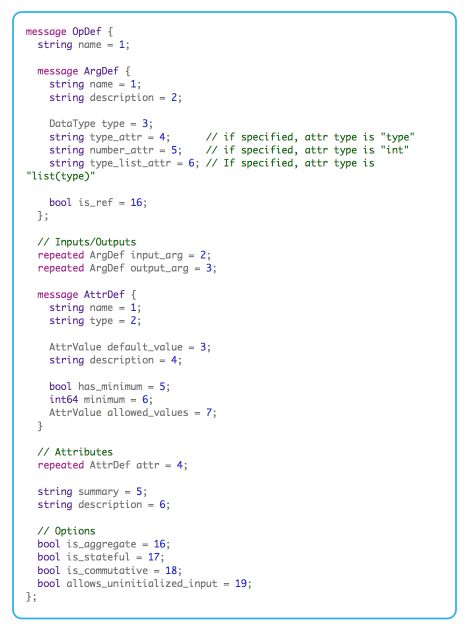
控制依赖：控制节点的执行顺序，常用虚线表示。

特殊地，控制依赖边，其src\_output, dst\_input都为-1(Graph::kControlSlot)，暗喻控制依赖边不承载任何数据，仅仅表示计算的依赖关系。

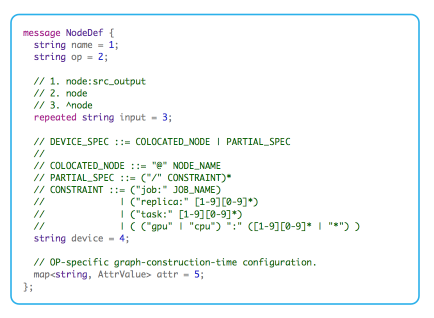
TensorFlow架构与设计：图模块： https://www.jianshu.com/p/a6d18c144052

### 2.3.OP数据结构

（1）OpDef定义



（2）NodeDef定义



（3）输入列表

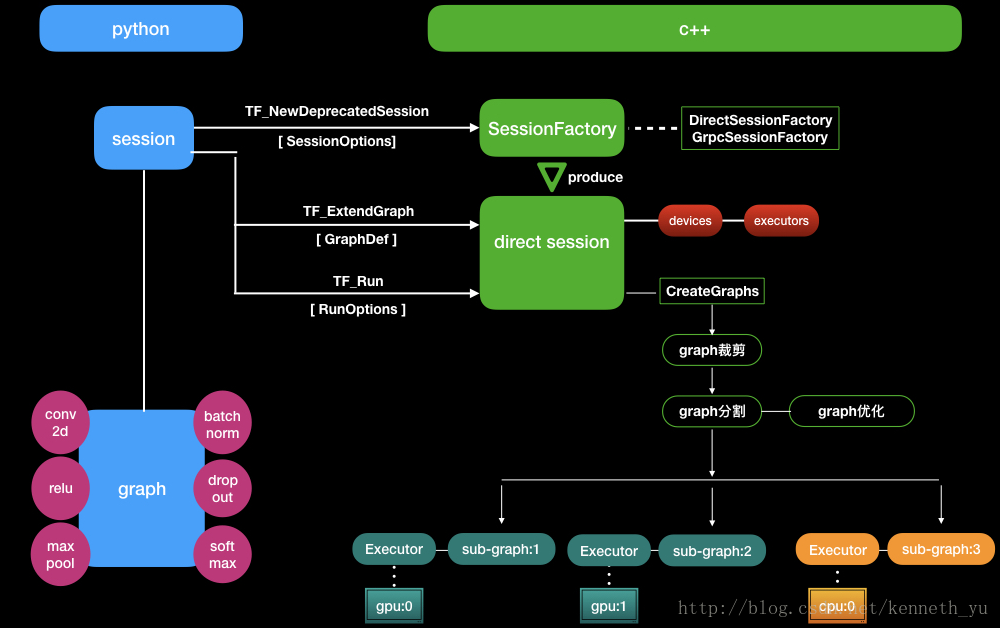
通过input指定节点的输入列表，它也是构造计算图最重要的知识所在。它存在2种情况，分别表示普通边与控制依赖边。按照约定，为了解析方便，input列表前面存储普通边，随后存储控制依赖边。

