目录

[ **系统目标：** 2](#_Toc446684177)

[ **系统总体设计：** 2](#_Toc446684178)

[ **系统详细设计：** 4](#_Toc446684179)

[**一、** **主流程设计** 4](#_Toc446684180)

[**1.** **增加WServer流程。** 4](#_Toc446684181)

[**3.** **为WServer做备份流程。** 7](#_Toc446684182)

[**6.** **Controler查看同步状态流程。** 11](#_Toc446684183)

[**7.** **主Wserver宕机修复流程。** 11](#_Toc446684184)

[将数据库修改掉，确认主Writer和从Writer的顺序互换。 11](#_Toc446684185)

[需要提供API接口实现数据库修改。 11](#_Toc446684186)

[**8.** **辅助Wserver宕机修复流程。** 11](#_Toc446684187)

[**9.** **CenterServer宕机修复流程。** 11](#_Toc446684188)

[**10.** **ClientServer宕机修复流程。** 11](#_Toc446684189)

[为client设计定时同步机制（见之前的设计），无论是否重启。Client的基本流程都是，请求controler获取仓库信息、服务器信息以及关联信息，并进行同步流程。如果client彻底宕机，或者重新部署client，则只需将原有repo目录删除，然后重新启动client即可。 11](#_Toc446684190)

[**11.** **Controler宕机修复流程。** 11](#_Toc446684191)

[需要为controler设计共享锁，保证操作使用单机进行，读操作使用多机部署。因此如果有其中一台宕机，其余的controler可继续工作。同时如果controler关掉的话。客户端则不做任何操作。 11](#_Toc446684192)

[**二、** **辅助流程设计（未完待续）** 11](#_Toc446684193)

[**1.** **创建和管理模板流程。** 11](#_Toc446684194)

[**2.** **客户端模板拼接流程。** 11](#_Toc446684195)

[**3.** **Pop节点配置check配置流程。** 11](#_Toc446684196)

[**4.** **Pop节点配置reload配置过程。** 12](#_Toc446684197)

[ **要点说明** 12](#_Toc446684198)

[**一、** **数据库详细设计** 12](#_Toc446684199)

[**1.** **模式表**：存放程序工作模式 ，mode值为0或1 ,0代表可以同步，1代表不能同步。 12](#_Toc446684200)

[**2.** **写服务器表：**存放写服务器信息 12](#_Toc446684201)

[**3.** **中心仓库服务器表：**存放中心仓库服务器信息 12](#_Toc446684202)

[**4.** **仓库表：**repos 13](#_Toc446684203)

[**5.** **客户端表：**client 13](#_Toc446684204)

[**6.** **仓库和中心服务端关联表：**repo\_and\_csserver 13](#_Toc446684205)

[**7.** **仓库和客户端同步关联表：**repo\_and\_client 13](#_Toc446684206)

[**二、** **接口详细设计** 14](#_Toc446684207)

[**1.** **Contorler提供相关接口。** 14](#_Toc446684208)

[**2.** **WriteServer提供相关接口。** 15](#_Toc446684209)

[**3.** **CenterServer提供相关接口。** 16](#_Toc446684210)

* **系统目标：**

1、需要设计一套通用文件配置系统，可以灵活控制agent所需要同步的数据。  
2、实现远端服务器配置同步和应用的过程分开，且各自执行需要有反馈信息。  
3、可实现快速的配置回滚机制。  
4、为以上操作记录服务器变更操作日志。  
5、配置文件更新需要实现有限更新，而非全量更新，尽量降低配置文件更新影响面。  
6、对配置者来说配置文件本身只读，配置者只能根据相应的配置模板来进行配置更新。  
7、可以查看服务器的各个仓库版本信息，也可和最新仓库版本进行对比，对比结果显示在界面上。  
8、监控同步客户端的状态。  
9、提供所有以上功能的API，并对API进行加密。

* **系统总体设计：**

该系统的主要部分是关于如何使用git工具构建一个集群同步系统，这个过程架构如下：



主要服务器如下：

1、中心控制端（controler）

2、仓库写入服务（Wserver）

3、中心机房仓库集群服务（CenterServer）

4、客户端集群服务（PopServer、agent）

其中部署有中心控制端（controler），该控制端实现所有用户请求的入口，提供大量api支持用户操作。

Wserver是仓库写入服务器，其功能主要提供权限设置、仓库建立和管理、同步到CenterServer集群等功能。

CenterServer是在中心机房建立的仓库集群服务，通过配置https的web服务（不同运营商分别使用不同的域名），为客户端提供资源下载功能。

PopServer部署在每台服务器上，通过拉取控制信息，获取自身需要进行的各种任务。

整个系统除了建立git集群同步体系之外，还将提供一些辅助功能，以实现配置检查、更新、模板处理等操作。

* **系统详细设计：**

为了让流程说明过程更加清晰，将流程分为两部分，其中主流程设计部分集中将使用git工具建立整个同步架构体系进行重点说明。辅助流程部分则进一步说明该系统其余部分功能。另外，在每个部分将流程拆分为多个部分以便进行分别说明。每个部分都有相应的数据库设计，服务器配置相关、api设计部分等。

