宝兰德分布式缓存软件 3.1.0用户手册

北京宝兰德软件股份有限公司
Beijing Baolande Software Corporation

目录

第1章 前 言

本文档是宝兰德分布式缓存软件（BES CacheServer）的用户手册，详细介绍BES CacheServer的配置和管理。本手册的组织结构与管理控制台的布局基本上对应，每章都以概念性信息开头，随后的部分说明如何使用管理控制台进行特定的操作。

本手册适合的对象

本手册主要适用于使用BES CacheServer的系统管理员和开发人员。

本手册假定您已经具备如下技能：

1. 操作系统的基础操作。
2. JDK的安装。

约定

BES CacheServer定义了一些变量来表示BES CacheServer目录等信息，本文档中涉及到的有：

表 1-1 BES CacheServer变量说明

变量	说明
BCS_HOME	文档中借用该值表示BES CacheServer的安装目录，该变量实际上并不存在。

技术支持

BES CacheServer提供全方位的技术支持，获得技术支持的方式有：

网址：www.bessystem.com

Support Email：support@bessystem.com

Support Tel：400 650 1976

在取得技术支持时，请提供如下信息：

1. 姓名
2. 公司信息及联系方式
3. 操作系统及其版本
4. BES CacheServer版本
5. 日志等错误的详细信息

第2章 产品介绍

2.1 关于BES CacheServer

BES CacheServer（简称BCS）是一款宝兰德自研的分布式高性能KV存储数据库，可完全兼容Redis协议标准，支持基于内存和文件的持久化存储，保证数据的安全可靠。主要解决高并发、大数据量场景下的数据访问性能问题，具有高性价比、高可靠、弹性伸缩、高可用等特点。

2.2 支持的平台环境

BES CacheServer认证的平台环境如表所示：

表 2-1 BES CacheServer认证的平台环境

操作系统	RedHat系列、SUSE系列、银河麒麟操作系统、CentOS系列、深度操作系统、麒麟操作系统、一铭操作系统、统信操作系统.....
芯片	支持国内主流X86和ARM架构芯片：海光、华为鲲鹏、龙芯、飞腾、申威、兆芯.....
浏览器	IE：10.0+ Chrome：73.0+ Firefox：60+
JDK	JDK1.8+

2.3 使用场景

1）缓存热点数据

所有大中型网站都在使用的技术，不仅能够提高网站的访问速度，还能大大降低数据库的压力。用于缓存热点数据（经常会被查询，不经常被修改或者删除的数据），可以使用String数据类型，如果要设置过期时间，建议将热点数据过期时间错开，以防同一时间全部失效，造成缓存雪崩。

2）计数器

诸如统计点击数、访问数、点赞数、评论数、浏览数，都可以使用BCS实例提供自增命令实现计数器功能，内存操作，因为BCS实例高频率读写的特征可以完全发挥作为内存数据库的高效，数据类型String、Hash和Zset都提供了incr方法用于原子性的自增操作，下面列举下各自的使用场景：

- 如果应用需要显示每天的注册用户数，便可以使用string作为计数器，设定一个名为REGISTERED_COUNT_TODAY的key，初始化时给它设置一个到凌晨0点的过期时间，每当用户注册成功后便使用incr使其增加1。
- 每条微博都有点赞数、评论数、转发数和浏览数四条属性，这时用hash进行计数会更好，将该计数器的key设为weibo:weibo_id，hash的field为like_number、comment_number、forward_number和view_number，在对应操作后通过hincrby使hash中的field自增。
- 如果应用有一个发帖排行榜的功能，便选择zset吧，将集合的key设为POST_RANK。当

用户发帖后，使其zincrby将该用户id的score增加1，zset会重新进行排序，用户所在排行榜的位置也就会得到实时的更新。

3）Session共享

在分布式应用系统中，Session统一储存在内存数据库中，方便快速读写和集中管理，防止应用单点故障导致的Session丢失，可以使用BCS实例的String数据类型。

4）分布式锁

分布式锁是用来解决分布式应用中并发冲突的一种常用手段，可以使用基于BCS实例的String数据类型。setnx编写分布式锁，如果返回1说明获取锁成功，否则失败。下面列出分布式锁的两种策略：

- 抢不到锁的请求，允许丢弃。
- 抢不到锁的情况，继续重试策略，这里建议锁要设置过期时间，防止一致获取不到锁。

5）排行榜

很多网站目前都有排行榜的应用，如淘宝的年度/每日销量榜单、商品按时间的上新排行榜等。利用BCS的zset就能实现各种复杂的排行榜。比如使用zset和一个计算热度的算法便可以轻松打造一个热度排行榜，zrevrange可以得到以分数倒序排列的序列，zrevrank可以得到成员的排名。

6）社交网络

使用哈希、集合等数据结构能很方便的实现社交网站的点赞，粉丝，共同好友，推送，下拉刷新等基本功能。例如共同好友，可以使用set数据类型，把张三添加的还有通过sadd zhangsan haoyou1 haoyou2添加进去；同样，李四的好友通过sadd lisi haoyou2 haoyou3添加进去，然后sinter筛选出共同的好友。

7）消息队列

提供pub/sub方式或者基于list方式，来实现一个简单的消息队列。

- 使用list数据类型当作队列，比如一个客户端使用lpush生产数据到BCS实例，另一个客户端使用rpop取出数据进行消费，非常方便。但要注意的是，使用list当作队列，缺点是没有ack机制和不支持多个消费者。没有ack机制会导致从BCS取出的数据后，如果客户端处理失败了，取出的这个数据相当于丢失了，无法重新消费。所以使用list用作队列适合于对于丢失数据不敏感的业务场景。但它的优点是，因为都是内存操作，所以非常快和轻量。
- 使用pub/sub方式，可以支持多个消费者消费，生产者发布一条消息，多个消费者同时订阅消费。但是它的缺点是，如果任意一个消费者挂了，在这期间的生产者数据就丢失了。pub/sub只把数据发给在线的消费者，消费者一旦下线，就会丢失数据。另一个缺点是，PubSub中的数据不支持数据持久化，当BCS实例宕机恢复后，其他类型的数据可以从RDB和AOF中恢复，但pub/sub不行，它就是简单的基于内存的多播机制。

8）地理位置信息

诸如附近好友、摇一摇功能，这些都依赖于地理位置信息。可以使用BCS实例提供的geo数据类型来实现。例如附近好友功能，先把不同好友经纬度信息通过geoadd添加进去，然后通过geosearch功能查询方圆几里离你多远的符合条件的好友。

第3章 产品安装

请您联系BES CacheServer的销售代表，获取适用于您操作系统的安装包。

3.1 安装前检查

1) 卸载老版本的BES CacheServer

建议在安装BES CacheServer前先卸载老版本的BES CacheServer，保留以前版本的BES CacheServer可能会造成潜在的版本冲突，影响BES CacheServer的正常运行。有关老版本BES CacheServer的卸载步骤，请参考相应版本的安装手册。

2) 安装用户权限

在UNIX/Linux平台上，确保具有合适权限的用户来安装BES CacheServer，即确认用户对安装目录具有写权限。

3.2 解压产品安装包

```
mkdir /home/bes/BCS

tar -zxvf  BES-CACHESERVER-3.1.0-RHEL6-X64.tar.gz -C /home/bes/BCS
```

第4章 产品注册

BES CacheServer安装完成后自带的License是试用版本，是一个会过期的版本，用户必须进行申请许可、激活产品，才能正常使用BES CacheServer。

1. BES CacheServer提供的License包括以下几种：

1) 试用版本：在使用一段时间后自动过期，用户将无法使用产品。

2) 正式版本：不过期，支持用户永久使用的版本。

2. 产品注册具体步骤为：

1) 用户必须向北京宝兰德软件股份有限公司申请License。

2) 用户运行lmadm import-lic命令将收到的License文件导入BES CacheServer中，假设用户将License文件放在/home/bes/目录下，在BCS_HOME/bin通过下列命令导入License：

```
[bes@Linux17209 bin]$ ./lmadm import-lic --sourcepath=/home/bes/bcs.lic.txt
License has been imported into /home/bes/saber/BCS/server/license/bcs.lic suc」
↪  cessfully.
```

3) License导入成功后，通过lmadm view-lic命令读取License文件：

```
[bes@Linux17209 bin]$ ./lmadm view-lic
Customer.name=BES
Product.title=BES
Product.version=9.5.5
Product.type=Enterprise
License.type=PRIVATE
Register.feature=WEB
Serial.number=CN0Q-XW9JKM-TMSKA8-4EMN
Project.name=internal
Register.description=PRIVATE
```

第5章 管理中心

BES CacheServer管理中心是一个基于WEB浏览器的图形化管理工具，用户通过管理中心对BES CacheServer提供的主机、节点、实例进行配置和管理。

5.1 启动

1. BCS_HOME/bin下执行startManagement启动管理中心。

```
[bes@Linux17209 bin]$ ./startManagement
Starting BES Cache Server...
More information refer to server log (default:/home/bes//bcs/logs/serv」
↪  er.log)
```

5.2 登录

管理中心访问方式：

在浏览器中输入http://hostName:port/console以访问管理中心。其中hostName是服务器所在机器的主机名，或是服务器所在机器的IP地址；port是管理中心的管理端口，默认为4900，默认用户名和密码为：admin/B#2008_2108#es。

图 5-1 登录页面

5.3 注销

BES CacheServer管理中心提供用户注销功能，用户在控制台右上角用户名下的“**注销**”按钮即可退出登录。

5.4 使用指引

管理中心提供了“**使用指引**”，可以方便用户快速构建生产和测试环境。

图 5-2 管理控制台

第6章 主机管理

6.1 主机列表

在管理中心查看主机列表：

1) 在管理中心左侧导航区点击“**主机管理**”，右侧会显示所有已添加到管理中心的主机。

2) 主机列表显示当前主机名称、主机名/IP、主机类型、远程登录方式和操作栏。

主机可以是一台物理机，虚拟机或其他提供主机服务的设备。

图 6-1 主机列表

注：管理中心默认提供一个本地主机Local，Local主机默认不可删除。

6.2 新建主机

主机支持四种形式的远程登录方式：

1) 用户名密码登录，适用于知道主机用户名和密码的机器。
2) 证书登录，适用于不知道用户名密码，但可以获得允许访问主机的证书和密码，安全性较高。
3) 交互模式登录，适用于允许交互模式的机器。
4) 免登录（需手动安装节点）。

在管理中心新建主机：

1) 在管理中心左侧导航区点击“**主机管理**”，进入主机列表页面。
2) 点击“**新建**”按钮，进入“**主机信息**”页面，在当前页面录入主机的信息，包括：名称、主机名（或IP），操作系统，远程登录方式，SSH端口，用户名和密码。
3) 在远程登录方式勾选为“**证书登录**”时，存在证书路径和证书密码选项。

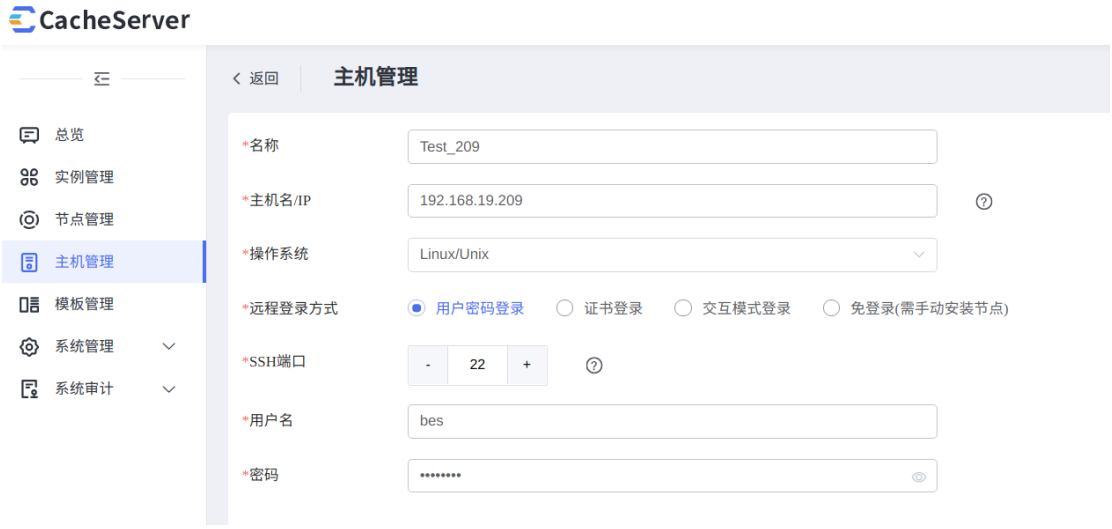

图 6-2 新建主机

4) 点击“**保存**”按钮即可添加成功。

6.3 编辑主机

在管理中心编辑主机：

1) 在管理中心左侧导航区点击“**主机管理**”，进入主机列表页面。
2) 点击主机列表主机操作栏里的“**编辑**”超链接，进入主机信息页面。
3) 可配置项：主机名/IP，操作系统，远程登录方式，SSH端口，证书路径，证书密码，用户名和密码。
4) 用户可以在右下角点击“**ping**”按钮测试主机是否可以ping通。

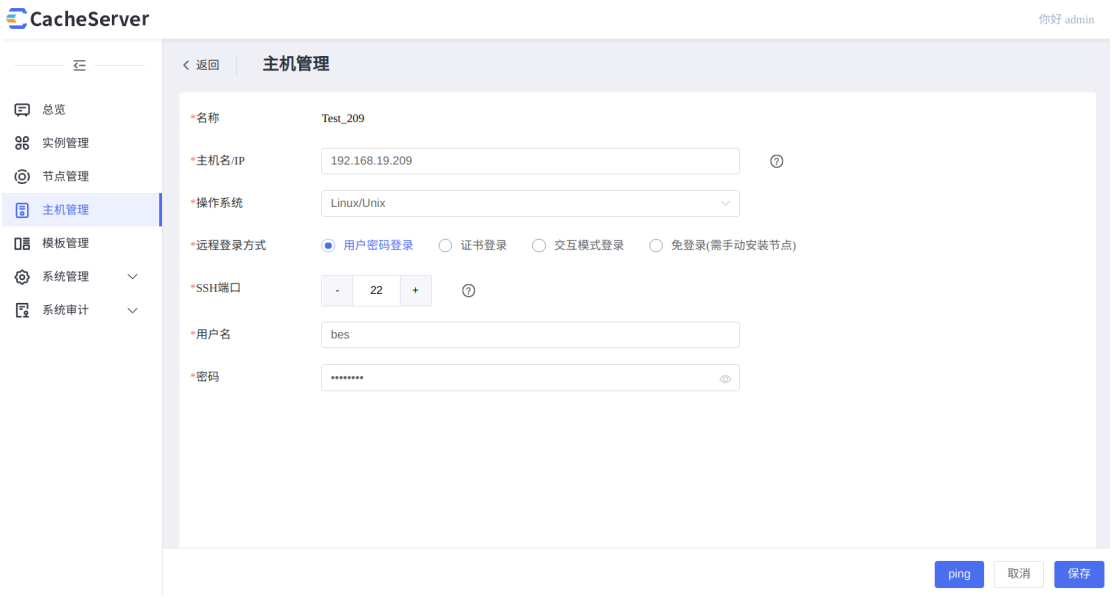

图 6-3 编辑主机

5) 修改主机信息，点击“**保存**”按钮保存修改的属性。

6.4 删除主机

在管理中心删除主机：

1) 在管理中心左侧导航区点击“**主机管理**”，进入主机列表页面。

2) 勾选要删除的主机，点击“**删除**”按钮弹出提示对话框。

3) 点击“**确认**”按钮即可删除主机。

注意：删除主机时，主机下不能包含节点。

第7章 节点管理

7.1 节点管理器

每个BES CacheServer实例都需要托管到一台物理计算机上，BES CacheServer新建了一个轻量级的托管代理进程（即节点管理器）来管理BES CacheServer实例的生命周期。节点管理器功能：

1) 启动、停止、新建和删除BES CacheServer实例。
2) 重新启动发生故障的BES CacheServer实例。

7.2 节点列表

在管理中心查看节点列表：

1) 在管理中心左侧导航区点击“**节点管理**”，操作区域显示节点列表页面，显示所有已添加到管理中心的节点。页面展示了当前节点名称、主机、管理端口、节点版本、节点状态、服务状态、节点位置和操作栏。
2) 用户可以对节点进行新建、删除、强制删除、启动、停止、重启、注册服务和删除服务操作。

图 7-1 节点列表

3) 点击操作栏里的“**高级编辑**”可以对节点配置信息进行编辑。
4) 点击操作栏里的“**下载日志**”可以下载节点的日志内容。

7.3 新建节点

节点依赖于主机，新建节点之前，需要先建好主机。

在管理中心新建节点：

1) 在管理中心左侧导航区点击“**节点管理**”，进入节点列表页面。
2) 点击“**新建**”按钮，在新建节点页面录入节点的信息，包括：节点名称，节点版本，主机，配置模板，管理端口，节点目录和JAVA HOME。其中，节点目录是指安装节点所在主机的绝对路径。