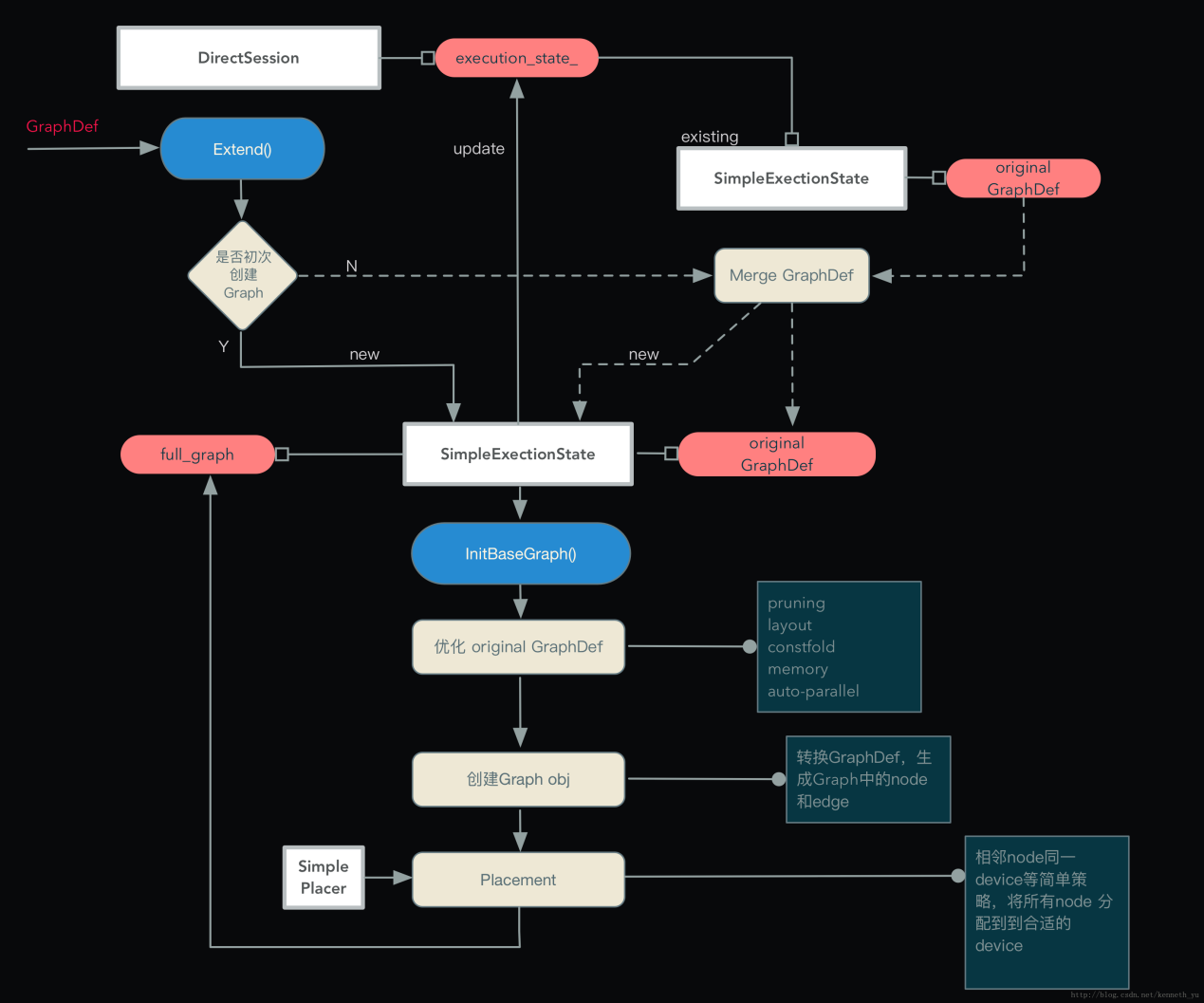
node:src\_output表示此边为普通边，承载Tensor的数据流。其中，node为前驱节点的名称，src\_output为前驱节点输出边的索引。特殊地，当src\_output为0时，可以略去0。

