[storm介绍（一）](https://www.cnblogs.com/Jack47/p/storm_intro-1.html)

# 什么是流式计算

随着WEB2.0的出现，数据步入了井喷的时代，每天都有大量的数据产生，如何让数据说话，更及时的指导业务是每个企业都在重点关注的方向。

尽管传统数据仓库的T+1（以天为计算周期对数据进行批处理）机制一定程度上能满足业务的日常需求，但是这种处理方式毕竟具有一定的滞后性，针对统计分析时效性要求更高T+0（实时计算）的业务场景，诸如618、双11大促期间的实时大屏、实时预测、风控预警等诸多业务场景下，T+1的处理方式不免显得有些力不从心。然而，对于企业的决策者而言，他们深知在当下激烈竞争角逐的市场环境下，随着时间的流逝，数据的业务价值也迅速降低，最佳的处理方式是在数据产生后能尽快的进行计算和分析，从而能够时刻掌握公司的运营状况，对当前状况了如指掌，及时制定决策、发现问题并快速响应。

从而需要一种全新的实时计算方式出现，降低整个数据处理链路的延迟，实时化计算，来弥补离线计算的不足。

什么是流式计算？

简而言之，就是对持续生成的，无穷尽的数据集合，秒级甚至毫秒级尽可能低延迟的逐条地进行加工、计算。每个环节都需要近实时的处理，就需要各种技术手段、引擎工具配合处理，这些引擎统一到一起，就形成了流式计算平台。举个例子：客户通过PC、APP、微信手Q等渠道下单购物，这些订单无时无刻不在产生，为了及时统计订单量，我们就要思考用什么样的技术手段和工具来处理，这种计算方式就是流式计算。

流式计算有以下几个特点：

实时(realtime)且无穷(infinite)的数据

数据以连续数据流的形态高速产生，只要数据源在不断生产数据，数据流就会持续不断，并且不断动态变化。同时数据量也无法准确预估，可能是无穷大的。数据并不是一次全部可用，不能够一次得到全量数据，只能在不同的时间点，以增量的方式，逐步产生。

高效（efficient）的计算

数据生命周期的时效性往往很短，数据的时间价值也更重要。不论数据生产的数据量多大，数据流到来后，均需要实时高效处理，这样才能保证实时产生相应结果，进行反馈。

流式(streaming)且实时的数据集成

虽然部分数据可能以批量的形式被存储下来，但也只是为了满足后续其他场景下的应用需求。例如将计算后的数据直接写入JimDB进行报表展示，因此流数据的计算结果可以类似流式数据一样持续写入目的数据存储。

# 流式计算与离线计算的区别

在流式计算出现之前，传统数据仓库的计算方式完全依赖于离线计算。为了让大家对流式计算和离线计算有更清晰的认识，下面简单介绍京东流式计算与离线计算的区别。

离线计算：

在Hadoop出现之前，传统的数据仓库（OLAP）分析大多借助ETL工具将数据加载到基于Oracle等关系型数据库或者一体机来进行数据存储，然后构建数据仓库模型，再通过在线的数据服务Ad-Hoc、DashBoard等，构造SQL语言来进行即席查询。也就是将数据搬向计算节点，我们暂且称之为计算密集型-OLAP，这种方式在大数据产生之前的阶段得到广泛使用。

随着数据的井喷式增长，计算密集型的数据仓库耗时较高，用这种方式去处理海量的数据，无论是从数据的抽取，还是到数据的计算加工，数据量越来越大，对机器的性能也要求越来越高，越来越凸显出弊端。Hadoop技术的出现，打破了这一瓶颈，这种将计算分布到各个数据节点去分布式计算的方式，使得海量数据的加工、处理变成了可能，我们暂且称之为存储密集型-OLAP。

