

**FastDFS讲义**

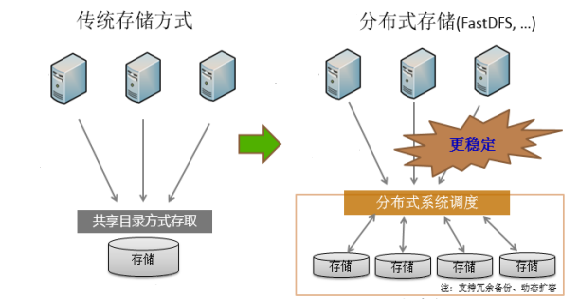
# FastDFS入门

## 分布式文件系统

分布式文件系统 (Distributed File System) 是一个软件/软件服务器，这个软件可以用来管理文件。但这个软件所管理的文件通常不是在一个服务器节点上，而是在多个服务器节点上，这些服务器节点通过网络相连构成一个庞大的文件存储服务器集群，这些服务器都用于存储文件资源，通过分布式文件系统来管理这些服务器上的文件。

常见的分布式文件系统有：FastDFS、GFS、HDFS、Lustre 、Ceph 、GridFS 、mogileFS、TFS等。

分布式文件系统与传统文件系统对比



传统方式弊端

* + - 如果用户数量多，IO操作比较多，对磁盘访问压力很大
    - 如果磁盘发生故障，会造成数据丢失
    - 存储容量有限

## FastDFS简介

FastDFS是一个开源的轻量级分布式文件系统，为互联网应用量身定做，简单、灵活、高效，采用C语言开发，由阿里巴巴开发并开源。

FastDFS对文件进行管理，功能包括：文件存储、文件同步、文件访问（文件上传、文件下载、文件删除）等，解决了大容量文件存储的问题，特别适合以文件为载体的在线服务，如相册网站、文档网站、图片网站、视频网站等等。

FastDFS充分考虑了冗余备份、线性扩容等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

## FastDFS发展历史

2008年4月项目启动，7月发布第一个版本V1.00，两年时间内持续升级到V1.29

2010年8月推出V2.00

2011年6月推出V3.00

2012年10月推出V4.0.0

2013年12月推出V5.0.0

截止目前最新版是V5.11（2017年6月发布）

FastDFS系统架构从第一个版本发布后一直没有大的调整，高版本完全兼容低版本的数据，可以做到平滑升级，推荐更新升级到最新版本

FastDFS代码托管在github上：<https://github.com/happyfish100/fastdfs>

## FastDFS整体架构

FastDFS文件系统由两大部分构成，一个是客户端，一个是服务端

客户端通常指我们的程序，比如我们的Java程序去连接FastDFS、操作FastDFS，那我们的Java程序就是一个客户端，FastDFS提供专有API访问，目前提供了C、Java和PHP几种编程语言的API，用来访问FastDFS文件系统。

服务端由两个部分构成：一个是跟踪器（tracker），一个是存储节点（storage）

跟踪器（tracker）主要做调度工作，在内存中记录集群中存储节点storage的状态信息，是前端Client和后端存储节点storage的枢纽。因为相关信息全部在内存中，Tracker server的性能非常高，一个较大的集群（比如上百个group）中有3台就足够了。

存储节点（storage）用于存储文件，包括文件和文件属性（meta data）都保存到存储服务器磁盘上，完成文件管理的所有功能：文件存储、文件同步和提供文件访问等。

## FastDFS线上使用者

UC （<http://www.uc.cn/> ，存储容量超过10TB）

支付宝（<http://www.alipay.com/>）

京东商城（<http://www.jd.com/>）

淘淘搜（<http://www.taotaosou.com/>）

飞信（<http://feixin.10086.cn/>）

赶集网（<http://www.ganji.com/>）

淘米网（<http://www.61.com/>）

迅雷（<http://www.xunlei.com/>）

蚂蜂窝（<http://www.mafengwo.cn/>）

5173（<http://www.5173.com/>）

华师京城教育云平台（<http://www.hsjdy.com.cn/>）

视友网（<http://www.cuctv.com/>）

搜道网（<http://www.sodao.com/>）

58同城（<http://www.58.com/>）

商务联盟网（<http://www.biz72.com/>）

中青网（<http://www.youth.cn/>）

保利威视（<http://www.freeovp.com/>）

梦芭莎（<http://www.moonbasa.com/>）

51CTO（<http://www.51cto.com/>）

搜房网（<http://www.soufun.com/>）

# FastDFS环境搭建

## FastDFS安装

### 安装前的准备

#### 检查Linux上是否安装了 gcc、libevent、libevent-devel

* + yum list installed | grep gcc
  + yum list installed | grep libevent
  + yum list installed | grep libevent-devel