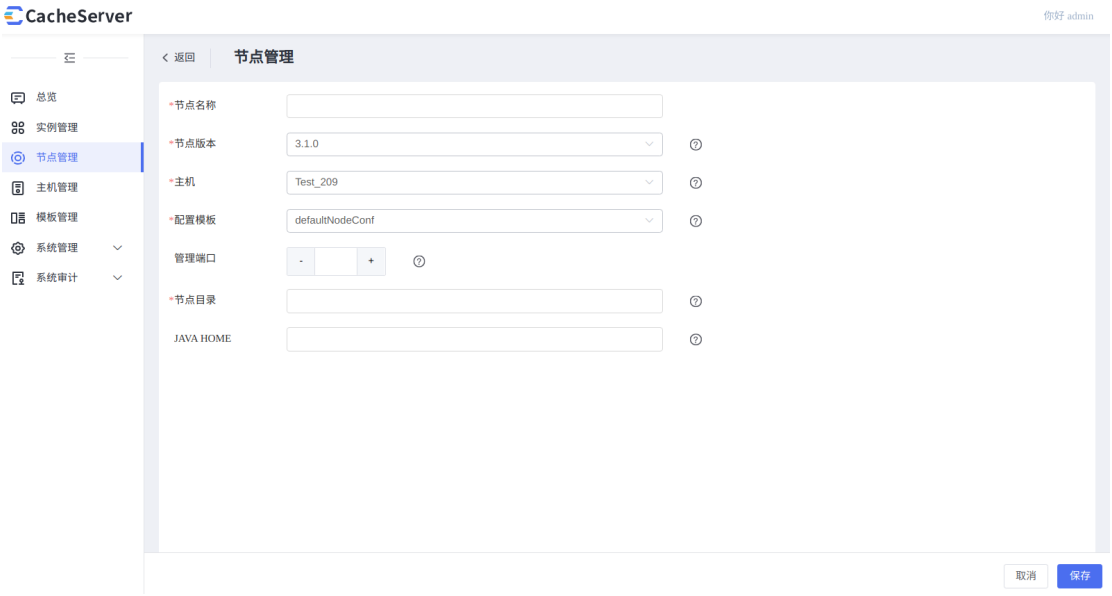

图 7-2 新建节点

3) 点击“**保存**”按钮即可新建成功。

注意：节点的管理端口，如不指定，系统会自动生成一个可用的端口，默认值：3100，需要确保节点使用的端口没有被占用。

7.3.1 安装免登录类型的节点

当主机所在机器不允许远程登录时，可以使用免登录类型的安装方式，实现节点的安装，可以按照以下步骤操作：

1. 在远程机器上新建节点目录，例如：/home/test/node/na，此处的na是节点名称，控制台添加节点需要和这个名称保持一致。
2. 手动将BCS_HOME/media/BCS-NODE-3.1.0.zip解压到节点目录na。
3. 在/home/test/node/na/bin下执行configserver命令

```
[bes@Linux17209 bin]$ ./configserver
=====================================================
Authentication
----------------------
Enter current admin user name :admin          -->用户名，默认为admin
Enter current admin password :                -->密码，
↪  默认为B#2008_2108#es

Authentication success.
----------------------
=====================================================
Enter the value for the dmshost option<default:localhost> :192.168.17.」
↪  209       -->管理中心控制台IP地址
Enter the value for the dmsport option<default:4900> :                 」
↪           -->管理中心控制台端口，
↪  默认4900
```

```
Enter the value for the nodehost option<default:0.0.0.0> :
↪          -->节点监听地址
Enter the value for the nodeport option<default:3100> :
↪          -->节点监听端口
==================================================
Initialization Detail
----------------------
Node Dir:/home/test/node                                    -->节点目录
New User:admin
New Password:**********
DMS Host:192.168.17.209
DMS Port:4900
Node Host:0.0.0.0
Node Port:3100
Node Name:na                                                        -->
↪  节点名称,
↪  和当前节点目录名称一致

==================================================
Initialization Complete.
--------------------------------------------------
```

配置完成后，代表节点安装已经完成。

7.3.2 启动免登录类型的节点

免登录类型的节点,不支持在管理中心和iastool中启动,需要在节点的安装目录/home/test/node/na/bin下，使用如下命令:

启动:

```
./startNode --user admin --password B#2008_2108#es
```

停止:

```
./stopNode --user admin --password B#2008_2108#es
```

7.3.3 注册服务/删除服务

节点提供注册服务功能，将节点注册为服务，机器重启后，节点会自动启动并将节点下所有实例一起启动。将节点注册为服务，需要使用root用户或者具有root权限的用户。对于普通节点，在“**节点列表**”页面即可进行注册服务/删除服务。

注意：对于免安装类型的节点，在节点目录下执行如下命令:

注册服务:

```
NODE_HOME/bin/service --action=register
```

删除服务:

```
NODE_HOME/bin/service --action=unregister
```

7.3.4 删除/强制删除节点

删除节点前要确保节点下不存在实例，并且节点处于停止状态。强制删除节点会先停止节点，然后删除节点。

7.4 基本信息

在管理中心查看节点基本信息：

1) 在管理中心左侧导航区点击“**节点管理**”，进入节点列表页面。

2) 点击一个节点操作栏里的“**编辑**”超链接，进入节点的编辑页面。

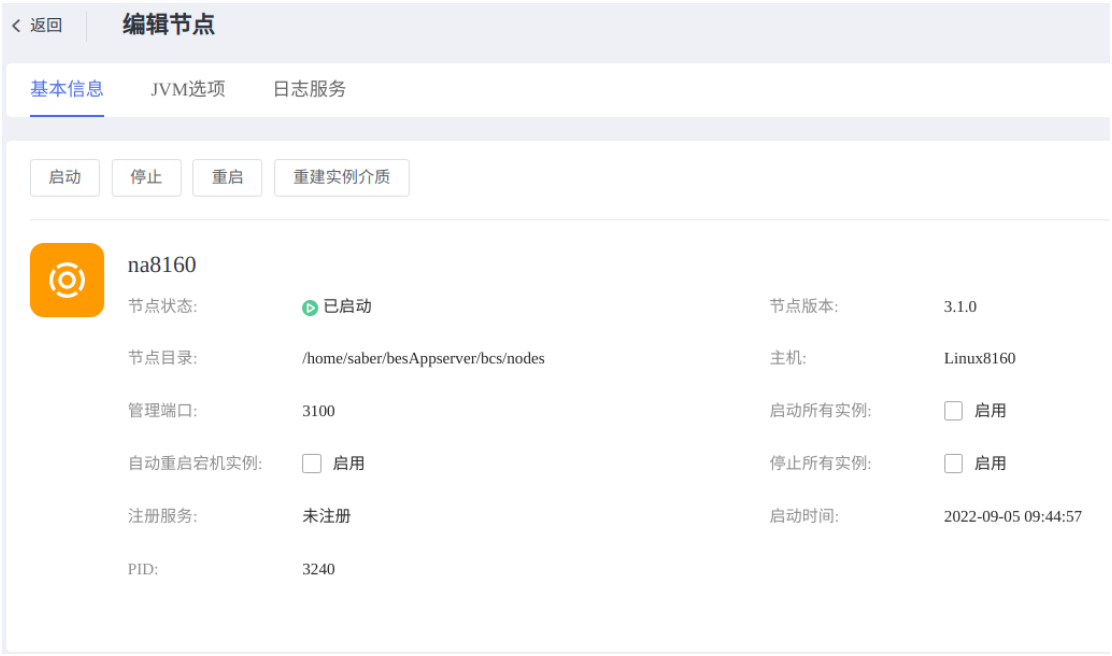

图 7-3 节点基本信息

表 7-1 基本信息配置项

配置项名称	说明	默认值
启动所有实例	节点启动时，将会启动该节点下的所有实例。	禁用
停止所有实例	节点停止时，将会停止该节点下的所有实例。	禁用
自动重启宕机实例	节点启动后，当实例非正常停止时，是否重新启动该节点下的实例。	禁用
注册服务	节点是否注册为服务	未注册
启动时间	节点的启动时间	节点启动时间
PID	节点进程ID	节点进程ID

7.5 日志服务

日志服务用来配置节点的日志信息。

在管理中心配置节点日志服务：

1. 在管理中心左侧导航区点击“**节点管理**”，进入“**节点列表**”页面。
2. 点击一个节点操作栏里的“**编辑**”超链接，进入节点的编辑页面，选择“**日志服务**”页面。

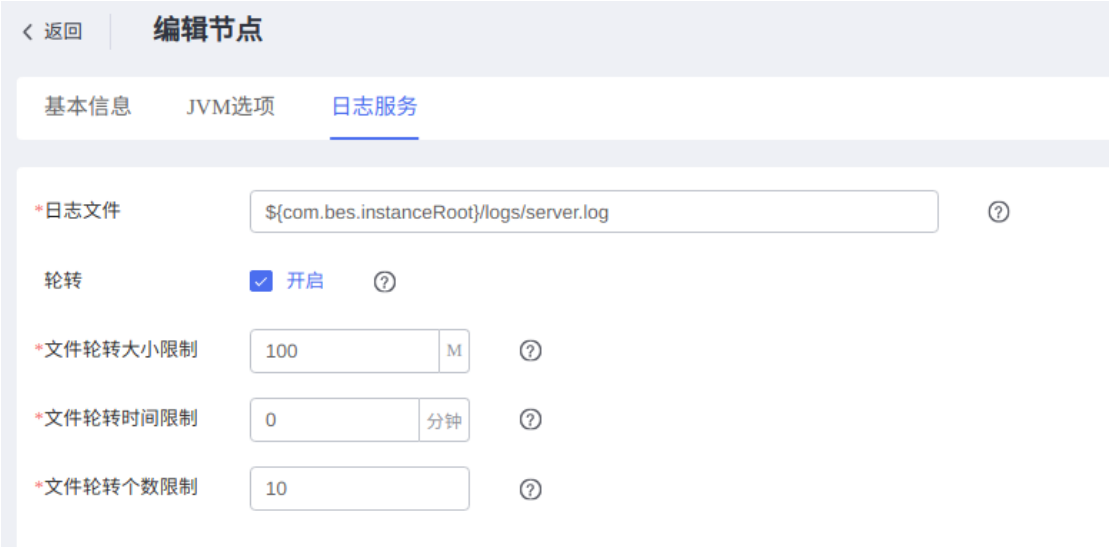

图 7-4 节点日志服务

表 7-2 日志服务配置项

配置项名称	说明
日志文件	日志文件的绝对路径。默认值：${com.bes.instanceRoot}/logs/server.log。
轮转	若启用，在满足条件时生成新的日志文件。默认值：启用。
文件轮转大小限制	指定日志文件的最大容量，达到上限后创建新的日志文件。合法值：1-2047，默认值：100，单位：兆。
文件轮转时间限制	日志文件轮转的时间间隔。当文件轮转时间间隔为0时，按文件轮转大小限制进行轮转。合法值：0-2147483647，默认值：0，单位：分钟。
文件轮转个数限制	日志文件轮转的最大个数，达到上限后删除时间最早的日志文件。合法值：1-2147483647，默认值：10。

第8章 实例管理

8.1 实例组列表

在管理中心查看实例组：

1) 在管理中心左侧导航区点击“**实例管理**”，进入实例组列表页面。
2) 在当前页面可以看到实例组名称、类型、创建人、集群状态、最后更新时间，并且可以构建集群、启动、停止、重启、新建实例组和删除实例组。
3) 实例组列表也提供简单的命令命中率监控，方便用户查看。

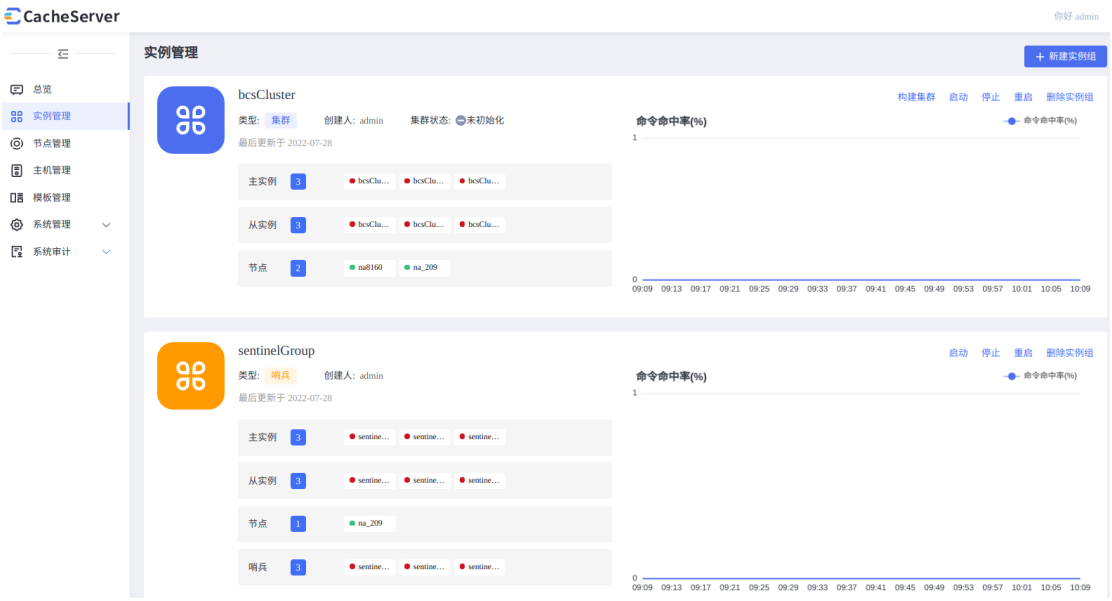

图 8-1 实例组列表

8.2 新建实例组

8.2.1 主从模式

BES CacheServer具有非常快的读取写入速度，这可能会造成非常大的读写压力。主从模式可以很好的解决这个问题，BES CacheServer的主从架构支持一主多从或者级联架构。主从模式可以做到读写分离，从而提高服务器性能。

在管理中心新建实例组：

1. 在管理中心左侧导航区点击“**实例管理**”，进入实例组列表页面。
2. 点击右上角“**新建实例组**”按钮，进入新建实例组页面，输入实例组名称，选择模式（可选值：主从模式、哨兵模式和集群模式），配置模板和描述。
3. 新建实例组页面提供了跳转到新建模板的链接，用户可以点击“**新建配置**”跳转到新建模板页面。

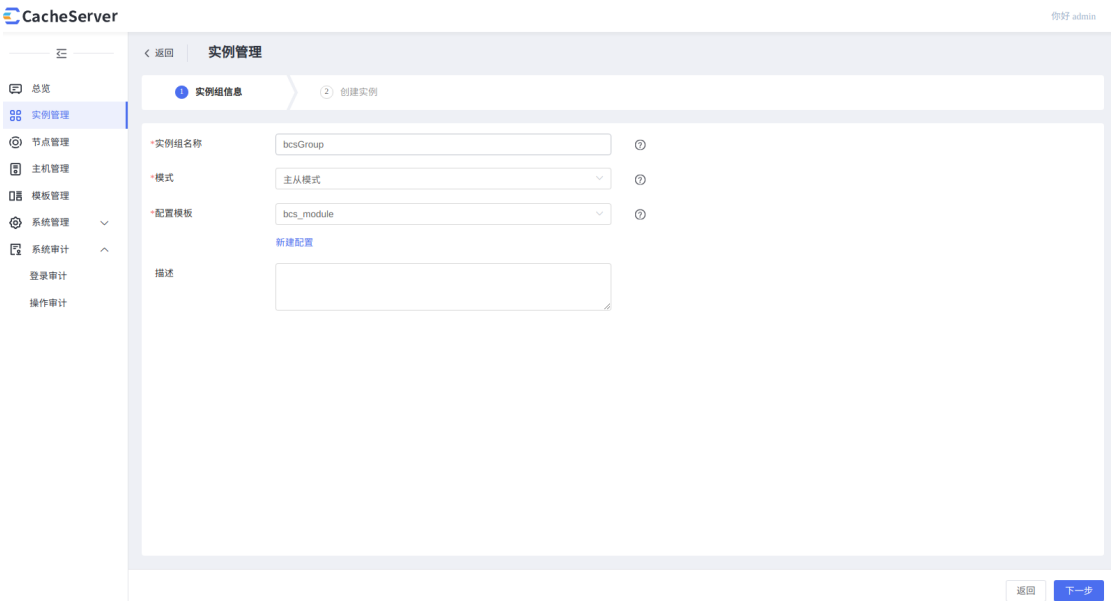

图 8-2 新建实例组页面

4. 点击“**下一步**”按钮进入创建实例页面，输入实例名称，选择实例所在节点，实例端口，安装路径，密码，点完成即可完成实例组新建。

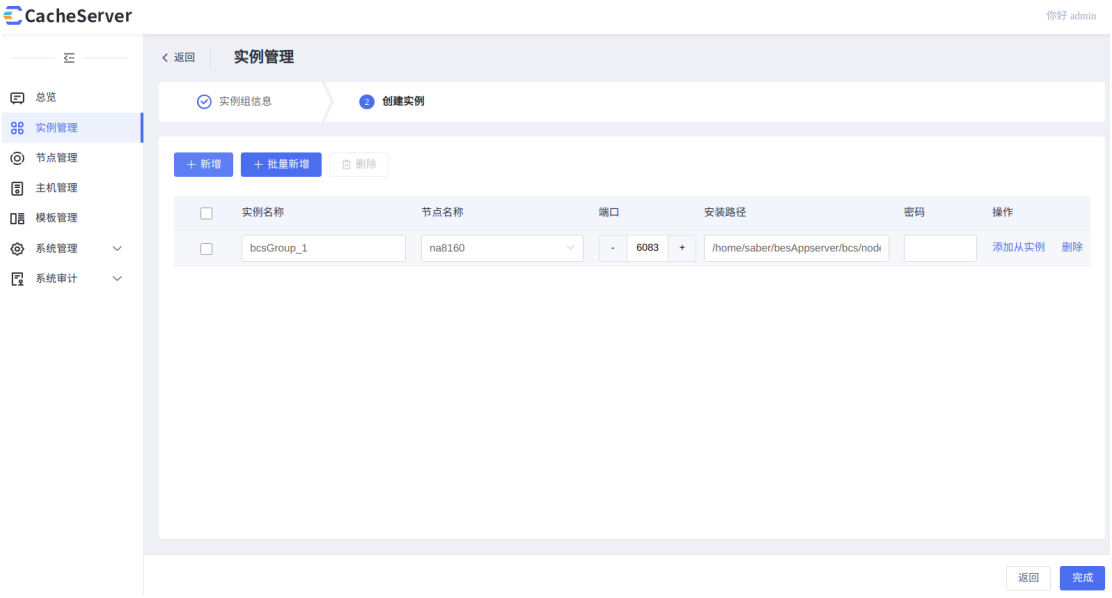

图 8-3 新建实例页面

5. 用户也可以选择“**批量新增**”功能创建主从实例。可以输入主实例名称前缀、节点、安装路径（默认在当前节点的bcses目录下）、起始端口、连接密码和新建实例数。