不论是计算密集型-OLAP还是存储密集型-OLAP，其核心处理方法都是基于一套离线计算的方法。下面我们看看这种离线计算方法的数据处理方式：

1、通过ETL工具，预先加载数据到数据仓库。

2、定义复杂的计算规则，预先计算数据到数据模型，整个过程可能耗时数小时，才能提供服务给前端应用。

3、一旦无法满足需要修改规则及模型，再次耗时重新计算，以满足查询。

4、由于以上环节耗时较长并且数据预先加载，处理的数据已经是历史数据，无法计算最新的数据。

由此，我们可以看出离线计算的特点是：批量计算、高延迟、规则复杂的计算方式。

流式计算：

如果说离线计算是批量计算方式的加工利器，那么流式计算的出现，则是对离线计算的完美补充。不同于离线计算，流式计算更注重于强调计算的低延迟，由于需要有java等编程语言开发功底，因此具有一定的开发门槛。

流式计算有以下几个特点：

1、预先设定计算规则，提交计算作业。

2、等待数据流流经作业进行计算，一旦流过，很难再重新计算之前数据。

3、持续不断输出计算结果。



由以上对比可以看出，在绝大多数大数据处理场景下，受限于流计算的整个计算模型较为简单，流计算是批量计算的有效补充，尤其在对于高数据处理时效性、计算复杂度相对较低的需求上，流式计算则更为合适 。

# 流式计算

BDP隆重推出流式计算平台JRC，平台覆盖了主流的Storm、Spark Streaming、Flink流式计算引擎，用于流式计算。

流式计算平台JRC是为实时计算设计的一款可视化配置管理系统，实现了资源管理，任务上传、发布、管理服务，支持程序包版本管理，实时查看任务监控、日志搜索和操作记录查看功能，实现了公司人力、技术、硬件等资源利用最大化。助力业务更加敏捷、准确的获取数据，迅速提取、挖掘数据价值，高效驱动业务增长。

下图为三种流式计算框架的对比，请大家针对业务场景，酌情使用。



## Storm

Storm比较侧重极低延迟的流处理方式，也是在业务上要求近实时处理的工作的最佳选择。该框架可处理海量数据，与其他方案相比具有更高吞吐量和更低延迟的特点。

Storm的流处理可对框架中名为Topology（拓扑）的DAG（有向无环图）进行编排。这些拓扑描述了当数据片段进入系统后，需要对每个传入的片段执行的不同转换或步骤。

拓扑包含：

1）Stream：普通的数据流，这是一种会持续抵达系统的无边界数据。

2）Spout：位于拓扑边缘的数据流来源，例如可以是API或查询等，从这里可以产生待处理的数据。

3）Bolt：Bolt代表需要消耗流数据，对其应用操作，并将结果以流的形式进行输出的处理步骤。Bolt需要与每个Spout建立连接，随后相互连接以组成所有必要的处理。在拓扑的尾部，可以使用最终的Bolt输出作为相互连接的其他系统的输入。

Storm背后的想法是使用上述组件定义大量小型的离散操作，随后将多个组件组成所需拓扑。默认情况下Storm提供了“至少一次”的处理保证，这意味着可以确保每条消息至少可以被处理一次，但某些情况下如果遇到失败可能会处理多次。Storm无法确保可以按照特定顺序处理消息。

为了实现严格的一次处理，即有状态处理，可以使用一种名为Trident的抽象。严格来说不使用Trident的Storm通常被视为Core Storm。Trident会对Storm的处理能力产生极大影响，是一种为处理提供状态牺牲效率，使用微批模式代替逐项处理的纯粹流处理模式。

为避免这些问题，通常建议Storm用户尽可能使用Core Storm。然而也要注意，Trident对内容严格的一次处理保证模式在某些情况下也比较有用。

Trident拓扑包含：

1）流批（Stream batch）：这是指流数据的微批，可通过分块提供批处理语义。

2）操作（Operation）：是指可以对数据执行的批处理过程。

优势和局限

目前来说Storm可能是近实时处理领域的最佳解决方案。该技术可以用极低延迟处理数据，可用于希望获得最低延迟的工作负载。如果处理速度直接影响用户体验，例如需要将处理结果直接提供给访客打开的网站页面，此时Storm将会是一个很好的选择。