#### 如果没有安装，则需进行安装

yum install gcc libevent libevent-devel -y

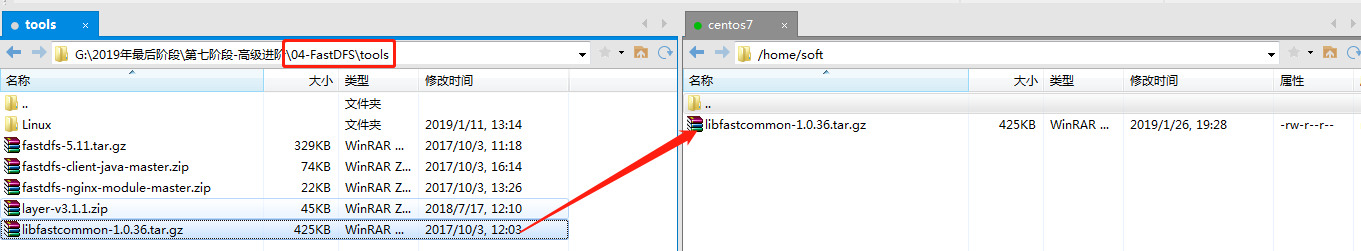


### 安装 libfastcommon 库

libfastcommon 库是 FastDFS 文件系统运行需要的公共 C 语言函数库

**注意：目前最新版本的v1.0.39和最新版的FastDFS5.11不兼容，所有我们这里使用的版本是v1.0.36** 下载地址：<https://github.com/happyfish100>

#### 将下载好的libfastcommon文件上传到Linuxs（/home/soft）



#### 解压下载下来的tar.gz压缩包到当前目录

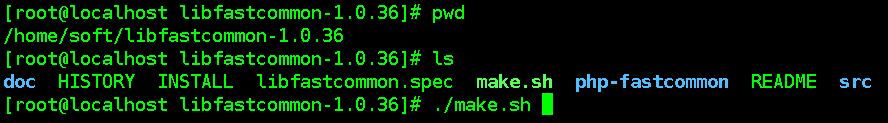
tar -zxvf libfastcommon-1.0.36.tar.gz

#### 切换到解压后的libfastcommon目录

cd libfastcommon-1.0.36

#### 执行make脚本进行编译

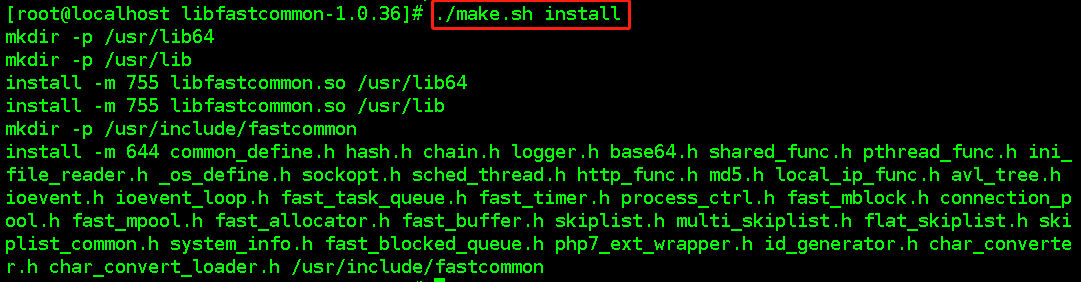
./make.sh



**注意： make编译的时候如果报错，需解决错误后再次进行make，通常发生错误是由于Linux缺少某些依赖库导致，根据错误提示解决错误**

#### 执行make install进行安装

./make.sh install



**至此 libfastcommon 库安装完毕**

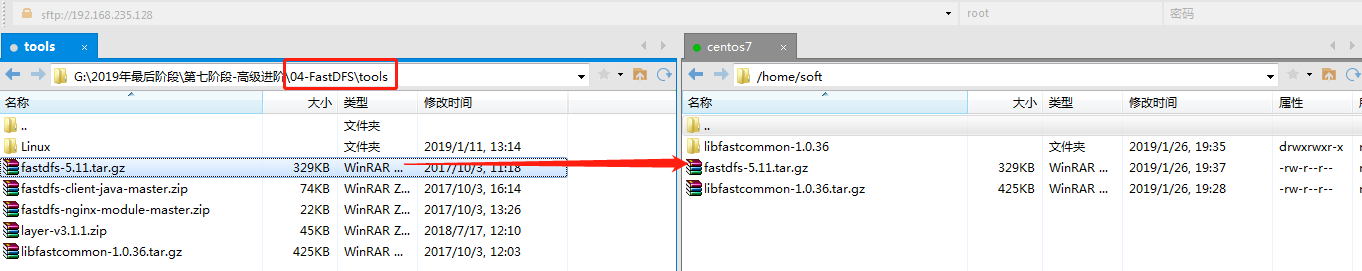
### 安装FastDFS

FastDFS没有Windows版本，不能在Windows下使用。

FastDFS需要安装部署在Linux环境下，我们这里使用的是fastdfs-5.11版本（201901）

下载地址：<https://github.com/happyfish100/fastdfs/archive/V5.11.tar.gz>

#### 将下载好的FastDFS文件上传到Linux（home/soft）



#### 解压下载下来的tar.gz压缩包到当前目录

tar -zxvf fastdfs-5.11.tar.gz

#### 切换到解压后FastDFS的目录

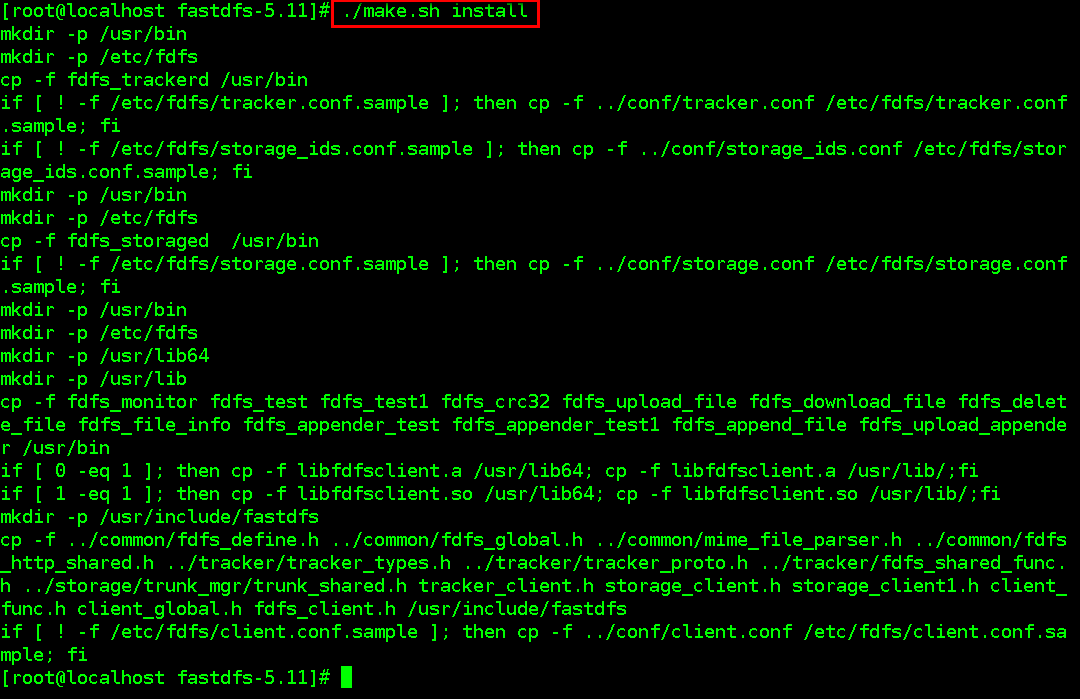
cd fastdfs-5.11

#### 执行make脚本进行编译

./make.sh

#### 执行make install进行安装

./make.sh install



**至此FastDFS安装完成**

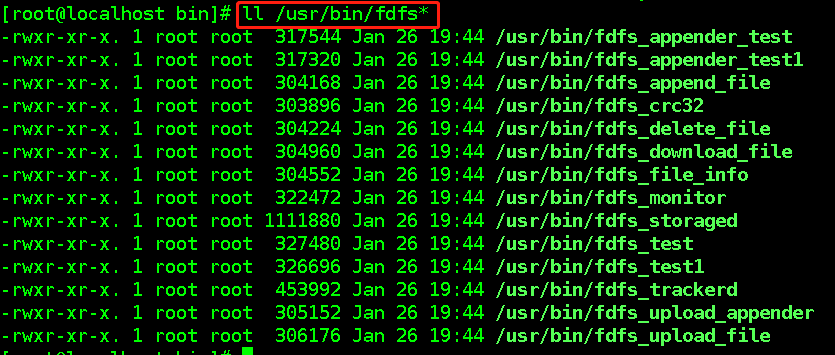
**所有编译出来的文件存放在/usr/bin目录下**

**所有配置文件存放在/etc/fdfs目录下**

#### 查看安装后的效果

##### 查看FastDFS相关的可执行程序

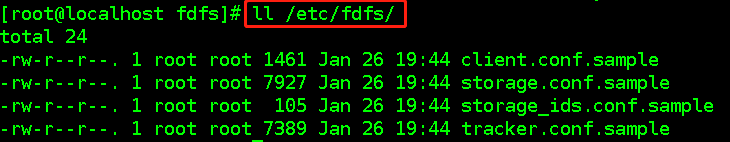
ll /usr/bin/fdfs\*



/usr/bin是Linux的环境变量，可通过echo $PATH查看

##### 查看FastDFS的配置文件

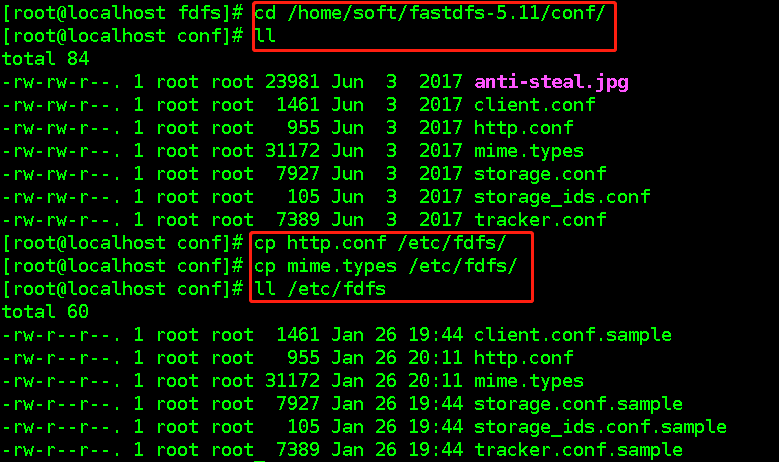
ll /etc/fdfs/



#### 另外注意需要把解压后的fastdfs-5.11/conf目录下的两个文件拷贝到/etc/fdfs/ ，否则后续会有很多奇怪问题不好解决

cp http.conf /etc/fdfs/

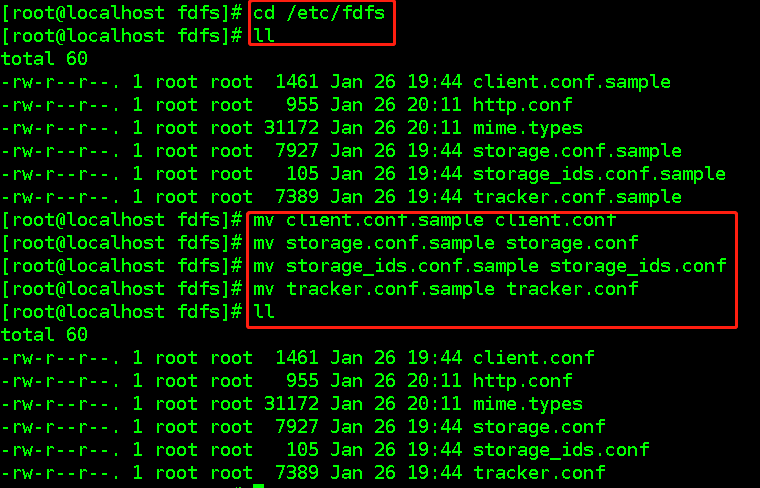
cp mime.types /etc/fdfs/



**这两个文件后续需要用到，所以先拷贝过去**

## FastDFS配置

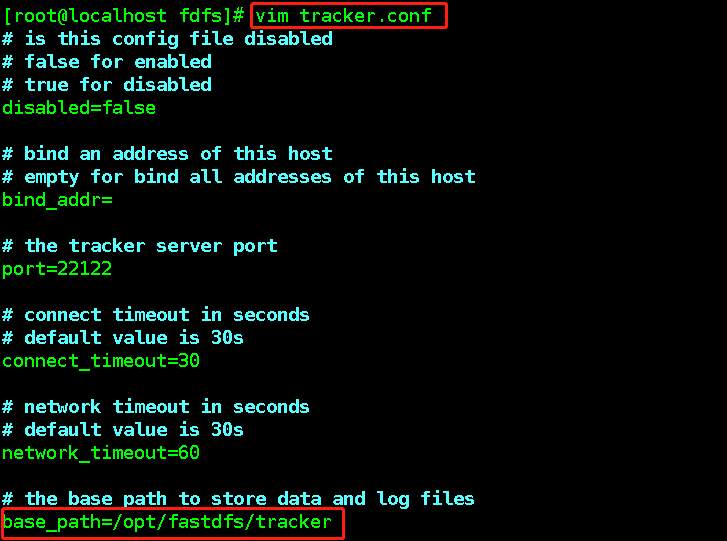
### 去掉/etc/fdfs/目录下FastDFS配置文件的后缀名



### 修改tracker.conf文件

**默认指向的FastDFS作者余庆的目录，因为在我们的机器上不存在，所有手动改一下**

base\_path=/opt/fastdfs/tracker #配置tracker存储数据的目录

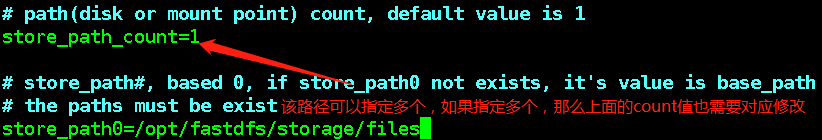


### 修改storage.conf文件

base\_path=/opt/fastdfs/storage #storage存储数据目录

store\_path0=/opt/fastdfs/storage/files #真正存放文件的目录

tracker\_server=192.168.235.128:22122 #注册当前存储节点的跟踪器地址



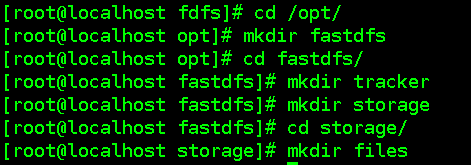


### 在Linux服务器上创建上面指定的目录

/opt/fastdfs/tracker

/opt/fastdfs/storage

/opt/fastdfs/storage/files



### 然后启动FastDFS

## FastDFS启动

FastDFS服务启动需要启动两个脚本：

### 启动FastDFS的tracker服务

在任意目录下执行：fdfs\_trackerd /etc/fdfs/tracker.conf

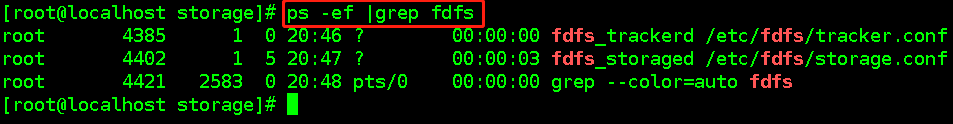


### 启动FastDFS的storage服务

在任意目录下执行：fdfs\_storaged /etc/fdfs/storage.conf



### 查看启动进程



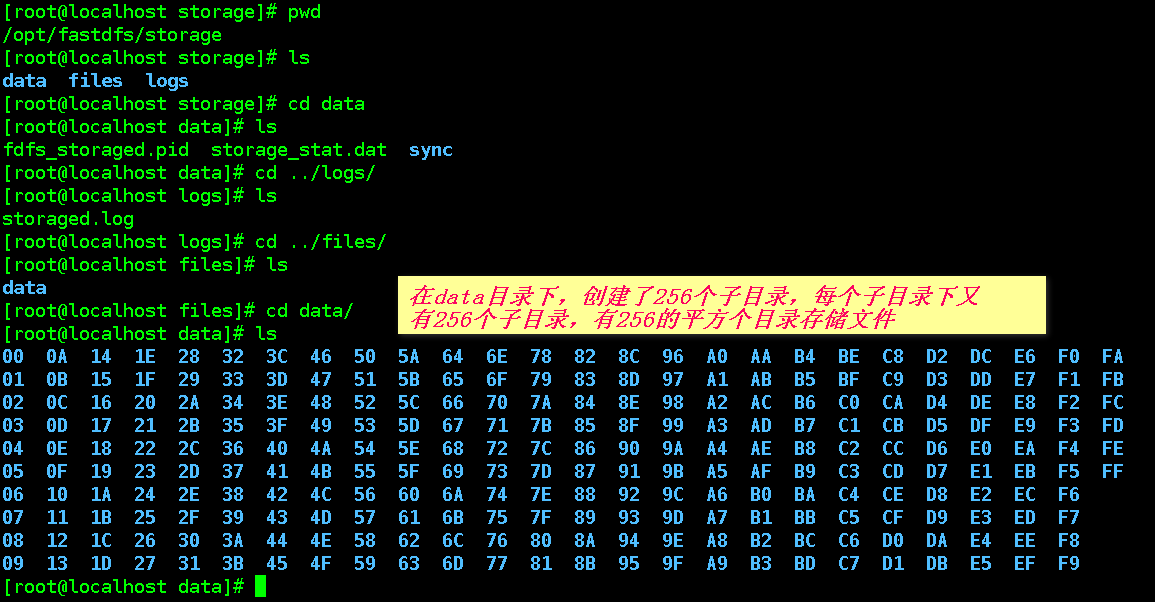
有启动的执行命令即为启动成功

### 查看storage是否已经注册到了tracker下

fdfs\_monitor /etc/fdfs/storage.conf



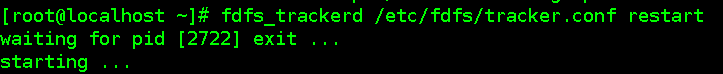
### 首次启动storage后，会在配置的路径下创建存储文件的目录



## FastDFS重启

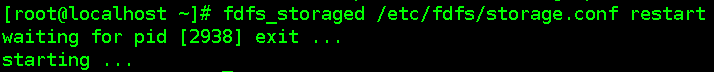
### 重启tracker

fdfs\_trackerd /etc/fdfs/tracker.conf restart



### 重启storage

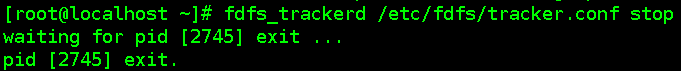
fdfs\_storaged /etc/fdfs/storage.conf restart



## FastDFS关闭

### 关闭tracker执行命令

在任意目录下执行：fdfs\_trackerd /etc/fdfs/tracker.conf stop



### 关闭storage执行命令

在任意目录下执行：fdfs\_storaged /etc/fdfs/storage.conf stop



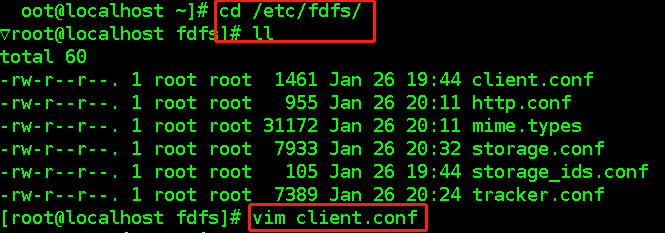
### 或者kill关闭fastdfs，但不建议在线上使用 kill -9 强制关闭，因为可能会导致文件信息不同步问题