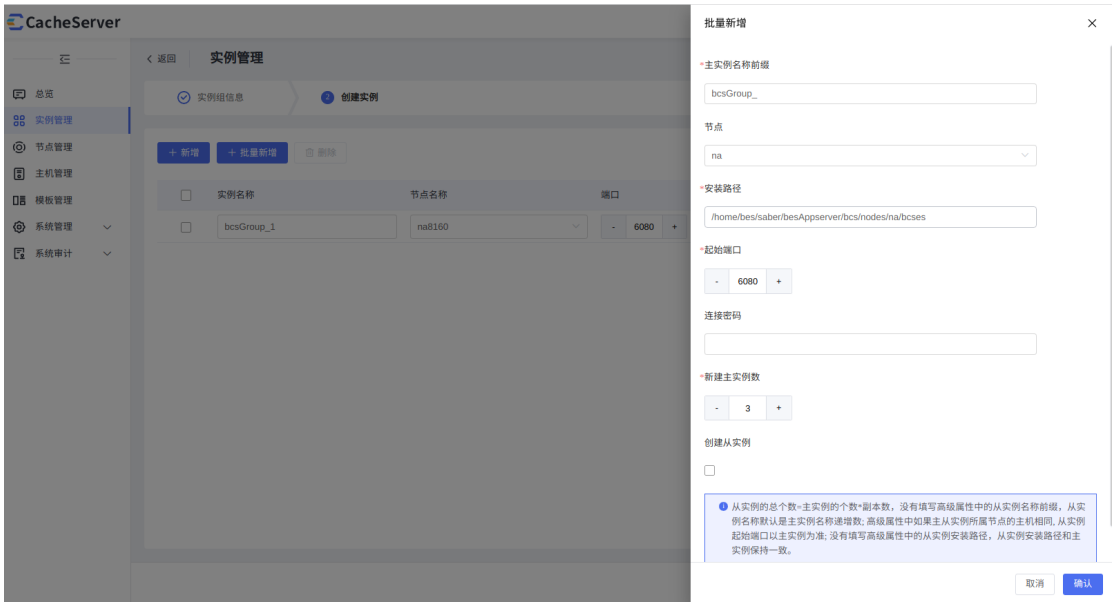

图 8-4 批量新建主实例页面

6. 勾选批量新增上面的“**创建从实例**”选项，输入主从复制比例，从实例节点，从实例名称前缀，从实例起始端口和从实例安装路径。

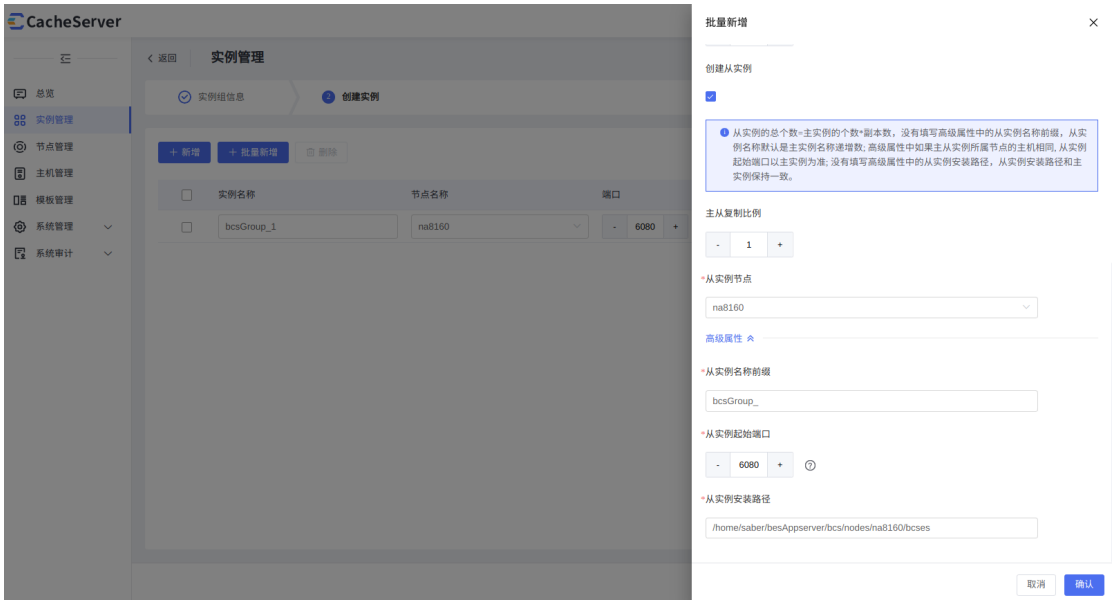

图 8-5 批量新建从实例页面

7. 点击“**确认**”按钮完后，批量新增的实例会出现在界面上。

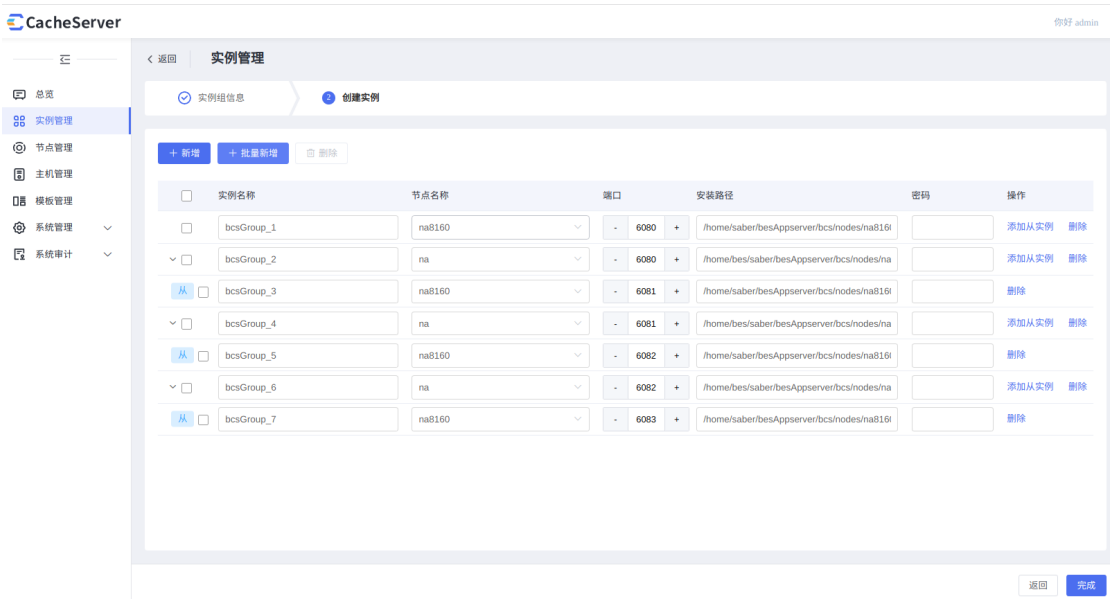

图 8-6 批量新建实例结果

8. 点击“**完成**”按钮完后，完成新建实例组。

9. 启动实例组，主从实例日志出现如下信息即代表主从模式组建成功。

主实例日志信息如下：

```
14520:M 30 Jul 2022 15:17:53.510 * Synchronization with replica 192.168.8.160
↪  :6379 succeeded    --->此处是从实例的IP和端口
```

从实例日志信息如下：

```
339553:S 30 Jul 2022 15:17:40.842 * Connecting to MASTER 192.168.19.30:6080
↪                      ---->此处是主实例的IP和端口
339553:S 30 Jul 2022 15:17:40.842 * MASTER <-> REPLICA sync started
339553:S 30 Jul 2022 15:17:40.875 * Non blocking connect for SYNC fired the e
↪  vent.
339553:S 30 Jul 2022 15:17:40.909 * Master replied to PING, replication can c
↪  ontinue...
339553:S 30 Jul 2022 15:17:40.974 * Partial resynchronization not possible (n
↪  o cached master)
339553:S 30 Jul 2022 15:17:41.008 * Full resync from master: 1881968cbe65545c
↪  2e3076ff10bddf7ccc3bced9:0
339553:S 30 Jul 2022 15:17:41.054 * MASTER <-> REPLICA sync: receiving 175 by
↪  tes from master to disk
339553:S 30 Jul 2022 15:17:41.055 * MASTER <-> REPLICA sync: Flushing old data
339553:S 30 Jul 2022 15:17:41.055 * MASTER <-> REPLICA sync: Loading DB in me
↪  mory
339553:S 30 Jul 2022 15:17:41.085 * Loading RDB produced by version 6.2.4
339553:S 30 Jul 2022 15:17:41.085 * RDB age 0 seconds
339553:S 30 Jul 2022 15:17:41.085 * RDB memory usage when created 1.83 Mb
339553:S 30 Jul 2022 15:17:41.085 * MASTER <-> REPLICA sync: Finished with su
↪  ccess
```

8.2.2 哨兵模式

BES CacheServer支持高可用解决方案，由一个或者多个哨兵实例组成哨兵系统，这些哨兵系统可以监视多个主服务器及其从服务器，当主服务器发生宕机时，会自动从某个从服务器选举出新的主服务器。

（1）哨兵模式和主从模式新建的方式相同，不同的是多了一个添加哨兵实例的“**哨兵信息**”标签页，用户可以在当前页面选择新增或者批量新增，完成哨兵实例的新建。

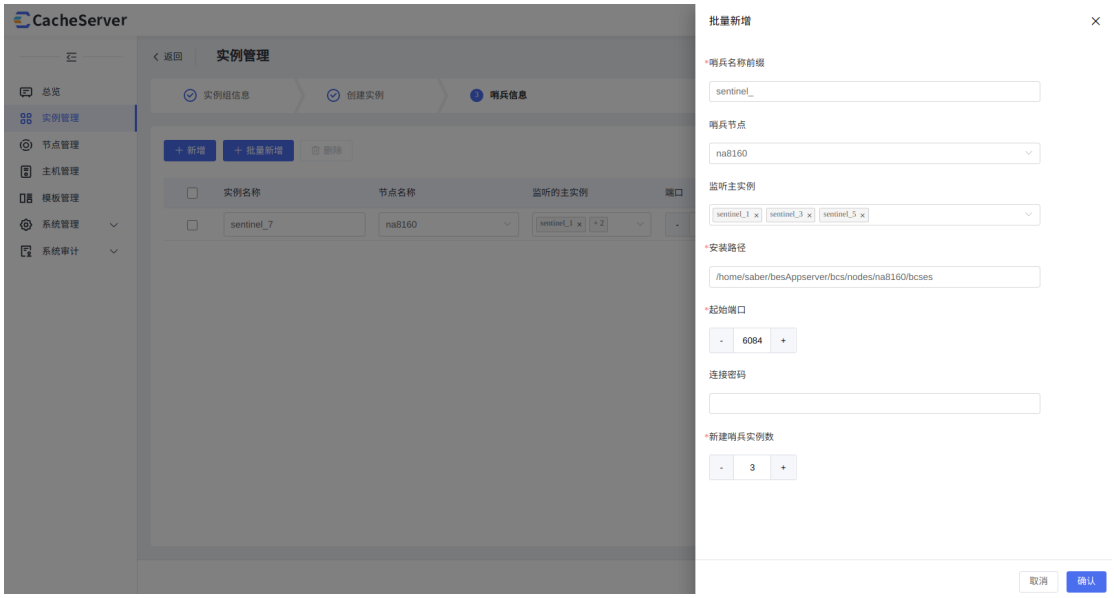

图 8-7 新增哨兵

（2）点击“**确认**”按钮完后，批量新增的哨兵实例会出现在界面上。

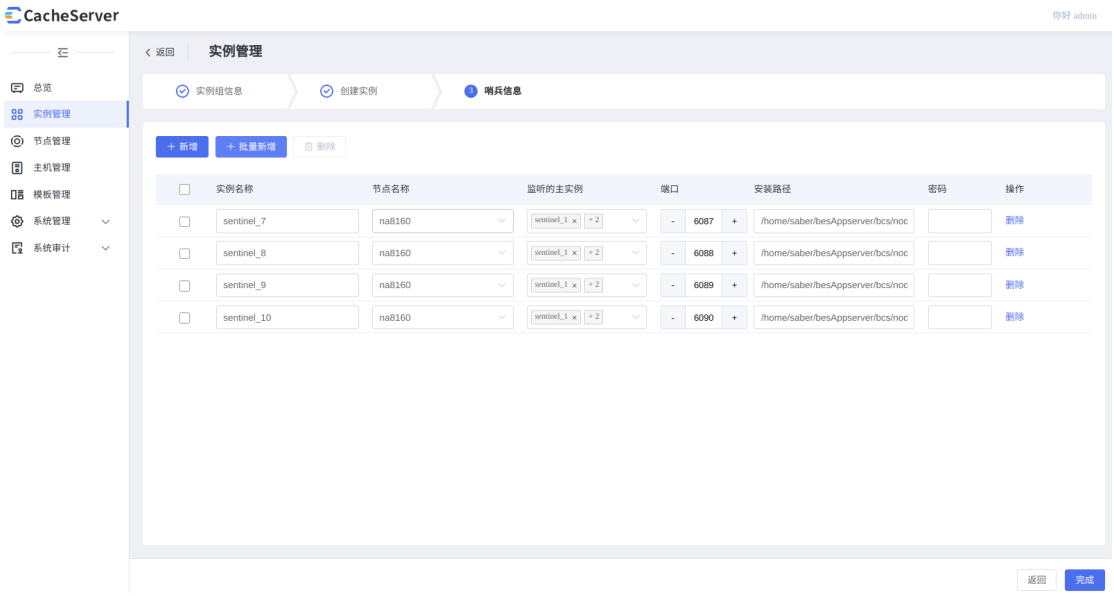

图 8-8 批量新建哨兵

（3）点击“**完成**”按钮完后，完成新建哨兵模式实例组。

（4）启动哨兵实例组，哨兵实例日志出现如下信息即代表哨兵模式组建成功。

```
499046:X 30 Jul 2022 16:25:09.252 # +monitor master sentinel_1 192.168.8.160
↪  6382 quorum 2
499046:X 30 Jul 2022 16:25:09.262 * +slave slave 192.168.8.160:6383 192.168.8
↪  .160 6383 @ sentinel_1 192.168.8.160 6382
499046:X 30 Jul 2022 16:25:11.119 * +sentinel sentinel e3e6232b74daa06c2e62b0
↪  e969263bbe47ebd65b 192.168.8.160 6386 @ sentinel_1 192.168.8.160 6382
499046:X 30 Jul 2022 16:25:11.184 * +sentinel sentinel 0a2be7cfc15543f2efd73b
↪  155b3b19eecdb8db2b 192.168.8.160 6385 @ sentinel_1 192.168.8.160 6382
```

8.2.3 集群模式

哨兵模式下，读写都是在主服务器上，这样子势必会造成性能瓶颈。BES CacheServer的集群模式很好的 解决了这个问题，集群模式是去中心实例方式实现的，每个实例之间都是相互连接、交换相互的状态并保存自己与其他实例的信息。建议用户采用三主三从6个实例实现集群模式。

