**BRFS文件系统概要设计**

**Version 1.0.0**

**北京博睿宏远数据科技股份有限公司**

**变理记录：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **版本号** | **修改时间** | **修改内容** | **修改人** |
| 1 | 1.0.0 | 2017-12-26 | 文档初始化 | 魏征,陈俞朋,朱成岗,张念礼 |

目录

[1 引言 4](#_Toc351122288)

# 引言

## 背景

由于博睿的迅速发展，内部出现了很多产品（SDK、Browser、Server等）。这些产品都有快照存储的问题（由于文件小且数量多，导致在存储上会有很多的问题），当然这些问题使用目前开源的文件系统也能解决问题，但毕竟是开源的东西，使用和部署对于咱们来说都不是很方便。所以咱们应该做一个自己的文件系统，首先是方便各个产品使用，而且等成熟以后可以开源到github，以提高博睿在业界里的形象。

## 目的

本篇的目的是从产品规划的角度，给出博睿文件系统项目的应用范围、产品形态、开发技术框架、主要模块技术架构、业务定制及模块扩展的方式、可用的技术框架等进行通盘考虑，以便为后续开发工作提供指南，并为详细设计文档提供重要参考。

预期读者：产品经理、市场经理、售前人员、开发人员、测试人员。

## 术语定义

***StorageName***：存储空间名称，简称sn，用户在存储数据的时候，必须指定一个StorageName。

***ServerID***：服务标识，每个服务在启动的时候，必须用有一个服务标识。

***FID***：文件标识，由一系列编码过的字符串组成，用户需要提供相应的FID来读取数据。格式：sn\_time\_combinerFileName\_ServerIDn\_offset\_size。

# 总体设计

为了满足现在各产品线的需要，以系统高稳定、可扩展为需求，以相关开源产品（FastDFS等）为参考， 对BRFS文件系统进行设计，从而达到目前博睿各产品线上对小文件数据保存的功能及性能需求。

## 需求规定

参见《BRFS系统需求规格说明书.doc》

## 运行环境

### 硬件环境

|  |  |
| --- | --- |
| 设备 | 规格要求 |
| 应用服务器 | 中档LINUX企业服务器  4CPU， 内存8G以上  硬盘：多块,高速（10000rpm以上）1T-4T  千兆以太网卡，光纤接口卡 |
| 客户端 | 支持网络访问 |

### 操作系统及软件环境

|  |  |
| --- | --- |
| 用途 | 要 求 |
| 服务器操作系统 | LINUX操作系统(centos5.5) |
| zookeeper | 3.4.6版本 |
| Jdk | 1.8版本 |
| Hadoop-hdfs | 2.6.0版本 |
| nginx | 1.8.5版本 |
| netty | 4.1.19版本 |

## 系统设计

### 系统架构

系统内部也全部采用插件化方式，方便扩展，便于维护。系统在系统安全上从操作系统层、软件安全、通信系统本身做了程序监控、系统日志进行了异常追踪，事后排查。图1示意系统的整个架构图。

系统整体构架在系统软件基础上，通过相应的系统功能模块的定制，并遵循国际标准，利用组件化手段加以实现，使得各功能模块之间在物理上独立、在逻辑上统一，彼此留有接口，达到无缝的结合，易于扩展。



图 1系统框架图

### 模块简介

* Client：

该模块以jar包的形式被其它系统引用。

通过http方式使用pb协议与Server端通信。

提供了权限验证，写数据，读数据，删除数据等方法。

* Server：

该模块作为服务端运行在服务器上，提供client端的相关服务。

提供了权限验证，写数据，读数据，副本保存，副本自动平衡，删除数据，归并数据，环境监控等功能。

* nginx：

开源组件。

用于负载均衡Client请求。

也可用于Server端部署在安全区无法外网访问时的跳转代理。

* zookeeper：

开源组件。

用于各server的分布式管理及协作工作。

* hdfs：

开源组件。

可选模块。

用于归并后的数据存储。

* 可视图化监控与报警：

该模块以web的形式提供服务。

调用server的相关接口获取监控数据，把server的文件系统（资源信息）的相关信息实时的展现出来，方便于管理；另如发现相关信息超过了配置的预警阀值，就会进行邮件或短信报警。