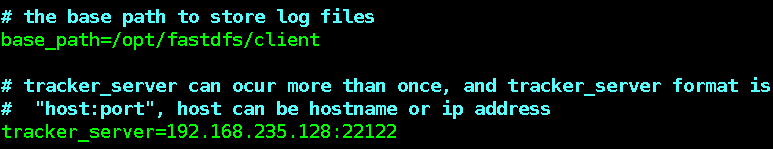
## FastDFS测试

FastDFS安装完成之后，可以使用fdfs\_test脚本测试文件上传

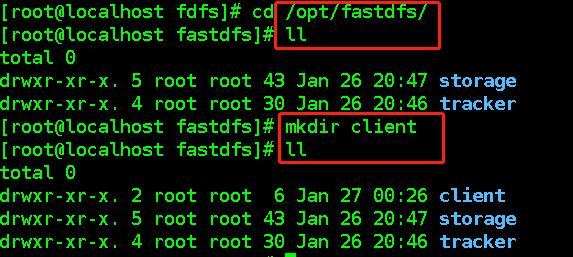
### 测试之前，需要修改client.conf配置文件，修改两个配置

* base\_path=/opt/fastdfs/client
* tracker\_server=192.168.179.128:22122





* 在/opt/fastdfs/目录下创建client

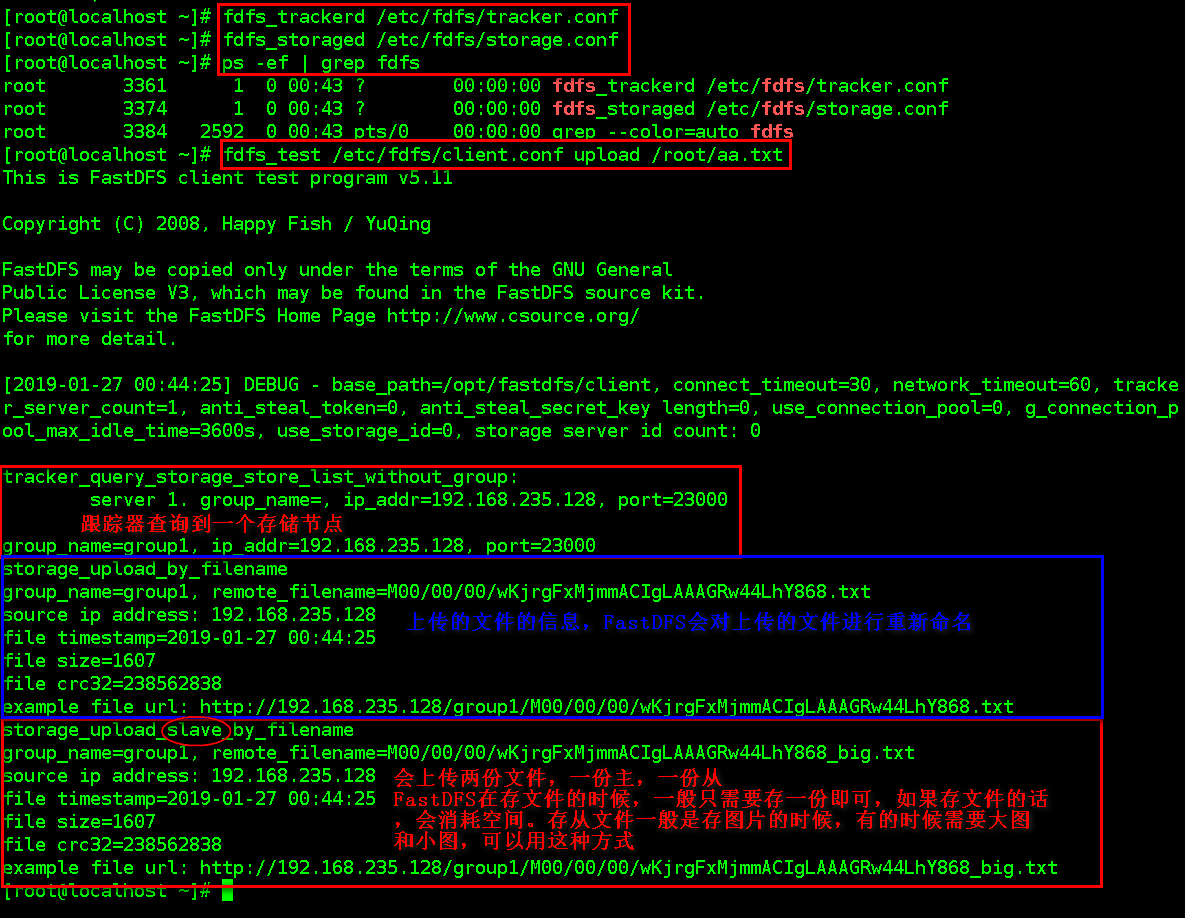


### 测试文件上传

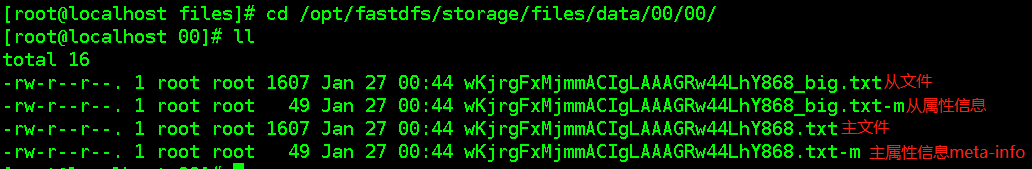
* 准备需要上传的文件



* 执行上传命令fdfs\_test /etc/fdfs/client.conf upload /root/aa.txt



* 切换到存储目录查看文件上传情况

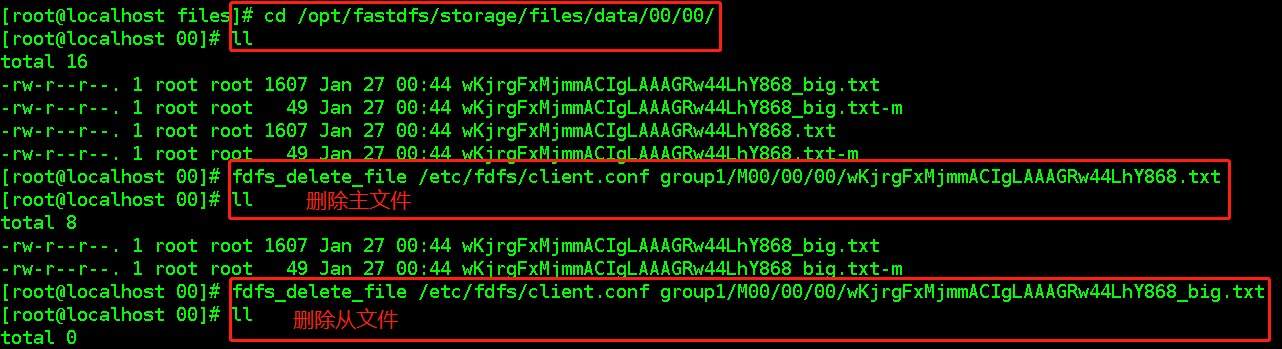


### FastDFS生成的文件目录结构及名称示例



### 测试文件删除

**fdfs\_delete\_file /etc/fdfs/client.conf group1/要删除的文件路径**



### 注意

* 没有搭建集群默认只有一个组group1
* 后缀名包含-m的为属性文件(meta)
* 在Linux中并没有磁盘一说,是虚拟的

# 分布式文件系统FastDFS的HTTP访问

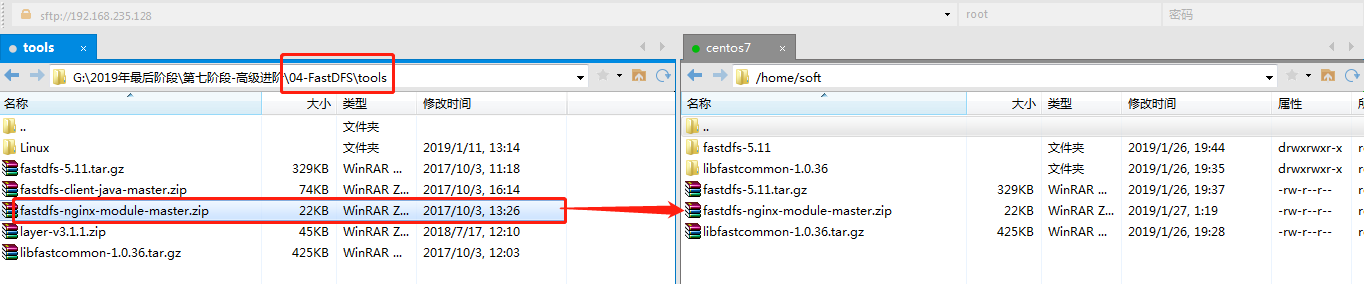
## 概述

在文件上传的时候，上传成功的信息中有提示我们可以通过某个路径去访问上传的文件，但是我们直接访问这个路径，却不可以，那么已经上传到FastDFS文件系统中的文件，我们如何在浏览器中访问呢？

FastDFS提供了一个Nginx扩展模块，利用该模块，我们可以通过Nginx访问已经上传到FastDFS上的文件

## 前期准备工作

### 将Fastdfs的Nginx扩展模块源代码上传到Linux上



### 解压下载下来的fastdfs-nginx-module-master.zip 文件

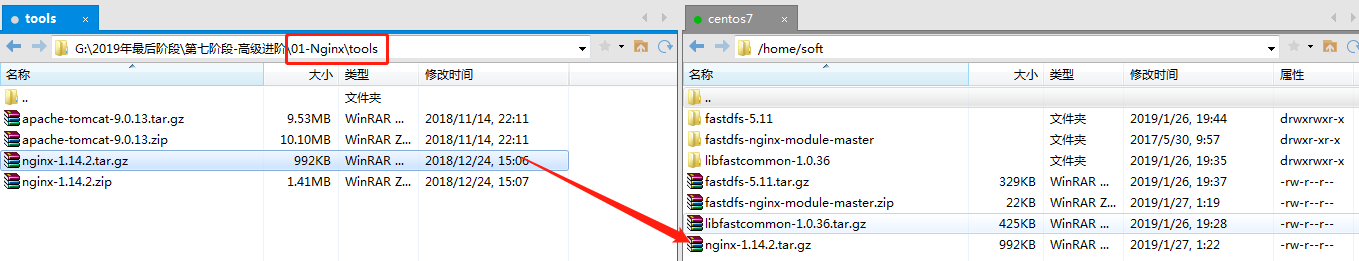
unzip fastdfs-nginx-module-master.zip



## 安装Nginx并且添加fastDFS模块

**因为这个模块必须在Nginx的安装的过程中才能添加，所有我们需要重新安装一个nginx，为了和原来已安装的Nginx进行区分，我们把新安装的Nginx取名为nginx\_fdfs**

### 将Nginx的tar包上传到Linux上



### 解压上传的Nginx文件



### 切换至解压后的Nginx主目录，执行配置操作

cd nginx-1.14.2

./configure --prefix=/usr/local/nginx\_fdfs --add-module=/home/soft/fastdfs-nginx-module-master/src

* --prefix是指定nginx安装路径
  + --add-module指定fastDFS的nginx模块的源代码路径



### 执行命令进行编译

make

### 执行命令进行安装

make install

### 以上安装Nginx的FastDFS扩展模块注意事项

我们知道，Nginx的安装需要Linux安装相关的几个库，否则编译会出现错误，这几个库分别是：

* gcc编译器是否安装

检查是否安装：yum list installed | grep gcc

执行安装：yum install gcc -y

* openssl库是否安装

检查是否安装：yum list installed | grep openssl

执行安装：yum install openssl openssl-devel -y

* pcre库是否安装

检查是否安装：yum list installed | grep pcre

执行安装：yum install pcre pcre-devel -y

* zlib库是否安装

检查是否安装：yum list installed | grep zlib

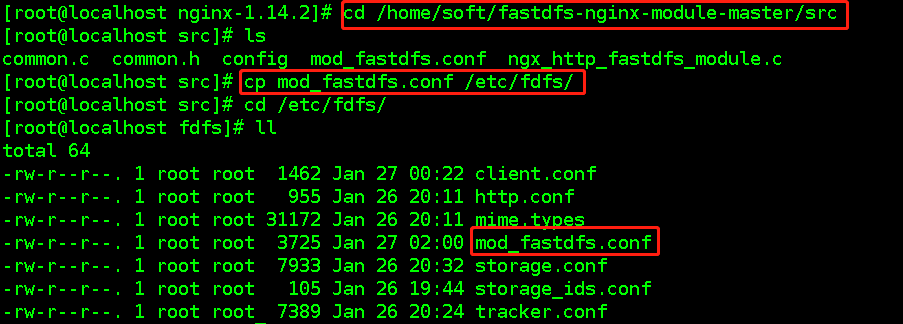
执行安装：yum install zlib zlib-devel -y

yum install gcc openssl openssl-devel pcre pcre-devel zlib zlib-devel –y

## FastDFS的Nginx访问配置

### 将/home/soft/fastdfs-nginx-module-master/src(自己实际存放Nginx扩展模块的目录)目录下的mod\_fastdfs.conf文件拷贝到 /etc/fdfs/目录下，这样才能正常启动Nginx

cp /home/soft/fastdfs-nginx-module-master/src/mod\_fastdfs.conf /etc/fdfs/



### 修改mod\_fastdfs.conf配置文件

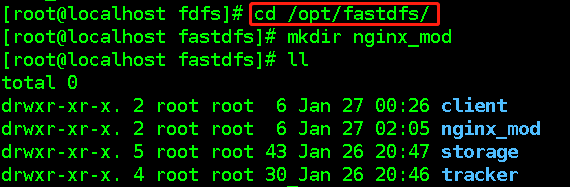
base\_path=/opt/fastdfs/nginx\_mod

tracker\_server=192.168.235.128:22122

url\_have\_group\_name = true

store\_path0=/opt/fastdfs/storage/files

### 在/opt/fastdfs/目录下创建nginx\_mod目录



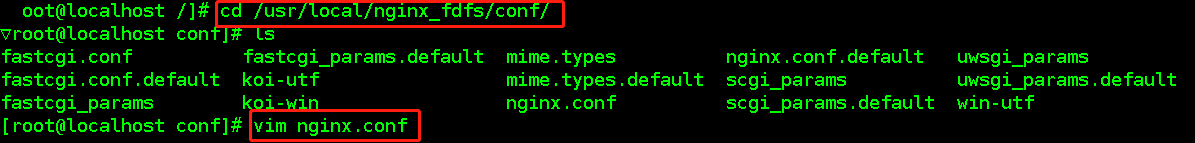
### 配置Nginx的配置文件

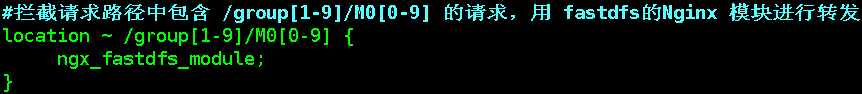
#拦截请求路径中包含 /group[1-9]/M0[0-9] 的请求，用 fastdfs的Nginx 模块进行转发

location ~ /group[1-9]/M0[0-9] {

ngx\_fastdfs\_module;

}

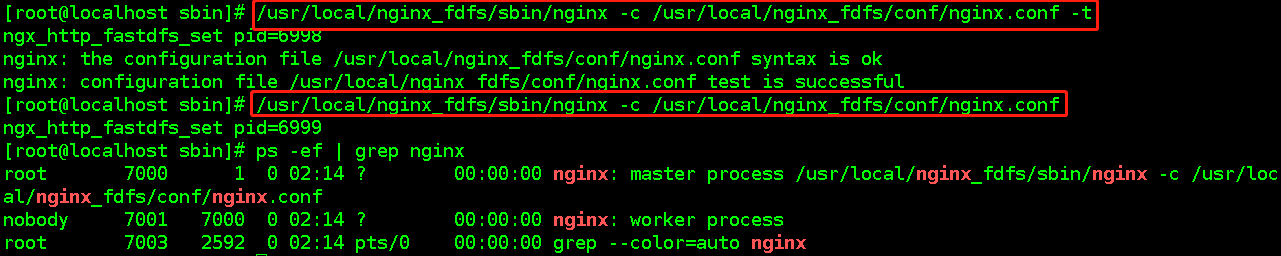




ngx\_fastdfs\_module; #这个指令不是Nginx本身提供的，是扩展模块提供的，根据这个指令找到FastDFS提供的Nginx模块配置文件，然后找到Tracker，最终找到Stroager

## FastDFS的Nginx访问启动与测试

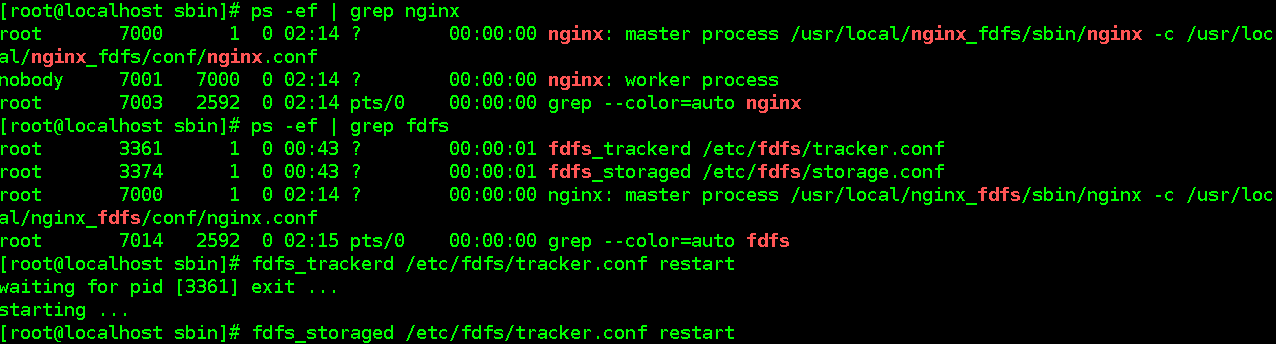
### 启动带有Fastdfs模块的Nginx



### 重启或启动FastDFS服务进程

fdfs\_trackerd /etc/fdfs/tracker.conf restart

fdfs\_storaged /etc/fdfs/storage.conf restart



### 上传一个文件进行测试验证

fdfs\_test /etc/fdfs/client.conf upload /root/aa.txt



### 在浏览器访问上传的文件

当遇到400错误，检查配置/etc/fdfs/mod\_fastdfs.conf url\_have\_group\_name=true

该配置表示访问路径中是否需要带有group1，改为true表示路径中需要有group1

### 扩展

**模拟大型网站用户头像的处理方式，上传一张图片，然后自己写一个html页面，src指向上传的图片**

# FastDFS在Java项目中开发示例

## 在实际项目开发中，FastDFS提供的主要功能

* + upload：上传文件
  + download：下载文件
  + delete：删除文件

## FastDFS文件系统的Java客户端

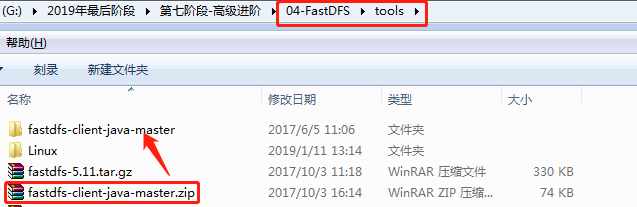
FastDFS文件系统Java客户端是指采用Java语言编写的一套程序，专门用来访问fastDFS文件系统，其实就是一个jar包。