（1）集群模式的创建和主从模式界面一致，创建完成后，需要先启动任意一个实例。

（2）在实例组页面点击创建好的集群，进入集群概览页面。可以看到当前集群状态是未初始化，点击“**构建集群**”按钮，开始构建集群。

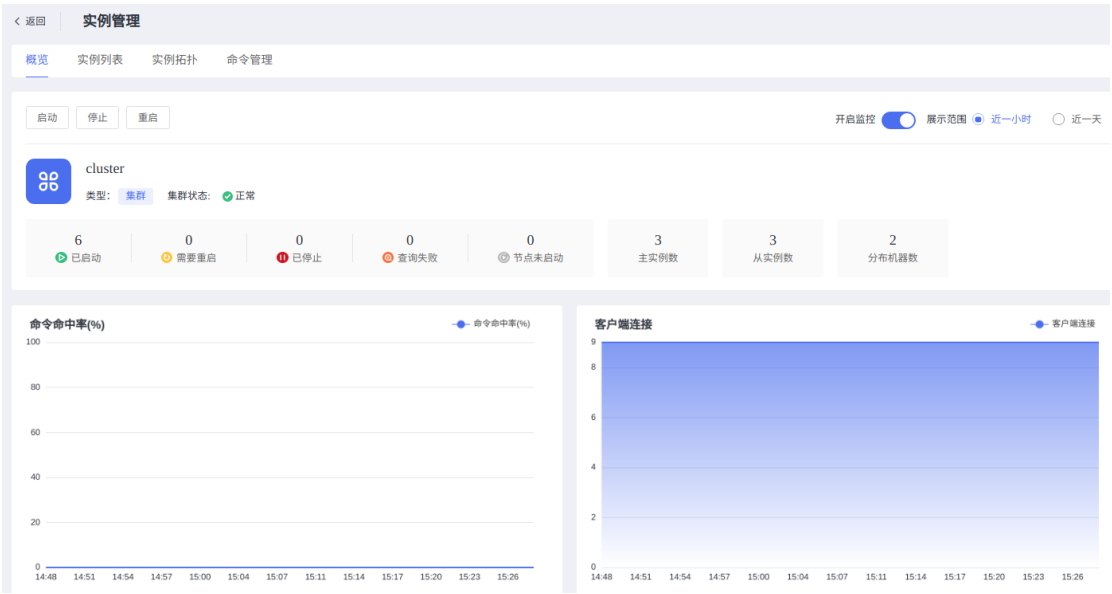

图 8-9 集群概览页面

（3）构建完城后，在实例列表页面，点击任意实例超链接，在命令执行页面执行如下命令，出现“**cluster_state:ok**”即代表集群构建成功。

图 8-10 集群状态

（4）构建完城后，集群实例列表如下，可以看到槽位范围，并进行移动槽位操作。

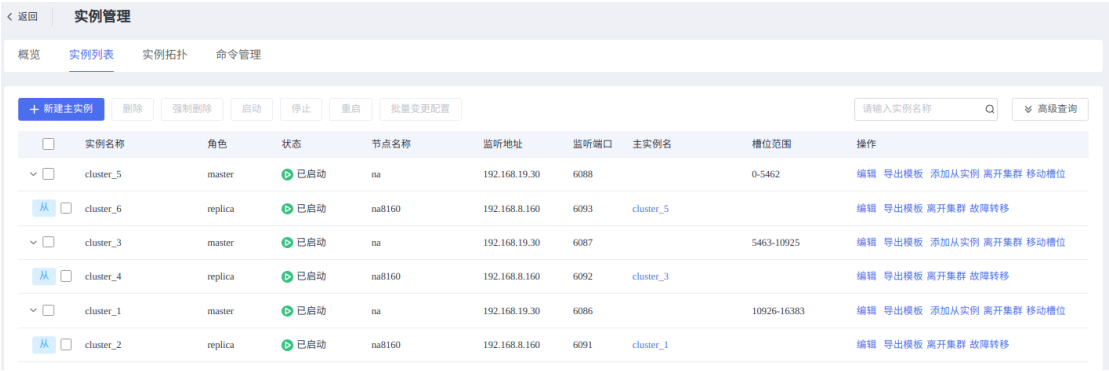

图 8-11 集群实例列表

（5）点击“**移动槽位**”后，可以选择目标实例、起始槽位和结束槽位。移动槽位后，会将当前的槽位范围移动到目标实例上。

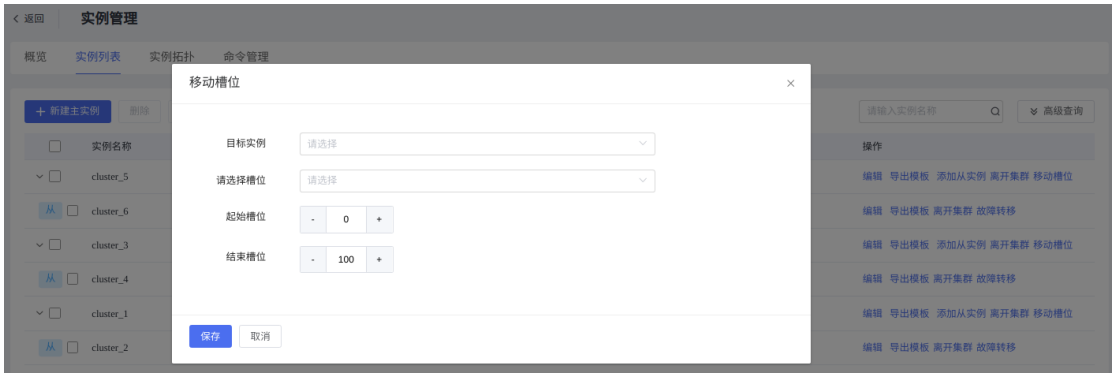

图 8-12 移动槽位

8.3 实例配置

8.3.1 概览

实例概览页面会展示当前实例组的基本信息，包括实例组类型，实例状态，实例操作，开启监控等。

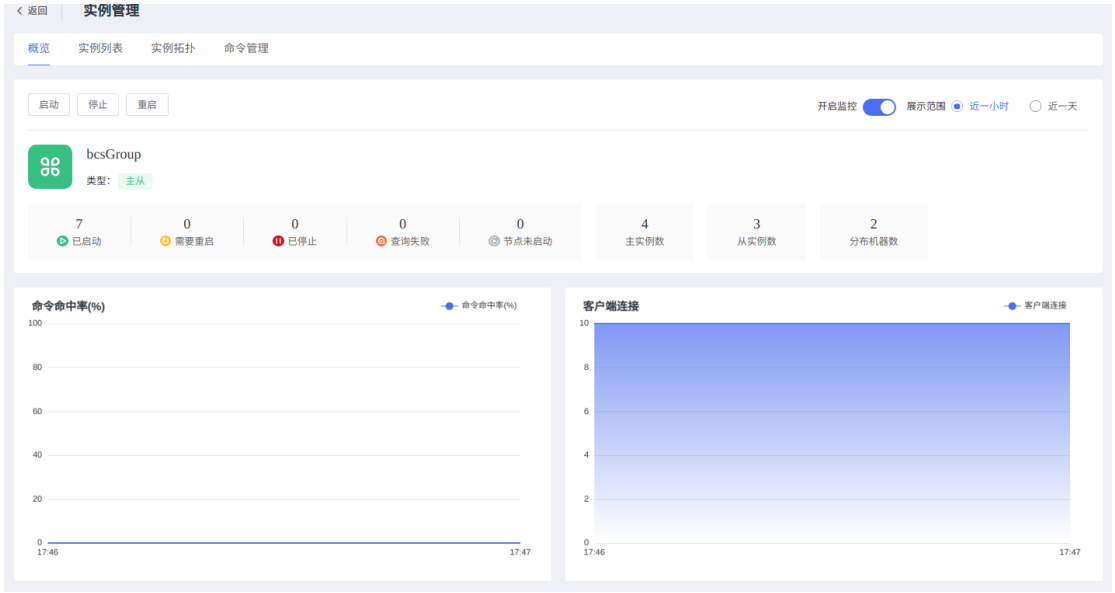

图 8-13 概览

8.3.2 实例列表

实例列表展示了当前实例组下的所有实例，并且可以对实例进行单独的操作，如导出模板、添加从实例、删除、强制删除、启动、停止、重启、批量变更配置和查询等。

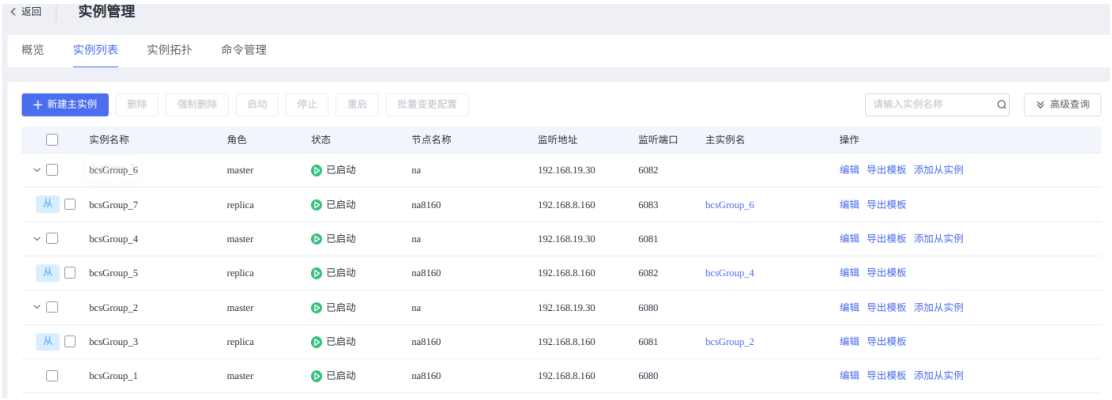

图 8-14 实例列表

8.3.3 实例配置批量变更

勾选要批量变更配置的实例，点击“**批量变更配置**”按钮，弹出批量变更配置对话框，用户可以选择添加或者删除配置，也可以自定义配置信息。

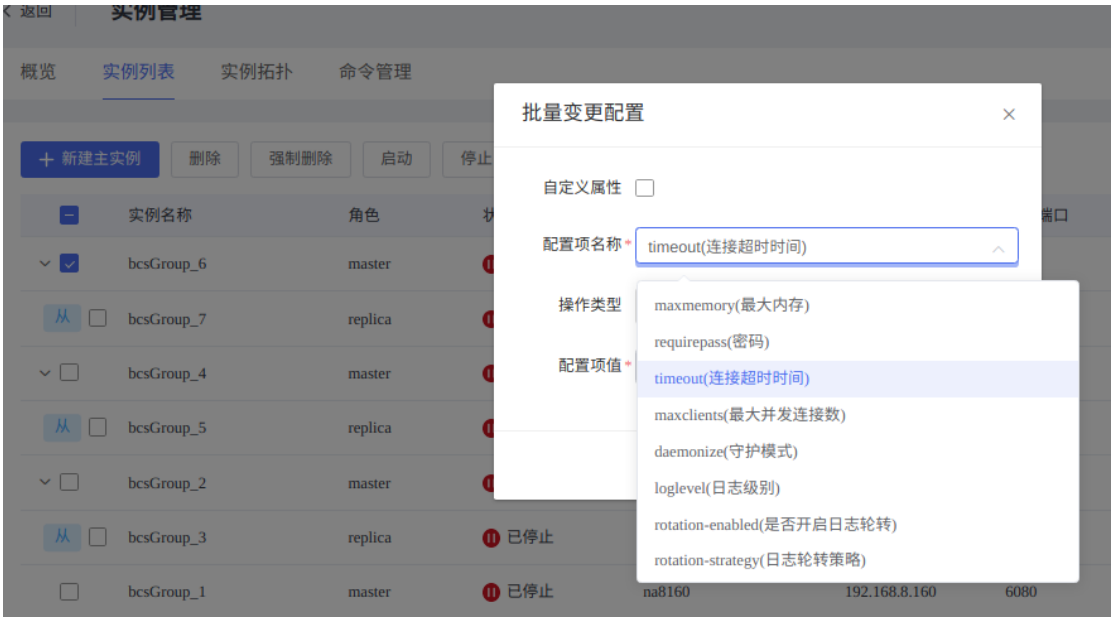

图 8-15 批量变更实例配置

点击任意实例名称后面的“**编辑**”超链接，进入实例的配置页面

图 8-16 实例基础配置

表 8-1 基础配置项

配置项名称	说明
监听地址	实例的监听地址
监听端口	实例的监听端口
最大内存	达到内存后，会尝试清除已到期或即将到期的Key，清除后，仍然超过最大内置设置的，将无法再进行写入操作。合法值：1 - 9007199254740992
连接密码	日志文件轮转的时间间隔。当文件轮转时间间隔为0时，按文件轮转大小限制进行轮转。合法值：0-2147483647，默认值：0，单位：分钟。
连接超时时间	客户端闲置时间超过该值后，自动断开连接，默认是0，表示不关闭，单位：秒
最大并发连接数	允许同时连接的客户端的最大数，默认不限制
存放路径	

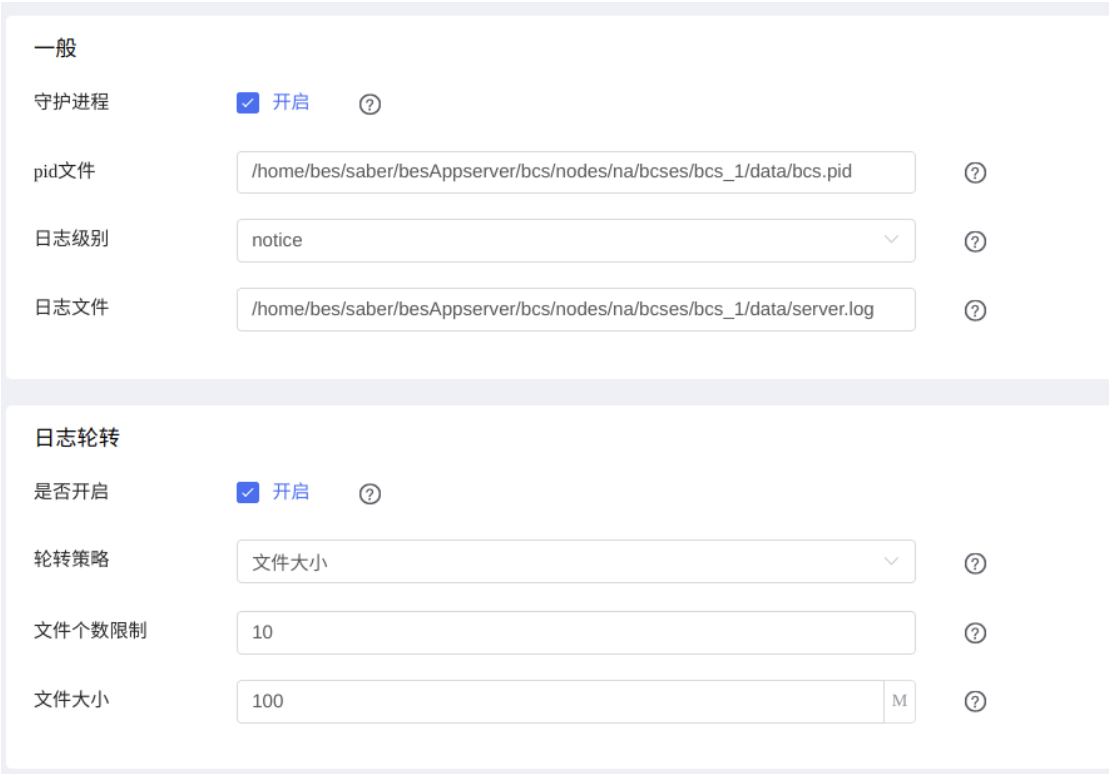

图 8-17 实例一般配置

表 8-2 一般配置项

配置项名称	说明
守护进程	如以守护进程运行，则需要指定pid文件，默认值：$instance_home/logs/bcs.pid\|\|pid$文件|若启用，在满足条件时生成新的日志文件。默认值：启用。|

表 8-3 日志轮转配置项

配置项名称	说明
是否开启	如以守护进程运行，则需要指定pid文件，默认值：${instance_home}/logs/bcs.pid
轮转策略	选择时间时，只根据时间进行轮转，选择文件大小时，根据日志文件的大小进行轮转，选择时间+文件大小时，任意一个满足条件即触发轮转
时间间隔	日志文件轮转的时间间隔。合法值：1~525600，默认值：1440，单位：分钟
文件个数限制	日志文件轮转的最大个数，达到上限后删除时间最早的日志文件。合法值：1-2147483647，默认值：10。
文件大小	指定日志文件的最大容量，达到上限后创建新的日志文件。合法值：1-2047，默认值：100，单位：M

图 8-18 主从复制配置

表 8-4 主从复制配置项

配置项名称	说明
是否只读	当实例角色为replica时，该配置才生效
允许脏数据	当从实例失去和主实例的连接后，是否继续处理客户端请求，存在访问到过期数据的风险，当实例角色为replica时，该配置才生效
心跳时间	以定义的心跳时间定期向主实例发送ping，默认值：10秒，当实例角色为replica时，该配置才生效。合法值：1 - 2147483647
复制超时时间	主从实例之间复制的超时时间，当检测超时后，会关闭当前连接，由从实例重新发起和主实例建立连接的请求,默认值：60秒。合法值：1 - 2147483647

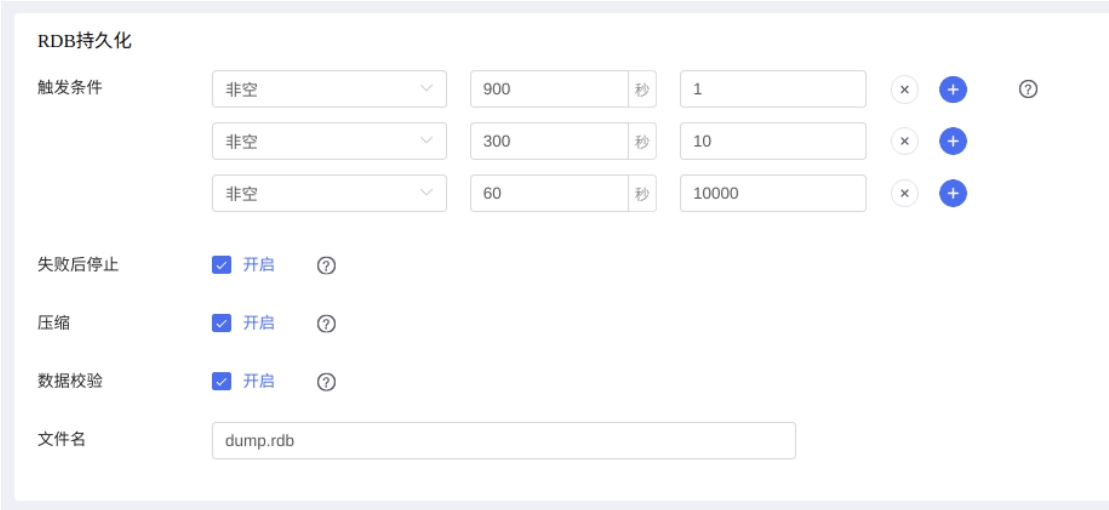

图 8-19 RDB持久化配置

表 8-5 RDB持久化配置项

配置项名称	说明
触发条件	在统计周期内，至少有指定数目的Key发生变化，才会触发写数据到磁盘，支持设置多个条件，满足任一一个即被触发
失败后停止	持久化失败后，实例停止接受更新操作，默认值：启用
压缩	储存在磁盘上中的快照，可以采用LZF算法进行压缩，默认值：启用
数据校验	使用CRC64算法进行数据校验，会增加一定的性能损耗，默认值：启用
文件名	持久化文件名称，默认值：dump.rdb