Storm与Trident配合使得用户可以用微批代替纯粹的流处理。虽然借此用户可以获得更大灵活性打造更符合要求的工具，但同时这种做法会削弱该技术相比其他解决方案最大的优势。话虽如此，但多一种流处理方式总是好的。

Core Storm无法保证消息的处理顺序。Core Storm为消息提供了“至少一次”的处理保证，这意味着可以保证每条消息都能被处理，但也可能发生重复。Trident提供了严格的一次处理保证，可以在不同批之间提供顺序处理，但无法在一个批内部实现顺序处理。

在互操作性方面，Storm可与Hadoop的YARN资源管理器进行集成，因此可以很方便地融入现有Hadoop部署。除了支持大部分处理框架，Storm还可支持多种语言，为用户的拓扑定义提供了更多选择。

总结

对于延迟需求很高的纯粹的流处理工作负载，Storm可能是最适合的技术。该技术可以保证每条消息都被处理，可配合多种编程语言使用。由于Storm无法进行批处理，如果需要这些能力可能还需要使用其他软件。如果对严格的一次处理保证有比较高的要求，此时可考虑使用Trident。不过这种情况下其他流处理框架也许更适合。

## Spark

Apache Spark是一种包含流处理能力的批处理框架。

Spark 运行在现有的 Hadoop 分布式文件系统基础之上（HDFS）提供额外的增强功能。它支持将 Spark 应用部署到现存的 Hadoop v1 集群（with SIMR – Spark-Inside-MapReduce）或 Hadoop v2 YARN 集群甚至是Apache Mesos之中。其意图并非是替代 Hadoop，而是为了提供一个管理不同的大数据用例和需求的全面且统一的解决方案。

批处理模式

与MapReduce不同，Spark的数据处理工作全部在内存中进行，只在一开始将数据读入内存，以及将最终结果持久存储时需要与存储层交互。所有中间态的处理结果均存储在内存中。

虽然内存中处理方式可大幅改善性能，Spark在处理与磁盘有关的任务时速度也有很大提升，因为通过提前对整个任务集进行分析可以实现更完善的整体式优化。

流处理模式

流处理能力是由Spark Streaming实现的。Spark本身在设计上主要面向批处理工作负载，为了弥补引擎设计和流处理工作负载特征方面的差异，Spark实现了一种叫做微批（Micro-batch）的概念。在具体策略方面该技术可以将数据流视作一系列非常小的“批”，借此即可通过批处理引擎的原生语义进行处理。

Spark Streaming会以亚秒级增量对流进行缓冲，随后这些缓冲会作为小规模的固定数据集进行批处理。这种方式的实际效果非常好，但相比真正的流处理框架在性能方面依然存在不足。

优势和局限

使用Spark而非Hadoop MapReduce的主要原因是速度。在内存计算策略和先进的DAG调度等机制的帮助下，Spark可以用更快速度处理相同的数据集。

Spark的另一个重要优势在于多样性。该产品可作为独立集群部署，或与现有Hadoop集群集成。该产品可运行批处理和流处理，运行一个集群即可处理不同类型的任务。

除了引擎自身的能力外，围绕Spark还建立了包含各种库的生态系统，可为机器学习、交互式查询等任务提供更好的支持。相比MapReduce，Spark任务更是“众所周知”地易于编写，因此可大幅提高生产力。

为流处理系统采用批处理的方法，需要对进入系统的数据进行缓冲。缓冲机制使得该技术可以处理非常大量的传入数据，提高整体吞吐率，但等待缓冲区清空也会导致延迟增高。这意味着Spark Streaming可能不适合处理对延迟有较高要求的工作负载。