**注意：大家如果能连上 mvnrepository上搜索到的用友云提供的fastdfs-client，那大家就下载那个jar包使用，如果连不上，这个jar包需要我们自己来打**

### 下载官方的源代码

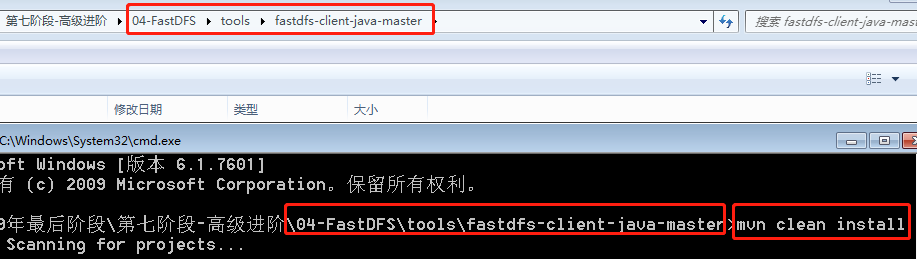
从<https://codeload.github.com/happyfish100/fastdfs-client-java/zip/master> 上下载FastDFS源代码到本地

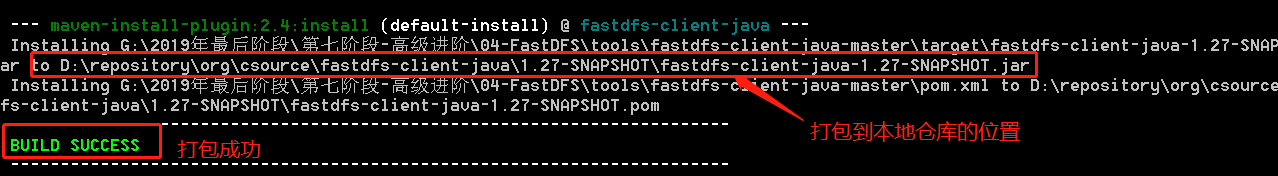
### 解压



### 采用maven命令编译成jar安装到本地maven库

mvn clean install





### 在Java程序中使用它提供的API来访问FastDFS文件系统

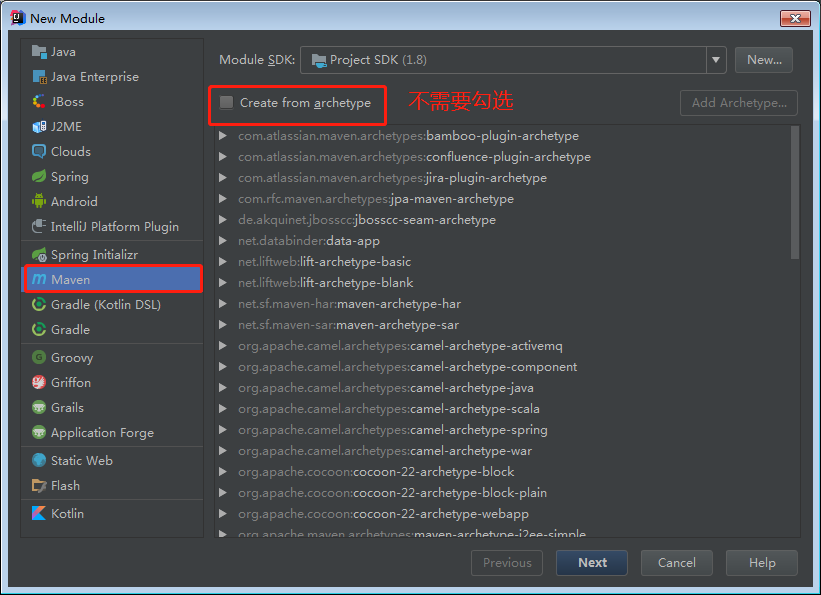
## 文件上传功能的实现

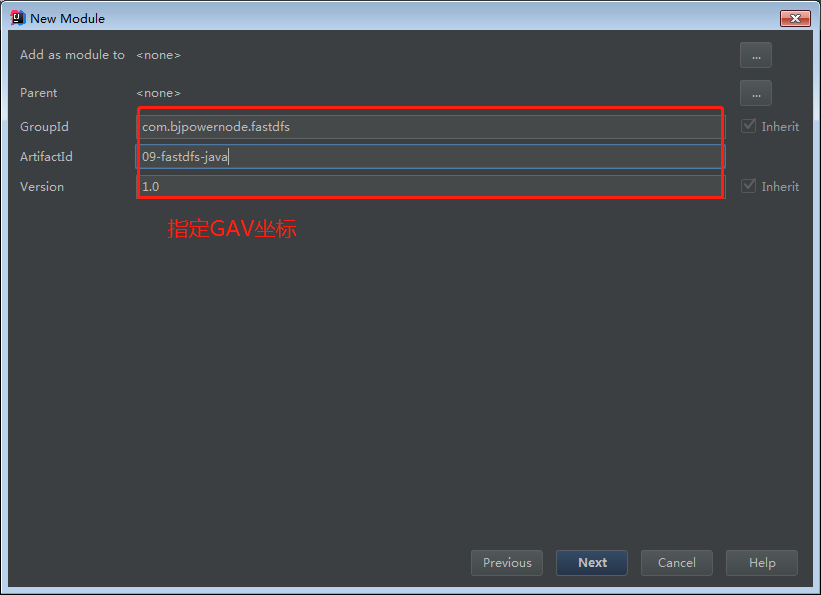
### **需求**

使用Java客户端，编程操作fastDFS分布式文件系统，上传本地文件到FastDFS服务器上

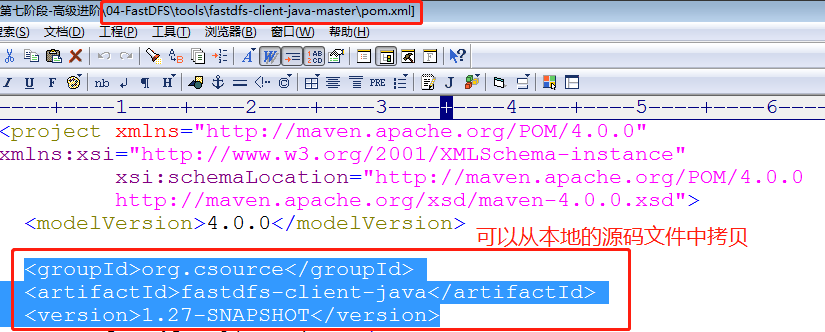
### 实现步骤

#### 使用IDEA创建普通的maven项目，不需要使用



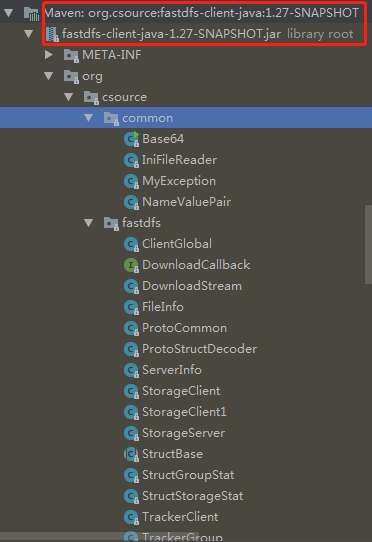


#### 在pom.xml文件中添加我们打包好的FastDFS本地仓库的jar包（FastDFS的java客户端依赖）

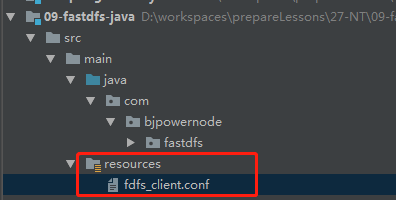
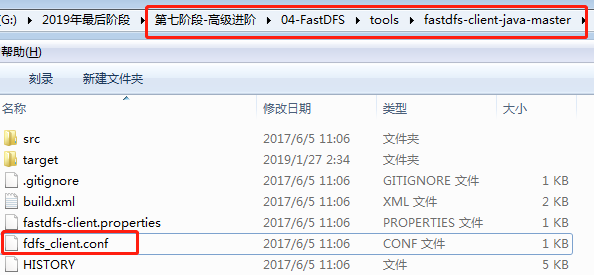


<!--加入FastDFS的java客户端依赖-->  
<dependencies>  
 <dependency>  
 <groupId>org.csource</groupId>  
 <artifactId>fastdfs-client-java</artifactId>  
 <version>1.27-SNAPSHOT</version>  
 </dependency>  
</dependencies>

可以在这查看jar包里面的内容



#### 拷贝源代码包中的fdfs\_client.conf文件到resources目录下，在里面主要配置tracker地址



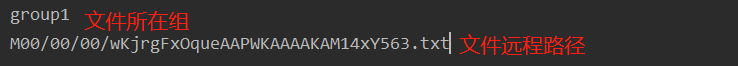
tracker\_server = 192.168.235.128:22122

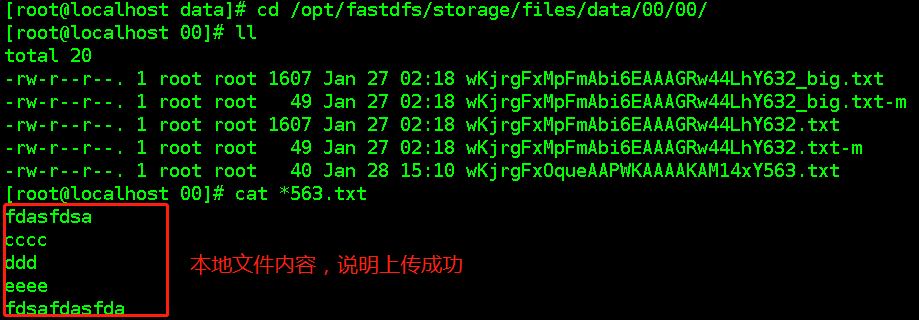
#### 编写代码，进行上传测试

在com.bjpowernode.fastdfs包下创建FastDFS类，在其中编写上传代码

package com.bjpowernode.fastdfs;  
import org.csource.common.MyException;  
import org.csource.fastdfs.\*;  
import java.io.IOException;public class FastDFS {  
 public static void main(String[] args) {  
 *fileUpload*();  
 }  
 //上传文件的方法  
 public static void fileUpload(){  
 TrackerServer trackerServer = null;  
 StorageServer storageServer = null;  
 try {  
 //1.加载配置文件，默认去classpath下加载  
 ClientGlobal.*init*("fdfs\_client.conf");  
 //2.创建TrackerClient对象  
 TrackerClient trackerClient = new TrackerClient();  
 //3.创建TrackerServer对象  
 trackerServer = trackerClient.getConnection();  
 //4.创建StorageServler对象  
 storageServer = trackerClient.getStoreStorage(trackerServer);  
 //5.创建StorageClient对象，这个对象完成对文件的操作  
 StorageClient storageClient = new StorageClient(trackerServer,storageServer);  
 //6.上传文件 第一个参数：本地文件路径 第二个参数：上传文件的后缀 第三个参数：文件信息  
 String [] uploadArray = storageClient.upload\_file("D:/aa.txt","txt",null);  
 for (String str:uploadArray) {  
 System.*out*.println(str);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (MyException e) {  
 e.printStackTrace();  
 } finally {  
 if(storageServer != null){  
 try {  
 storageServer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 if(trackerServer != null){  
 try {  
 trackerServer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
}

#### 运行程序，在Linux上，FastDFS存储目录下查看上传文件内容





## 将文件上传功能中的公共代码进行封装

#### 需求

因为使用FastDFS进行文件操作代码大多都是通用的，所以我们这里在FastDFS类中将通用的功能进行封装，并提供上传、下载、删除文件的方法

**注意：这里只是简单的封装，如果多线程会有问题**

#### 实现步骤

##### 抽取获取StorageClient的方法

public static StorageClient getStorageClient() throws IOException, MyException {  
 //1.加载配置文件，默认去classpath下加载  
 ClientGlobal.*init*("fdfs\_client.conf");  
 //2.创建TrackerClient对象  
 TrackerClient trackerClient = new TrackerClient();  
 //3.创建TrackerServer对象  
 *trackerServer* = trackerClient.getConnection();  
 //4.创建StorageServler对象  
 *storageServer* = trackerClient.getStoreStorage(*trackerServer*);  
 //5.创建StorageClient对象，这个对象完成对文件的操作  
 StorageClient storageClient = new StorageClient(*trackerServer*,*storageServer*);  
 return storageClient;  
}

##### 定义两个全局变量

private static TrackerServer *trackerServer* = null;  
private static StorageServer *storageServer* = null;

##### 抽取关闭资源的方法

public static void closeFastDFS() {  
 if (*storageServer* != null) {  
 try {  
 *storageServer*.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 if (*trackerServer* != null) {  
 try {  
 *trackerServer*.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

##### 改造文件上传的方法

public static void fileUpload(){  
 try {  
 //1. 获取StorageClient对象  
 StorageClient storageClient = *getStorageClient*();  
 //2.上传文件 第一个参数：本地文件路径 第二个参数：上传文件的后缀 第三个参数：文件信息  
 String [] uploadArray = storageClient.upload\_file("D:/aa.txt","txt",null);  
 for (String str:uploadArray) {  
 System.*out*.println(str);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (MyException e) {  
 e.printStackTrace();  
 } finally {  
 *closeFastDFS*();  
 }  
}

##### 下载文件的方法

//下载文件的方法  
public static void fileDownload(){  
 try {  
 //1. 获取StorageClient对象  
 StorageClient storageClient = *getStorageClient*();  
 //2.下载文件 返回0表示成功，其它均表示失败  
 int num = storageClient.download\_file("group1",  
 "M00/00/00/wKjrgFxOqueAAPWKAAAAKAM14xY563.txt","E:/bb.txt");  
 System.*out*.println(num);  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (MyException e) {  
 e.printStackTrace();  
 } finally {  
 *closeFastDFS*();  
 }  
}

##### 删除文件的方法

//删除文件的方法  
public static void fileDelete(){  
 try {  
 //1. 获取StorageClient对象  
 StorageClient storageClient = *getStorageClient*();  
 //2.删除文件 返回0表示成功，其它均表示失败  
 int num = storageClient.delete\_file("group1",  
 "M00/00/00/wKjrgFxOqueAAPWKAAAAKAM14xY563.txt");  
 System.*out*.println(num);  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (MyException e) {  
 e.printStackTrace();  
 } finally {  
 *closeFastDFS*();  
 }  
}