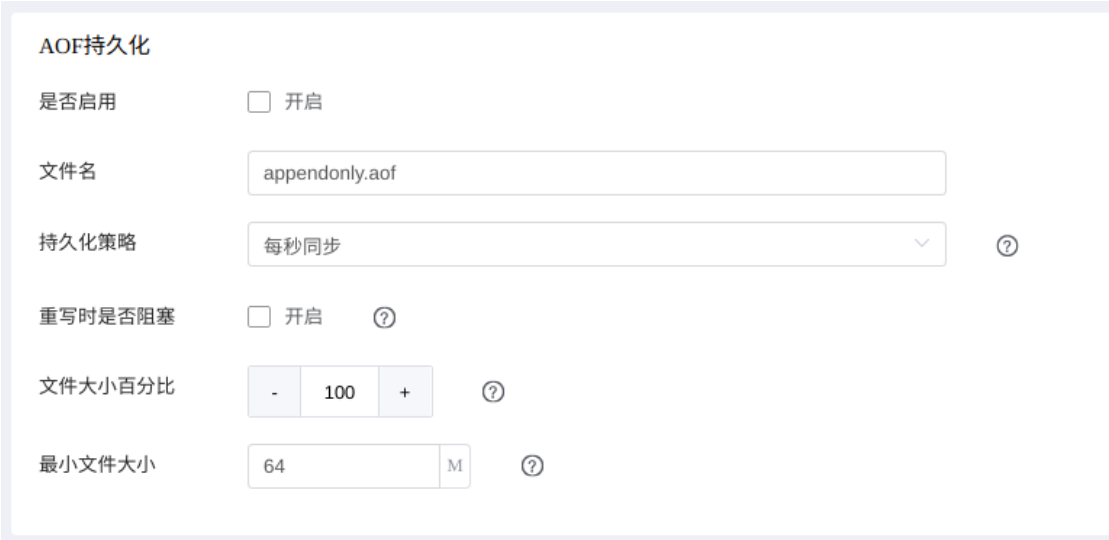

图 8-20 AOF持久化配置

表 8-6 AOF持久化配置项

配置项名称	说明
是否启用	AOF持久化是否开启，默认值：禁用
文件名	持久化文件名称，默认值：appendonly.aof
持久化策略	可选值：每次同步、每秒同步、操作系统控制。其中，每次同步：每次发生数据变更都会被立即记录到磁盘，性能差但数据完整性好;每秒同步：每秒记录，如果数据一秒内宕机，会导致少量数据丢失;操作系统控制：将缓存回写的策略交给系统，Linux默认是30秒
重写时是否阻塞	在AOF重写时，对新写入操作执行fsync会导致阻塞过长时间，对于延迟要求很高的应用，可以选择不勾选，新写入操作的记录会暂时存在内存中，等重写完成后再写入到磁盘。默认值：启用
文件大小百分比	当AOF文件大小超过上次重写的AOF文件大小的百分之多少时，自动触发重写，默认值：100，即当前AOF文件大小是上次日志重写时的2倍，自动启动新的日志重写过程

配置项名称	说明
最小文件大小	允许重写的最小文件大小，避免达到文件大小百分比后，文件很小的情况也触发重写

图 8-21 慢日志配置

表 8-7 慢日志配置项

配置项名称	说明
阈值	执行时间超过该阈值的，才会被记录成慢日志
最大条数	允许记录的最多慢请求条数，超过后会将最早的慢请求删除

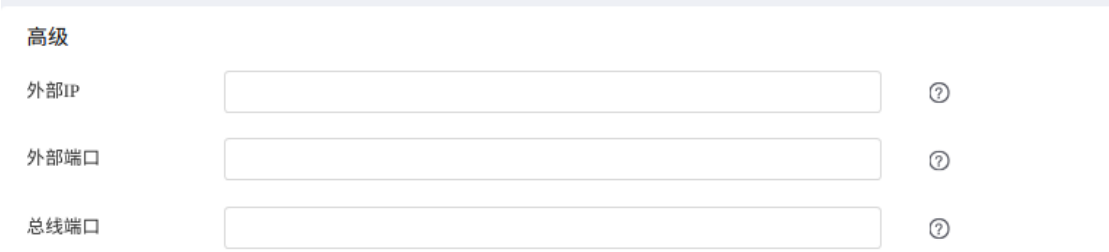

图 8-22 高级配置

表 8-8 高级配置项

配置项名称	说明
外部IP	在端口转发或者NAT网络环境中，实例有多个IP时，可以指定具体的IP地址
外部端口	在端口转发或者NAT网络环境中，指定实例的端口
总线端口	在端口转发或者NAT网络环境中，指定实例的总线端口

如果是从实例模式，实例会存在如下配置

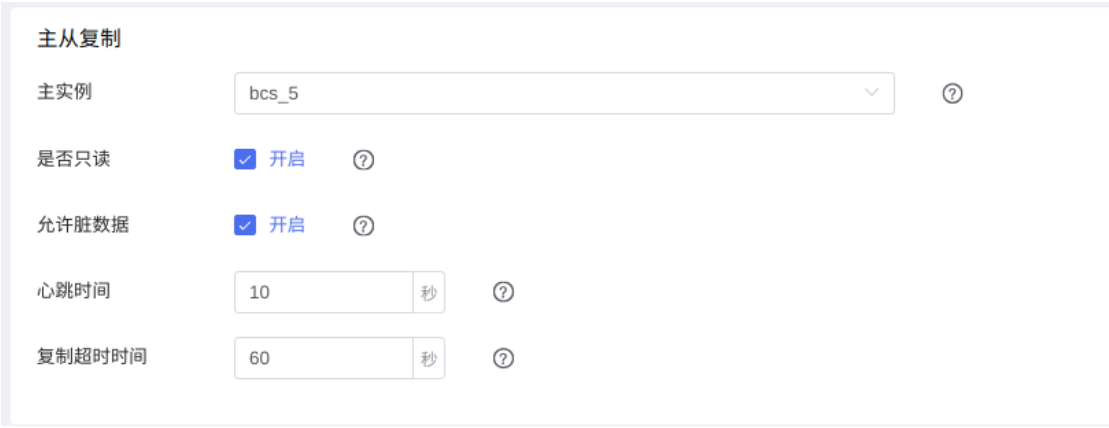

图 8-23 从实例配置

表 8-9 从实例配置项

配置项名称	说明
主实例	连接主实例的IP地址信息
是否只读	只读模式下，不允许客户端连接
允许脏数据	当从实例失去和主实例的连接后，是否继续处理客户端请求，存在访问到过期数据的风险
心跳时间	以定义的心跳时间定期向主实例发送ping，默认值：10秒
复制超时时间	主从实例之间复制的超时时间，当检测超时后，会关闭当前连接，由从实例重新发起和主实例建立连接的请求,默认值：60秒

如果是集群模式，实例会存在如下配置

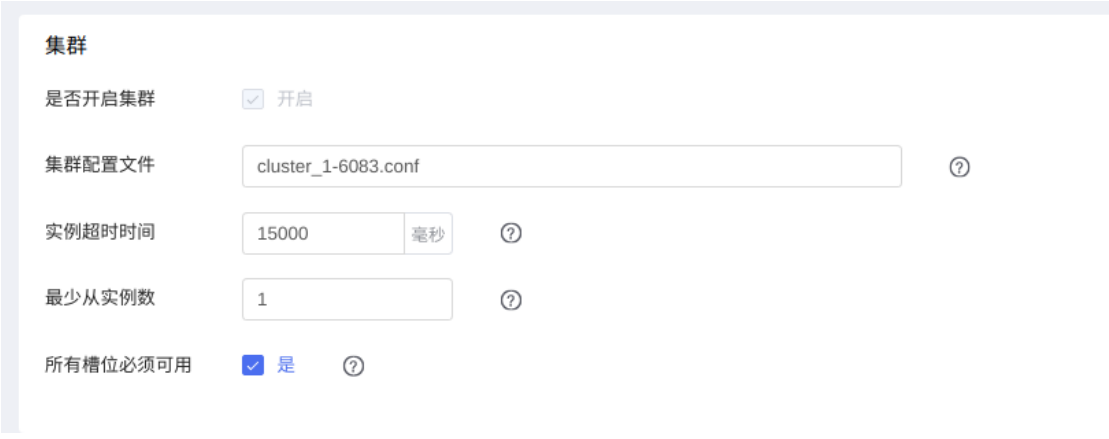

图 8-24 集群实例配置

表 8-10 集群实例配置项

配置项名称	说明
是否开启集群	是否开启集群
集群配置文件	每个集群实例都会有一个集群配置文件，由实例自动创建和维护
实例超时时间	集群实例能够失联的最长时间，超过后会被认为故障，如果主实例超过该时间还是不可达，则升级从实例成为主实例
最少从实例数	只有该主实例超过拥有该设置值数量的正常状态的从实例时，才会分配其下的从实例给集群中孤立的主实例。默认值：1，合法值：1 - 2147483647
所有槽位必须可用	检测到部分槽位没有可用的实例提供服务时，集群整体是否可用，默认值：是，即所有槽位必须可用，否则集群整体也将不可用

8.3.4 实例拓扑

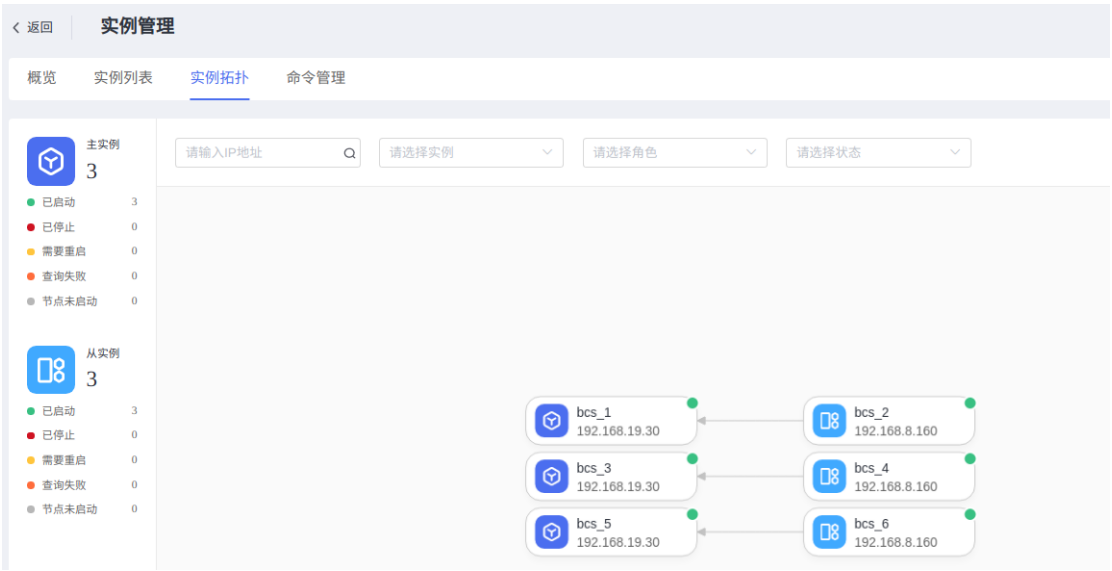

图 8-25 实例拓扑图展示

8.3.5 命令管理

BES CacheServer为了方便用户管理命令，可以针对一些命令进行禁用和改名。

图 8-26 命令列表

注意：重命名原命令名称，新的命令名将代替原命令名，原命令名将无法使用。

禁用命令之后，原命令也无法使用。

图 8-27 编辑命令

命令改名效果：

```
192.168.19.30:6080> kehuduan info
"id=5 addr=192.168.19.30:56538 laddr=192.168.19.30:6080 fd=10 name= age=1 idl
↪  e=0 flags=N db=0 sub=0 psub=0 multi=-1 qbuf=28 qbuf-free=40926 argv-mem=1
↪  2 obl=0 oll=0 omem=0 tot-mem=61468 events=r cmd=client user=default redir
↪  =-1\n"
192.168.19.30:6080> client info
ERR unknown command `client`, with args beginning with: `info`,
```

命令禁用效果：

```
192.168.19.30:6080> client info
ERR unknown command `client`, with args beginning with: `info`,
```

目前支持如下命令：

```
bgsave、client、command、config、debug、failover、flushall、flushdb、keys、
↪  lastsave、latency、module、monitor、pfdebug、pfselftest、reset、restore、
↪  restore-asking、save、sort、swapdb。
```

8.3.6 监控

在监控首页，可以看到已执行命令数、key总数、命令命中率、客户端连接数和每秒并发操作数，并提供资源使用、命令执行、Key统计、网络流量和客户端的详细监控。

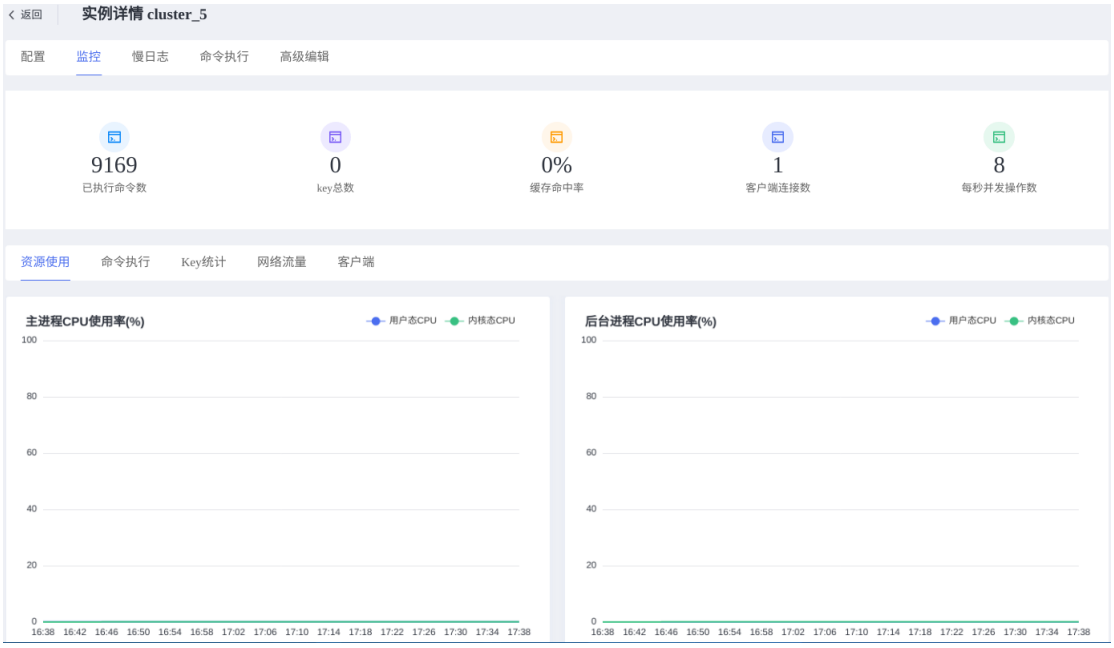

图 8-28 监控首页

8.3.6.1 资源使用

资源使用可以查看当前实例的主进程CPU使用率、后台进程CPU使用率、内存使用率和内存碎片使用率。

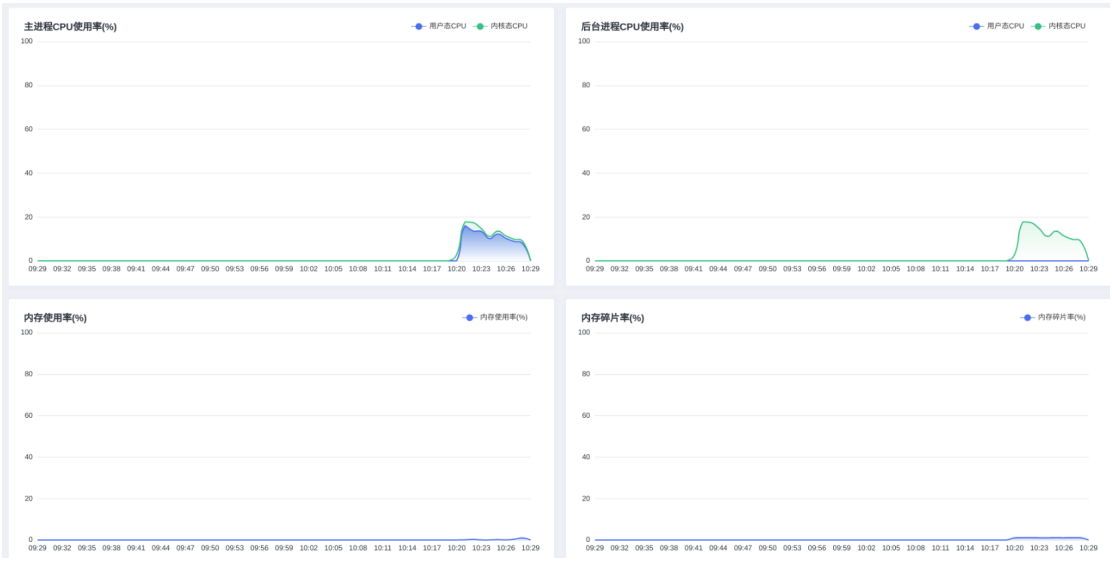

图 8-29 资源使用

8.3.6.2 命令执行

命令执行主要监控BES WebServer支持的各种命令，包括：执行命令数、每秒并发操作数、键命令数、String命令数、Hash命令数、List命令数、Set命令数和SortedSet命令数、