由于内存通常比磁盘空间更贵，因此相比基于磁盘的系统，Spark成本更高。然而处理速度的提升意味着可以更快速完成任务，在需要按照小时数为资源付费的环境中，这一特性通常可以抵消增加的成本。

Spark内存计算这一设计的另一个后果是，如果部署在共享的集群中可能会遇到资源不足的问题。相比Hadoop MapReduce，Spark的资源消耗更大，可能会对需要在同一时间使用集群的其他任务产生影响。从本质来看，Spark更不适合与Hadoop堆栈的其他组件共存一处。

总结

Spark是多样化工作负载处理任务的最佳选择。Spark批处理能力以更高内存占用为代价提供了无与伦比的速度优势。对于重视吞吐率而非延迟的工作负载，则比较适合使用Spark Streaming作为流处理解决方案。

## Flink

Flink是一种可以处理批处理任务的流处理框架。该技术可将批处理数据视作具备有限边界的数据流，借此将批处理任务作为流处理的子集加以处理。为所有处理任务采取流处理为先的方法会产生一系列有趣的副作用。

Flink的流处理模型在处理传入数据时会将每一项视作真正的数据流。Flink提供的DataStream API可用于处理无尽的数据流。Flink可配合使用的基本组件包括：

1）Stream（流）是指在系统中流转的，永恒不变的无边界数据集

2）Operator（操作方）是指针对数据流执行操作以产生其他数据流的功能

3）Source（源）是指数据流进入系统的入口点

4）Sink（槽）是指数据流离开Flink系统后进入到的位置，槽可以是数据库或到其他系统的连接器

为了在计算过程中遇到问题后能够恢复，流处理任务会在预定时间点创建快照。为了实现状态存储，Flink可配合多种状态后端系统使用，具体取决于所需实现的复杂度和持久性级别。

此外Flink的流处理能力还可以理解“事件时间”这一概念，即事件实际发生的时间，此外还对处理会话有良好的支持。这意味着可以通过某种有趣的方式确保执行顺序和分组。

批处理模型

Flink的批处理模型在很大程度上仅仅是对流处理模型的扩展。此时模型不再从持续流中读取数据，而是从持久存储中以流的形式读取有边界的数据集。Flink会对这些处理模型使用完全相同的运行时。

Flink可以对批处理工作负载实现一定的优化。例如由于批处理操作可通过持久存储加以支持，Flink可以不对批处理工作负载创建快照。数据依然可以恢复，但常规处理操作可以执行得更快。

另一个优化是对批处理任务进行分解，这样即可在需要的时候调用不同阶段和组件。借此Flink可以与集群的其他用户更好地共存。对任务提前进行分析使得Flink可以查看需要执行的所有操作、数据集的大小，以及下游需要执行的操作步骤，借此实现进一步的优化。

优势和局限

Flink目前是处理框架领域一个独特的技术。虽然Spark也可以执行批处理和流处理，但Spark的流处理采取的微批架构使其无法适用于很多用例。Flink流处理为先的方法可提供低延迟，高吞吐率，近乎逐项处理的能力。

Flink的很多组件是自行管理的。虽然这种做法较为罕见，但出于性能方面的原因，该技术可自行管理内存，无需依赖原生的Java垃圾回收机制。与Spark不同，待处理数据的特征发生变化后Flink无需手工优化和调整，并且该技术也可以自行处理数据分区和自动缓存等操作。

Flink会通过多种方式对工作进行分解进而优化任务。这种分析在部分程度上类似于SQL查询规划器对关系型数据库所做的优化，可针对特定任务确定最高效的实现方法。该技术还支持多阶段并行执行，同时可将受阻任务的数据集合在一起。对于迭代式任务，出于性能方面的考虑，Flink会尝试在存储数据的节点上执行相应的计算任务。此外还可进行“增量迭代”，或仅对数据中有改动的部分进行迭代。