* 日志模块

该模块主要用于收集各server的日志，方便日志管理，排查问题。

# 系统设计原则

## 可扩展性

方案的设计应考虑对业务种类和数据种类的扩展需求，以及业务量增长和数据量增长对存储容量、处理性能等各方面的扩展要求。

## 稳定性

方案设计必须顾及系统的稳定运行，当一个项目出现问题时尽量不影响其他专题。

应从方案上减少一些不必要的专题间的依赖。

## 安全性

本系统用户按角色权限设置对系统的操作，各用户在自己角色的权限内操作相关功能，且本方案的设计应考虑到每个服务内容管理（删除、修改）的权限、操作日志等。

## 可维护性

方案的设计应考虑维护的便利性，考虑采用相关机制来实现对问题的快速定位和解决等方面的要求。

系统记录运行日志，方便排查系统运行出现的异常状况。

## 可靠性

本系统从设计上考滤备份和冗余机制，从而保障了系统在高压力的时候可以支持负载，出错后可以回滚，且输出相关日志，以及从系统监控等多方面的手段来保障系统的稳定可靠；

## 系统性能

本系统对服务请求能快速响应；访问记录、报表部分由于访问历史数据会比较大，在记录流水表里数据只保存近6个月的数据， 超过日期范围的数据定期删除，以提高系统的性能。

# 系统模块设计

## 客户端

### 功能概述

提供给用户，用来访问服务端，进行文件数据的读写删等操作。

### 运行方式

以jar包形式提供，用户使用该包和服务端交互。

初次讨论设定为如下：

BRFSClient client=new BRFSClient (property)

BRFSAdmin admin=client.getAdmin()

Admin.creatSn/admin.updateSn/admin.deleteSn()

BRFSDataOpt opt=client.connect(sn,passwd)

Opt.write/opt.read/opt.delete(bt,et)

### 操作数据

在sn管理方面，可以创建sn，修改sn，删除sn。

在数据操作方面，可以写数据，读数据，删除数据。

### 相关接口

客户端主要提供的接口：

sn创建接口

Sn修改接口

Sn删除接口

Sn认证接口

数据写入接口

数据读取接口

数据删除接口

### 处理流程

主要涉及到以下的处理流程。

#### 创建storageName流程

创建storageName流程大致如下：



（1）合适的Server选取流程：

Client端会维护一个Server的信息列表，信息包括磁盘空闲空间大小、CPU和内存的使用率、网络及磁盘IO等，Client端需要对Server端发送操作请求时，都会根据各个Server当前的资源信息使用情况进行排序，确定出每个Server的优先级，优先级越高的Server可利用的资源越多，排序完毕后，选择优先级最高的Server进行操作。（下同）

（2）StorageName属性包括：StorageName名称、TTL、副本数。

#### 更新storageName流程



#### 删除storageName流程



#### 写数据流程

客户端写数据流程大致如下：



（1）数据封装包括数据校验码生成和数据序列化等操作。

#### 读数据流程

客户端读数据流程大致如下：



（1）合适的Server：

从FID解析出的所有的ServerID，选择其中合适的一个Server进行读取。

#### 删除数据流程

客户端删除数据的流程大致如下：



（1）客户端的删除结果返回可以分同步和异步，同步方式表示Server端真正删除数据后才会返回结果，异步方式表示Server端只要接收到删除请求即可返回。

## 服务端

### 功能概述

服务端主要对客户端发送来的各种请求进行处理，如创建SN，写文件数据等。Server支持分布式部署，并且各个server是对等的。通过zookeeper之间进行各个服务之间的协调性。服务端支持多副本，支持数据ttl设置，支持文件归并。

### 运行方式

运行部署简单，通过修改配置文件，然后用启动脚本，启动服务即可。

### 操作数据

在sn管理中，服务端提供创建sn，修改sn，删除sn等操作。

在data管理中，服务端提供数据写入，数据读取，数据删除等操作。

### 相关接口

客户端和服务端涉及到以下接口：

创建sn接口

修改sn接口

删除sn接口

Sn认证接口

数据写入接口

数据读取接口

数据删除接口

服务端内部接口:

数据校验接口

数据序列化接口

数据转发接口

文件存储接口

ZK交互接口

操作日志记录接口

副本恢复接口

资源调度接口

### 处理流程

主要涉及到以下的处理流程。

#### 服务启动流程

启动服务的大致流程如下：



注意点：由于在进行一系列任务的时候，如删除任务，会将任务记录到ZK中，服务端在启动的时候，必须在ZK上获取未执行的任务，并执行该任务，达到数据一致性。

#### storageName认证流程

storageName认证的流程大致如下：



#### 创建storageName流程

创建storageName的流程大致如下：



注意点：此处会通过ZK的watch机制通知其他的服务端进行sn的更新，如果未通知成功或者通知未完成，则需要服务端显示的去ZK上获取sn列表。

#### 修改storageName属性流程

修改storageName属性的流程大致如下：



注意点：这里的操作都是会获取分布式锁来确保信息安全性的。此处并不是所有的属性都可以修改，目前可以修改的属性为sn的ttl，passwd。

#### 删除storageName流程

删除storageName的流程大致如下：



注意点：任何一个节点具有该SN的数据，该sn是不能直接被删除的，必须先删除所有的数据，才能删除SN。

#### 写数据流程

服务端写数据的流程大致如下：



注意点：CRC校验可以通过用户在使用客户端时，传递参数来确定是否开启校验。

预写入文件指的是本次需要往那个文件里写入数据，普通情况下，一个sn会在当前服务端维护一份预写入文件。预写入文件有大小限制，若使用HDFS进行归并，则可以设置预写入文件为64MB。当预写入文件大小接近64MB时，会自动生成新的预写入文件。

副本选择的规则会根据当前的硬盘容量大小，来选择一个除了本节点的其他节点作为副本存储。

#### 读数据流程

服务端读数据的流程大致如下：



注意点：使用zookeeper获取到的服务列表，可直接定位到相应的服务端，但是有的用户会配置了Nginx，则需要对请求进行转发。也有可能该服务端IO资源紧缺，也可转发请求到副本请求进行读取。

#### 副本保存流程

副本保存流程分为同步副本保存和异步副本保存。

同步副本保存的流程大致如下：



异步副本保存的流程大致如下：



#### 副本自动平衡流程

副本自动平衡流程大致如下：



副本恢复需要满足以下几点：

1. 副本恢复必须保证各个副本在不同的节点上。
2. 副本恢复必须保证多次副本恢复。
3. 副本恢复必须尽可能保证副本在各个服务中，均匀存在。
4. 副本恢复必须保证恢复后的副本能正常访问。

#### 删除数据流程

服务端删除数据的流程大致如下：



注意点：由于用户删除的任务优先级最高，则此处可以设计同步删除和异步删除。同步删除需要所有数据删除成功，返回结果。异步则只需要创建任务即可，后面通过回调来检查删除结果。

#### 定时删除数据流程

Server 定时删除数据流程的流程大致如下：

创建删除任务流程图：



删除任务创建说明：

1. 删除任务由定时服务触发，可配置
2. 任务控制检查，根据当前服务的cpu，内存，任务的优先级，以及制定的策略，判断任务是否创建。
3. 任务在zk上是有序的;

删除数据流程度：



1. 定期发起删除任务，可配置
2. 由任务控制根据cpu 内存，任务优先级，删除任务的数量 和制定的策略 决定是否删除任务
3. 任务在zk上是有序的;

#### 定时归并文件流程

归并任务创建流程



创建归并任务：

1. 归并任务为定时触发，可配置
2. 由任务控制根据cpu 内存，任务优先级，任务的数量和制定的策略 决定是否执行任务；
3. 归并任务创建时，需要将sn配置的归并校验信息存入任务信息中。一般不建议文件校验，影响效率。
4. 任务在zk上是有序的;

归并任务执行流程图



归并任务说明：

创建归并任务：

1. 归并任务为定时触发，可配置
2. 由任务控制根据cpu 内存，任务优先级，任务的数量和制定的策略 决定是否执行任务；
3. 归并任务可校验文件正确性，一般不建议文件校验，影响效率。
4. 归并后的文件，由sn的TTL控制文件的删除，这样可保证归并期间的数据的查询效率，
5. 任务在zk上是有序的;

