Beatles小记（分布式数据流分析框架）

Author：放翁（文初）

Email：[fangweng@taobao.com](mailto:fangweng@taobao.com)

Mblog: weibo.com/fangweng

Blog: <http://blog.csdn.net/cenwenchu79/>

概述 2

建议 2

背景 2

整体设计 4

**任务调度抽象** 4

**任务抽象** 6

**统计模型抽象** 6

**整体组件和流程设计** 8

局部设计 11

待续 12

# 概述

**重要：全文内容都是参照这个源码地址内容所写，因此两边对着看会更清晰明了：**[**https://github.com/cenwenchu/beatles**](https://github.com/cenwenchu/beatles)

这篇小记主要处于两方面考虑：首先，希望打破一提到海量数据分析，就只有hadoop基础上的一系列工具，更多的时候很多企业需要的是更轻量的设计（办喜酒杀猪杀鸡未必都要用一把刀），因此将开放平台基础分析组件重构版本beatles的设计写出来，给出更多的思考空间。其次，也是希望推广一种思想，所有的系统，框架设计简化（可扩展），小部件精致化，这样才能让很多项目能够整体灵活，细节给力。

## 建议

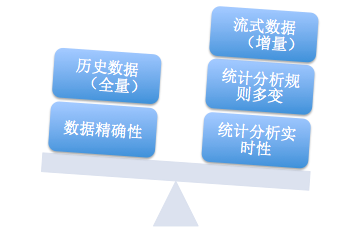
这篇小记一共分成4部分，概述，整体设计，局部设计，待续。如果你只想了解个背景，那么看完概述即可，如果对于流式分析的大框架设计感兴趣（看看省略了分布式计算集群的什么？核心设计是怎么样的），请仔细看完整体设计。如果还对代码优化有兴趣，那请看局部设计（细到代码功能级别）。最后留下的待续，将会增加后续的一些扩展，及同学看完后提出问题的解答（比较通用的一些问题，例如容灾啊，啥啥啥很多被认为很重要的东西）

## 背景

07年底开始做开放平台，当时每天访问量在4kw左右，考虑到开放平台的数据透明化需求，开始考虑如何做统计分析，当时需求是一天出一次结果即可，因此自己摸索搭建Hadoop迷你集群，开始了分析之路。09年公司调整加入淘宝开放平台，当时每天服务调用量2亿，数据分析要求比较散，从服务的系统数据统计到业务趋势统计都有涉及，而且统计需求变化较多，因此开始筹备自己写简单的统计抽象模型来规避MapReduce类，提高适应变化的能力，同时出于简化系统设计维护的要求，直接将每日分析数据放置到集中服务器上，每日拉取，切割，分析（统计分析引擎抽象完成）。2010年开放平台基础体系开始建立，对于服务质量，应用行为，用户安全都提到了较高的要求上来，分析结果从原来的统计分析，扩展到了监控告警，每日分析转变为增量分析（频度1小时左右），简化的任务调度模型抽象出来，同时服务调用量增涨到了9亿。2011年平台数据统计分析及时性要求更高，同时开始开放统计数据给外部开发者（系统可用性和效率要求更高），整体框架和局部设计不断优化和改进，截至今年11月，单日最大服务调用量19亿，增量统计实时性要求在2分钟内（包含数据分析和数据产出，低峰期1分钟，高峰期1分半），系统可用性要求高于99.6％，而投入的服务器比起动则几十台甚至上百上千的Hadoop系统来说，就是一个迷你集群（一台Master实体机（16核，16g内存），12台虚拟机（虚后5核，8g内存，实际为4台实体机）），每天负责600g增量数据分析，产出1.5g数据。

很多时候很多开发人员会问到说在业务和代码结构优化冲突的时候怎么办，老板要结果，而程序员要的是看起来不恶心的代码，但很多时候，我们就是在摸索中做事。上面描述的背景就好比开始买的是件夏天穿的短袖，然后天气不断变冷，开始给短袖加袖子，然后在身上贴补丁，但最后真的要到冬天的时候，应该怎么办，在秋天和冬天之间，作为核心代码负责人，就应该保证系统可用性的情况下做好另一手准备（简单来说，时间不是别人给的，而是自己给的，天晴补漏，雨天不愁）。因此年末的两周将2年中断断续续走过的路，重新整理了一遍，取名为beatles（甲壳虫），因为甲壳虫虽小，但聚集起来能够吃掉一大片叶子（业务系统各种需要分析的数据），因此这个框架首先是个很小的内核（希望有更多扩展和参与者），其次不是一个从头开始的项目，而是一个两年多断断续续演进产品的积累。

