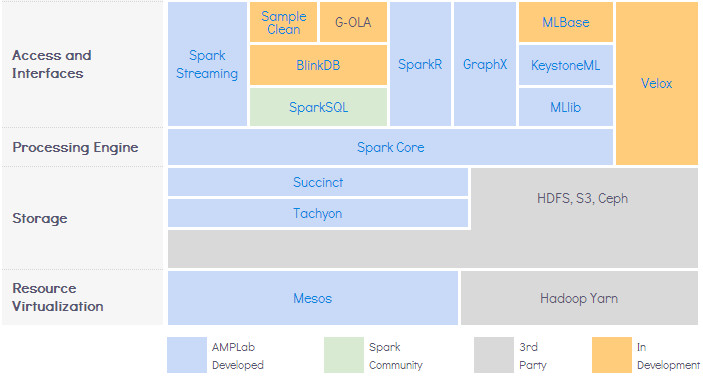
Apache Spark简介

Apache Spark是专为大规模数据处理而设计的快速通用的计算引擎，类似于MR的通用并行框架，拥有Hadoop MapReduce所具备的优点，但是不同于MapReduce的是Job中间输出结果可以保存到内存中，从而不再需要读写HDFS，因此Spark能够更好的适用于数据挖掘与机器学习等需要的迭代的MapReduce算法。尽管Spark是为了支持分布式数据集上的迭代作业，但是实际上它是对Hadoop的补充，可以在Hadoop文件系统中并行运行。

# Spark生态系统

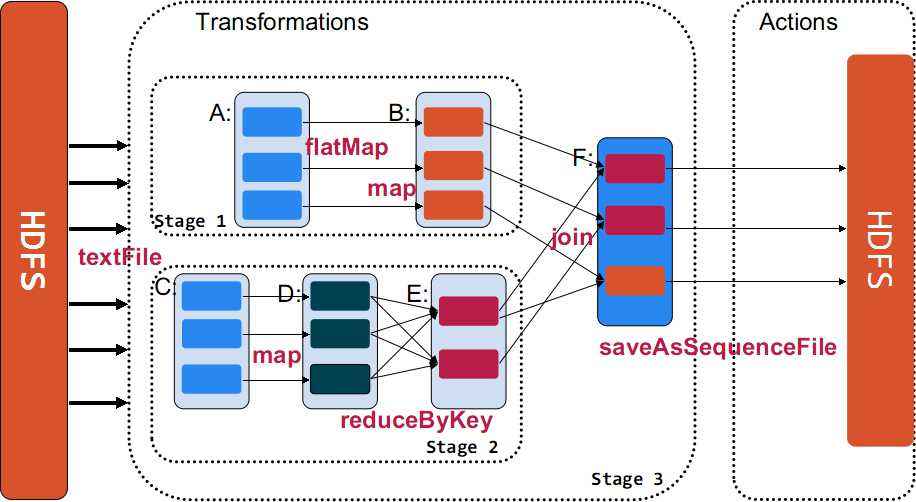
Spark在2009年诞生于加州伯克利AMPLab实验室，经过多年的发展，已经拥有一套生态系统，如下图所示：



其核心框架是Spark，同时Spark生态系统还涵盖了结构化SQL查询（Shark，已经不再维护）Spark SQL，提供机器学习功能的系统Mlbase及底层的分布式机器学习库Mlib，并行图计算框架GraphX，流计算框架Spark Streaming、采样近似计算查询引擎BlinkDB，内存分布式文件系统Alluxio，资源管理框架YARN及Mesos等，这些子项目在Spark上层提供了更高层、更丰富的计算范式。

## Spark Core

Spark 是一个大数据分布式编程框架，不仅实现了MapReduce的map和reduce函数，还提供了如filter,join,groupbykey等丰富的算子。Spark将分布式数据抽象为弹性分布式数据集RDD，实现了应用任务调度、RPC、序列化和压缩，并为运行在其上的组件提供API。其底层采用Scala这种函数式语言书写而成，提供的API深度借鉴Scala函数式的编程思想，提供与Scala类似的编程接口，处理流程如下图(数据源以HDFS为例)所示：



Spark将数据在分布式环境下分区，然后将作业转化为有向无环图，并分阶段进行DAG调度和任务的分布式并行处理。

## Spark SQL

Spark SQL的前身是Shark，主要用于结构化数据处理和对Spark数据执行类SQL查询，通过Spark SQL可以针对不同格式的数据执行ETL操作，如JSON,Parqent，数据库等进行特定的查询操作。Spark SQL使用Catalyst做查询解析和优化器，并在底层使用Spark作为执行引擎实现SQL的Operator，用户可以在Spark上直接写SQL，相当于Spark扩充了一套SQL算子，如下图所示：