在用户工具方面，Flink提供了基于Web的调度视图，借此可轻松管理任务并查看系统状态。用户也可以查看已提交任务的优化方案，借此了解任务最终是如何在集群中实现的。对于分析类任务，Flink提供了类似SQL的查询，图形化处理，以及机器学习库，此外还支持内存计算。

Flink能很好地与其他组件配合使用。如果配合Hadoop 堆栈使用，该技术可以很好地融入整个环境，在任何时候都只占用必要的资源。该技术可轻松地与YARN、HDFS和Kafka 集成。在兼容包的帮助下，Flink还可以运行为其他处理框架，例如Hadoop和Storm编写的任务。

目前Flink最大的局限之一在于这依然是一个非常“年幼”的项目。现实环境中该项目的大规模部署尚不如其他处理框架那么常见，对于Flink在缩放能力方面的局限目前也没有较为深入的研究。随着快速开发周期的推进和兼容包等功能的完善，当越来越多的组织开始尝试时，可能会出现越来越多的Flink部署。

总结

Flink提供了低延迟流处理，同时可支持传统的批处理任务。Flink也许最适合有极高流处理需求，并有少量批处理任务的场景。该技术可兼容原生Storm和Hadoop程序，可在YARN管理的集群上运行，因此可以很方便地进行评估。快速进展的开发工作使其值得被大家关注。

# 接入storm

[Storm消费JDQ3.0集群](http://bdp.jd.com/helpCenter/front/showDocumentList.html?docId=227)

<http://bdp.jd.com/jrdw/jrctask/storm/overview.html>

可以查询运行日志

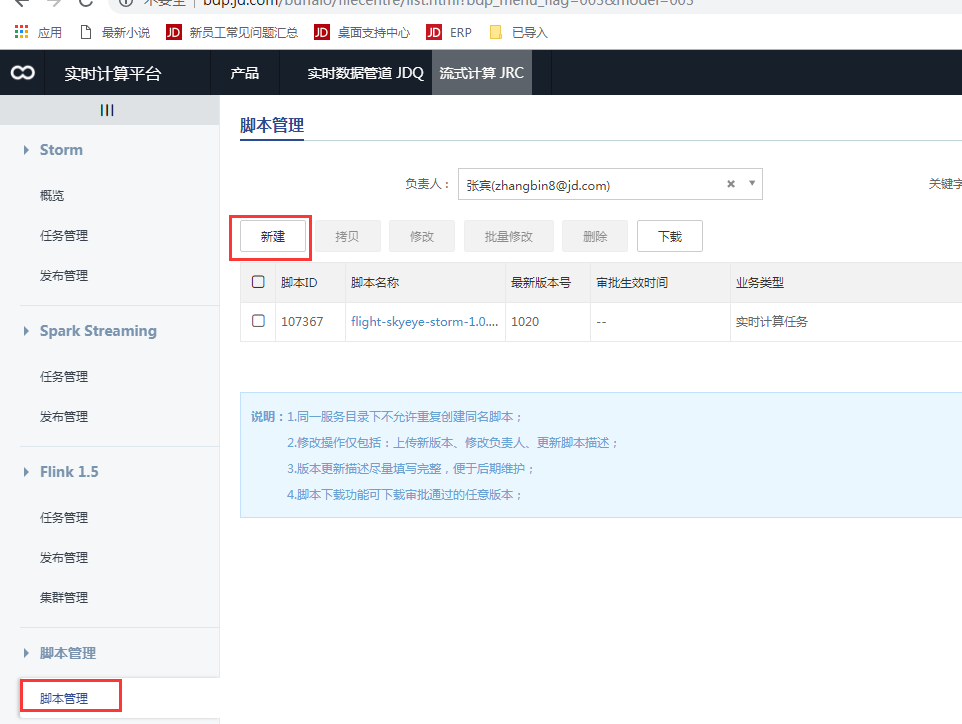
<http://bdp.jd.com/jrdw/jrctask/storm/listStorm.html?status=1>