^node表示该边为控制依赖边。其中，node为前驱节点的名称。

TensorFlow架构与设计：OP本质论：https://www.jianshu.com/p/236335897b30

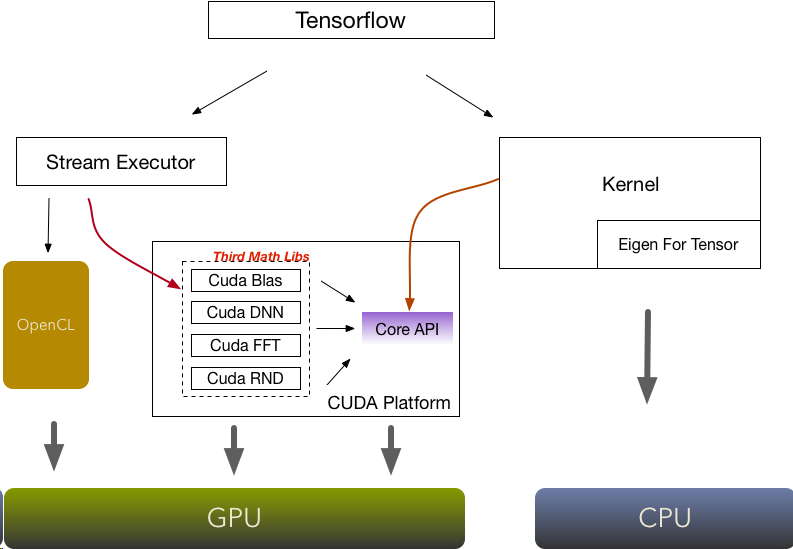
### 2.4.网络构建分割及优化

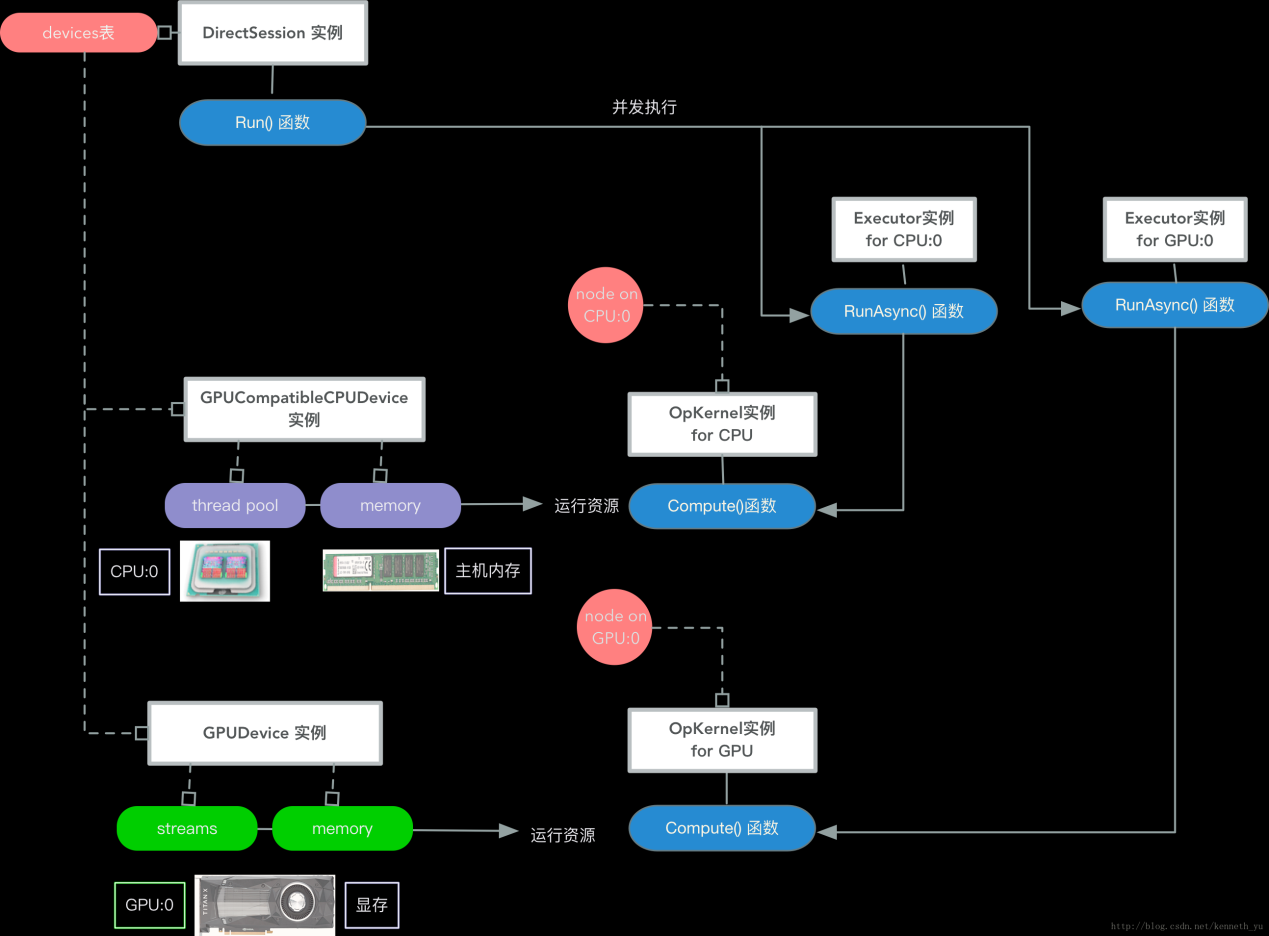




TensorFlow 核心流程剖析 -- 2 