Spark Core是基于RDD编程，但是Spark SQL是基于Data Frame编程。DataFrame的底层封装了RDD，是一个分布式的数据容器，其更像传统数据库的二维表格，除了数据以外，还掌握数据的结构信息（Schema,比如对列的描述）。

## Spark Streaming

Spark Streaming通过将流数据按指定的时间片累积成RDD，然后将每个RDD进行批处理，进而实现大规模的流处理，Storm处理数据方式是以条为单位，而Spark Streaming基于单位时间处理数据的。

Spark Streaming支持从多种数据源获取数据，包括Kafka、Flume、Twitter、ZeroMQ及TCP sockets等，从数据源获取数据之后，可以使用诸如map、Reduce、Join等高级函数进行复杂算法处理，最后把处理结果存储到文件系统，数据流图如下所示：



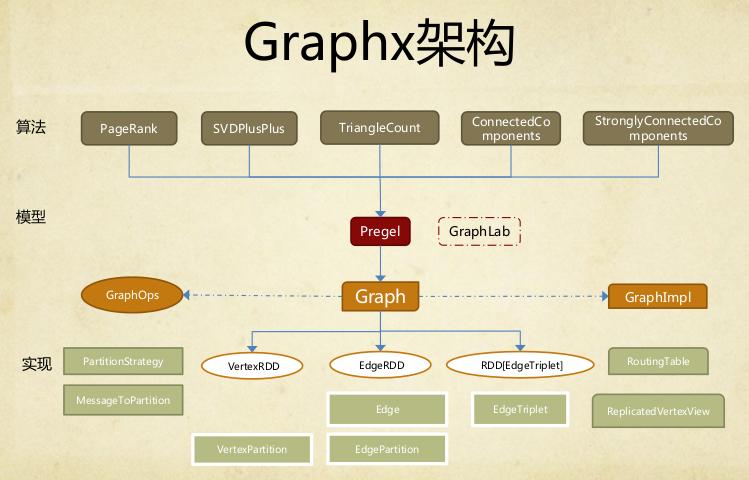
Spark Streaming在内部的处理机制是，接收实时流的数据，并根据一定的时间间隔拆分成一批批的数据，然后通过Spark Engine处理这些批数据，最终得到处理后的一批批结果数据，如下图所示：



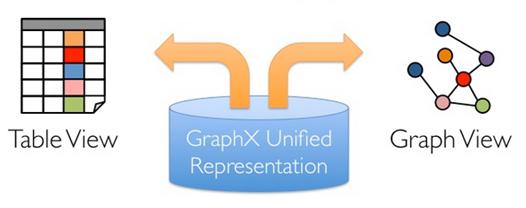
与Spark批数据处理内核对应的RDD实例，对应流数据Dstream可以看做是一组RDDs，及RDD序列，在流数据分成批后，通过一个FIFO队列，然后通过Spark Engine从该队列中依次取出批数据，再把批数据封装成RDD，然后进行处理。