##### 主方法调用不同的方法进行测试

# FastDFS在web项目中的应用

## 需求

对P2P项目合同进行管理，在WEB项目中实现对文件的上传下载和删除操作

## 名词解释

* 有一些债权：投资人有该债务的权利

通常隐含的意思就是：一笔借款常被称为一个债权

* 一个债权会有一个合同
* 合同是pdf文件
* 债权是债务的对应词，但是在P2P项目中，我们管理的债权，以及合同一般指的是借款人的信息，所以在我们下面创建的creditor\_info表中存的是借款人信息

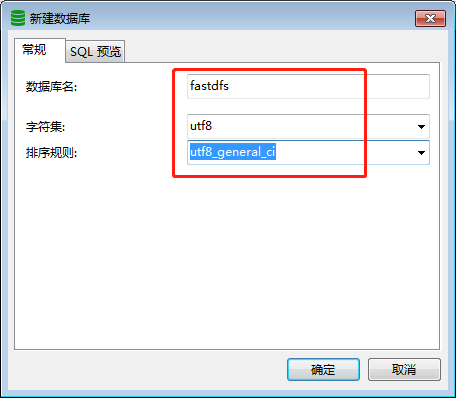
## 目标

* + 实现对pdf文件上传、下载、删除
  + 熟练一下Springboot+thymeleaf

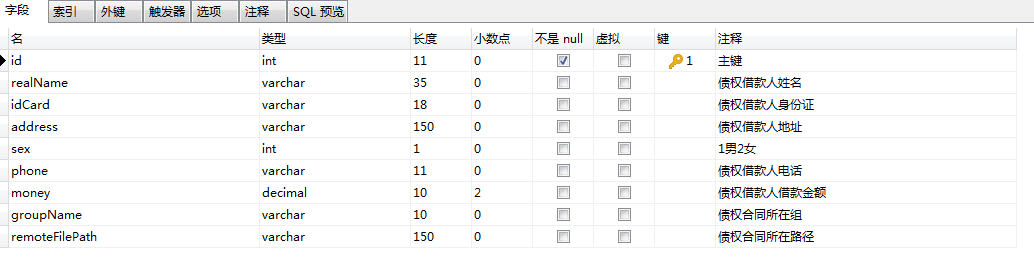
## 案例实现步骤

### 数据库环境搭建

#### 创建数据库fastdfs



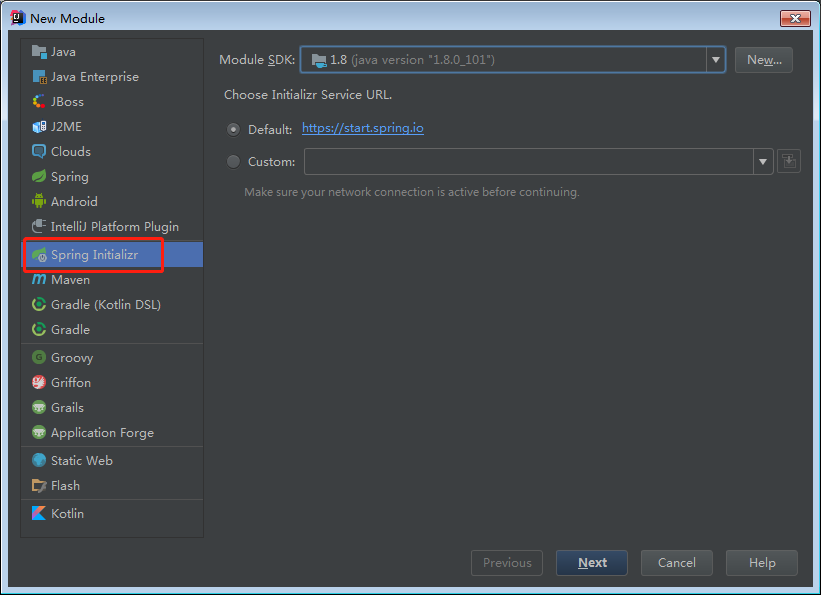
#### 在该库下创建creditor\_info表

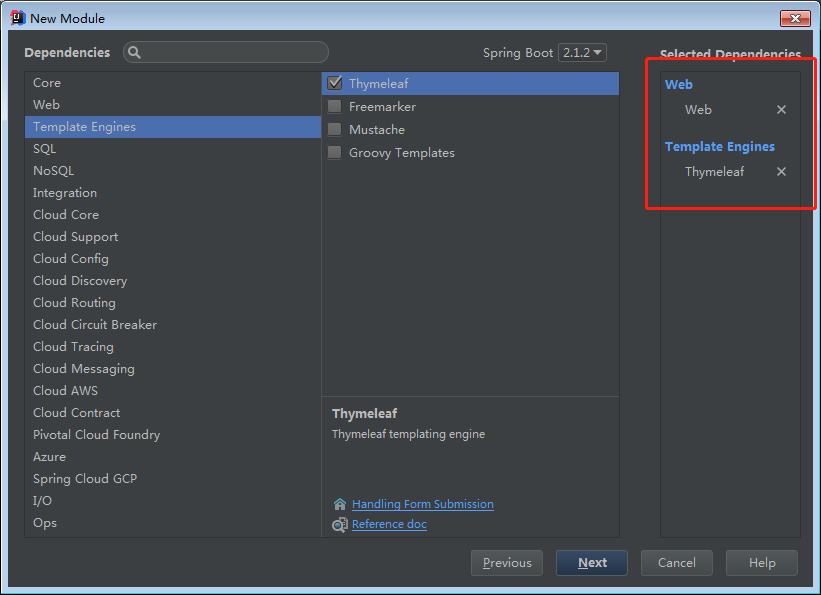


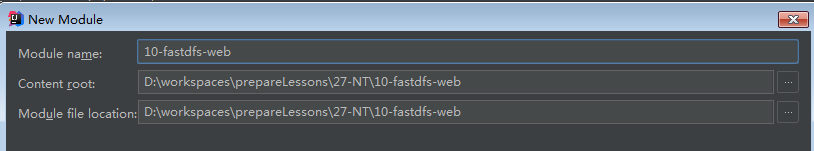
CREATE TABLE `creditor\_info` (  
`id` int(11) NOT NULL AUTO\_INCREMENT COMMENT '主键',  
`realName` varchar(35) DEFAULT NULL COMMENT '债权借款人姓名',  
`idCard` varchar(18) DEFAULT NULL COMMENT '债权借款人身份证',  
`address` varchar(150) DEFAULT NULL COMMENT '债权借款人地址',  
`sex` int(1) DEFAULT NULL COMMENT '1男2女',  
`phone` varchar(11) DEFAULT NULL COMMENT '债权借款人电话',  
`money` decimal(10,2) DEFAULT NULL COMMENT '债权借款人借款金额',  
`groupName` varchar(10) DEFAULT NULL COMMENT '债权合同所在组',  
`remoteFilePath` varchar(150) DEFAULT NULL COMMENT '债权合同所在路径',  
PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8

### 开发环境搭建

#### 创建SpringBoot项目10-fastdfs-web，添加Web和Thymeleaf依赖







#### 在pom.xml文件中添加Mybatis依赖及MySQL依赖

<!-- 加载mybatis整合springboot -->  
<dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <!--在springboot的父工程中没有指定版本，我们需要手动指定-->  
 <version>1.3.2</version>  
</dependency>  
<!-- MySQL的jdbc驱动包 -->  
<dependency>  
 <groupId>mysql</groupId>  
 <!--在springboot的父工程中指定了版本，我们就不需要手动指定了-->  
 <artifactId>mysql-connector-java</artifactId>  
</dependency>

#### 在pom.xml文件中添加resources，指定编译的位置

<resources>  
 <resource>  
 <directory>src/main/java</directory>  
 <includes>  
 <include>\*\*/\*.xml</include>  
 </includes>  
 </resource>  
 <resource>  
 <directory>src/main/resources</directory>  
 <includes>  
 <include>\*\*/\*.\*</include>  
 </includes>  
 </resource>  
 <!--如果存在jsp，需要指定jsp文件编译的位置-->  
</resources>

#### 在SpringBoot主配置文件application.properties中添加数据库配置信息

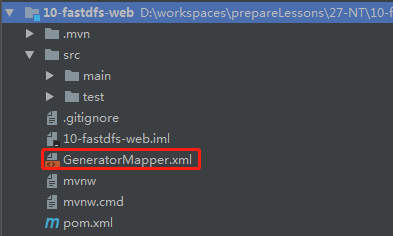
#数据库的连接配置信息  
spring.datasource.username=root  
spring.datasource.password=123456  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://192.168.235.128:3306/fastdfs?useUnicode=true&characterEncoding=utf8&useSSL=false

#### 使用Mybatis反向工程，生成实体类及mapper映射（参照SpringBoot附录教程）

##### 在pom.xml文件中添加反向工程插件

<!--mybatis代码自动生成插件-->  
<plugin>  
 <groupId>org.mybatis.generator</groupId>  
 <artifactId>mybatis-generator-maven-plugin</artifactId>  
 <version>1.3.7</version>  
 <configuration>  
 <!--配置文件的位置-->  
 <configurationFile>GeneratorMapper.xml</configurationFile>  
 <verbose>true</verbose>  
 <overwrite>true</overwrite>  
 </configuration>  
</plugin>

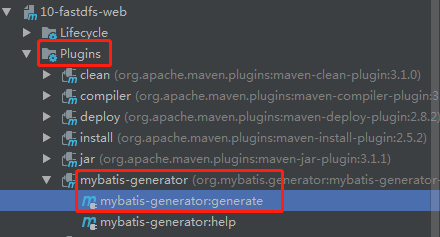
##### 从03-springboot-web中复制GeneratorMapper.xml到当前项目下



##### 修改GeneratorMapper.xml配置文件内容

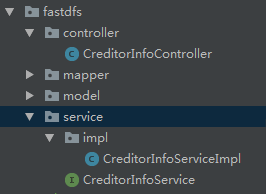
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE generatorConfiguration  
 PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  
 "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  
<generatorConfiguration>  
 <!-- 指定连接数据库的JDBC驱动包所在位置，指定到你本机的完整路径 -->  
 <classPathEntry location="D:/repository/mysql/mysql-connector-java/8.0.13/mysql-connector-java-8.0.13.jar"/>  
 <!-- 配置table表信息内容体，targetRuntime指定采用MyBatis3的版本 -->  
 <context id="tables" targetRuntime="MyBatis3">  
 <!-- 抑制生成注释，由于生成的注释都是英文的，可以不让它生成 -->  
 <commentGenerator>  
 <property name="suppressAllComments" value="true" />  
 </commentGenerator>  
 <!-- 配置数据库连接信息 注意：使用高版本的驱动 url后面应该加属性nullCatalogMeansCurrent=true，否则生成有问题 -->  
 <jdbcConnection driverClass="com.mysql.cj.jdbc.Driver"  
 connectionURL="jdbc:mysql://192.168.235.128:3306/fastdfs?nullCatalogMeansCurrent=true"  
 userId="root"  
 password="123456">  
 </jdbcConnection>  
 <!-- 生成model类，targetPackage指定model类的包名， targetProject指定生成的model放在eclipse的哪个工程下面-->  
 <javaModelGenerator targetPackage="com.bjpowernode.fastdfs.model" targetProject="src/main/java">  
 <property name="enableSubPackages" value="false" />  
 <property name="trimStrings" value="false" />  
 </javaModelGenerator>  
 <!-- 生成MyBatis的Mapper.xml文件，targetPackage指定mapper.xml文件的包名， targetProject指定生成的mapper.xml放在eclipse的哪个工程下面 -->  
 <sqlMapGenerator targetPackage="com.bjpowernode.fastdfs.mapper" targetProject="src/main/java">  
 <property name="enableSubPackages" value="false" />  
 </sqlMapGenerator>  
 <!-- 生成MyBatis的Mapper接口类文件,targetPackage指定Mapper接口类的包名， targetProject指定生成的Mapper接口放在eclipse的哪个工程下面 -->  
 <javaClientGenerator type="XMLMAPPER" targetPackage="com.bjpowernode.fastdfs.mapper" targetProject="src/main/java">  
 <property name="enableSubPackages" value="false" />  
 </javaClientGenerator>  
 <!-- 数据库表名及对应的Java模型类名 -->  
 <table tableName="creditor\_info"  
 domainObjectName="CreditorInfo"  
 enableCountByExample="false"  
 enableUpdateByExample="false"  
 enableDeleteByExample="false"  
 enableSelectByExample="false"  
 selectByExampleQueryId="false"/>  
 </context>  
</generatorConfiguration>

##### 双击生成



#### 创建相关的包和类

* 在com.bjpowernode.fast包下创建controller ，service 包，及其子包impl
* 创建CreditorInfoController类
* 创建CreditorInfoService接口
* 创建CreditorInfoServiceImpl实现类



### 功能实现-展示所有债权信息

#### 在CreditorInfoController类中创建index方法，将CreditorInfoService注入到controller中

@Controller  
public class CreditorInfoController {  
 @Autowired  
 private CreditorInfoService creditorInfoService;  
  
 @GetMapping("/fastdfs/index")  
 public String index(Model model){  
 List<CreditorInfo> creditorInfoList = creditorInfoService.getAllCreditorInfo();  
 model.addAttribute("creditorInfoList",creditorInfoList);  
 //模板页面，不是jsp  
 return "index";  
 }  
}

#### 在CreditorInfoService中提供getAllCreditorInfo方法

public interface CreditorInfoService {  
 */\*\*  
 \* 获取所有债权信息  
 \** ***@return*** *\*/* List<CreditorInfo> getAllCreditorInfo();  
}

#### 在CreditorInfoServiceImpl中对getAllCreditorInfo方法进行实现

@Service  
public class CreditorInfoServiceImpl implements CreditorInfoService {  
 @Autowired  
 private CreditorInfoMapper creditorInfoMapper;  
 @Override  
 public List<CreditorInfo> getAllCreditorInfo() {  
 return creditorInfoMapper.selectAllCreditorInfo();  
 }  
}