神经网络模型的构建、分割和优化：https://blog.csdn.net/kenneth\_yu/article/details/76824513

### 2.5.Device管理

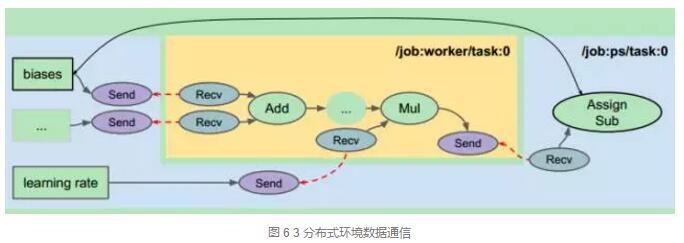


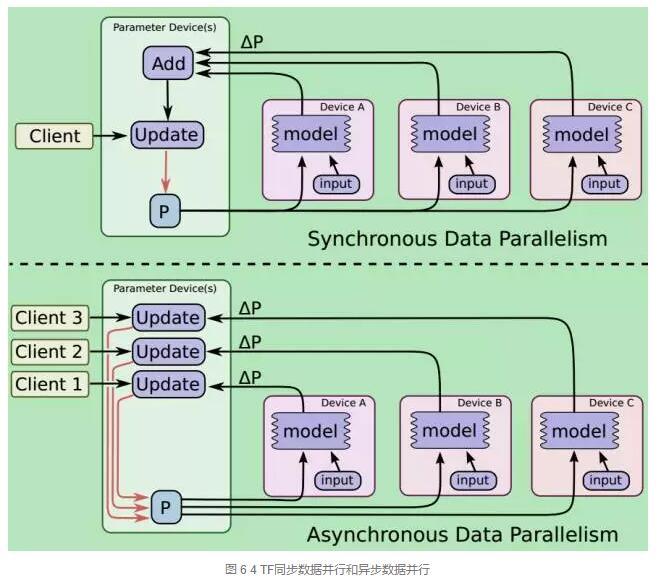


Tensorflow 核心流程剖析 3-- 运行设备Device的生成和管理：https://blog.csdn.net/kenneth\_yu/article/details/77098192