## GraphX

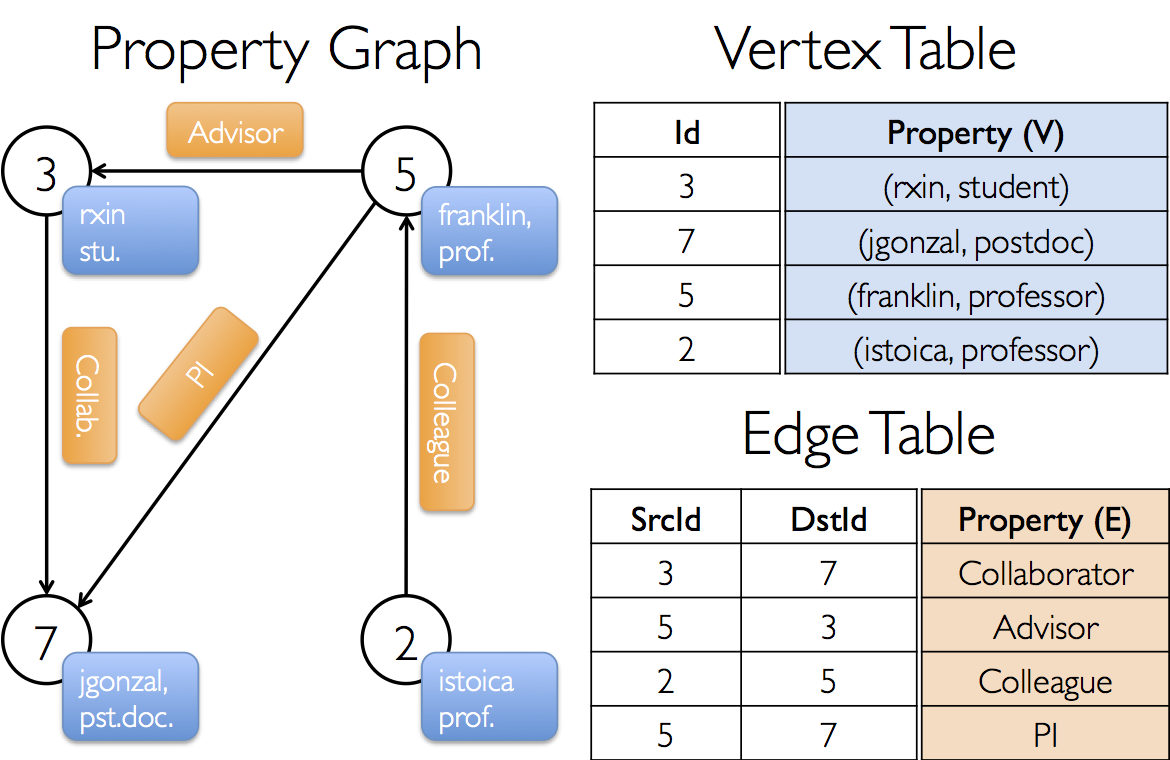
Spark GraphX是一个分布式式图处理框架，是基于Spark平台提供对图计算和图挖掘简洁易用的而丰富的接口，图的分布式或者并行处理其实是把图拆分成很多的子图，然后分别对这些子图进行计算，架构如下图所示：



Graphx的核心抽象是Resilient Distributed Property Graph，一种点和边都带属性的有向多重图，其扩展了Spark RDD抽象，有Table和Graph两种视图，而只需要一份物理存储，两种视图都有自己独有的操作符，从而获得灵活操作和执行效率。



对Graph视图的所有操作，最终都会转换成关联的Table视图的RDD操作来完成。这样对一个图计算，最终在逻辑上，等价于一系列RDD转换过程。对于GraphX中两种视图底层，共用一份物理数据，由RDD[Vertex-Partition]和RDD[EdgePartition]这两个RDD组成。点和边都不是以表Collection[Tuple]的形式存储的，而是由VertexPartition/EdgePartition在内部存储一个带索引结构的分片数据块，以加速不同视图下的遍历速度，不变的索引结构在RDD转换过程中是共用的，降低存储和计算开销，如下图所示：



图分布式存储采用点分割模式，而且使用partitionBy方法，由用户指定不同的划分策略，将边分到EdgePartition，顶点Master分配到各个VertexPartition，EdgePartition也会缓存本地边关联点的Ghost副本。划分策略的不同会影响到所需要缓存的Ghost副本数量，以及每个EdgePartition分配的边的均衡程度，需要根据图的结构特征选取最佳策略。

## Mlbase/MLib

Mlib是Spark的可扩展的机器学习库，由一些通用的学习算法和工具组成，包括分类、回归、聚类、协同过滤、降维等，同时还包括底层的优化原语和高层的管道API。目前Mlib包括spark.mlib，基于RDD的原始算法API和spark.ml基于DataFrames高层次的API，可以用来构建机器学习管道。

Mlbase通过边界定义，力图将Mlbase打造成一个机器学习平台，让不了解机器学习的用户也能方便的使用Mlbase这个工具来处理自己的数据：

1）ML Optimizer优化器选择最合适的，已经实现好的机器学习算法和相关参数

2）MLI是一个进行特征抽取和高级ML编程抽象的算法实现的API或平台

3）MLlib基于Spark的底层分布式机器学习库，可以不断扩充算法

4）MLRuntime，基于Spark计算框架，将Spark分布式计算应用到机器学习领域。

## Alluxio

Alluxio是以内存为中心的分布式存储系统，统一了数据访问的方式，为上层计算框架和底层存储系统构建了桥梁，应用只需要连接Alluxio即可访问存储在底层的任意存储系统中的数据。Alluxio可以为Spark提供可靠的数据共享层，使Spark专注于应用逻辑而Alluxio用于处理存储。Spark使用Alluxio的好处如下：