1. **主流程设计**
2. **增加WServer流程。**



功能：增加一台WServer，并记录入数据库.

系统准备工作：在需要添加的WenterServer上，部署WriteServer agent.(简称WSagent) 端口8077

第一步：调用controler **API： :8067/api/wservercreate ，参数：WShostname\WSip\WSType（主备）\ Rebuild:参数值true or false。。**

第二步：controler调用WSagent 的**API：http://WSip:8077/api/wscreate ，参数： Rebuild：参数值true or false。**

第三步：WSagent自动生成，/root/.ssh/gitwriter 公钥秘钥对（公钥秘钥内容固定，并存放到/export/servers/WSserver/.ssh/下面），首先判断Rebuild值，如果是true，则首先判断公钥私钥两个文件和config文件是否存在，如果不存在（情况1）则将程序目录下的公钥私钥拷贝到/root/.ssh/下面，并配置config文件。Config内容如下：

**Config文件内容**

Host gitwriter103（CSserver服务器标识）

HostName 192.168.56.103（CSserver服务器ip）

User root

Port 22

IdentityFile ~/.ssh/gitwriter

Host gitwriter104

HostName 192.168.56.104

User root

Port 22

IdentityFile ~/.ssh/gitwriter

Host gitwriter105

HostName 192.168.56.105

User root

Port 22

IdentityFile ~/.ssh/gitwriter

**以上数据根据从数据库csserver表中获得，需要实现强制刷新接口API，以便后续CenterServer调用。**

如果存在（情况2）则删除公钥私钥后，重新建立。如果Rebuild值是false则第二种情况不作处理。（Rebuild 的值为true的过程，一般用于强制修复过程）

第六步：返回成功给controler。controler接收反馈后，根据成功和失败状态判断是否需要入库处理。

第七步: 入库过程描述.

库表：write\_server

字段：id、 hostname 、ip 、type（主或备份）

如果失败，将可重复上一过程。此时已经建立起了writer控制单元。可进一步做下一步的动作。

1. **增加CenterServer流程。**



功能：增加一台CenterServer，并记录入数据库.

系统准备工作：在需要添加的CenterServer上，预装nginx和gitolite，并部署CenterServer agent.(简称CSagent) 端口8087

第一步，调用controler **API： :8067/api/cservercreate ，参数：CShostname\CSip\CSdomain。**

第二步,controler调用CSagent 的**API：http://CSip:8087/api/cscreate ，参数： Rebuild：参数值true or false。首先要能获取repo仓库表里面的数据。**

第三步：CSagent自动生成，/root/.ssh/gitwriter.pub 公钥（公钥内容固定，并存放到/export/servers/CSserver/.ssh/下面），首先判断Rebuild值，如果是true，则首先判断公钥文件是否存在，如果不存在（情况1）则将程序目录下的公钥拷贝到/root/.ssh/下面。如果存在（情况2）则删除公钥后，重新建立。如果Rebuild值是false则第二种情况不作处理。（Rebuild 的值为true的过程，一般用于强制修复过程）

第四步：CSagent 通过gitolite生成 repository仓库。首先判断Rebuild值，如果是true，则首先判断root下.gitolite 、repository目录是否存在，如果不存在（情况1）则执行gitolite通过公钥建库过程。（${HOME}/bin/gitolite setup -pk /root/.ssh/gitwriter.pub）如果存在（情况2）则删除，/root/下面的 .gitolite、 repository 仓库，重新建立仓库。如果Rebuild值是false则第二种情况不作处理。（Rebuild 的值为true的过程，一般用于强制修复过程）同时在本地拉取gitolite-admin包，并将私钥分别拷贝到各个包的key目录下，并做一次推送操作。（保证权限被设置）

**第五步：如果发现repo仓库list是空列表，就不做任何调用，如果不是，就调用WServer API强制刷新一下Wserver上的config文件。然后根据仓库列表恢复裸仓库，并进行一次推送过程（前提是WServer已经建设完毕，且权限已经刷新好）。**

第六步：CSagent 返回成功给controler。

第七步：controler接收反馈后，根据成功和失败状态判断是否需要入库处理。

第八步: 入库过程描述.