### 2.6.分布式架构

分布式架构涉及到任务调度、数据传输、节点进程监控、容错机制等。TF分布式环境采用master-slave架构，由master service完成主控操作。





https://zhuanlan.zhihu.com/p/26031658?utm\_source=itdadao&utm\_medium=referral

http://www.tensorflow123.cn/baihuatfdistribute.html

### 2.7.Profiler

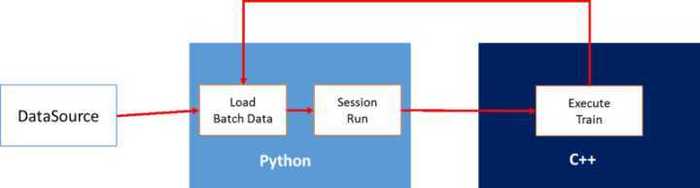
tensorflow 核心流程剖析 4-- 使用profiler检测神经网络模型的运行性能：https://blog.csdn.net/kenneth\_yu/article/details/77466776

### 2.8.XLA

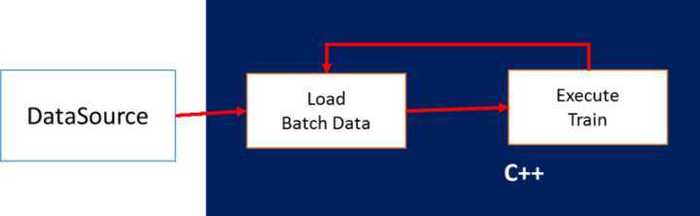
http://cwiki.apachecn.org/pages/viewpage.action?pageId=10029447

### 2.9.纯C++高性能训练

下图为Python API的运算架构：在模型训练的每次迭代中，程序通过Python API读取Batch Data，然后通过TensorFlow Session Run接口，传递数据给C++，并触发神经网络训练。如下图所示：



下图为C++ API的运算架构：在模型训练的每次迭代中，通过C++ API读取Batch Data后，直接触发模型训练。减少了不同语言间API接口的循环调用及数据传输。如下图所示：



https://www.jianshu.com/p/e811cc365959?utm\_campaign=maleskine&utm\_content=note&utm\_medium=seo\_notes&utm\_source=recommendation

1、常用术语的解释及对应的代码实现细节

例如，graph表示里面的node和边，什么是普通边，什么是逻辑边，在代码中怎样表示

collective，scoped...

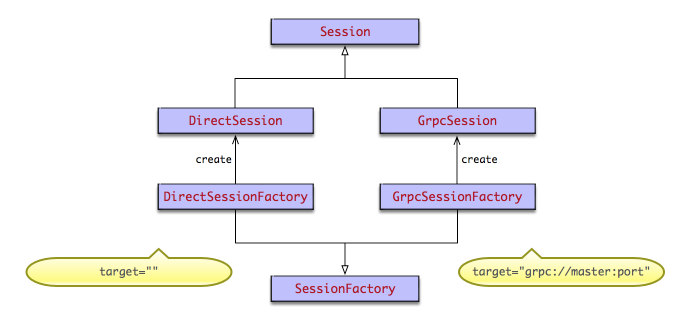
2、tensorflow整体框架原理

session管理, Threadpool ， graph拆分

device管理，memory管理，graph表示

分布式训练，Rendezvous

3、复杂类之间的关系：继承,组合,友元,静态...



4、关键算法的实现细节

例如，显存管理BFC，graph优化及划分costmodel placer，分布式通信ring\_all\_reduce，性能评估benchmark等

5、关键类的主要功能接口

Session：Newsession，Reset

Create，Extend，Run，Close

PRunSetup，PRun， ListDevices，LocalDeviceManager, CallableHandle

6、编程通识

bazel test使用，C++11新增语法 lamda表达式，多线程，线程锁，swig...

mutex

10、Rendezvous

11、memory

12、threadpool

13、Stream Executor & Eigen

https://blog.csdn.net/raintungli/article/details/78860611

14、gpucc

https://github.com/henline/streamexecutordoc

15、llvm&clang

16、timeline

https://www.zhihu.com/question/268375146/answer/340097912