#### 因为是SpringBoot项目，所以需要在Mapper接口上加一个Mapper注解

@Mapper  
public interface CreditorInfoMapper {

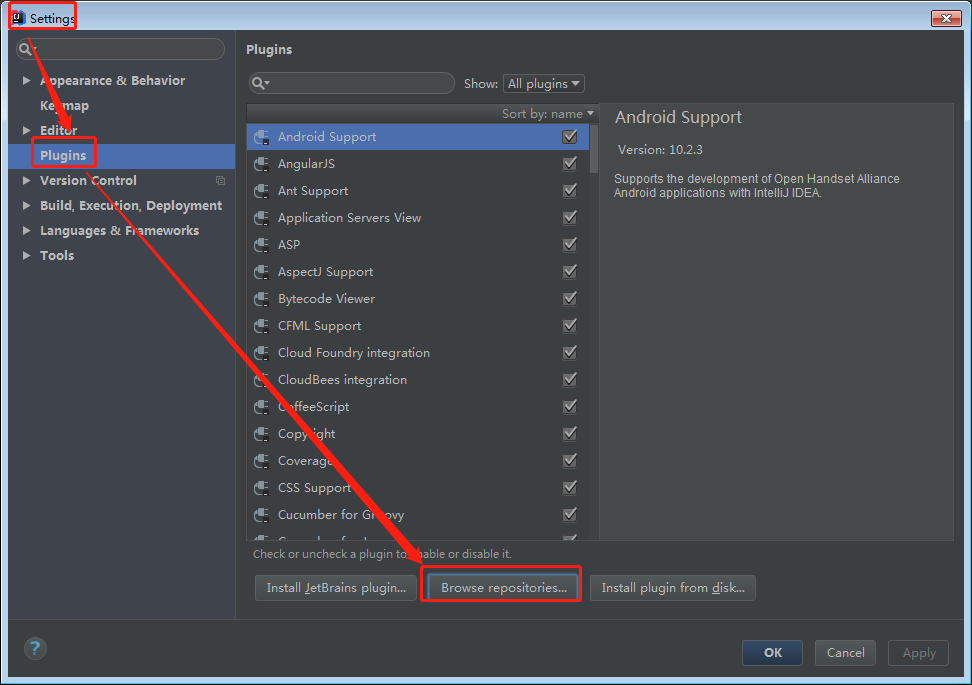
#### 在CreditorInfoMapper类中添加selectAllCreditorInfo方法

List<CreditorInfo> selectAllCreditorInfo();

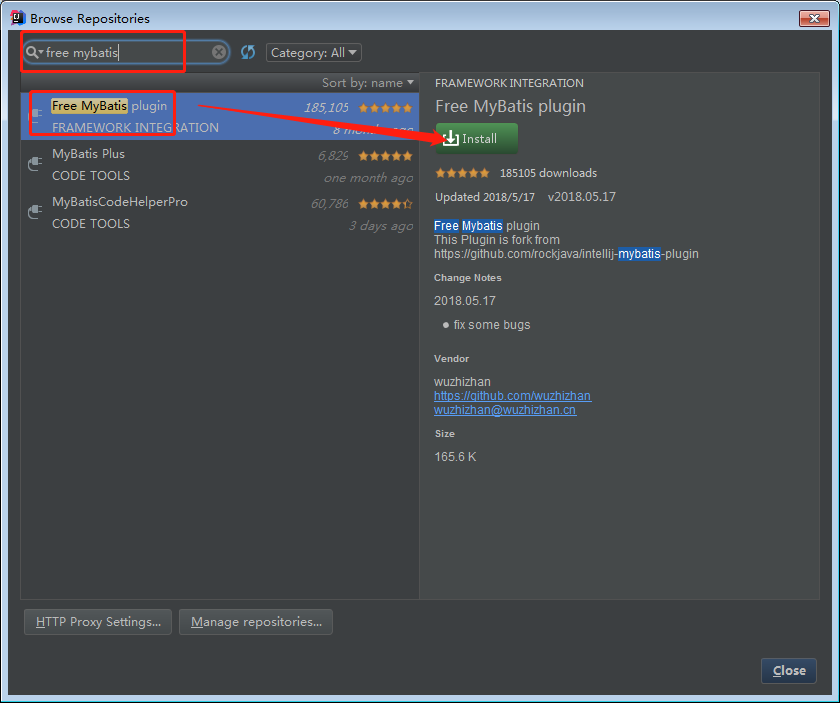
#### 在IDEA中安装free Mybatis插件

**该插件可以通过点击Mapper接口中的方法，进入到.xml文件**

##### Settings🡪plugins🡪Browse repositories



##### 在插件库中搜索，free mybatis安装



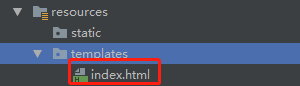
##### 插件安装完毕，需要重启IDEA

#### 在CreditorInfoMapper.xml文件中添加SQL语句

<select id="selectAllCreditorInfo" resultMap="BaseResultMap">  
 select  
 <include refid="Base\_Column\_List"/>  
 from creditor\_info  
</select>

#### 展示页面的设计

##### 在项目的templates目录下创建index.html



##### 百度搜索bootstrap表格，挑选自己喜欢风格的表格，将代码拷贝到index.html中

我这里使用的是<http://www.runoob.com/try/try.php?filename=bootstrap3-table-striped>表格进行改写

##### 在html标签上加上Thymeleaf的命名空间

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.thymeleaf.org">

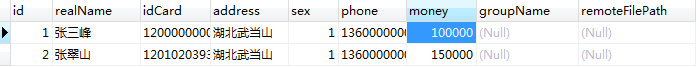
##### 修改index.html内容

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="utf-8">  
 <title>债权合同管理</title>  
 <link rel="stylesheet" th:href="@{/css/bootstrap-3.3.7.min.css}">  
 <script th:src="@{/js/jquery-2.1.1.min.js}"></script>  
 <script th:src="@{/js/bootstrap-3.3.7.min.js}"></script>  
</head>  
<body>  
<table class="table table-striped">  
 <caption>债权合同信息列表</caption>  
 <thead>  
 <tr>  
 <th>序号</th>  
 <th>债权借款人姓名</th>  
 <th>债权借款人身份证</th>  
 <th>债权借款人住址</th>  
 <th>债权借款人手机号</th>  
 <th>债权借款人性别</th>  
 <th>债权借款人借款金额</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr th:each="creditorInfo:${creditorInfoList}">  
 <td th:text="${creditorInfoStat.count}"></td>  
 <td th:text="${creditorInfo.realname}"></td>  
 <td th:text="${creditorInfo.idcard}"></td>

<td th:text="${creditorInfo.address}"></td>  
 <td th:text="${creditorInfo.phone}"></td>  
 <td th:text="${creditorInfo.sex == 1 ?'男':'女'}"></td>  
 <td th:text="${creditorInfo.money}"></td>  
 </tr>  
 </tbody>  
</table>  
</body>  
</html>

**注意：我们从网络上拷贝过来的内容css，js等是联网获取的，我们这里可以从04-FastDFS\resources获取，并放在项目的static的相关目录下，在页面上引用**

#### 向数据库中加几条数据



#### 启动项目，访问<http://localhost:8080/fastdfs/index> 查看效果



#### 调整页面样式

<body style="margin: 50px">

### 功能实现-为某一个债权合同上传文件

#### 在index.html中添加操作列

<th>合同管理</th>

<td>  
 <!--为哪个合同上传，需要将合同的id传递过去-->  
 <a th:href="@{'/fastdfs/toUpload?id=' + ${creditorInfo.id}}">上传</a>  
 下载  
 删除  
</td>

#### 在CreditorController中添加跳转到上传页面的方法

@GetMapping("/fastdfs/toUpload")  
public String toUpload(Model model, @RequestParam("id") Integer id){  
 model.addAttribute("id",id);  
 return "upload";  
}

#### 在templates下创建upload.html页面

在网上搜索bootstrap表单，并对其内容进行修改，我这里使用的是

<http://www.runoob.com/try/try2.php?filename=bootstrap3-form-inline>

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="utf-8">  
 <title>债权合同上传</title>  
 <link rel="stylesheet" th:href="@{/css/bootstrap-3.3.7.min.css}">  
 <script th:src="@{/js/jquery-2.1.1.min.js}"></script>  
 <script th:src="@{/js/bootstrap-3.3.7.min.js}"></script>  
</head>  
<body>  
 <form th:action="@{/fastdfs/upload}" class="form-inline" role="form" method="post" enctype="multipart/form-data">  
 <div class="form-group">  
 <label class="sr-only" for="fileName">文件输入</label>  
 <input type="file" id="fileName" name="fileName">  
 </div>  
 <input type="hidden" name="id" th:value="${id}">  
 <button type="submit" class="btn btn-default">提交</button>  
 </form>  
</body>  
</html>

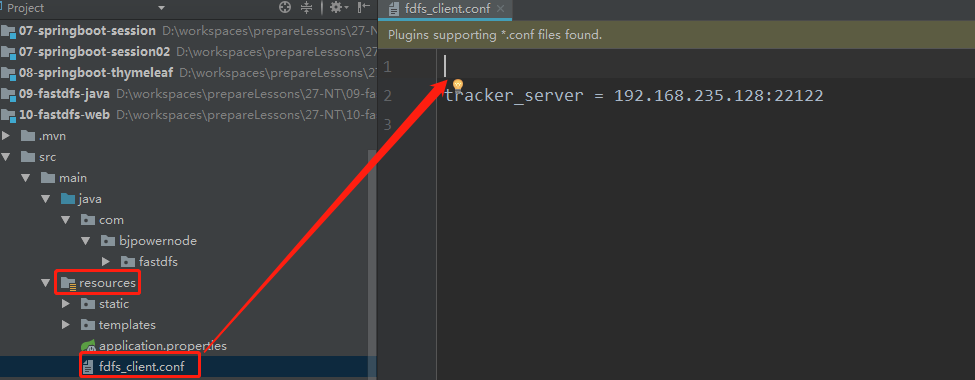
**注意**

* + - 文件上传必须是post请求
    - enctype必须为multipart/form-data
    - 合同的id通过隐藏域传递

#### 在pom.xml文件中加入FastDFS客户端的jar包依赖

<!--加入FastDFS的java客户端依赖-->  
<dependency>  
 <groupId>org.csource</groupId>  
 <artifactId>fastdfs-client-java</artifactId>  
 <version>1.27-SNAPSHOT</version>  
</dependency>

#### 将FastDFS客户端的配置文件fast\_client.conf拷贝到resources目录下



#### 将原来我们封装的FastDFS类拷贝到fastdfs包下，修改其中的file\_upload方法，定义一些参数

去掉主方法，新的fileUpload写法如下：

//上传文件的方法  
public static String[] fileUpload(byte[] fileBytes,String fileExt){  
 String [] uploadArray = null;  
 try {  
 //1. 获取StorageClient对象  
 StorageClient storageClient = *getStorageClient*();  
 //2.上传文件 第一个参数：本地文件路径 第二个参数：上传文件的后缀 第三个参数：文件信息  
 uploadArray = storageClient.upload\_file(fileBytes,fileExt,null);  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (MyException e) {  
 e.printStackTrace();  
 } finally {  
 *closeFastDFS*();  
 }  
 return uploadArray;  
}

#### 在CreditorController中添加处理上传文件的方法

@PostMapping("/fastdfs/upload")  
public @ResponseBody String upload(@RequestParam("id") Integer id, @RequestParam("fileName") MultipartFile file){  
 //原来文件上传是将文件写到本地或者远程服务器的某个目录下  
 //现在的文件上传是将文件上传到fastdfs文件服务器上  
 //1表示上传失败 0表示成功  
 int result = 1;  
 //abc.txt -->txt  
 String fileExt = file.getOriginalFilename().substring(file.getOriginalFilename().indexOf(".") + 1);  
 try {  
 String[] uploadArray = FastDFS.*fileUpload*(file.getBytes(),fileExt);  
 if(uploadArray != null && uploadArray.length ==2){  
 //文件上传到fastDFS成功 ,将合同文件路径更新到债权记录中  
 CreditorInfo creditorInfo = new CreditorInfo();  
 creditorInfo.setId(id);  
 creditorInfo.setGroupname(uploadArray[0]);  
 creditorInfo.setRemotefilepath(uploadArray[1]);  
 int updateRow = creditorService.updateCreditorInfo(creditorInfo);  
 //数据库更新成功  
 if(updateRow > 0){  
 result = 0;  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return "<script>window.parent.uploadOK('"+result+"')</script>";  
}

#### 在CreditorInfoService中添加updateCreditorInfo方法

*/\*\*  
 \* 更新债权信息  
 \** ***@param*** *creditorInfo  
 \** ***@return*** *\*/*int updateCreditorInfo(CreditorInfo creditorInfo);

#### 在CreditorInfoServiceImpl中添加updateCreditorInfo方法实现

@Override  
public int updateCreditorInfo(CreditorInfo creditorInfo) {  
 return creditorInfoMapper.updateByPrimaryKeySelective(creditorInfo);  
}

#### 在upload.html做一个类似ajax的页面不刷新效果

* 在upload.html页面中加一个iframe
* upload.html页面中的form中的target设置为iframe的name
* 在iframe的父页面中，写一个函数，处理上传结果

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="utf-8">  
 <title>债权合同上传</title>  
 <link rel="stylesheet" th:href="@{/css/bootstrap-3.3.7.min.css}">  
 <script th:src="@{/js/jquery-2.1.1.min.js}"></script>  
 <script th:src="@{/js/bootstrap-3.3.7.min.js}"></script>  
</head>  
<body>  
 <form th:action="@{/fastdfs/upload}" class="form-inline" role="form" method="post" target="uploadFrame" enctype="multipart/form-data">  
 <div class="form-group">  
 <label class="sr-only" for="fileName">文件输入</label>  
 <input type="file" id="fileName" name="fileName">  
 </div>  
 <input type="hidden" id="id" name="id" th:value="${id}">  
 <button type="submit" class="btn btn-default">提交</button>  
 </form>  
 <iframe name="uploadFrame" style="display: none;"></iframe>  
 <script type="text/javascript" th:inline="javascript">  
 **function** uploadOK(result){  
 **if**(result == 0){  
 //文件上传成功  
 alert("文件上传成功");  
 **var** contextPath = [[${#request.getContextPath()}]];  
 window.location.href = contextPath + "/fastdfs/index";  
 }**else**{  
 alert("文件上传失败");  
 }  
 }  
 </script>  
</body>  
</html>

#### 如果上传文件超出了1M，需要在application.properties中配置SpringBoot上传文件的最大限制

#SpringBoot上传文件的最大限制  
spring.servlet.multipart.max-file-size=10MB

**注意：如果提示找不到tracker\_server,看看是否编译到target中**

### 功能实现-下载某一个债权合同

#### 修改index.html页面，下载加连接，并做判断

<td>  
 <span th:if="${creditorInfo.getGroupname() ne null && creditorInfo.remotefilepath ne null}">  
 <a th:href="@{'/fastdfs/download?id=' + ${creditorInfo.id}}">下载</a>  
 删除  
 </span>  
 <span th:unless="${creditorInfo.getGroupname() ne null && creditorInfo.remotefilepath ne null}">  
 <!--为哪个合同上传，需要将合同的id传递过去-->  
 <a th:href="@{'/fastdfs/toUpload?id=' + ${creditorInfo.id}}">上传</a>  
 </span>  
</td>