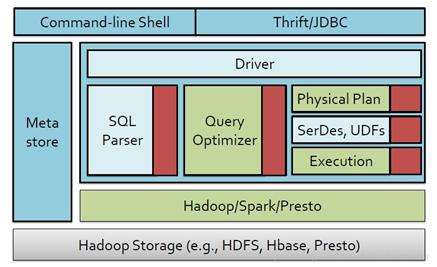
1. 数据可以长期存储在内存中，多个应用可以共享缓存数据
2. 数据存储在JVM外部，可以减少程序的GC时间
3. 缓存的数据不会因为程序的以外崩溃而消失
4. Alluxio与Hadoop兼容，需要修改Spark代码，即可直接运行

## BlinkDB

BlinkDB是一个用于在海量数据上进行交互式SQL的近似查询引擎，允许用户在查询准确性和响应时间之间做出权衡来完成近似查询。其数据的精度被控制在允许的误差范围内，为了达到这个目标，两个核心思想：

1. 通过一个自适应优化框架，随着时间的推移，从原始数据建立并维护一组多维样本
2. 通过一个动态样本选择策略，选择一个适当大小的示例，然后基于查询的准确性和响应时间满足用户查询需求。

BlinkDB的架构如下所示：



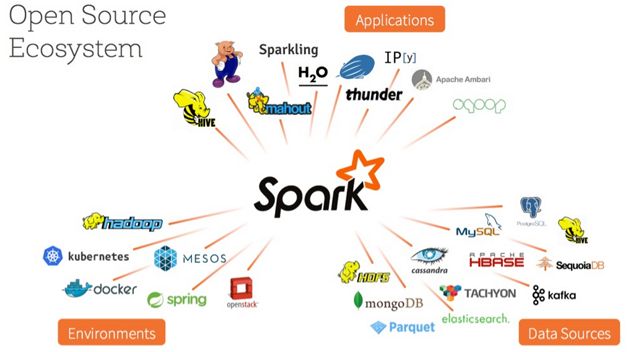
## Spark R

SparkR是AMPLab开发的一个R开发包，提供轻量级的方式使得可以在R语言中使用Apache Spark，特性如下：

1. 提供Spark中弹性分布式数据集RDD的API，用户可以在集群中通过R shell交互性的运行Spark Job
2. 支持序列化闭包功能，可以将用户定义函数所引用的变量自动化发送到集群中的其他机器上
3. SparkR很容易调用R开发包，只需要在集群上执行操作前includePackge读取R开发包就可以

# Spark Use-Cases

从下图中，可以看到无论是在容器领域火爆的Kubernet及Docker，还是在NoSQL领域的HBase及Cassandra，亦或其他一些如消息队列Kafka，分布式搜索引擎Elasticsearch及各集群学习框架都与Spark产生了联系，并且这种趋势在发展中。这意味着，Spark可能出现在大数据处理的各个领域：



Spark提供了与Hadoop相似的开源集群计算环境，但基于内存和迭代优化的设计，因此Spark在某些工作负载表现更优秀，Spark具有以下几点优势：

1）轻量级快速处理，Spark允许Hadoop集群中的应用程序在内存中以100倍的速度允许，即使在磁盘上也能快10倍，Spark通过减少磁盘IO来达到性能提升，将中间数据全部放到内存中。Spark使用了RDD的理念，可以透明的在内存中存储数据，只在需要时才持久化到磁盘。

2）易于使用，Spark支持多语言，Java、Scala及Python，这些允许开发者在自己熟悉的语言环境下进行工作，其自带了80多个高等级操作符，允许在Shell中进行交互式查询。

3）支持复杂查询，在简单的map和reduce操作之外，Spark还支持SQL查询、流式查询即复杂查询，比如开箱即用的机器学习图算法，同时用户可以在同一个工作流中无缝的搭配这些能力。

4）实时流处理，对比MR只能处理离线数据，Spark支持实时的流计算，Spark依赖Spark Streaming对数据进行实时的处理。

5）可以与Hadoop和已有Hadoop数据整合，Spark可以独立运行，除了可以运行在当下的YARN集群之外，它还可以读取已有的任何Hadoop数据。

6）活跃和无限壮大的社区

目前大数据在互联网公司主要应用在广告、报表、推荐系统等业务上。在广告业务方面需要大数据做应用分析、效果分析、定向优化等，在推荐系统方面则需要大数据优化相关排名、个性化推荐以及热点点击分析等。这些应用场景的普遍特点是计算量大、效率要求高。Spark恰恰满足了这些要求，该项目一经推出便受到开源社区的广泛关注和好评。并在近两年内发展成为大数据处理领域最炙手可热的开源项目。下面将列举国内外应用Spark的成功案例：

1）腾讯

广点通是最早使用Spark的应用之一。腾讯大数据精准推荐借助Spark快速迭代的优势，围绕“数据+算法+系统”这套技术方案，实现了在“数据实时采集、算法实时训练、系统实时预测”的全流程实时并行高维算法，最终成功应用于广点通pCTR投放系统上，支持每天上百亿的请求量。