库表：center\_server

字段：id、 hostname 、ip 、domain、gitservername（服务器标识）

如果失败，将可重复上一过程。

1. **为WServer做备份流程。**

第一步：按照创建Wserver流程部署新的Wserver端，注意创建过程中需要将**WSType设定为backup。**

第二步：将创建好的Wserver建立和主Wserver的备份关系。每一秒同步一份儿全库过来。。。。。。方式待定。

1. **创建仓库库流程。**



此过程是在以上过程都完成之后继续进行。

第一步：通过调用controler API 启动创建仓库过程,返回json串给controler。

**API： :8067/api/repocreate ，参数：Reponame、Repocomment、**

第二步：判断仓库表中是否有该仓库，如果没有，就创建仓库表，并建立关联表数据。置关联表状态为0.如果有，则判断CSserver和仓库关联表中该仓库数据是否状态都为1，如果是，则返回已经存在，如果没有则继续。

第三步：选出状态为0的部分。按顺序调用Wserver按执行修改各个CSserver上的gitolite内容并推送到各个CSserver过程。（失败重试3次）,彻底失败，则记录创建权限失败错误，并报警。

第三步：对已经成功配置权限的部分，进一步创建本地仓库，记录仓库hash值并上传到各个CSserver服务器（失败重试3次），彻底失败，记录创建失败错误，并报警。

第五步：controler接收到建库结果后，根据检查关联表中的建库状态是否都为1而判断出本次操作的成功与失败。

以上过程可重复进行，直到所有的CS服务器都建库成功。

数据库内容如下：

库表：repos

字段：id、 name 、download\_path 、dispatch\_path 、hash\_value、repo­\_comment

库表：center\_server

字段：id、 hostname 、ip 、domain、git\_servername（服务器标识）

库表：repo\_and\_csserver

字段：id、 repo\_id 、cserver\_id 、cstatus（创建库使用）、mstatus（留作修改库操作使用）。

1. **修改仓库流程。**



用户的操作部分，我们放到后面辅助功能部分进行讲解，总之，用户的操作需要记录入用户操作表格，并为该表格内的数据分配taskid（任务编号，根据commit过程来进行记录）。

每30秒，在Wserver端，对各个仓库进行一次统一的commit过程，并对该过程设置一个任务编号，将任务编号自增加入到任务表，另外将该任务编号，绑定用户操作库，以便将本次任务绑定到相应的用户操作上，同时commit后将Wserver上面的仓库的hash值记录入仓库表。

执行commit并入库后，进一步分别对各个CenterServer上面的仓库进行push操作，并将push过程成功和失败状态记录入，CSserver和Repo关联表。（repo\_and\_csserver.mstatus）。

PopServer（预先部署客户端）每10秒定时请求CSserver获取仓库数据。CSServer

但是约定同步的两项约束条件：

1. **Mode表内容。**

**mode是否开启同步。**

1. **判断服务器和仓库的关联关系中的auto值，如果auto值被设定为0，则该服务器对该仓库的同步跳过。**

客户端流程（定时执行）：

第一步：每次请求过程中请求contorler获取repo最新版本数据以及现在的工作模式。对工作模式为0的不做同步，如果工作模式为1，就继续从关联表中取出需要同步的仓库（auto值）。请求API如下：

**API： :8067/api/ getrepoinfo，参数：仓库名称列表，获取仓库名称列表和对应的最新版本号。**

对每个仓库都进行同步，同步后将版本数据和repo版本数据进行对比，然后返回json对比数据和时间戳给contorler。反馈API如下：

**API： :8067/api/syncresult ，参数：仓库列表id和server id，status，timestamp。**

定时同步

数据库内容如下：

库表：mode

字段：id、mode、comment

库表：repos

字段：id、 name 、download\_path 、dispatch\_path 、hash\_value、repo­\_comment

库表：client

字段：id、 hostname 、ip 、domain、git\_servername（服务器标识）

库表：repo\_and\_client

字段：id、 repo\_id 、client\_id 、auto（是否跳过该仓库的同步）、status。

1. **Controler查看同步状态流程。**

只需关联查询repo\_and\_client 和repo、client三个表，即可得到现有的同步状态，其中时间戳部分，可以通过Api进行查询。

以上已经将主流程都定义完成。下面将进入特殊流程说明一下流程都可以通过先将mode修改为关闭状态，然后进行。

1. **主Wserver宕机修复流程。**

将数据库修改掉，确认主Writer和从Writer的顺序互换。

需要提供API接口实现数据库修改。

1. **辅助Wserver宕机修复流程。**
2. **CenterServer宕机修复流程。**

CenterServer如果是断网，则应该会有报警。

如果重新部署，则按照重新部署流程走即可。Rebuild = true

接下来等待30秒writerserver继续同步即可。

1. **ClientServer宕机修复流程。**

为client设计定时同步机制（见之前的设计），无论是否重启。Client的基本流程都是，请求controler获取仓库信息、服务器信息以及关联信息，并进行同步流程。如果client彻底宕机，或者重新部署client，则只需将原有repo目录删除，然后重新启动client即可。