#### 任务控制检查流程

任务控制检查流程图



1. 集群的任务信息 都存放在zookeeper
2. 任务的执行与否 与cpu，内存，IO，任务优先级，正在执行的任务的种类及数量，未执行的任务的种类及数量和制定的任务执行策略相关
3. 每个server的任务，由自身创建
4. 每个server取任务时，都只取自己对应标识的任务。
5. 每种类型的任务都是顺序存储
6. 当资源充足的情况下，任务的优先级为，备份>归并>用户删除>定时删除;
7. 当资源不充足情况下，任务优先级为，用户删除>定时删除>归并>备份，同时发送报警；
8. 任务的状态有，未执行，执行中，完成，异常，重试

任务切换流程



# 异常处理

## 出错处理设计及安全保密设计

* 努力提高系统容错性，避免因用户的疏忽对系统造成的损坏；
* 采用友善界面的形式为用户提供警告信息；
* 一些关键性操作（比如：更新、删除操作），都应提供留痕机制；
* 对系统的维护指定专人进行管理，并且导出的数据具有一定的保密设置。

## 异常处理设计

### 架构上异常处理设计：

1. 当系统出现异常时，系统自身会通过短信或邮件的形式发送异常信息给运维人员并抄送给相关技术人员，使异常情况尽快得到处理解决。
2. 当出现第三方资源不可调配的情况，或是由终端传输过来的指令解析出现错误等非程序的错误时，系统都会给出友好提示，告知终端用户请求失败原因。
3. 记录详细完善的系统操作日志，方便运维人员或相关技术人员查看和了解系统运行状况。

### 代码上异常处理设计：

1. 捕获了异常，就要对它进行处理，不要捕获异常之后又把它丢弃，不予理睬。
2. 在catch语句中尽可能指定具体的异常类型，不要不管什么类型的异常都写成Excetpion，必要时使用多个catch。不要试图处理所有可能出现的异常。
3. 要尽量减小try块的体积，不要将全部代码写在一个try...catch中。
4. 不要过分细化异常。不要在每个语句上都加上异常处理，最好将一个工作任务放在try块内。如果其中有一项操作失败，可以随即放弃任务。
5. 保证所有资源都被正确释放。充分运用finally关键词。
6. 在异常处理模块中提供适量的错误原因信息，组织错误信息使其易于理解和阅读。
7. 全面考虑可能出现的异常以及这些异常对执行流程的影响。
8. 异常不能影响对象的状态。
9. 不要同时使用异常机制和异常机制的返回值来进行异常处理。
10. 不要用try...catch参与控制程序流程，异常控制的根本目的是处理程序的非正常情况。

## 补救措施

说明故障出现后可能采取的变通措施。包括：

* 后备技术

说明准备采用的后备技术，当原始系统数据万一丢失时启用的副本的建立和启动的技术，例如周期性地把磁盘信息记录到磁带上去就是对于磁盘媒体的一种后备技术。

* 降效技术

说明准备采用的后备技术，使用另一个效率稍低的系统或方法来求得所需结果的某些部分，例如一个自动系统的降效技术可以是手工操作和数据的人工记录。

* 恢复及再启动技术

说明将使用的恢复再启动技术，使软件从故障点恢复执行或使软件从头开始重新运行的方法。

# 系统维护设计

1. 在设计角度上采用了以插件的方式进行整体系统架构，从而保障了系统具有良好的可扩展性和松耦合的特点。
2. 在终端与后台的消息传递方面,使用了基于JMS开源的消息中间件产品ActiveMQ，并采用它的主题方式进行广播通信，方便了备份平台对接信息，保障了信息不会出现单点，出现事故可查询。
3. 易接入第三方接口，如:

易接入短信接口

易接入业务处理接口

易接入指令解析接口

1. 针对高负载和应备份的模块进行了单独设计，如:

HTTP通信模块，可以采用nginx进行负载均衡和备份实现；

socket服务器，通过内存表进行合适分配，且支持在线扩展。

# 参考资料

《BRFS文件系统需求规格说明书.doc》

《FsatDFS架构设计》

《zookeeper分布式过程协同技术》

《Druid架构设计》

《HDFS架构设计》

【文终】