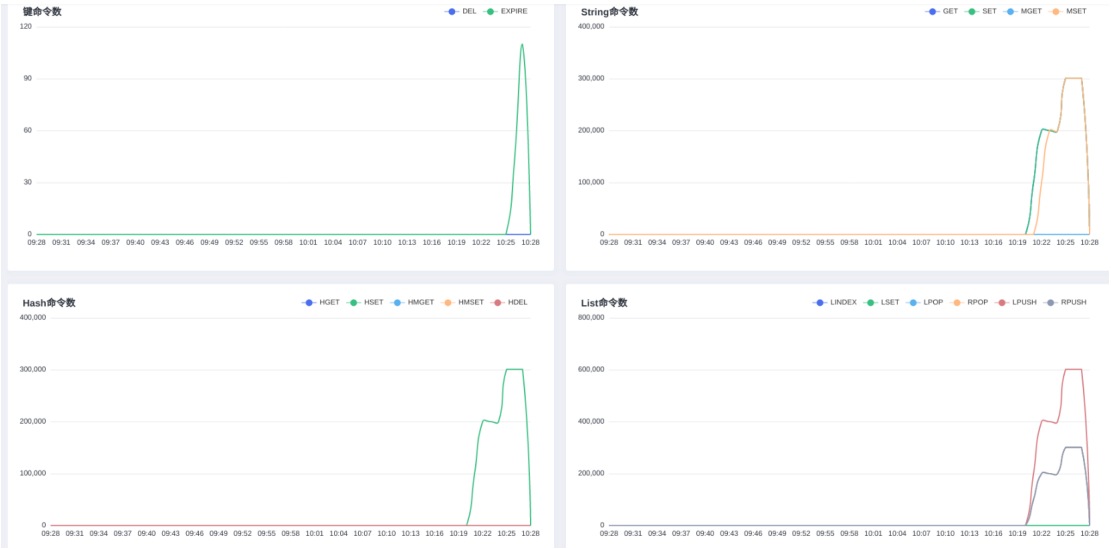

图 8-30 命令执行监控

8.3.6.3 Key统计

Key统计主要是监控Key总数、过期Key数、被删除Key数和命中情况。

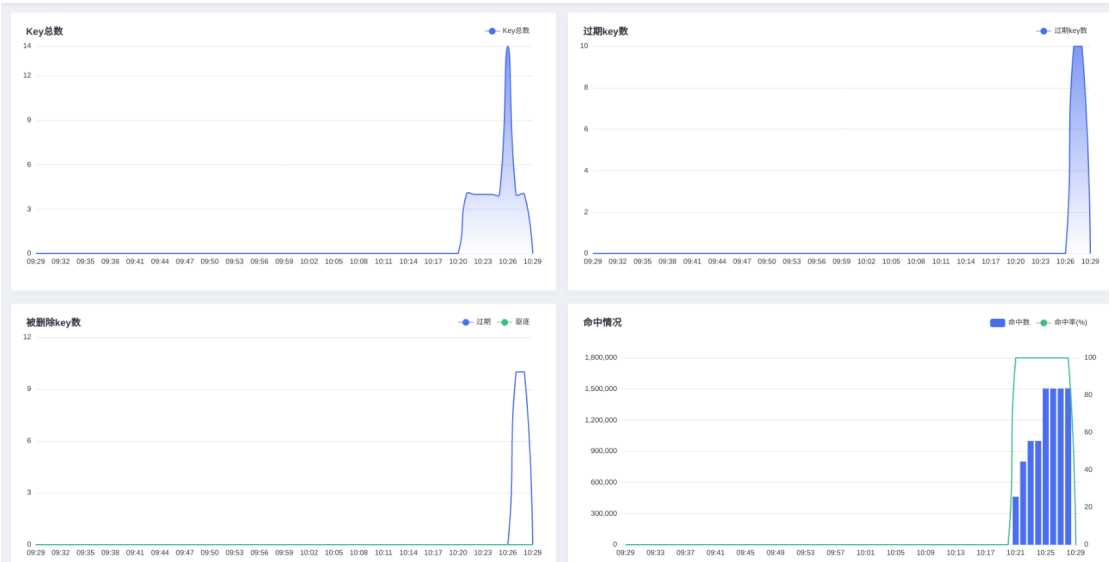

图 8-31 Key统计监控

8.3.6.4 网络流量

网络流量主要统计输入字节数和输出字节数。

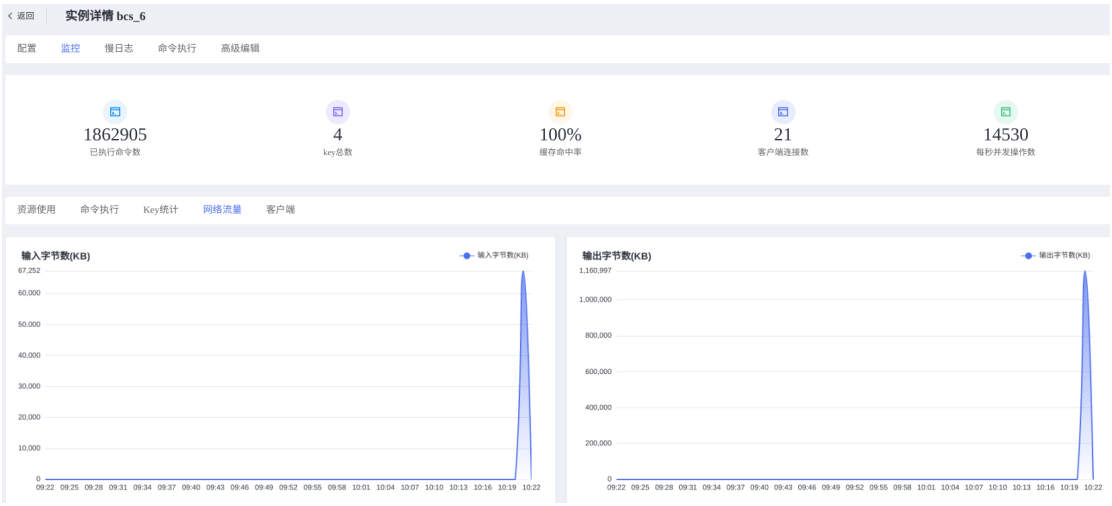

图 8-32 网络流量监控

8.3.6.5 客户端

客户端主要统计活跃的客户端连接、堵塞的客户端连接、新建连接和拒绝连接。

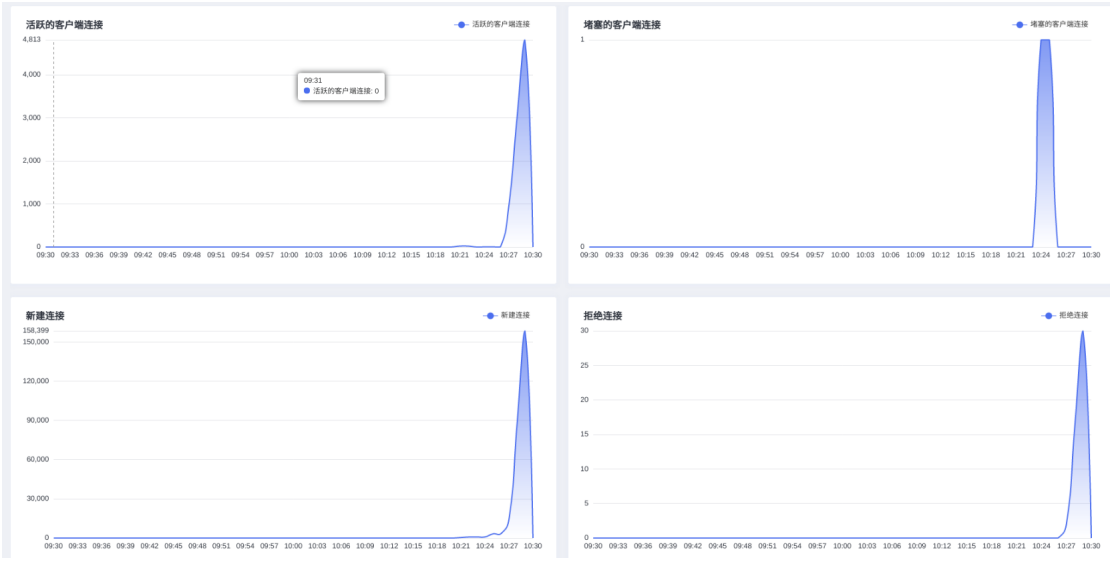

图 8-33 客户端监控

8.3.7 慢日志

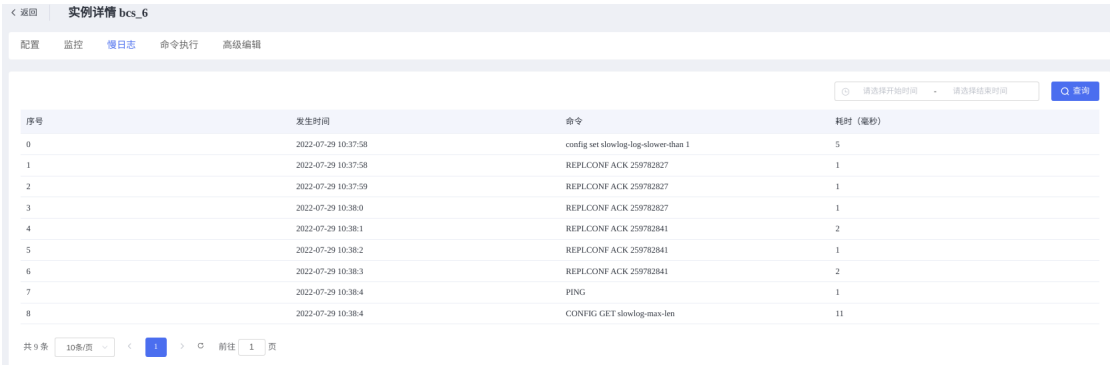

图 8-34 慢日志

8.3.8 命令执行

图 8-35 命令执行

8.3.9 高级编辑

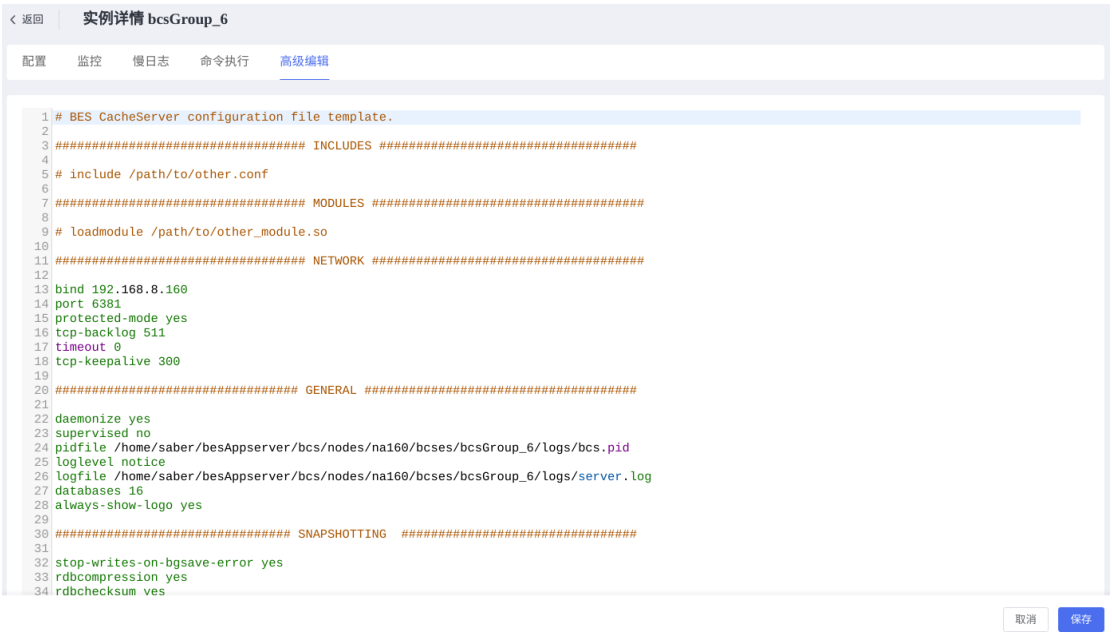

图 8-36 高级编辑

8.4 升级实例

在管理中心升级实例：

1) 替换当前BCS_HOME/media/下的实例介质，命名规则参考BCS-3.0.2-64.tar.gz。

2) 重启管理控制台。

3) 停止实例，然后在“**节点管理**”->“**编辑节点**”页面，点击“**重建实例介质**”按钮即可。

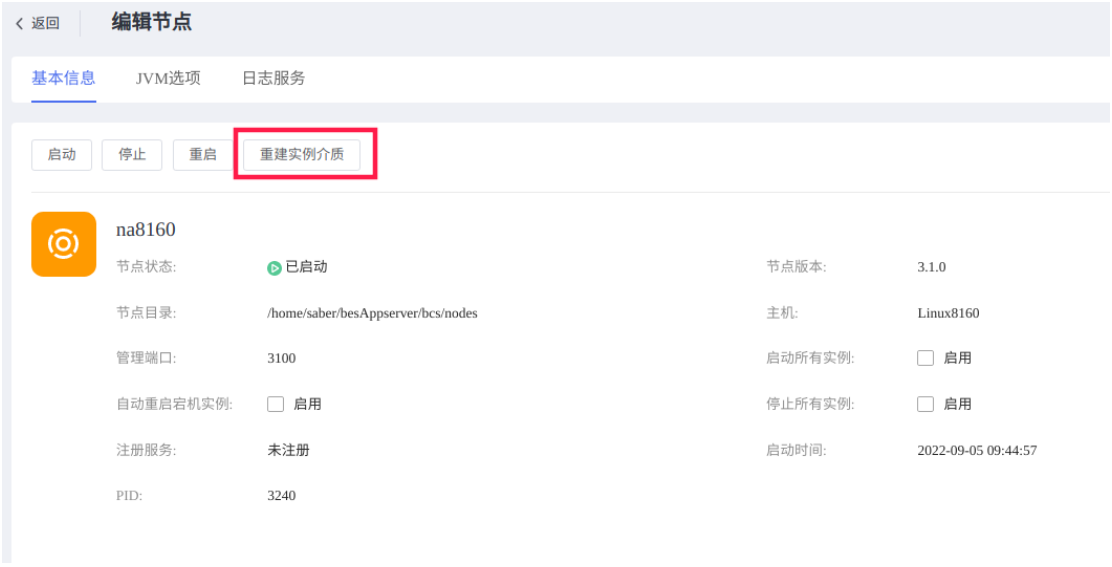

图 8-37 重建实例介质

第9章 模板管理

如果用户想要创建多个具有同样配置的节点、BCS实例，可以使用模板功能。模板功能允许用户自定义所有配置。

9.1 模板列表

在管理中心查看模板列表：

1) 在管理中心左侧导航区点击“**模板管理**”，操作区域显示模板列表页面，显示所有已添加到管理中心的模板。

图 9-1 模板列表

2) 点击模板列表操作栏里的“**高级编辑**”链接，进入“**编辑模板**”页面，该页面是模板的基本配置，在该页面可修改模板的配置信息。

3) 点击“**导出模板**”可以将当前模板导出。

9.2 新建模板

在管理中心新建模板：

1) 在管理中心左侧导航区点击“**模板管理**”，进入“**模板列表**”页面。

2) 在“**模板列表**”页面，点击“**新建**”按钮，进入“**新建模板**”页面，可配置项如下：

图 9-2 新建模板

表 9-1 新建模板配置项

配置项名称	说明
模板名称	模板的唯一标识。
类型	新建模板的类型。可选值为：主从实例配置、集群实例配置、哨兵实例配置和节点配置。
创建模式	可选值为：基于已有模板或者实例。导入模板。
参考模板	选择管理中心的配置文件中已存在的配置模板，模板将在选择配置内容基础上进行新建。可选值为：defaultBCSMasterConf(主从实例配置) defaultBCSClusterConf(集群实例配置) defaultBCSSentinelConf（哨兵实例配置）和 defaultNodeConf（节点配置模板）。
描述	模板的描述信息。

