FastDFS架构设计文档

2019年3月

目录

[一、FastDFS基本特性 3](#_Toc4243516)

[1.1 FastDFS基本组成 3](#_Toc4243517)

[1.2 FastDFS特性 3](#_Toc4243518)

[1.3 FastDFS优点 3](#_Toc4243519)

[1.4 FastDFS缺点 4](#_Toc4243520)

[二、分布式文件存储系统设计实现 5](#_Toc4243521)

[2.1 需求分析 5](#_Toc4243522)

[2.1.1 需求来源 5](#_Toc4243523)

[2.1.2 FastDFS的引入 5](#_Toc4243524)

[2.2 架构设计 5](#_Toc4243525)

[2.3 集群服务安装 6](#_Toc4243526)

[三、Java通过API的文件操作 8](#_Toc4243527)

[3.1 Jar包引入 8](#_Toc4243528)

[3.2 添加FastDFS的配置 8](#_Toc4243529)

[3.3 公共类实现 8](#_Toc4243530)

# 一、FastDFS基本特性

## FastDFS基本组成

Fast文件系统主要由**跟踪服务器(Tracker Server)，存储服务器(Storage Server)和客户端(Client)**三部分组成，在底层存储上通过逻辑的分组概念，使得通过在同组内配置多个Storage，从而实现软[RAID10](https://www.cnblogs.com/ivictor/p/6099807.html)，提升简单负载均衡、并发IO的性能、及数据的冗余备份；同时通过线性的添加新的逻辑存储组，从容实现存储容量的线性扩容。：

1. **Tracker Server：**负责管理、调度和均衡所有的存储服务器和存储服务器组。
2. **Storage Server：**提供容量和备份服务（简单来说就是用来存储文件的服务器）；以 group 为单位，每个 group 内可以有多台 storage server，数据互为备份；每个 storage 在启动后会连接 Tracker，告知自己所属 group 等信息，并保持周期性心跳。
3. **Client:**客户端，上传下载数据的服务器，也就是我们自己的项目所部署在的服务器。

## FastDFS特性

**跟踪服务器(Tracker Server)**作为整个系统的核心枢纽，其完成了访问调度(负载均衡)，监控管理Storage服务器，由此可见Tracker的作用至关重要，也就增加了系统的单点故障，为此FastDFS支持多个备用的Tracker，虽然实际测试发现备用Tracker运行不是非常完美，但还是能保证系统可用。

**存储服务器(Storage Server)**作为上传文件实际的存储，支持同组间的文件备份（binlog的方式实现），由于目前底层对同步后的文件不做正确性校验，因此这种同步方式仅适用单个集群点的局部内部网络。

## FastDFS优点

1）支持在线扩容机制，增强系统的可扩展性  
2）实现了软RAID，增强系统的并发处理能力及数据容错恢复能力  
3）支持主从文件，支持自定义扩展名  
4）主备Tracker服务，增强系统的可用性

5）支持断点续传

6）系统无需支持POSIX(可移植操作系统)，降低了系统的复杂度，处理效率更高

7）文件下载上，除了支持通过API方式，目前还提供了apache和[nginx](https://www.baidu.com/s?wd=nginx&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)的插件支持

## FastDFS缺点

1）对跨公网的文件同步，存在较大延迟，需要应用做相应的容错策略  
2）同步机制不支持文件正确性校验，降低了系统的可用性

3）不支持POSIX通用接口访问，通用性较低

4）通过API下载，存在单点的性能瓶颈

# 二、分布式文件存储系统设计实现

## 需求分析

### 需求来源

1、CRM和云客服系统邮件中存在的附件存放在代码服务器存在存储空间占用过多的问题；

2、CRM和云客服系统在邮件发送时附件需要从本地上传到服务器然后上传到邮件服务器， 两次上传操作浪费时间和服务器内存资源；

3、服务器在不正常挂机的时候会造成文件丢失。

4、所有的文件无统一处理，维护代价比较大。

### FastDFS的引入

由于FastDFS有原文件存储和在线扩容的功能，可以解决代码服务器存储空间占用过多的问题；

FastDFS文件下载除了支持通过API方式，目前还提供了apache和[nginx](https://www.baidu.com/s?wd=nginx&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)的插件支持，转移我们只需要一步上传到文件服务器上，然后通过http路径或者提供的API直接获取就可以添加文件到邮件的附件中，相比于之前的两步上传文件节省了很多时间和操作总的内存和CPU的占用。