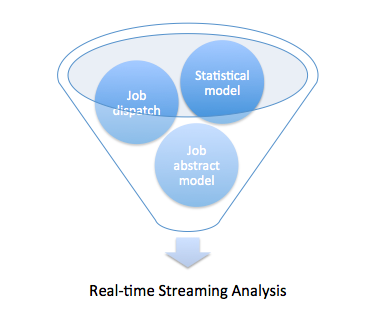
Beatles不是一个万能的技术产品，它出生和发展就为它适合的场景做了定义，因此使用和扩展的时候需要明确的了解是否合适，避免勿用，下面两个图会大志说一下它的特点和适用场景。



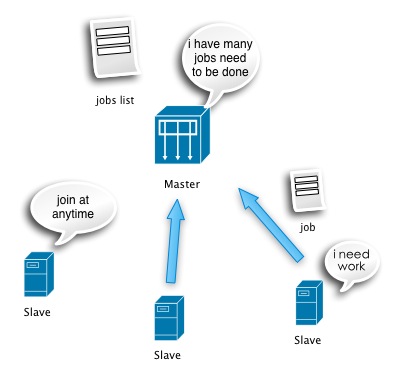
上面这张图左边部分是Beatles可以局部牺牲的，右边部分是场景要求的。由于是对流式数据的增量分析，因此对于历史数据的全量挖掘无能为例（这部分完全可以用Hadoop这种离线分析系统来做）。数据精确性要求所有数据在分析的任何一个环节都要做好保护（数据输入，分析过程，数据输出），而这种强完整性要求势必会使得系统的效率和可用性降低（和右边实时性矛盾），因此会被放低一些要求（类似于计费结算等就直接一天走一次分析即可）。在左面所看重的三个部分大致分布的场景为：监控告警（业务，系统，用户，平台透明化），业务即时分析对比（ABtest），系统灰度发布对比，用户实时统计展示（非金额等数据一致性要求较高的内容展示）。

# 整体设计

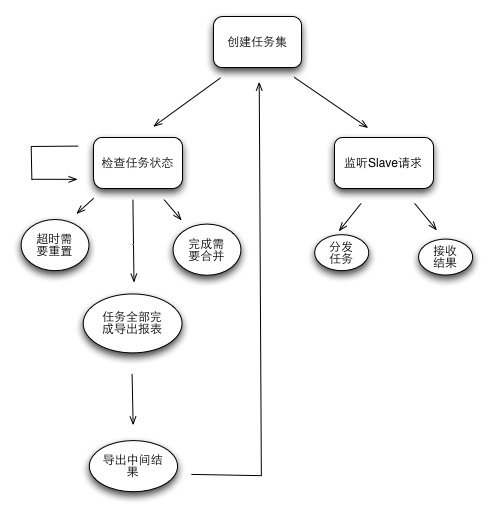
要满足上面所说的场景，实时流式数据分析需要做哪些功能？



## **任务调度抽象**

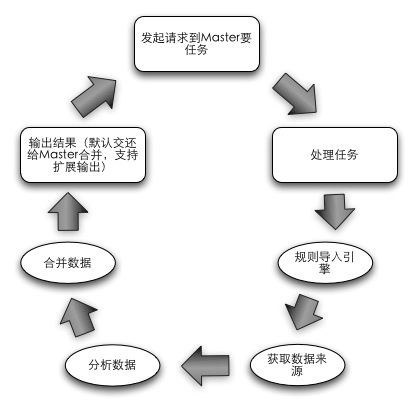


Beatles的任务调度十分简单，遵循两个原则：1.按需分配（Slave的多少及Slave自身执行任务的快慢自然促成的分配方式）2.任务粒度细化，粗暴简单的任务重置（通过透明化监控任务可能出现的问题，避免集群陷入一个任务的纠结中）。优势：简单，高效，易扩展（Slave随时来，随时走）。劣势：对于任务执行可控度较弱（通过任务细粒度和粗暴重置状态的方式来降低风险，增加的只是节点重复计算的浪费可能性）。



master处理流程

可以看到Master整体就两部分工作，对内部任务的管理维护，对外部slave请求的处理（请求获取任务，返回处理后的结果（Slave也可以不返回结果，根据Job定义来判断，防止Master变重））。Master单点并不可怕，只要遵循两个原则：现场可快速恢复，分析流程可追赶。因此做到Master所有状态定期外移和实时监控，即可满足这种简单的Master可用性需求。

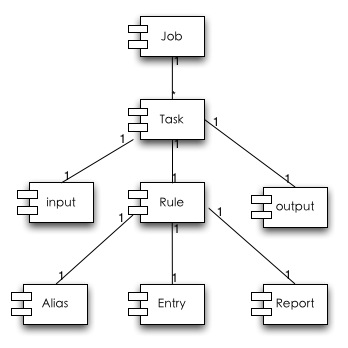


slave处理流程

Slave更为单纯，整个生命周期就是获取任务，分析任务，返回任务结果的一个环，内置一个分析引擎和交互组件，根据任务的定义来无差别化的处理各种分析工作（Job定义了数据的输入来源，输出目标地址，分析规则）。Slave的设计主要考虑如何做到无业务规则侵入和数据来源限制，满足了这些需求的情况下才能够实现节点处理无差别性，各种分析任务可以跑在一个集群上（实现计算节点可复用）。