基于日志数据的快速查询系统业务构建于Spark之上的Shark，利用其快速查询以及内存表等优势，承担了日志数据的即席查询工作。在性能方面，普遍比Hive高2-10倍，如果使用内存表的功能，性能将会比Hive快百倍。

2）Yahoo

Yahoo将Spark用在Audience Expansion中的应用。Audience Expansion是广告中寻找目标用户的一种方法：首先广告者提供一些观看了广告并且购买产品的样本客户，据此进行学习，寻找更多可能转化的用户，对他们定向广告。Yahoo采用的算法是logistic regression。同时由于有些SQL负载需要更高的服务质量，又加入了专门跑Shark的大内存集群，用于取代商业BI/OLAP工具，承担报表/仪表盘和交互式/即席查询，同时与桌面BI工具对接。目前在Yahoo部署的Spark集群有112台节点，9.2TB内存。

3）淘宝

阿里搜索和广告业务，最初使用Mahout或者自己写的MR来解决复杂的机器学习，导致效率低而且代码不易维护。淘宝技术团队使用了Spark来解决多次迭代的机器学习算法、高计算复杂度的算法等。将Spark运用于淘宝的推荐相关算法上,同时还利用Graphx解决了许多生产问题，包括以下计算场景：基于度分布的中枢节点发现、基于最大连通图的社区发现、基于三角形计数的关系衡量、基于随机游走的用户属性传播等。

4）优酷土豆

优酷土豆在使用Hadoop集群的突出问题主要包括：第一是商业智能BI方面，分析师提交任务之后需要等待很久才得到结果；第二就是大数据量计算，比如进行一些模拟广告投放之时，计算量非常大的同时对效率要求也比较高，最后就是机器学习和图计算的迭代运算也是需要耗费大量资源且速度很慢。

最终发现这些应用场景并不适合在MapReduce里面去处理。通过对比，发现Spark性能比MapReduce提升很多。首先，交互查询响应快，性能比Hadoop提高若干倍；模拟广告投放计算效率高、延迟小（同hadoop比延迟至少降低一个数量级）；机器学习、图计算等迭代计算，大大减少了网络传输、数据落地等，极大的提高的计算性能。目前Spark已经广泛使用在优酷土豆的视频推荐（图计算）、广告业务等。

# Spark新特性

## Spark-packages.org

Spark-package.org是使用spark的开源库的社区package索引。Spark package可以使用不同的数据源、管理工具、higher-level的domain-specific lib、集群学习算法，code samples等。例如Spark 2.0中增加的Spark-CSV，Spark ML融合包包括GraphFrames和TensorFrames。

## Tungsten

Tungsten是对Spark优化内存和CPU使用的Project，对Spark SQL优化最多，主要的优化点在于：

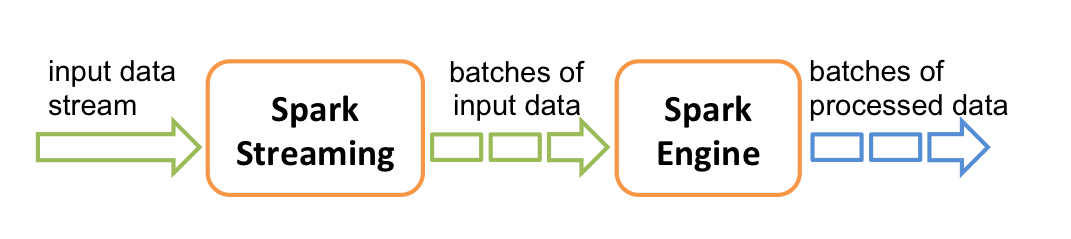
1. Memory Management和Binary Processing，利用应用的语义来更明确地管理内存，同时消除JVM对象模型和垃圾回收开销
2. Cache-aware computation，使用算法和数据结构来实现内存分级结构
3. 代码生成，Code generation，使用代码生成来利用新型编译器和CPU