1. **Controler宕机修复流程。**

需要为controler设计共享锁，保证操作使用单机进行，读操作使用多机部署。因此如果有其中一台宕机，其余的controler可继续工作。同时如果controler关掉的话。客户端则不做任何操作。

1. **辅助流程设计（未完待续）**
2. **创建和管理模板流程。**
3. **客户端模板拼接流程。**
4. **Pop节点配置check配置流程。**
5. **Pop节点配置reload配置过程。**

* **要点说明**

1. **数据库详细设计**
2. **模式表**：存放程序工作模式 ，mode值为0或1 ,0代表可以同步，1代表不能同步。

CREATE TABLE `mode` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`mode` int(11) NOT NULL,

`comment` varchar(32) NOT NULL,

PRIMARY KEY (`id`),

KEY `idx\_mode\_id` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

1. **写服务器表：**存放写服务器信息

CREATE TABLE `write\_server` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`hostname` int(11) NOT NULL,

`ip` varchar(32) NOT NULL,

`type` varchar(32) NOT NULL,

`create\_time` timestamp DEFAULT NULL,

`update\_time` timestamp DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

1. **中心仓库服务器表：**存放中心仓库服务器信息

CREATE TABLE `center\_server` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`hostname` int(11) NOT NULL,

`ip` varchar(32) NOT NULL,

`domain` varchar(32) NOT NULL,

`gitservername` varchar(32) NOT NULL,

`create\_time` timestamp DEFAULT NULL,

`update\_time` timestamp DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

1. **仓库表：**repos

CREATE TABLE `repos` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` int(11) NOT NULL,

`download\_path` varchar(32) NOT NULL,

`dispatch\_path` varchar(32) NOT NULL,

`hash\_value` varchar(32) NOT NULL,

`repo\_comment` varchar(32) NOT NULL,

`check\_func` varchar(32) NOT NULL,

`reload\_func` varchar(32) NOT NULL,

`create\_time` timestamp DEFAULT NULL,

`update\_time` timestamp DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

1. **客户端表：**client

CREATE TABLE `client` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`hostname` varchar(32) NOT NULL,

`ip` varchar(32) NOT NULL,

`domain` varchar(32) NOT NULL,

`git\_servername` varchar(32) NOT NULL,

`create\_time` timestamp DEFAULT NULL,

`update\_time` timestamp DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

1. **仓库和中心服务端关联表：**repo\_and\_csserver

CREATE TABLE `repo\_and\_csserver` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`repo\_id` int(11) NOT NULL,

`cserver\_id` int(11) NOT NULL,

`cstatus` int(2) NOT NULL,

`mstatus` int(2) NOT NULL,

`create\_time` timestamp DEFAULT NULL,

`update\_time` timestamp DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `idx\_repo\_cserver` (`repo\_id`,`cserver\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

1. **仓库和客户端同步关联表：**repo\_and\_client

CREATE TABLE `repo\_and\_client` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`repo\_id` int(11) NOT NULL,

`client\_id` int(11) NOT NULL,

`auto` int(2) NOT NULL,

`status` int(2) NOT NULL,

`check` int(2) NOT NULL,

`reload` int(2) NOT NULL,

`check\_status` int(2) NOT NULL,

`reload\_status` int(2) NOT NULL,

`create\_time` timestamp DEFAULT NULL,

`update\_time` timestamp DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `idx\_repo\_client` (`repo\_id`,`client\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

1. **接口详细设计**
2. **Contorler提供相关接口。**

url: http://ctrl.jdcdn.com/api/wservercreate （内部转发端口8067）

params: '{"WShostname":"BJ-UNI-1-WS-32","WSip":"192.168.56.102","WSType":"master","Rebuild":false}'

returned: '{"Code":0,"Msg":"create master wserver success"}'