## **任务抽象**

任务抽象设计比较简单，主要结合任务调度设计，实现计算节点的无差别化。



Job是一类分析的定义（例如对gc的日志分析，对服务调用日志分析可以定义为两个Job），Job中的Task表示对于这一类数据分析再次拆分任务，来分解海量数据处理，Task中继承了Job中的输入和输出，支持多种模式的数据来源和数据输出。Rule就是分析统计模型抽象部分主要分成：Alias（对于分析数据的列别名定义），Entry的MapReduce的定义，Report是Entry整合成用户可接受的Report的定义。

## **统计模型抽象**

统计模型抽象主要分为两部分：统计模型抽象和统计流程抽象。统计模型抽象就是将MapReduce的Key-Value统计，转化成为传统意义上的报表结构。

分析的输入:（弱业务含义的大表）

c1,c2,c3,c4,c5,c6

c1,c2,c3,c4

c1,c2,c3,c4,c5,c6,c7

……

MapReduce可以处理的：

如下图，传统报表的一行可以看作是多个相同key但不同统计字段组合的结果。

例如：输入的数据结构如下：

服务名称，服务类型，服务上行数据流量，服务处理结果（错误码），服务耗时

真实日志如下：（分隔符可在分析时指定，这里用逗号作演示）

taobao.user.get,read,100,0,20

taobao.product.add,write,1000,0,50

……

那么定制如下MapReduce组合：

Key：服务名称，Value：服务上行数据流量总和。

Key：服务名称，Value：服务耗时总和。

Key：服务名称，Value：服务平均耗时。

Key：服务名称，Value：服务最大耗时。

Key：服务名称，Value：服务最小耗时。

那么将这些MapReduce处理后的Key-value在组合一次就可以得到：

Key：服务名称，Value：服务上行数据流量总和，服务耗时总和，服务平均耗时，服务最大耗时，服务最小耗时。

简单来说其实就是类似于SQL中的Groupby的方式，将一堆<key，value> groupby key。

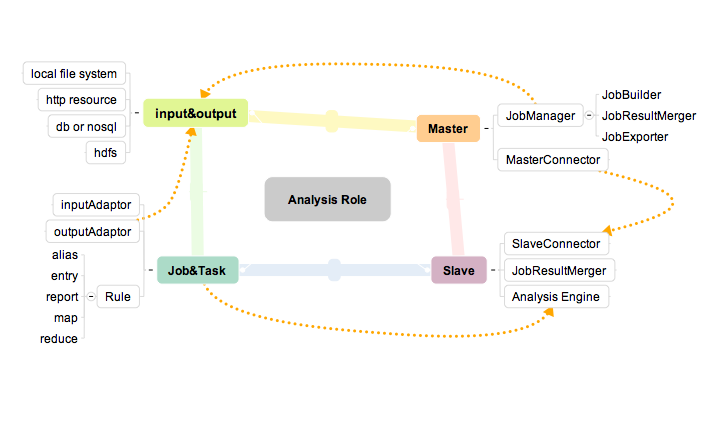
分析流程抽象如下：

分析流程抽象

流程中可以扩展的在第三步和第四步，第三步影响了Key的生成（当简单的列组合成字符串无法满足生成key的情况下可扩展），第四步影响value的生成。（当map的value生成以及Reduce无法满足需求的情况下可扩展），要使用min，max…以外的reduce，可以直接在ReduceClass中作处理，然后使用plain输出实现。

这种流程比传统的MapReduce的写法好处在于可以对输入只读取一次（海量的日志文件为了多种条件分析，反复读取本身就是最大的损耗）。可以看到在文件IO操作上，不会随着分析模型配置的增多而增长，中间数据也不会随着报表组合的不同而过快膨胀（只要报表复用Entry足够多）。