之所以大幅度聚焦内存和CPU的利用，其最大原因就在于对比IO和网络通信，Spark在CPU和内存上遭遇到的瓶颈最多。

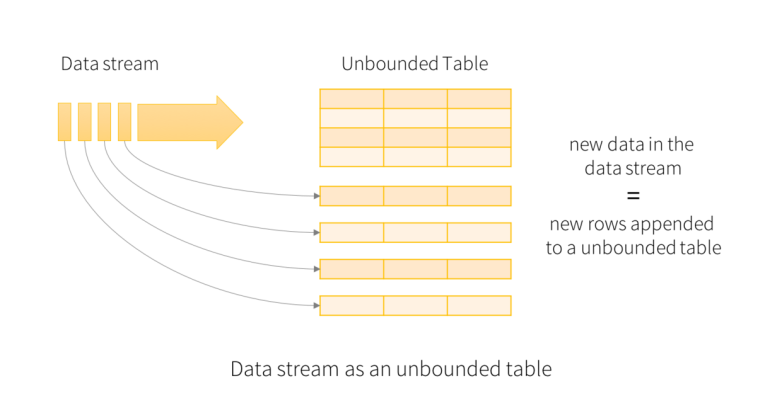
## Structured Streming

Spark 2.0将流式计算统一到DataFrame里，提出Structured Streaming概念，将数据源映射为一张无线长度的表，同时将流式计算的结果映射为另外一张表，完全以结构化的方式去操作流式数据，复用其对象Catalyst引擎。

作为Spark平台的流式实现，Spark Streaming是有单独一套抽象和API，如下所示：



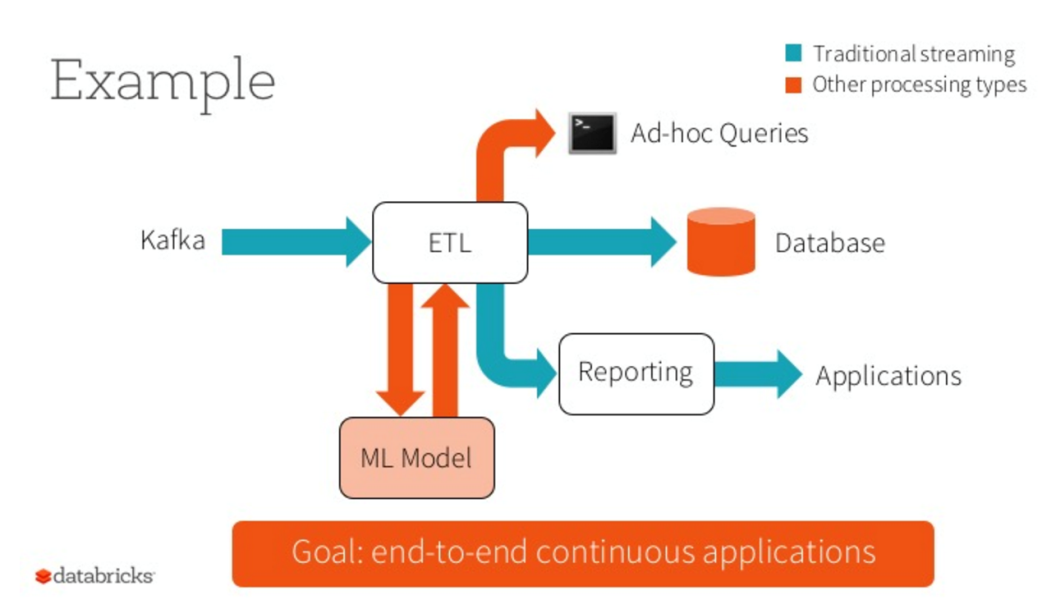
在Spark 2.0后，Structured Streaming构建在Spark SQL引擎上的流式数据处理引擎，在概念上流式抽象成一张无限大的表，如下所示：



当流数据连续不断的产生时，Spark SQL将会增量，持续不断的处理这些数据并将结果更新到结果集中。

## Continuous Applications

Continuous Application可以在real-time data执行类似于batch计算的查询，例如执行ETL，生成报告，更新及track 数据流中的特定数据。可以使用Structured Stream实现batch和real-time的查询结合，如下图所示：



# Spark数据抽象详述

在Spark 2.0中，提供了RDD、DataFrame和DataSet三种API，有各自适合的使用场景，而且Spark程序可以在三种数据集之间无缝转换。

## RDD(弹性数据集)

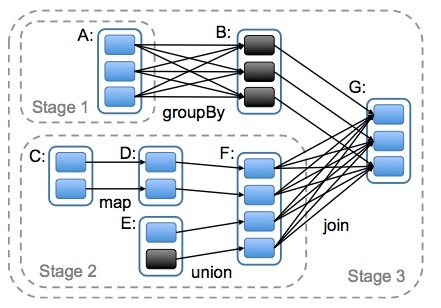
Apache Spark的核心概念是RDD，弹性数据集，指的是一个只读的，可分区的分布式数据集，这个数据集的全部或者部分可以缓存在内存中，在多次计算间重用。RDD是一种有容错机制的特殊集合，可以分布在集群的节点上，以函数式操作集合的方式，进行各种并行操作。使用RDD的场景：