#### 在CreditorController中，完成下载的请求

* ResponseEntity通常用于返回文件流
* @ResponseBody可以直接返回Json结果，
* @ResponseEntity不仅可以返回json结果，还可以定义返回的HttpHeaders和HttpStatus
* ResponseEntity的优先级高于@ResponseBody。在不是ResponseEntity的情况下才去检查有没有@ResponseBody注解。如果响应类型是ResponseEntity可以不写@ResponseBody注解，写了也没有关系。

@GetMapping("/fastdfs/download")  
public ResponseEntity<byte[]> download(@RequestParam("id") Integer id){  
 //根据债权id获取 债权对象  
 CreditorInfo creditorInfo = creditorInfoService.getCreditorInfoById(id);  
 String extName = creditorInfo.getRemotefilepath().substring(creditorInfo.getRemotefilepath().indexOf("."));  
 byte [] fileBytes = FastDFS.*fileDownload*(creditorInfo.getGroupname(),creditorInfo.getRemotefilepath());  
  
  
 HttpHeaders httpHeaders = new HttpHeaders();  
 httpHeaders.setContentType(MediaType.*APPLICATION\_OCTET\_STREAM*);//流类型  
 httpHeaders.setContentDispositionFormData("attachment",System.*currentTimeMillis*() + extName);  
  
 ResponseEntity<byte[]> responseEntity = new ResponseEntity<byte[]>(fileBytes,httpHeaders, HttpStatus.*OK*);  
 return responseEntity;  
}

#### 在CreditorService接口中添加getCreditorInfoById的方法

*/\*\*  
 \* 根据合同id获取债权信息  
 \** ***@param*** *id  
 \** ***@return*** *\*/*CreditorInfo getCreditorInfoById(Integer id);

#### 在CreditorServiceImpl中添加getCreditorInfoById方法的实现

@Override  
public CreditorInfo getCreditorInfoById(Integer id) {  
 return creditorInfoMapper.selectByPrimaryKey(id);  
}

#### 修改FastDFS类中fileDown方法的实现，传递参数

//下载文件的方法  
public static byte[] fileDownload(String group,String remoteFile){  
 byte[] fileBytes = null;  
 try {  
 //1. 获取StorageClient对象  
 StorageClient storageClient = *getStorageClient*();  
 //2.下载文件 返回0表示成功，其它均表示失败  
 fileBytes = storageClient.download\_file(group,remoteFile);  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (MyException e) {  
 e.printStackTrace();  
 } finally {  
 *closeFastDFS*();  
 }  
 return fileBytes;  
}

#### 浏览器访问下载测试效果

### 功能实现-删除某一个债权合同，使用ajax实现异步删除

#### 在index.html页面为删除加超链接

<span th:if="${creditorInfo.getGroupname() ne null && creditorInfo.remotefilepath ne null}">  
 <a th:href="@{'/fastdfs/download?id=' + ${creditorInfo.id}}">下载</a>  
 <a th:href="@{'javascript:deleteFile('+ ${creditorInfo.id} +')'}">删除</a>  
</span>

#### 在index.html页面提供js方法，并发送ajax请求，对响应结果进行处理

<script type="text/javascript" th:inline="javascript">  
 **function** deleteFile(id){  
 //获取项目的上下文根  
 **var** contextPath = [[${#request.getContextPath()}]];  
 $.ajax({  
 url:contextPath +"/fastdfs/fileDelete",  
 type:"post",  
 data:{  
 "id":id  
 },  
 success:**function**(responseMsg){  
 **if**(responseMsg==0){  
 alert("删除成功");

window.location.reload();  
 }**else**{  
 alert("删除失败");  
 }  
 }  
 });  
 }  
</script>

#### 在CreditorController中处理删除请求

**注意：删除FastDFS和清除数据库，所以我们将这些业务都放在service中进行事务的处理**

@RequestMapping("/fastdfs/fileDelete")  
public @ResponseBody String fileDelete(@RequestParam("id") Integer id){  
 int result = 1;  
 try {  
 result = creditorService.deleteContract(id);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return String.*valueOf*(result);  
}

#### 在CreditorService接口中加删除合同的方法deleteContract

**因为目前提供的方法，如果group和remoteFilePath为空就不更新，所以我们需要自己提供**

*/\*\*  
 \* 删除合同  
 \** ***@param*** *id  
 \** ***@return*** *\*/*int deleteContract(Integer id);

#### 在CreditorServiceImpl类中对deleteContract方法进行实现

@Override  
@Transactional //加上该注解控制事务  
public int deleteContract(Integer id) {  
 // 1 删除失败；0 删除成功  
 int result = 1;  
 //根据债权id获取债权信息  
 CreditorInfo creditorInfo = creditorInfoMapper.selectByPrimaryKey(id);  
 */\*\*  
 \* 注意：事务控制的数据库，所以我们先对数据库进行更新，在操作FastDFS  
 \* 如果操作FastDFS失败了，那么对数据库的操作回滚  
 \*/* //更新数据库债权表的合同路径及组  
 int updateRow = creditorInfoMapper.updateConstractById(id);  
 if(updateRow > 0){  
 //如果数据库更新成功，那么删除FastDFS上的文件  
 int num = FastDFS.*fileDelete*(creditorInfo.getGroupname(),creditorInfo.getRemotefilepath());  
 if(num == 0){  
 //如果删除成功，那么将整个操作结果设置为0，表示成功  
 result = 0;  
 }else{  
 //如果删除FastDFS上的文件失败，我们抛出一个运行异常，回滚事务  
 throw new RuntimeException("FastDFS文件删除失败");  
 }  
 }  
  
 return result;  
}

#### 在CreditorMapper类中添加更新的方法

*/\*\*  
 \* 根据债权的id，将组和合同路径更新为null  
 \** ***@param*** *id  
 \** ***@return*** *\*/*int updateConstractById(Integer id);

#### 在CreditorMapper.xml中添加更新的方法

<update id="updateConstractById" parameterType="java.lang.Integer">  
 update creditor\_info  
 set   
 groupName = NULL ,  
 remoteFilePath = NULL   
 where id = #{id,jdbcType=INTEGER}  
</update>

#### 修改FastDFS类中的fileDelete方法，提供参数

//删除文件的方法  
public static int fileDelete(String group ,String remoteFile){  
 int num = 1;  
 try {  
 //1. 获取StorageClient对象  
 StorageClient storageClient = *getStorageClient*();  
 //2.删除文件 返回0表示成功，其它均表示失败  
 num = storageClient.delete\_file(group,remoteFile);  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (MyException e) {  
 e.printStackTrace();  
 } finally {  
 *closeFastDFS*();  
 }  
 return num;   
}

#### 在Application类上开启事务支持

@SpringBootApplication  
@EnableTransactionManagement  
public class Application {  
 public static void main(String[] args) {  
 SpringApplication.*run*(Application.class, args);  
 }  
}

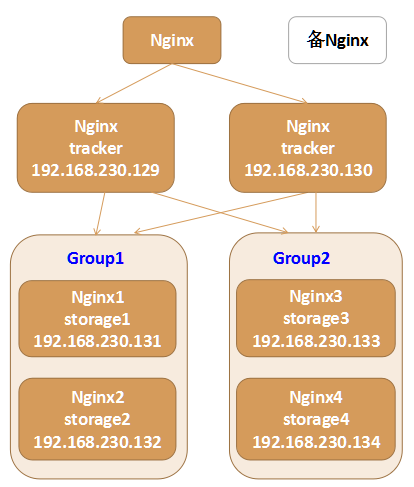
### 功能实现-弹层组建layer的使用（简单介绍）

官网：<https://www.layui.com/>

2018开源软件排行比较靠前

# FastDFS分布式文件系统集群

## 架构图



* 如果你公司刚好用这个，那你就会搭建集群
* 涉及到多个Linux，你可以更进一步熟悉一下Linux
* 提升自己驾驭复杂环境的能力

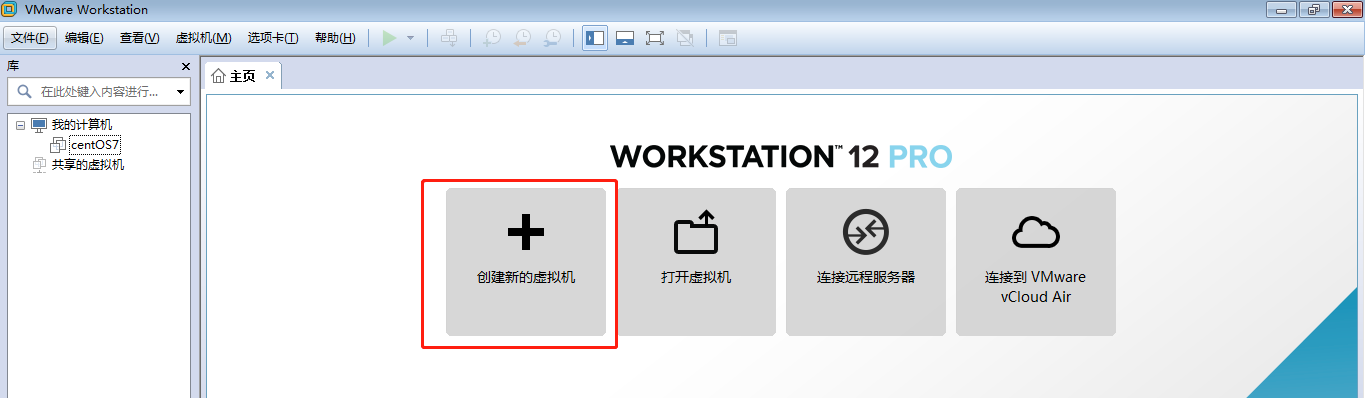
## 环境搭建步骤

搭建一个FastDFS分布式文件系统集群，推荐至少部署6个服务器节点

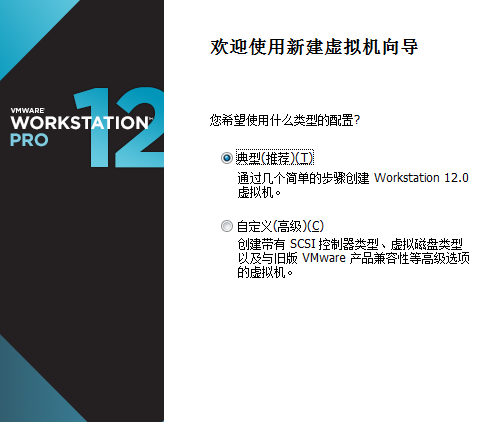
### 安装6个迷你版的Linux

迷你版Linux没有图形界面，占用磁盘及资源小，企业里面使用的Linux都是没有图形界面的Linux，在安装的时候，最少要给每一个虚拟机分配3.5G的内存空间，我这里分配5G，内存我这里分配的是512M