## **整体组件和流程设计**



角色定义

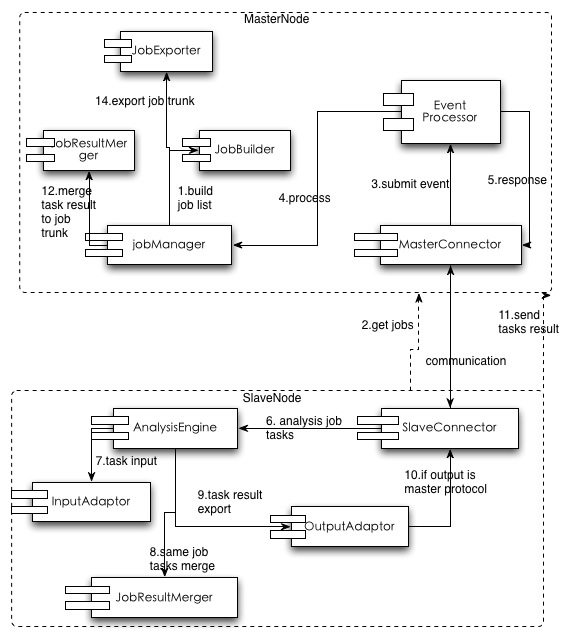
Beatles内部业务组件如上图。

Master包含两个子组件，JobManager用于管理任务，MasterConnector用于与Slave通信。

Slave包含三个组件，SlaveConnector用于与服务端通信，Analysis Engine用于数据分析，JobResultMerger可选，用于在客户端分担服务端汇总结果的压力，同时让Slave可以多线程并行执行任务。（当然单机可以跑多个Slave的实例）。

Job&Task已经提到过用于任务抽象，支持Slave的Analysis Engine的分析无差别性。

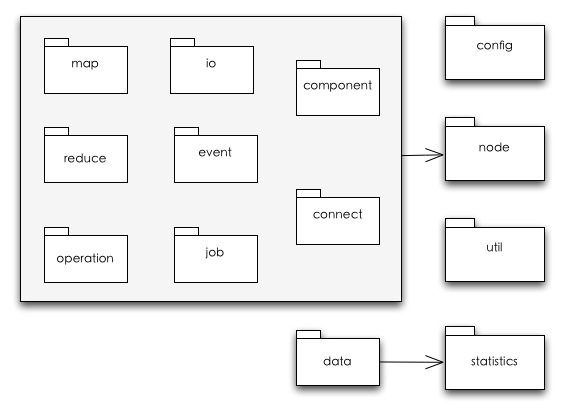
Input&output用于扩展整个框架的各种数据来源，例如job构建的来源，job的输入和输出等。



整体流程

1. Master利用jobManager通过JobBuilder来构建服务端的任务集合。
2. Slave向Master发起要任务的请求。
3. 通过Master和Slave的Connector来做交互。
4. MasterConnector向MasterNode内部的事件处理模块提交事件。
5. JobManager检查内部任务状态后返回未完成且符合条件的任务返回给Slave。
6. SlaveNode收到任务后调用内部分析引擎并行执行任务分析。
7. 分析引擎获得任务的数据来源，开始分析数据。
8. 如果是多个任务并行执行，合并同一个Job的多个Task的结果。
9. 导出分析后的结果
10. 如果是需要汇总到Master的话，利用SlaveConnector返回给Master。
11. MasterConnector获得返回的分析结果数据。
12. MasterNode类似走事件流程，然后调度到合并组件合并结果。
13. 当Job任务全部完成就调用JobExporter导出数据。

代码结构体系：sourcecode：（<https://github.com/cenwenchu/beatles>）



整体包结构

整个项目内容不多，根据包名的前缀可以发现主要分成两块：node，Statistics。前者是任务调度及任务抽象，后者是统计分析模型抽象。

Config中是多个角色各自的config定义，同时这些config会在一个实体里传播，例如MasterConfig就在MasterNode中传播到jobManager和MasterConnector组件中，SlaveConfig就在SlaveNode传播到分析引擎组件和SlaveConnector中。

Node中的结构如下：

Component：对Node的各个组件接口的实现。

Connect：Master与Slave交互的接口定义和实现。

Io：对于Job的输入输出来源的接口定义和默认几个实现。

Event：定义了Master和Slave这样的Node中需要处理的事件。

Job：任务抽象定义。

Map，Reduce：支持当分析引擎无法满足的Map，Reduce的情况。（足够通用的情况下可以被抽象到主框架中）

Operation：定义了Node结构中需要异步处理事件。（因为当前Node的Event是单线程处理的，因此事件执行如果比较消耗，则需要异步后台执行，或者并行执行）

Util包是一些工具类和定义类。

Staitistics是分析引擎接口和实现，其中Data中是分析规则的抽象。

至此为止，整体的结构设计就如上所述了，整体上结构比较简单直接，可扩展性为了支持分析规则扩展，不同计算场景扩展，效率和可靠性扩展。下一个部分将会细化到具体的模块代码设计上来谈优化和代码技巧。

# 局部设计

以下内容将会按照<https://github.com/cenwenchu/beatles> 的包顺序讲述一些设计和实现的detail，非适用于各种场景，但对java开发应该会有所启发。

具体参看文章：<http://blog.csdn.net/cenwenchu79/article/details/7057574>

# 待续