1. 使用Low-level的transformation和action来控制数据集
2. 数据集非结构化，如流媒体或者文本流
3. 使用函数式来操作数据，而不是特定领域语言DSL表达
4. 不在乎Schema，比如通过名字或者列处理数据属性不在意列式存储格式
5. 放弃使用DataFrame和DataSet来优化结构化和半结构化数据集

RDD提供的操作包括map,flatmap,filter等转换操作实现monad模式，较好的契合Scala集合操作。除此之外，还提供了join,groupBy,reduceByKey等更为方便的操作，以支持常见的数据运算。

### 4.1.1 RDD之间依赖

RDD作为数据结构，本质上只是一个只读的分区记录集合，一个RDD可以包含多个分区，每个分区都是一个dataset片段，而且RDD之间相互依赖，比如RDD的每个分区最多只能被一个Child RDD的一个分区使用，称为narrow dependency；若多个Child RDD分区都可以依赖，称为wide dependency。不同的操作依赖其特性，可能会产生不同的依赖，例如map操作产生的是narrow dependency，而join操作则产生wide dependency。Narrow与Wide dependency之间的区别如下图所示：



narrow dependencies可以支持在同一个cluster node上以管道形式执行多条命令，例如在执行了map后，紧接着执行filter。相反，wide dependencies需要所有的父分区都是可用的，可能还需要调用类似MapReduce之类的操作进行跨节点传递。从失败恢复的角度考虑，narrow dependencies的失败恢复更有效，因为它只需要重新计算丢失的parent partition即可，而且可以并行地在不同节点进行重计算。而wide dependencies牵涉到RDD各级的多个Parent Partitions。

### 4.1.2 RDD数据处理效率

RDD提供了两方面的特性：persistence和partitioning，分别对应persist和partitionBy函数来控制这两方面。RDD的分区特性和并行计算能力，使得Spark可以更好的利用可伸缩的硬件资源。

1. persist，将数据持久化，RDD的cache方法其实调用的就是persist方法，缓存策略均为MEMORY\_ONLY，可以根据需要设置不同的缓存级别
2. partitioning，待处理的原始数据按照相应的逻辑切分成n份，每份数据对应RDD中的一个Partition，Partition的数量决定task的数量，影响着程序的并行度。Partition和RDD是伴生的，每个RDD都有对应的Partition实现，其定义数据源如何被切分的逻辑。
3. Partitioner，分片函数，HashPartitioner和RangePartitioner

### 4.1.3 RDD操作

RDD的操作分为转换操作（transformation）和行动操作（action transformation），之所以将操作分成两类和RDD惰性计算有关，当RDD执行转化操作时，实际操作并没有被执行，只有当RDD执行action操作时才会触发任务的提交，执行相应的计算操作。转换操作和行动操作的区别在于转换操作从RDD产生一个新的RDD，而行动操作进行实际的计算，具体操作如下所示：

|  |  |  |
| --- | --- | --- |
| 操作类型 | 函数名 | 作用 |
| 转化操作 | map() | 参数是函数，函数应用于RDD每一个元素，返回值是新的RDD |
| flatMap() | 参数是函数，函数应用于RDD每一个元素，将元素数据进行拆分，变成迭代器，返回值是新的RDD |
| filter() | 参数是函数，函数会过滤掉不符合条件的元素，返回值是新的RDD |
| distinct() | 没有参数，将RDD里的元素进行去重操作 |
| union() | 参数是RDD，生成包含两个RDD所有元素的新RDD |
| intersection() | 参数是RDD，求出两个RDD的共同元素 |
| subtract() | 参数是RDD，将原RDD里和参数RDD里相同的元素去掉 |
| cartesian() | 参数是RDD，求两个RDD的笛卡儿积 |
| 行动操作 | collect() | 返回RDD所有元素 |
| count() | RDD里元素个数 |
| countByValue() | 各元素在RDD中出现次数 |
| reduce() | 并行整合所有RDD数据，例如求和操作 |
| fold(0)(func) | 和reduce功能一样，不过fold带有初始值 |
| aggregate(0)(seqOp,combop) | 和reduce功能一样，但是返回的RDD数据类型和原RDD不一样 |
| foreach(func) | 对RDD每个元素都是使用特定函数 |

### 4.1.4 RDD容错

RDD是基于Lineage的容错机制，RDD在转换时根据转换关系构成compute chain，这个chain可以认为是RDD之间演化的Lineage。在部分计算结果丢失时，只需要根据这个Lineage重算即可。在Spark的内部实现上，DAG被划分为不同的Stage，Stage之间的依赖关系可以认为是Lineage。