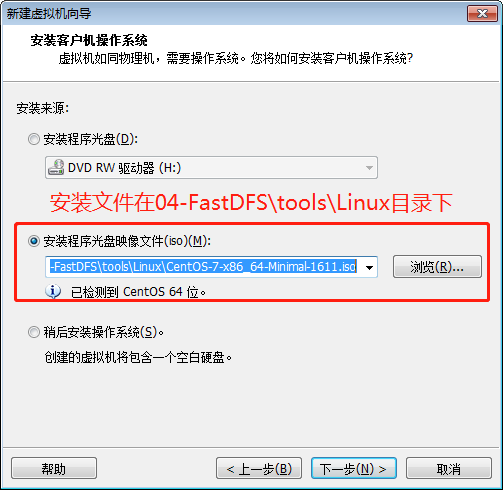
#### 打开Vmware，点击创建新的虚拟机



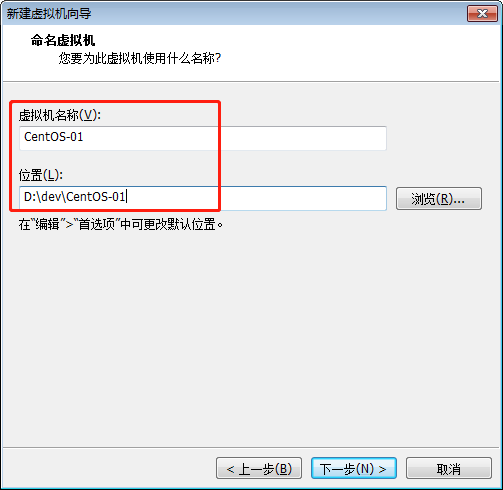
#### 选择典型安装



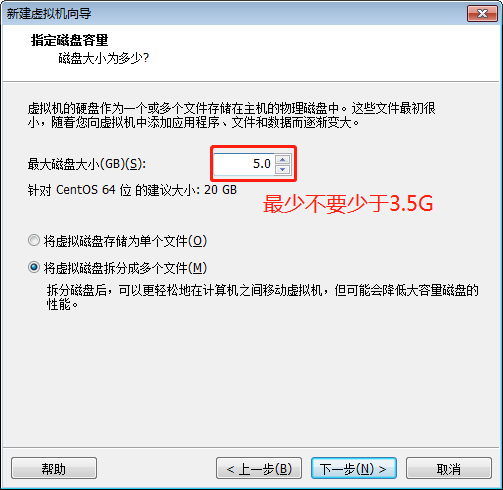
#### 选择安装Linux迷你版安装文件位置



#### 指定虚拟机名称及安装位置

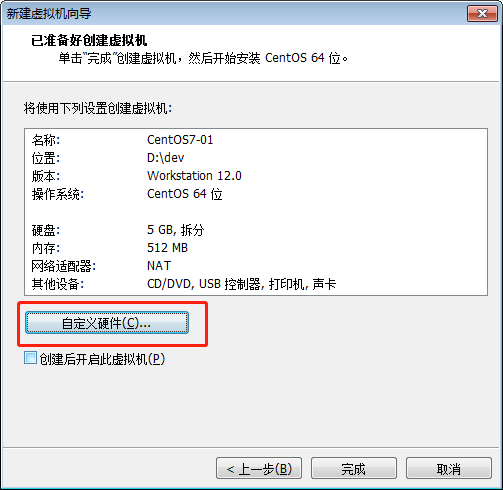


#### 为虚拟机分配磁盘空间



#### 为虚拟机分配内存

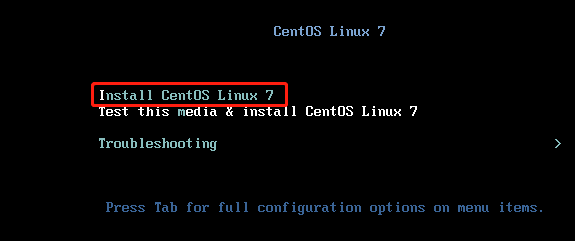
我是8G内存，每个虚拟机分配512M，因为是迷你版，所以内存消耗不会太大



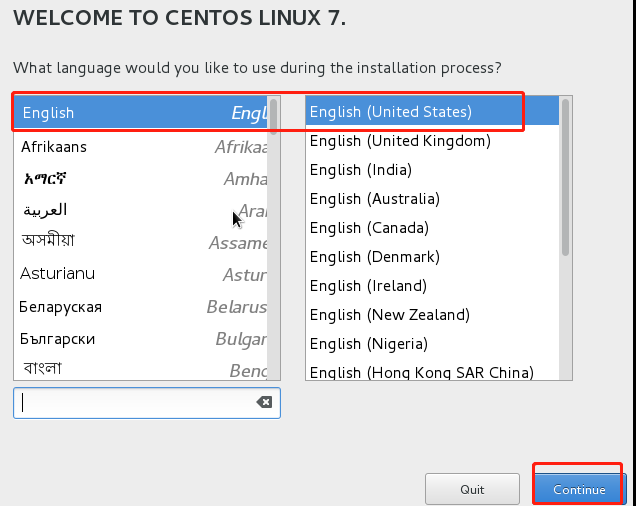
#### 开启此虚拟机



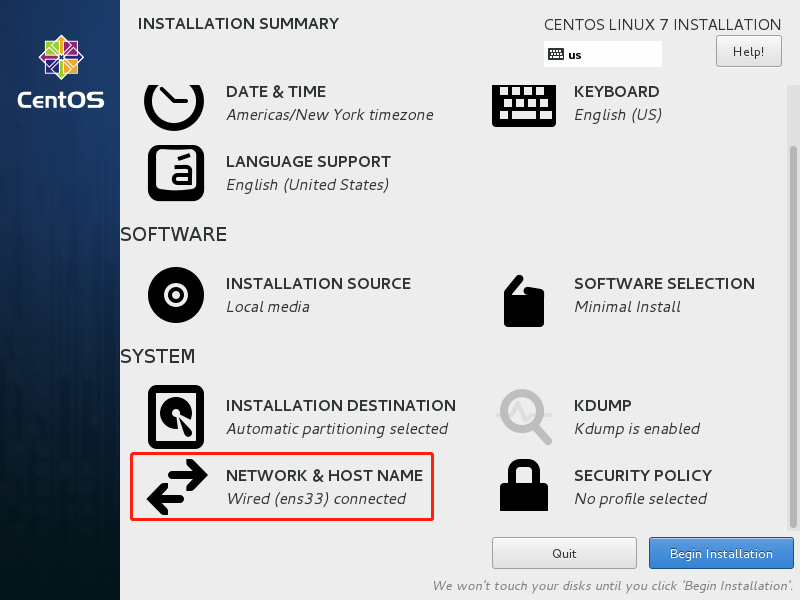
#### 开始安装

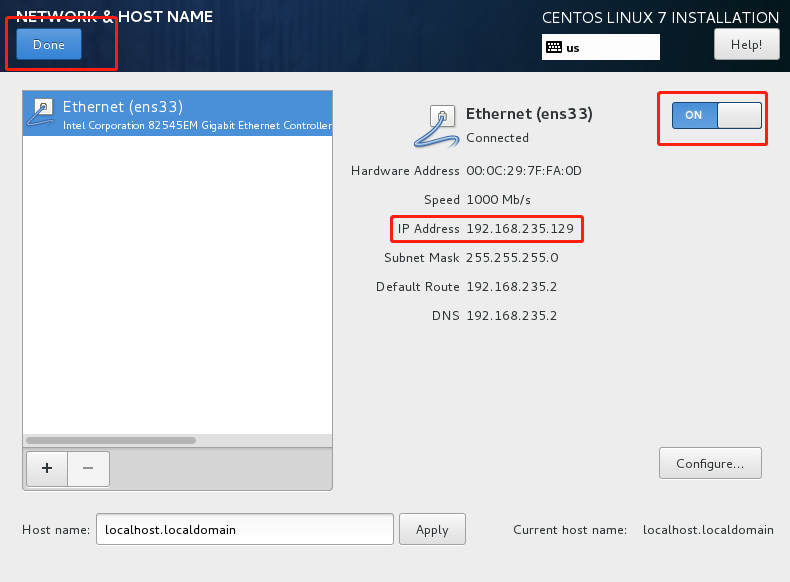


#### 选择语言为English



#### 打开网络连接（重要）





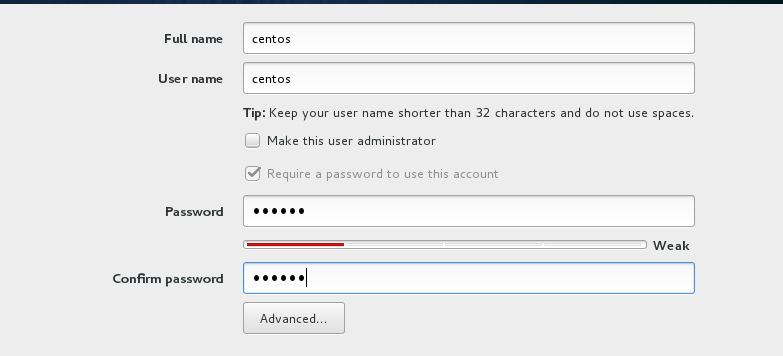


这个选项点进去完成一下就好，其它配置都可以默认，但是网络一定要打开，否则虚拟机之间无法通讯

#### 设置root用户密码为123456

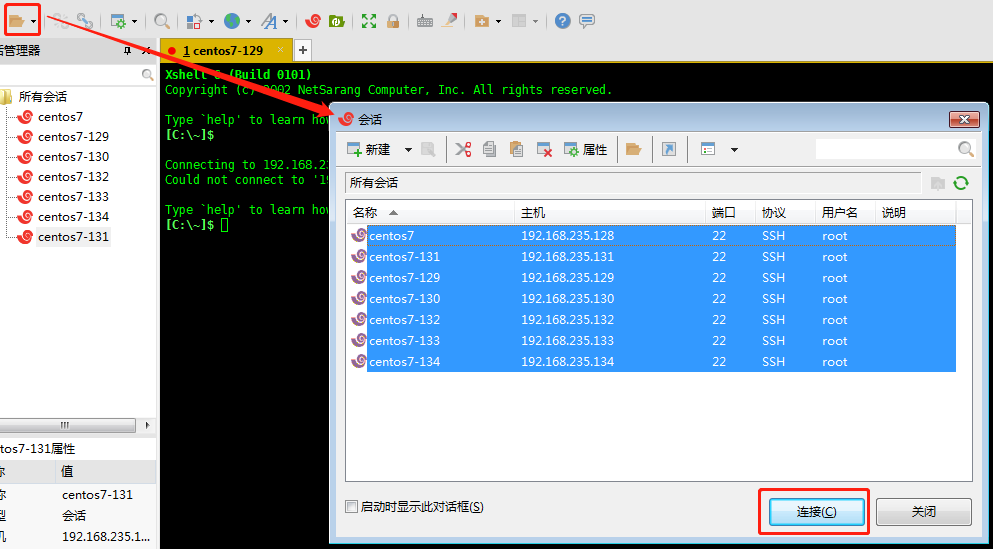


#### 创建普通用户centos，设置密码为123456



#### 安装完毕后，重启安装的CentOS系统

### 在Xshell中创建6个连接，分别连向不同的Linux



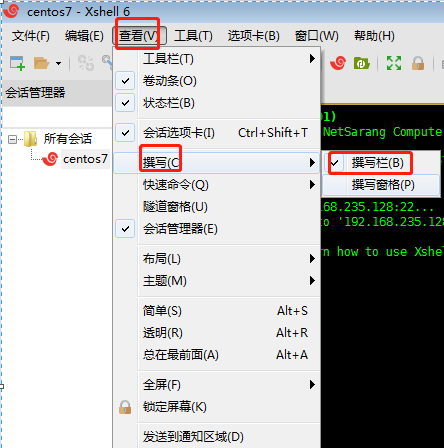
### 为迷你版的Linux安装常用工具库

由于迷你版Linux缺少一些常用的工具库，操作起来不方便，推荐安装如下的工具库

* 安装lrzsz， yum install lrzsz -y
* 安装wget, yum install wget -y
* 安装vim， yum install vim -y
* 安装unzip，yum install unzip -y
* 安装ifconfig，yum install net-tools -y

yum install lrzsz wget vim unzip net-tools –y

**打开撰写栏，方便批量执行命令**



### 按照课件上安装FastDFS的步骤在6个服务器节点安装FastDFS

### 配置两个tracker server服务器

**为了方便操作，我们再启动一次Xshell，一个Xshell操作Tracker，另一个操作Storage，**

**将Tracker和Storage分开**

* 把/etc/fdfs目录下的配置文件.sample后缀都去掉
* 修改两个tracker服务器的配置文件：

tracker.conf: 修改一个地方：

base\_path=/opt/fastdfs/tracker #设置tracker的数据文件和日志目录（需预先创建）

* 创建存放数据的目录

### 配置四个storage server服务器

每两个storage server为一组，共两个组（group1 和 group2），一个组内有两个storage server

#### 修改第一组group1的第一个storage server（修改storage.conf配置文件）

group\_name=group1 #组名，根据实际情况修改，值为 group1 或 group2

base\_path=/opt/fastdfs/storage #设置storage的日志目录（需预先创建）

store\_path0=/opt/fastdfs/storage/files #存储路径

tracker\_server=192.168.235.129:22122 #tracker服务器的IP地址和端口号,配2个

tracker\_server=192.168.235.130:22122

#### 第一个组的第二个storage按照相同的步骤操作

或者将第一组的配置文件下载到桌面上，然后上传覆盖第一组的第二个storage

#### 第二组的两个storage也按照相同的步骤操作；唯一的区别是group\_name=group2

可以在桌面上对第一个组的storage文件进行修改，将组名修改为group2，然后上传覆盖

**至此，一个FastDFS的分布式文件系统集群就搭建好了。**

#### 注意：配置文件中不要出现中文，另外别忘了创建配置文件中指定的目录

#### 启动两个tracker，再启动四个storage

#### 关闭6个Linux防火墙，通过09-fastdfs-java的代码进行测试

* 修改配置文件为两个tracker
* 测试负载均衡：tracker.conf文件 (修改 store\_lookup=0 表示轮训策略)

### 部署Http访问FastDFS上的文件

#### 先完成4个storage server的Nginx访问

**这4个Nginx需要去fastDFS找对应的文件，所以需要安装FastDFS的Nginx扩展模块**

##### 配置带有FastDFS扩展模块的Nginx

1、在四个storage server上安装Nginx，并且加入FastDFS扩展模块；

2、修改两组4个storage的nginx扩展模块配置文件 mod\_fastdfs.conf

base\_path=/opt/fastdfs/nginx\_mod #保存日志目录(需提前创建)

tracker\_server=192.168.230.129:22122 #tracker服务器的IP地址以及端口号

tracker\_server=192.168.230.130:22122

group\_name=group1 #当前服务器的group名，第二组应配置为group2

url\_have\_group\_name=true #文件url中是否有group名

store\_path\_count=1 #存储路径个数，需要和store\_path个数匹配（一般不用改）

store\_path0=/opt/fastdfs/storage/files #存储路径

group\_count = 2 #设置组的个数

在末尾增加2个组的具体信息：

[group1]

group\_name=group1

storage\_server\_port=23000

store\_path\_count=1

store\_path0=/opt/fastdfs/storage/files

[group2]

group\_name=group2

storage\_server\_port=23000

store\_path\_count=1

store\_path0=/opt/fastdfs/storage/files

##### 第一组的第二台和上面的配置一样

##### 第二组的两台只需要把group\_name=group2即可

##### 至此，mod\_fastdfs.conf就配置好了。

#### 配置两组4个storage的nginx拦截请求路径：

location ~ /group[1-9]/M0[0-9] {

ngx\_fastdfs\_module;

}

至此4个storage服务器上的Nginx就搭建配置OK了，

然后可以进行测试：

重启storage，启动Nginx

http://192.168.119.128:80/group1/M00/00/00/wKh3jVx6FUCARyK2AAAANaS4cxw338.txt http://192.168.92.132:80/group1/M00/00/00/wKhchFv7xQeAQbrBAAAALDPVPvc081.txt

http://192.168.92.133:80/group1/M00/00/00/wKhchFv7xQeAQbrBAAAALDPVPvc081.txt

<http://192.168.92.134:80/group1/M00/00/00/wKhchFv7xQeAQbrBAAAALDPVPvc081.txt>

注意：每一台都可以访问到，就算是当前组中没有改文件，因为向浏览器中发送请求的时候

* + 交给Nginx进行location匹配
  + 匹配上之后调用FastDFS的扩展模块
  + 扩展模块会读取扩展模块配置文件mod\_fast.conf
  + 通过扩展模块配置文件，找到对应的Tracker，从而找到对应的文件

#### 在两个tracker server安装Nginx

**这两个Nginx只需要做负载均衡，不要找文件，所以不需要安装扩展模块**

#### 配置两个tracker server服务器上的Nginx访问

部署配置nginx负载均衡:

upstream fastdfs\_storage\_server {

server 192.168.92.131:80;

server 192.168.92.132:80;

server 192.168.92.133:80;

server 192.168.92.134:80;

}

#nginx拦截请求路径：

location ~ /group[1-9]/M0[0-9] {

proxy\_pass http://fastdfs\_storage\_server;

}

至此，两个tracker服务器的Nginx就搭建配置OK了。

启动两个tracker服务器的Nginx进行测试。

http://192.168.92.129:80/group1/M00/00/00/wKhchFv7xQeAQbrBAAAALDPVPvc081.txt

http://192.168.92.130:80/group1/M00/00/00/wKhchFv7xQeAQbrBAAAALDPVPvc081.txt

#### 部署前端用户访问入口服务器，该Nginx负载均衡到后端2个tracker server

**这个Nginx也只需要做负载均衡，不要找文件，所以不需要安装扩展模块**

**可以对Windows上的的Nginx进行配置**

部署配置nginx负载均衡:

upstream fastdfs\_tracker\_server {

server 192.168.92.129:80;

server 192.168.92.130:80;

}

#nginx拦截请求路径：

location ~ /group[1-9]/M0[0-9] {

proxy\_pass http://fastdfs\_tracker\_server;

}

访问：http://127.0.0.1:80/group1/M00/00/00/wKhchFv7xQeAQbrBAAAALDPVPvc081.txt

至此，一个三层结构的Nginx访问架构就搭建配置OK了。为了保证高可用性，一般在入口出，会添加一个备用Nginx上，中间通过一个keepAlive软件连接。

**\*\*最后，为了让服务能正常连接tracker，请关闭所有机器的防火墙：\*\***

**systemctl status firewalld**

**systemctl disable firewalld**

**systemctl stop firewalld**

# 简历

熟悉分布式文件系统FastDFS的环境安装与部署，并具有项目应用经验