9.3 编辑模板

用户可以在模板编辑页面，自定义模板的配置，从而完成定制化实例或者节点的需求。

在管理中心编辑模板：

1. 在管理中心左侧导航区点击“**模板管理**”，进入模板列表页面。
2. 点击模板操作栏里的“**编辑**”超链接，进入编辑模板页面

图 9-3 编辑模板

3. 点击“**保存**”按钮完成修改。

9.4 删除模板

在管理中心删除模板：

1) 在管理中心左侧导航区点击“**模板管理**”，进入模板列表页面。

2) 勾选要删除的模板，点击“**删除**”按钮删除模板，默认模板defaultBCSMasterConf、defaultBCSClusterConf、defaultBCSSentinelConf、defaultNodeConf不可删除。

第10章 系统管理

10.1 用户管理

BES CacheServer提供默认的管理员admin（系统组、安全组和审计组）、审计员auditadmin（审计组）和安全管理员secadmin（安全组），这三个用户不可删除，并且默认密码都是B#2008_2108#es。

系统管理用户可以新建用户组，并赋予相应的角色。

10.1.1 用户列表

在管理中心左侧导航区点击“**系统管理**”->“**用户管理**”，操作区域显示用户列表页面，显示所有已添加到管理中心的用户。

图 10-1 用户列表

为方便管理用户密码，安全管理员secadmin可以在控制台“**系统管理**”->”**用户管理** “，使用“**安全策略**”对登录密码长度、连续登录失败次数、限制登录间隔和用户密码有效时间进行设置。

图 10-2 安全策略页面

10.1.2 新建用户

新建用户之前，需要先创建角色，有关角色相关介绍，请参考角色管理章节。

管理员在管理中心新建管理用户：

1) 在管理中心左侧导航区点击“**系统管理**”->“**用户管理**”，进入“**用户列表**”页面。

2) 在“**用户列表**”页面，点击“**新建**”按钮，进入“**新建用户**”页面，可配置项如下：

图 10-3 新建用户

表 10-1 新建用户配置项

配置项名称	说明
用户名称	用户的名称。
密码	用户密码。
确认密码	再次输入用户密码。
描述	用户的描述信息。

3) 配置上用户的所有属性，点击“**保存**”按钮完成新建用户。

10.1.3 分配角色

安全管理员新建完用户之后，需要在用户列表页面进行角色的分配，在用户列表页面，点击“**选择角色**”超链接，进入“**选择角色**”页面，为当前新建的用户选择角色。

图 10-4 选择角色页面

点击“**保存**”按钮完成角色的选择。

10.1.4 编辑用户

安全管理员在管理中心编辑用户：

1) 在管理中心左侧导航区点击“**系统管理**”->“**用户管理**”，进入用户列表页面。

2) 点击用户名称操作栏里的“**编辑**”超链接，进入“**编辑用户**”页面，可编辑项目如下：

图 10-5 编辑用户页面

安全管理员可以禁用当前用户，禁用之后，当前用户将无法登录。

3) 修改用户的属性，点击“**保存**”按钮保存修改的属性。

10.1.5 解锁用户

当用户登录时，连续输入密码错误超过安全策略设置的次数，用户会被锁定，可以使用安全管理员secadmin来解锁用户

1) 在管理中心左侧导航区点击“**系统管理**”->“**用户管理**”，进入“**用户列表**”页面。勾选要解锁的用户，点击“**解锁**”按钮即可。

图 10-6 解锁用户

10.1.6 修改用户密码

用户可以修改自己的密码，或者由安全管理员修改密码。

1) 安全管理员修改普通用户密码，在用户列表页面，点击要修改密码用户的操作栏“**修改密码**”超链接，进入修改密码页面，输入新密码和确认密码，点击“**保存**”按钮即可完成密码修改。

图 10-7 安全管理员修改普通用户密码

2) 普通用户修改自己的密码，可以在管理控制台右上角，点击“**个人信息**”->“**修改密码**“，进入”**修改密码**“详情页面。输入旧密码、新密码和确认密码完成密码修改。

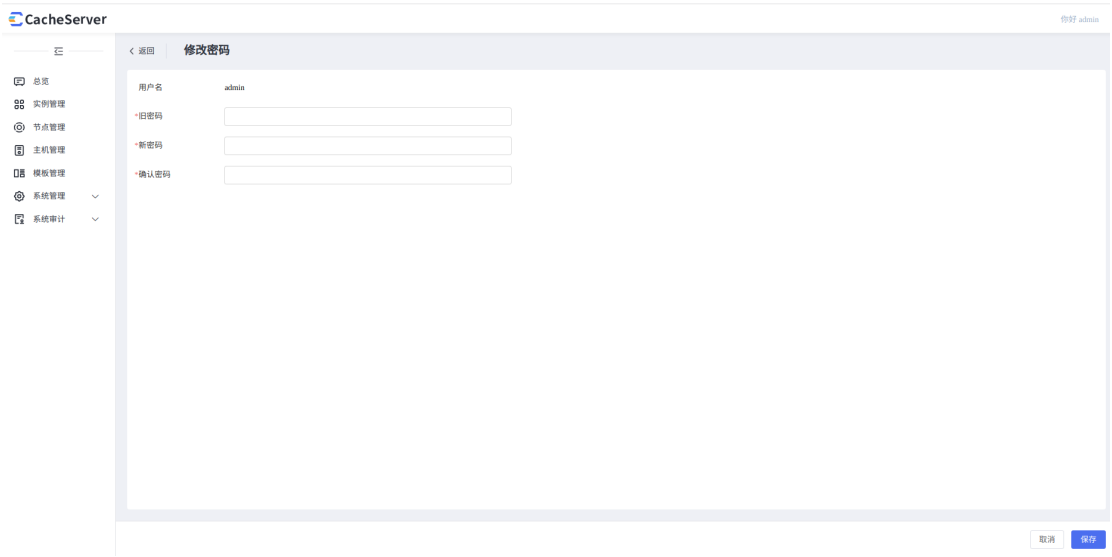

图 10-8 普通用户修改密码详情页

10.1.7 删除用户

安全管理员在管理中心删除用户：

1) 在管理中心左侧导航区点击“**系统管理**”->“**用户管理**”，进入用户列表页面。

2) 勾选要删除的用户，点击“**删除**”按钮删除即可。

10.2 角色管理

使用具有角色管理权限的用户登录管理中心，具有该权限的默认用户为secadmin，密码为B#2008_2108#es。

10.2.1 角色列表

在管理中心左侧导航区点击“**系统管理**”->“**角色管理**”，进入角色列表页面，显示所有已添加到管理中心的角色名。默认存在如下角色：

图 10-9 角色列表

用户可以根据自己的实际场景选择合适的角色或者新建角色分配相应的权限。

10.2.2 新建角色

在管理中心新建角色：

1) 在管理中心左侧导航区点击“**系统管理**”–>“**角色管理**”，点击“**新建**”按钮进入“**新建角色**”页面。

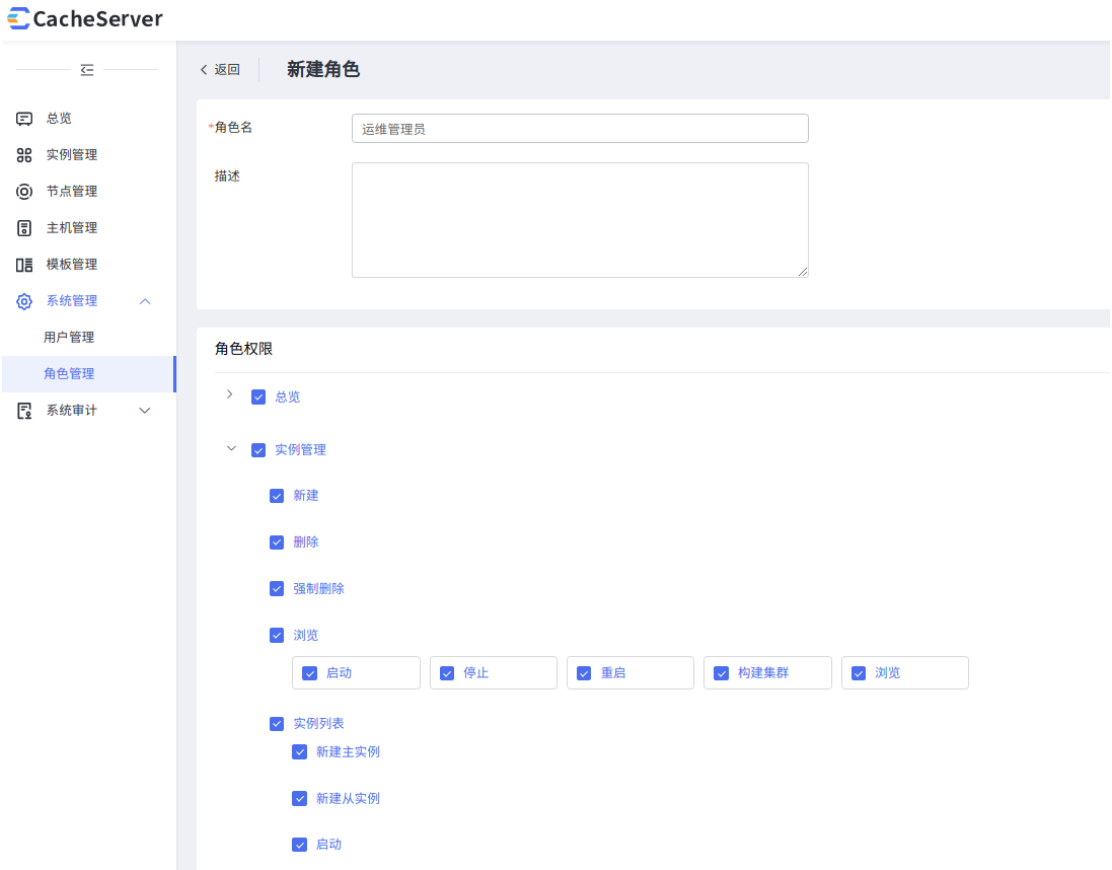

图 10-10 新建角色

2) 勾选当前用户要分配的权限，点击保存按钮，完成角色新建。

10.2.3 删除角色

管理员在管理中心删除角色：

1) 在管理中心左侧导航区点击“**系统管理**”->“**角色管理**”，进入“**角色列表**”页面。

2) 在“**角色列表**”页面，勾选要删除的角色，点击“**删除**”按钮删除即可。

第11章 系统审计

11.1 登录审计

BES CacheServer管理平台可以对用户登录和登出操作进行审计。主要支持按操作时间、按用户名和按操作类型进行查询，使用具有审计权限的用户登录管理中心，默认具有该权限的默认用户为auditadmin，密码为B#2008_2108#es。

审计主要内容如下：

开始时间、结束时间、用户、源IP、操作动作（登录和登出）、操作对象和状态。

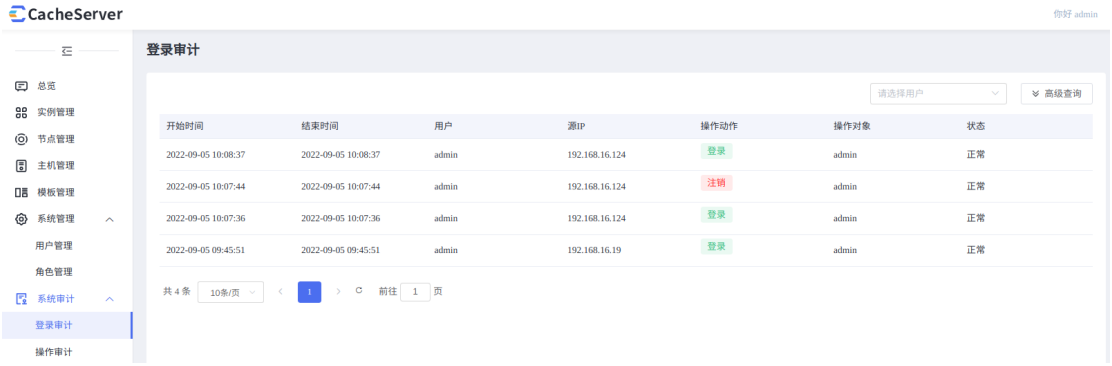

图 11-1 登录审计

11.2 操作审计

BES CacheServer管理平台可以对用户操作进行审计。主要支持按操作时间、按用户名、按功能模块和按操作类型进行查询。

审计主要内容如下：

开始时间、结束时间、用户、源IP、操作动作、操作对象和状态。

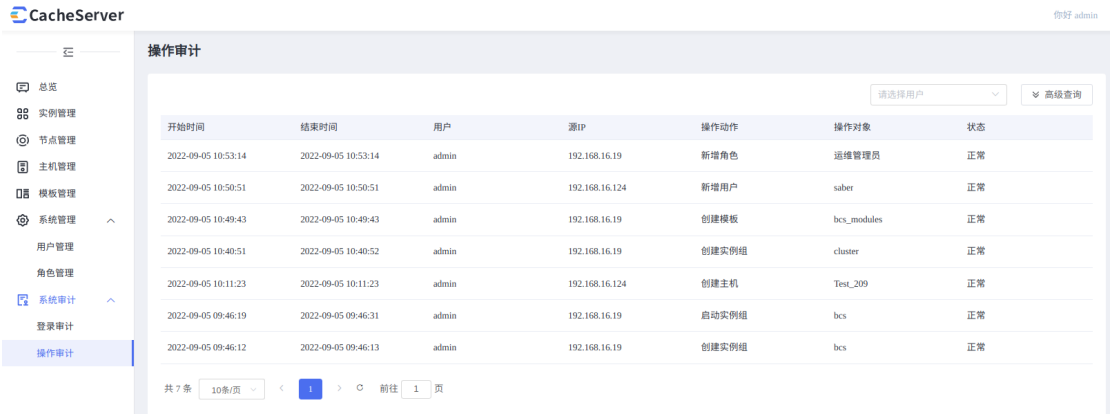

图 11-2 操作审计

审计日志默认7天删除,用户可以在BCS_HOME/system/console/WEB-INF/classes/config.properties文件中通过bcs.config.audit.storageTime 属性配置，默认单位是秒。配置7天=7*24*60*60。

第12章 补丁管理

12.1 关于补丁包

当管理中心或者节点发布新的特性时，有时会以补丁包的形式发布，管理中心提供了补丁包的安装工具Patch，将补丁包安装到管理中心上。

12.2 命名规则

管理中心补丁包都以zip文件形式存在，并符合固定的命名规则，如BCS3.1.0.5590.001.zip。

补丁的命名规则与补丁包基本一致,但以jar文件形式存在,并以V开头,如V3.1.0.5590.001.jar。

具体命名规则如下:

表 12-1 补丁命名规则

版本号	说明	示例
第一位版本号	代表软件更新换代。	如V**3.1.0**.5590
第二位版本号	软件有计划的大的功能升级。	如V3.1.0.5590
第三位版本号	软件有计划的小的功能升级。	如V3.1.0.5590
第四位版本号	发布的ID编号。	如V3.1.0.**5590**
在版本号后加小版本加T	临时紧急客户补丁编号。	如V3.1.0.5590.001.**T001**
在版本号后加小版本	正式紧急客户补丁编号。	如V3.1.0.5590.**001**

12.3 PATCH命令

在BES_HOME/bin目录下，运行patch.bat(Windows)或./patch(Linux/Unix)命令:

1) 多个指定的jar包一起打补丁，多个jar文件之间使用操作系统的系统分隔符分隔:

Windows：输入patch.bat -jar file1[;file2...]

Linux/Unix：输入./patch -jar file1[:file2...]

2) 多个指定的目录下的文件一起打补丁，多个目录之间使用操作系统的系统分隔符分隔:

Windows：输入patch.bat -path path1[;path2...]

Linux/Unix：输入./patch -path path1[:path2...]