如果DAG中的Lineage过长，例如PageRank计算，如果重算，则开销过大。可以在宽依赖上做Checkpoint获得的收益更大。Checkpoint，本质上是通过将RDD写入Disk做检查点，其是为了通过lineage做容错的辅助，在RDD中checkpoint方法相当于通过冗余数据来缓存数据，而Lineage是通过相当粗粒度的记录更新操作来实现容错的。

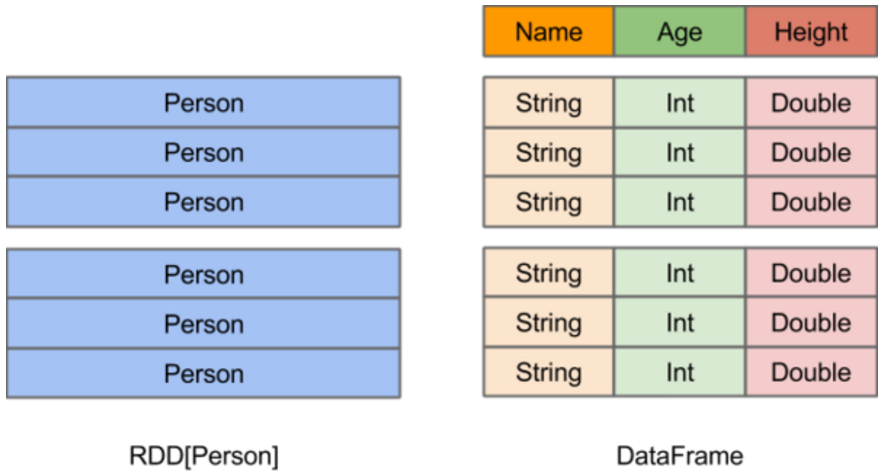
## DateFrame

DataFrame是以命名列方式组织的分布式数据集，等同于关系型数据库的表，也相当于R/Python中的data frames(但进行了更多的优化)。DataFrames可以广泛的支持任何第三方数据格式和数据源：



对于DataFrame，有丰富的操作函数，类似于RDD中的算子，DataFrames的支持能力允许应用程序可以轻松地组合不同数据源的数据。

DataFrame和RDD的主要区别在于，前者带有Schema元信息，即DataFrame所表示的二维数据表数据集的每一列都都有名称和类型：



这个特性使得Spark SQL可以洞察更多的结构信息，从而对藏于DataFrame背后的数据源以及作用于DataFrame之上的变换进行了针对性优化，反观RDD，由于无从得知所存储数据元素的内部结构，Spark Core只能在stage层面进行简单、通用的流水线优化。

## DataSet

DataSet是一个强类型的域特定对象的集合，可以使用功能或关系操作并行转换，每个数据集还有一个无类型的视图称为DataFrame，这是一个行的数据集。

DataSet结合了RDD和DataFrame的优点，DataSet API扩展了DataFrame API，支持静态类型和运行已经存在的Scala或Java语言的用户自定义函数，比RDD提供更好的内存管理，特别是在长任务中有更好的性能提升。DataSet效率的提升主要得益于Tungsten项目，其做的优化由Spark管理内存而不是使用JVM。

DataSet的核心是一个新的概念Encoder，编码器的作用是将JVM的对象与表结构进行转换。表结构采用的Tungsten的二进制格式存储，允许操作序列化数据，提高内存利用率。当序列化数据时，Encoder产生字节码与off-heap进行交互，能够达到按需访问数据的效果，而不用反序列化整个对象。

DataFrame和DatasetAPI获得空间效率和性能优化的两个原因：

1. DataFrame和DataSet API建立在Spark SQL引擎之上，会使用Catalyst优化器来生成优化过的逻辑计划和物理查询计划。R, Java,Scala或者Python的DataFrame和DataSet API使得查询都进行相同的代码优化以及空间和速度的效率提升
2. 其次，Spark作为编译器可以理解为Dataset类型的JVM Object，能映射特定类型的JVM Object到Tungsten的内存管理，使用Encoder。Tungsten的Encoder可以有效的序列化和反序列化JVM Object，生成字节码来提高执行速度。

参考链接：

1. Spark 组件介绍

https://databricks.com/blog/2016/06/22/apache-spark-key-terms-explained.html

1. Spark SQL:

http://www.cnblogs.com/BYRans/p/5057110.html

http://www.uml.org.cn/sjjm/2014071711.asp?artid=6201

3）Spark GraphX

http://blog.csdn.net/yirenboy/article/details/47844677

4）http://book.51cto.com/art/201502/465999.htm