用途：创建writeserver

url: http://ctrl.jdcdn.com/api/cservercreate （内部转发端口8067）

params: '{"CShostname":"BJ-UNI-1-CTS-33","CSip":"192.168.56.103","CSdomain":"cfuni.jd.com","Rebuild":false}'

returned: '{"Code":0,"Msg":"Create centorserver success"}'

用途：创建centerserver

url: http://ctrl.jdcdn.com/api/repocreate （内部转发端口8067）

params: '{"Reponame":"nginxandats","Repocomment":"联合配置1","DownloadPath":"/export/servers/gitconf/nginxandats","DispatchPath":"http://gitreader:git/nginxandats.git"}'

returned: '{"Code":0,"Msg":"Repo create success"}'

用途：创建仓库

url: http://ctrl.jdcdn.com/api/getrepoinfo （内部转发端口8067）

params: '{"hostname":"BJ-UNI-1-JCS-107"}'

returned: '{"Repos":[{"RepoName":"nginxandats","HashValue":"a123lkj12lk3j12l3789123lk",DispatchPath":"http://gitreader:git/nginxandats.git","CheckFunc":"","ReloadFunc":"","Check":true,"Reload":false},{"RepoName":"nginx","HashValue":"a123lkj12lk3j12l3789123lk",DispatchPath":"http://gitreader:git/nginxandats.git","CheckFunc":"","ReloadFunc":"","Check":true,"Reload":true]}'

用途：客户端获取需要同步的仓库信息

url: http://ctrl.jdcdn.com/api/syncresult （内部转发端口8067）

params: '{"hostname":"BJ-UNI-1-JCS-107","timestamp":1938912380,"Repos":[{"RepoName":"nginxandats","HashValue":"a123lkj12lk3j12l3789123lk","LocalHashValue":"a123lkj12lk3j12l3789123lk", "equal":true,"CheckStatus":1,"ReloadStatus":0]},{"RepoName":"ats","HashValue":"basdfasdfsadsdff3789123lk","LocalHashValue":"a123lkj12lk3j12l3789123lk","equal":false ,"CheckStatus":1,"ReloadStatus":1]}'

returned:'{"Code":0,"Msg":"Update repoandclient table success"}'

用途：更新仓库和客户端关联表

url: http://ctrl.jdcdn.com/api/getsyncresult （内部转发端口8067）

params: 可选 hostname、ip、

returned: '{"hostname":"BJ-UNI-1-JCS-107","timestamp":1938912380,"Repos":[{"RepoName":"nginxandats","HashValue":"a123lkj12lk3j12l3789123lk","LocalHashValue":"a123lkj12lk3j12l3789123lk", "equal":true},{"RepoName":"ats","HashValue":"basdfasdfsadsdff3789123lk","LocalHashValue":"a123lkj12lk3j12l3789123lk","equal":false ,"CheckResult":true,"ReloadResult":false}}'

用途：查看同步状态流程Api。

url: http://ctrl.jdcdn.com/api/changemaster （内部转发端口8067）

params: '{"MasterHostname":"BJ-UNI-1-WS-32","SlaveHostname":"BJ-UNI-1-WS-33"}'

returned:'{"Code":0,"Msg":"Change Writer Master and Slave Success"}'

用途：主Wserver宕机修复流程。将数据库修改掉，确认主Writer和从Writer的顺序互换。

1. **WriteServer提供相关接口。**

url: http://192.168.56.102:8077/api/wservercreate （内部转发端口8077）

params: '{"WShostname":"BJ-UNI-1-WS-32","WSip":"192.168.56.102","WSType":"master","Rebuild":false}'

returned: '{"Code":0,"WShostname":"BJ-UNI-1-WS-33","WSip":"192.168.56.102","WSType":"master","Msg":"Wserver create success"}'

用途：创建writeserver及授权

url: http://192.168.56.102:8077/api/refreshauth （内部转发端口8077）

params: '{"CShostname":"BJ-UNI-1-CTS-32","CSip":"192.168.56.102","CSdomain":"cfuni.jd.com","Rebuild":false}'

returned: '{"Code":0,"CShostname":"BJ-UNI-1-CTS-33","CSip":"192.168.56.102","CSdomain":"cfuni.jd.com","Msg":"Refresh success"}'

用途：刷新writeserver权限。（config文件）

1. **CenterServer提供相关接口。**

url: http://192.168.56.103:8087/api/cscreate （内部转发端口8077）

params: '{"CShostname":"BJ-UNI-1-CTS-33","CSip":"192.168.56.103","CSdomain":"cfuni.jd.com","Rebuild":false}'

returned: '{"Code":0,"Msg":"Create centorserver success"}'

用途：创建CenterServer及授权