FastDFS提供外部连接并提供负载均衡等服务的Tracker Server支持多台备用，提供存储文件的Storage Server支持组内文件静态共享，冗余配置可以在某一台服务器不正常挂机之后文件不丢失。

通过FastDFS统一管理所有文件，降低了CRM和云客服的文件维护成本。

因此，FastDFS分布式文件系统可以完美的解决CRM与云客服系统文件的处理。

## 架构设计

FastDFS可以实现组间负载均衡，组内文件自动共享备份。架构设计如下：

三台服务器节点ccs\_FDS\_01， ccs\_FDS\_02和ccs\_FDS\_03。

1、三台服务器各自部署一套**跟踪服务器(Tracker Server)**节点，保证一个跟踪节点不正常挂机后其他节点可以接入FastDFS。

2、存储的冗余设计：

共三个存储组（group1, group2,group3），相同组，数据自动共享备份。

节点一ccs\_FDS\_01: 安装存储组group1， group2

节点二ccs\_FDS\_02: 安装存储组group1， group3

节点三ccs\_FDS\_03: 安装存储组group2， group3

保证在某一台服务节点不正常挂机之后，各组数据不会丢失。

## 集群服务安装

请参考<FastDFS服务器集群安装手册.docx> 双击下方《FastDFS服务器集群安装手册》打开。



# 三、Java通过API的文件操作

## Jar包引入

该包是官网提供的一个fastdfs client jar包，直接依赖

|  |
| --- |
| <dependency>  <groupId>net.oschina.zcx7878</groupId>  <artifactId>fastdfs-client-java</artifactId>  <version>1.27.0.0</version> </dependency> |

## 添加FastDFS的配置

配置的主要内容为FastDFS的tracker server地址配置（fastdfs.properties），可以添加多个

|  |
| --- |
| fastdfs.tracker\_servers=192.168.0.220:22122,192.168.0.221:22122,192.168.0.222:22122 |