Storm的jar包上传

<http://bdp.jd.com/jrdw/jrctask/storm/stormPublishList.html>

先上传脚本





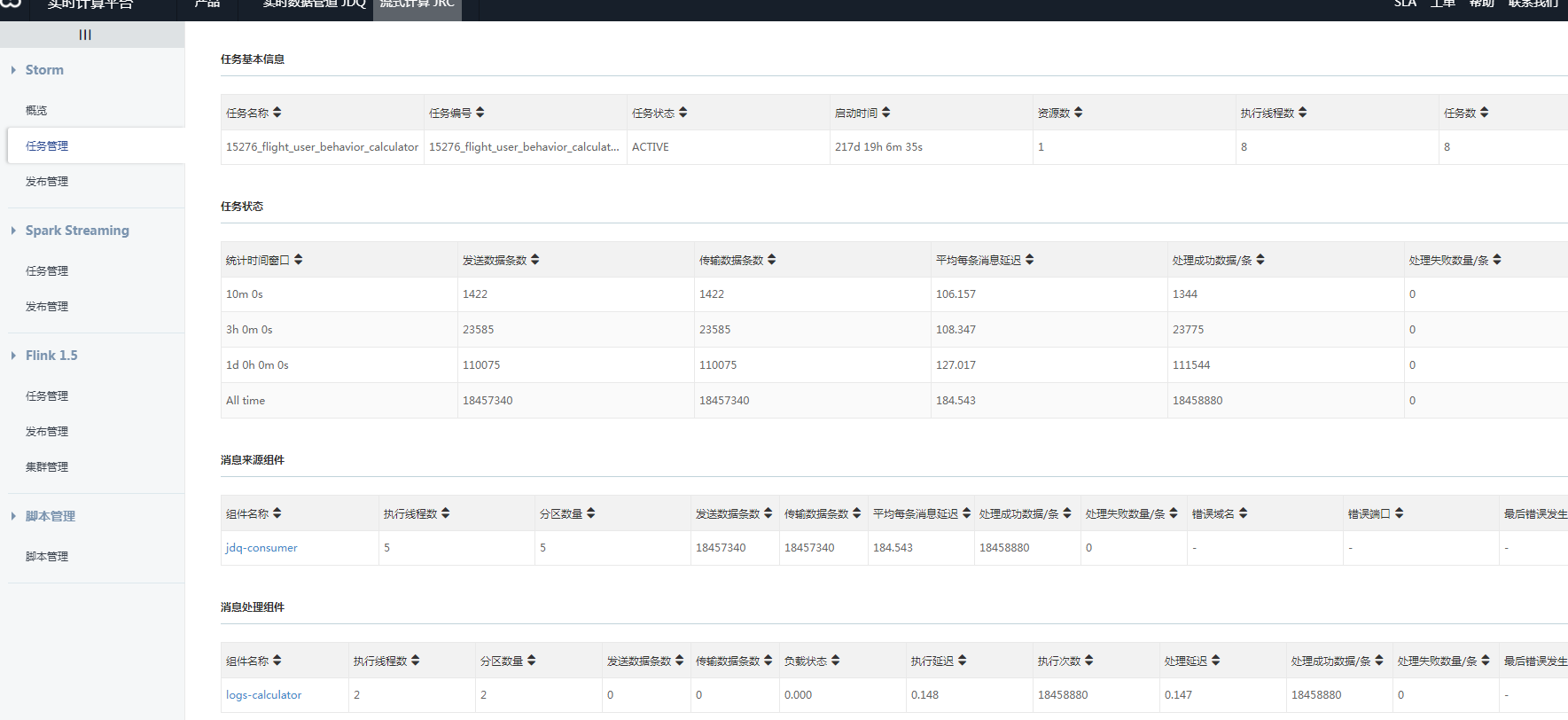
然后新建发布信息：



发布按钮发布任务



查看任务的基本信息



任务监控