## 公共类实现

|  |
| --- |
| public class FastDFSClient {   private TrackerClient trackerClient;  private TrackerServer trackerServer;  private StorageServer storageServer;  private StorageClient1 storageClient;   public FastDFSClient() {  try {  Properties properties = PropertiesLoaderUtils.*loadAllProperties*("com/tomtop/etc/conf/fastdfs.properties");  ClientGlobal.*initByProperties*(properties);  trackerClient = new TrackerClient();  trackerServer = trackerClient.getConnection();  storageServer = null;  storageClient = new StorageClient1(trackerServer, storageServer);  }catch (Exception e){  e.printStackTrace();  }   }   */\*\*  \* 上传文件方法  \* <p>Title: uploadFile</p>  \* <p>Description: </p>  \*  \** ***@param*** *fileName 文件全路径  \** ***@param*** *extName 文件扩展名，不包含（.）  \** ***@param*** *metas 文件扩展信息  \** ***@return*** *\** ***@throws*** *Exception  \*/* public String uploadFile(String fileName, String extName, NameValuePair[] metas) throws Exception {  String result = storageClient.upload\_file1(fileName, extName, metas);  return result;  }   */\*\*  \* 上传文件,传fileName  \*  \** ***@param*** *fileName 文件的磁盘路径名称 如：D:/image/aaa.jpg  \** ***@return*** *null为失败  \*/* public String uploadFile(String fileName) throws Exception {  return uploadFile(fileName, null, null);  }   */\*\*  \** ***@param*** *fileName 文件的磁盘路径名称 如：D:/image/aaa.jpg  \** ***@param*** *extName 文件的扩展名 如 txt jpg等  \** ***@return*** *null为失败  \*/* public String uploadFile(String fileName, String extName) throws Exception {  return uploadFile(fileName, extName, null);  }   */\*\*  \* 上传文件方法  \* <p>Title: uploadFile</p>  \* <p>Description: </p>  \*  \** ***@param*** *fileContent 文件的内容，字节数组  \** ***@param*** *extName 文件扩展名  \** ***@param*** *metas 文件扩展信息  \** ***@return*** *\** ***@throws*** *Exception  \*/* public String uploadFile(byte[] fileContent, String extName, NameValuePair[] metas) throws Exception {  String result = storageClient.upload\_file1(fileContent, extName, metas);  return result;  }   */\*\*  \* 上传文件  \*  \** ***@param*** *fileContent 文件的字节数组  \** ***@return*** *null为失败  \** ***@throws*** *Exception  \*/* public String uploadFile(byte[] fileContent) throws Exception {  return uploadFile(fileContent, null, null);  }   */\*\*  \* 上传文件  \*  \** ***@param*** *fileContent 文件的字节数组  \** ***@param*** *extName 文件的扩展名 如 txt jpg png 等  \** ***@return*** *null为失败  \*/* public String uploadFile(byte[] fileContent, String extName) throws Exception {  return uploadFile(fileContent, extName, null);  }   */\*\*  \* 文件下载到磁盘  \*  \** ***@param*** *path 图片路径  \** ***@param*** *output 输出流 中包含要输出到磁盘的路径  \** ***@return*** *-1失败,0成功  \*/* public int downloadFile(String path, BufferedOutputStream output) throws Exception {  int result = -1;  byte[] b = storageClient.download\_file1(path);  try {  if (b != null) {  output.write(b);  result = 0;  }  } catch (Exception e) {  throw e;  } //用户可能取消了下载  finally {  if (output != null)  try {  output.close();  } catch (IOException e) {  e.printStackTrace();  }  }  return result;  }   */\*\*  \* 获取文件数组  \*  \** ***@param*** *path 文件的路径 如group1/M00/00/00/wKgRsVjtwpSAXGwkAAAweEAzRjw471.jpg  \** ***@return*** *\*/* public byte[] downloadDytes(String path) {  byte[] b = null;  try {  b = storageClient.download\_file1(path);  } catch (IOException e) {  e.printStackTrace();  } catch (MyException e) {  e.printStackTrace();  }  return b;  }   */\*\*  \* 删除文件  \*  \** ***@param*** *group 组名 如：group1  \** ***@param*** *storagePath 不带组名的路径名称 如：M00/00/00/wKgRsVjtwpSAXGwkAAAweEAzRjw471.jpg  \** ***@return*** *-1失败,0成功  \*/* public Integer deleteFile(String group, String storagePath) {  int result = -1;  try {  result = storageClient.delete\_file(group, storagePath);  } catch (IOException e) {  e.printStackTrace();  } catch (MyException e) {  e.printStackTrace();  }  return result;  }   */\*\*  \** ***@param*** *storagePath 文件的全部路径 如：group1/M00/00/00/wKgRsVjtwpSAXGwkAAAweEAzRjw471.jpg  \** ***@return*** *-1失败,0成功  \** ***@throws*** *IOException  \** ***@throws*** *Exception  \*/* public Integer deleteFile(String storagePath) {  int result = -1;  try {  result = storageClient.delete\_file1(storagePath);  } catch (IOException e) {  e.printStackTrace();  } catch (MyException e) {  e.printStackTrace();  }  return result;  }   */\*\*  \* 获取远程服务器文件资源信息  \** ***@param*** *groupName 文件组名 如：group1  \** ***@param*** *remoteFileName M00/00/00/wKgRsVjtwpSAXGwkAAAweEAzRjw471.jpg  \** ***@return*** *\*/* public FileInfo getFile(String groupName,String remoteFileName){  try {  return storageClient.get\_file\_info(groupName, remoteFileName);  } catch (Exception e) {  e.printStackTrace();  }  return null;  } |