3) 列出所有已经安装的补丁:

Windows：输入patch.bat -list

Linux/Unix：输入./patch -list

4) 回退到某个生效的补丁，可以通过patch -list命令查看已经安装的补丁:

Linux/Unix：输入./patch -revert patchVersion

如：回退到1号正式补丁，1号补丁之后的补丁所作的修改将被回退:

```
patch.bat -revert BCS3.1.0.5590.001
```

5) 回退掉所有已经安装的补丁：

Windows：输入patch.bat -revert

Linux/Unix：输入./patch -revert

6) 补丁进程如果被意外终止，如：断电、内存溢出等情况，下次运行patch命令时程序会自动检测备份信息，并回退掉上次安装过程被异常中断的补丁。

7) 如果执行patch命令安装补丁过程中，需要安装的补丁包的优先级小于最大已安装的补丁优先级，则该补丁将被过滤而不会被安装。如：

```
patch.bat -jar BCS3.1.0.5590.002.zip

Patch D:/patch/BCS3.1.0.5590.002.zip is applied successfully.

patch.bat -jar BCS3.1.0.5590.001.zip

BCS3.1.0.5590.001.zip has been applied already, which will be ignore.
```

12.3.1 补丁种类

补丁分以下几种：

1) 临时紧急客户补丁：如V3.1.0.5590.001.T001。

2) 正式紧急客户补丁：如V3.1.0.5590.001。

12.3.2 过滤和排序

未来可能会出现很多临时和正式补丁，这些补丁并不需要用户关心和整理，补丁工具会自行判断哪些补丁应该生效。

具体过滤和排序机制如下：(示例中 <= 表示共存但有序 » 表示过滤)

1) 基于同一个正式版本的多个补丁，只有一个补丁生效。

2) 生效序列为：正式补丁 » 高版本临时补丁 » 低版本临时补丁。

如：Vxxx.001 » Vxxx.001.**T002** » Vxxx.001.**T001**

3) 基于不同正式版本的非累计补丁，不互相影响，但要按照正式版本由小到大的顺序应用补丁。

如：Vxxx.**003**.T001 <= Vxxx.**002**.T002 <= Vxxx.**001**

12.4 节点升级

当节点发布新的特性时，有时会以补丁包的形式发布，BES CacheServer提供了节点补丁升级命令patch –node、回滚命令revert –node、节点补丁工具patch，并支持在管理控制台对节点补丁进行升级维护。

节点补丁包都以zip文件形式存在，并符合固定的命名规则，如BCD3.0.2.5590.001.zip。

补丁的命名规则与补丁包基本一致,但以jar文件形式存在,并以V开头,如V3.0.2.5590.001.jar。

BES CacheServer的iastoo工具提供了patch –node和revert –node命令来升级/回滚补丁。

1) 将节点补丁文件放到BES_HOME/media/nodepatch目录中。

2) 停止要升级/回滚的节点。

3) 使用如下命令升级补丁:

```
BCS_HOME/bin/iastool patch --node --patchnames BCS3.0.2.5590.001.zip
node1,node2...
```

注意：如果不指定–patchnames参数，则默认升级nodepatch目录下的所有补丁。

4) 使用如下命令回滚补丁:

```
BCS_HOME/bin/iastool revert --node --patchversionname BCS3.0.2.5590.001
node1,node2...
```

注意：如果不指定–patchversionname参数，则默认回滚所有已升级的补丁。

第13章 实例特性

13.1 数据类型

BCS支持五种基本数据类型：string（字符串）、hash（哈希）、list（列表）、set（集合）及zset（sorted set：有序集合），以及四种特殊的数据类型：BitMap（位存储）、HyperLogLog（基数统计）、GEO（地理位置）、Stream（消息队列）。

13.1.1 String

String是BCS最基本的类型，可以理解成与Memcached一模一样的类型，一个key对应一个value。

String类型是二进制安全的。意思是BCS的String可以包含任何数据。比如字符串（简单的字符串、复杂的字符串（例如JSON、XML））、数字（整数、浮点数），甚至是二进制（图片、音频、视频）。

一个键最大能存储512MB。

13.1.1.1 示例

```
127.0.0.1:> set name "BCS"
OK
127.0.0.1:7878> get name
"BCS"
```

在以上实例中我们使用了String 的 **set** 和 **get** 命令。键为 name，对应的值为BCS。

注意：一个键最大能存储512MB。

13.1.1.2 应用场景示例

- 访问量统计：每次访问博客和文章使用 INCR 命令进行递增。
- 请求限速功能，防止大量恶意请求。
- 将数据以二进制序列化的方式进行存储，作缓存。
- 共享session。
- 分布式锁。

13.1.2 Hash

BCS Hash 是一个键值对集合，是一个String类型的field和value的映射表，hash特别适合用于存储对象。

13.1.2.1 示例

```
127.0.0.1:7878> hmset user:1 username bcs password 111111
OK
bcs 127.0.0.1:7878> hgetall user:1
1) "username"
2) "bcs"
3) "password"
4) "111111"
127.0.0.1:7878>
```

以上实例中 hash 数据类型存储了包含用户脚本信息的用户对象。实例中我们使用了Hash的 **hmset, hgetall** 命令，**user:1** 为键值，**username**和**password**为属性。每个 hash 可以存储 232 - 1 键值对（40多亿）。

13.1.2.2 应用场景

- 存储、读取、修改对象属性，比如：用户（姓名、性别、爱好），文章（标题、发布时间、作者、内容）。

13.1.3 List

BCS List是简单的字符串列表，按照插入顺序排序。可以添加一个元素导列表的头部（左边）或者尾部（右边）。

13.1.3.1 示例

```
127.0.0.1:7878> lpush listtest bcs
(integer) 1
127.0.0.1:7878> lpush listtest bes
(integer) 2
127.0.0.1:7878> lpush listtest besmq
(integer) 3
127.0.0.1:7878> lrange listtest 0 10
1) "besmq"
2) "bes"
3) "bcs"
127.0.0.1:7878>
```

列表最多可存储 232-1元素 (40多亿)。

13.1.3.2 应用场景

- 最新消息排行等功能（比如朋友圈的时间线）。
- 消息队列。

13.1.4 Set

BCS的Set是String类型的无序集合，是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

13.1.4.1 示例

```
127.0.0.1:7878> sadd settest bcs
(integer) 1
127.0.0.1:7878> sadd settest bes
(integer) 1
127.0.0.1:7878> sadd settest besmq
(integer) 1
127.0.0.1:7878> sadd settest besmq
(integer) 0
127.0.0.1:7878> smembers settest
1) "besmq"
2) "bes"
3) "bcs"
```

注意：以上实例中besmq添加了两次，但根据集合内元素的唯一性，第二次插入的元素将被忽略。集合中最大的成员数为 232 - 1 (40多亿个成员)。

13.1.4.2 应用场景

- 共同好友。
- 利用唯一性，统计访问网站的所有独立ip。
- 好友推荐时，根据tag求交集，大于某个阈值就可以推荐。

13.1.5 Zset

BCS Zset 和 set 一样也是string类型元素的集合，且不允许重复的成员。不同的是每个元素都会关联一个double类型的分数。bcs正是通过分数来为集合中的成员进行从小到大的排序。

Zset的成员是唯一的,但分数（score）却可以重复。

```
#添加元素到集合，元素在集合中存在则更新对应score
zadd key score member
```

13.1.5.1 示例

```
127.0.0.1:7878> zadd zsettest 0 bcs
(integer) 1
127.0.0.1:7878> zadd zsettest 0 bes
(integer) 1
127.0.0.1:7878> zadd zsettest 0 besmq
```

```
(integer) 1
127.0.0.1:7878> zadd zsettest 0 besmq
(integer) 0
127.0.0.1:7878> ZRANGEBYSCORE zsettest 0 1000
1) "bcs"
2) "bes"
3) "besmq"
```

注意：以上实例中 besmq添加了两次，但根据集合内元素的唯一性，第二次插入的元素将插入失败。

13.1.5.2 应用场景

- 排行榜，取TopN操作。

13.1.6 Bitmap

BCS Bitmap并不是实际的数据类型，是定义在String类型上的一个面向字节操作的集合，可以说是byte数组。字符串是二进制安全的块，他们的最大长度是512M。

Bitmap单独提供了一套命令，所以在BCS中使用Bitmap和使用字符串的方法不太相同。可以把Bitmap想象成一个以位为单位的数组，数组的每个单元只能存储0和1，数组的下标在Bitmap中叫做偏移量。

设置值:

```
setbit key offset value
```

BCS 的Bitmap让我们可以实时的进行统计，并且极其节省空间。

13.1.6.1 示例

```
127.0.0.1:7878> SETBIT date:20220331 0 1
(integer) 0
127.0.0.1:7878> SETBIT date:20220331 5 1
(integer) 0
127.0.0.1:7878> SETBIT date:20220331 9 1
(integer) 0
127.0.0.1:7878> GETBIT date:20220331 0
(integer) 1
127.0.0.1:7878> GETBIT date:20220331 5
(integer) 1
127.0.0.1:7878> GETBIT date:20220331 9
(integer) 1
```

13.1.6.2 应用场景

- 网站用户活跃数。

13.1.7 HyperLogLog

HyperLogLog并不是一种新的数据结构（实际类型为字符串类型，而是一种基数算法，通过HyperLogLog可以利用极小的内存空间完成独立总数的统计，有一定的误差率。

添加值

```
pfadd key element [element …]
```

13.1.7.1 示例

```
127.0.0.1:7878> PFADD date:20220331:id "id_1" "id_2" "id_3"
(integer) 1
127.0.0.1:7878> PFADD date:20220331:id "id_3"
(integer) 0
127.0.0.1:7878> PFCOUNT date:20220331:id
(integer) 3
```

13.1.7.2 应用场景

- 主要的应用场景就是进行基数统计，如统计网站访问量。

13.1.8 GEO

GEO（地理信息定位），数据类型为zset，支持存储地理位置信息用来实现诸如附近位置、摇一摇这类依赖于地理位置信息的功能。

13.1.8.1 示例

```
#增加地理位置信息  longitude、 latitude、 member分别是该地理位置的经度、纬度、
↪  成员,可以同时添加多个地理位置信息
geoadd key longitude latitude member [longitude latitude member ...]
```

```
127.0.0.1:7878> GEOADD city 116.28 39.55 beijing
(integer) 1
127.0.0.1:7878> GEOADD city 118.22 31.14 nanjing
(integer) 1
127.0.0.1:7878> GEOPOS city beijing nanjing
1) 1) "116.28000229597091675"
   2) "39.5500007245470826"
2) 1) "118.22000116109848022"
   2) "31.14000123027434341"
127.0.0.1:7878> geodist city beijing nanjing km
"951.7539"
```

13.1.8.2 应用场景

- 地理位置操作：附近的人(需要定位经纬度坐标)，相对距离等。

13.1.9 Stream

BCS Stream 主要用于消息队列（MQ，Message Queue），BCS本身是有一个发布订阅 (pub/sub) 来实现消息队列的功能，但它有个缺点就是消息无法持久化，如果出现网络断开、BCS 宕机等，消息就会被丢弃。

简单来说发布订阅 (pub/sub) 可以分发消息，但无法记录历史消息。

而 BCS Stream 提供了消息的持久化和主备复制功能，可以让任何客户端访问任何时刻的数据，并且能记住每一个客户端的访问位置，还能保证消息不丢失。

13.1.9.1 示例

```
127.0.0.1:7878> xadd mystream * field1 value1
"1648792865178-0"
127.0.0.1:7878> xadd mystream * field2 value2
"1648792871546-0"
127.0.0.1:7878> xadd mystream * field3 value3
"1648792877546-0"
127.0.0.1:7878> XDEL mystream 1648792865178-0
(integer) 1
127.0.0.1:7878> XRANGE mystream - +
1) 1) "1648792871546-0"
   2) 1) "field2"
      2) "value2"
2) 1) "1648792877546-0"
   2) 1) "field3"
      2) "value3"
127.0.0.1:7878> xread count 1 streams mystream 0-0
1) 1) "mystream"
   2) 1) 1) "1648792871546-0"
         2) 1) "field2"
            2) "value2"
```

13.1.9.2 应用场景

- 消息队列。

13.2 持久化

BCS提供了两种不同级别的持久化方式：

- **RDB** 持久化可以在指定的时间间隔内生成数据集的时间点快照。
- **AOF** 持久化记录服务器执行的所有写操作命令，并在服务器启动时，通过重新执行这些命令来还原数据集。AOF 文件中的命令全部以 Redis 协议的格式来保存，新命令会被

追加到文件的末尾。BCS 还可以在后台对 AOF 文件进行重写（rewrite），使得 AOF 文件的体积不会超出保存数据集状态所需的实际大小。BCS还可以同时使用 AOF 持久化和 RDB 持久化。在这种情况下，当 BCS 重启时，它会优先使用 AOF 文件来还原数据集，因为 AOF 文件保存的数据集通常比 RDB 文件所保存的数据集更完整。甚至可以关闭持久化功能，让数据只在服务器运行时存在。

了解 RDB 持久化和 AOF 持久化之间的异同是非常重要的，以下几个小节将详细地介绍这两种持久化功能，并对它们的相同和不同之处进行说明。

13.2.1 RDB

RDB持久化是把当前进程数据生成快照保存到硬盘的过程，触发RDB持久化过程分为手动触发和自动触发。

手动触发分别对应save和bgsave命令：

- save命令：阻塞当前BCS服务器，直到RDB过程完成为止，对于内存比较大的实例会造成长时间阻塞，**线上环境不建议使用**。
- bgsave命令：BCS进程执行fork操作创建子进程，RDB持久化过程由子进程负责，完成后自动结束。阻塞只发生在fork阶段，一般时间很短。

  bgsave命令是针对save阻塞问题做的优化。因此BCS内部所有的涉及RDB的操作都采用bgsave的方式，而save命令已经废弃。除了执行命令手动触发之外，BCS内部还存在自动触发RDB的持久化机制，例如：

```
save 900 1    #每个900秒内至少有1次键改动时自动转储数据集到磁盘
save 300 10   #每个300秒内至少有10次键改动时自动转储数据集到磁盘
save 60 10000 #每个60秒内至少有10000次键改动时自动转储数据集到磁盘
```

默认情况下，BCS保存数据集快照到磁盘，名为dump.rdb的二进制文件。可通过如下配置项进行配置：

```
dbfilename dump.db    #文件名称
dir %%%BCS_RDB_DIR%%% #文件保存路径
```

运维提示：

执行**lastsave**命令可以获取最后一次生成RDB的 时间，对应info [persistence]统计的rdb_last_save_time选项

127.0.0.1:7878> lastsave (integer) 1648863665

13.2.2 AOF

RDB不支持实时持久化，因此如果BCS因为某些原因而造成故障停机，那么服务器将丢失最近写入、且仍未保存到快照中的那些数据。尽管对于某些程序来说，数据是否全量持久化并不是最重要的考虑因素，但是对于那些追求全量持久化的程序来说，RDB功能就不太适用了。

所以 BCS增加了一种实时持久化方式：AOF（append only file）持久化，以独立日志的方式记录每次写命令，重启时再重新执行AOF文件中的命令达到恢复数据的目的，AOF的主要作用是解决了数据持久化的实时性。

可以通过修改配置文件来打开 AOF 功能：

```
# 打开 AOF
appendonly yes
#文件名称，文件保存路径使用RDB文件保存路径
appendfilename "appendonly.aof"
```

每当 BCS 执行一个改变数据集的命令时（比如 set），这个命令就会被追加到 AOF 文件的末尾。当 BCS重新启时，程序就可以通过重新执行 AOF 文件中的命令来达到重建数据集的目的。

可以配置 BCS 多久才将数据同步到磁盘一次。有三个选项：

```
#每秒fsync一次：足够快（和使用 RDB 持久化差不多），并且在故障时只会丢失 1 秒钟的数据。
appendfsync everysec
#每次有新命令追加到 AOF 文件时就执行一次fsync ：非常慢，也非常安全，不建议配置。
appendfsync always
#从不fsync：将数据交给操作系统来处理。更快，也更不安全的选择。
appendfsync no
```

推荐（并且也是默认）的措施为每秒fsync一次，这种fsync策略可以兼顾速度和安全性，总是fsync的策略在实际使用中非常慢。

重写机制

随着命令不断写入AOF，文件会越来越大，为了解决这个问题，BCS引入AOF重写机制压缩文件体积。AOF文件重写是把BCS进程内的数据转化为写命令同步到新AOF文件的过程。

AOF重写降低了文件占用空间，除此之外，另一个目的是：更小的AOF文件可以更快地被BCS加载。

AOF重写过程可以手动触发和自动触发：

- 手动触发：直接调用bgrewriteaof命令。
- 自动触发：根据auto-aof-rewrite-min-size和auto-aof-rewrite-percentage参数确定自动触发时机。
- auto-aof-rewrite-min-size：表示运行AOF重写时文件最小体积，默认为64MB。
- auto-aof-rewrite-percentage：代表当前AOF文件空间（aof_current_size）和上一次重写后AOF文件空间（aof_base_size）的比值。

  自动触发时机=aof_current_size>auto-aof-rewrite-min- size&&(aof_current_size-aof_base_size)/aof_base_size>=auto-aof-rewrite- percentage 其中aof_current_size和aof_base_size可以在info Persistence统计信息中查看。

13.3 主从复制

在分布式系统中为了解决单点问题，通常会把数据复制多个副本部署到其他机器，满足故障恢复和负载均衡等需求。BCS也是如此，它为我们提供了复制功能，实现了相同数据的多个BCS副本。复制功能是高可用BCS的基础，后面章节的哨兵和集群都是在复制的基础上实现高可用的。

13.3.1 配置

13.3.1.1 建立复制

参与复制的BCS实例划分为主节点（master）和从节点（slave）。默认情况下，BCS都是主节点。每个从节点只能有一个主节点，而主节点可以同时具有多个从节点。复制的数据流是单向的，只能由主节点复制到从节点。

配置复制的方式有以下三种:

- 在配置文件中加入replicaof {masterHost} {masterPort}随BCS启动生效。
- 在bes-cache-server启动命令后加入–replicaof {masterHost} {masterPort}生效。
- 直接使用命令：replicaof {masterHost} {masterPort}生效。

通过BCS管理控制中心可以很容易的创建主从实例。配置体现:

```
replicaof ip port
```

```
replicaof本身是异步命令，执行replicaof命令时，节点只保存主节点信息后返回，
↪ 后续复制流程在节点内部异步执行。主从节点复制成功建立后，
↪ 可以使用info replication命令查看复制相关状态。
```

13.3.1.2 断开复制

replicaof命令不但可以建立复制，还可以在从节点执行**replicaof no one**来断开与主节点复制关系。

断开复制主要流程： 1）断开与主节点复制关系。

2）从节点晋升为主节点。

从节点断开复制后并不会抛弃原有数据，只是无法再获取主节点上的数据变化。

通过**replicaof**命令还可以实现切主操作，所谓切主是指把当前从节点对主节点的复制切换到另一个主节点。执行**replicaof** {newMasterIp}{newMasterPort} 命令即可。

切主操作流程如下： 1）断开与旧主节点复制关系。

2）与新主节点建立复制关系。

3）**删除从节点当前所有数据**。

4）**对新主节点进行复制操作**。

运维提示：切主后从节点会清空之前所有的数据，线上人工操作时小心 replicaof 在错误的节点上执行或者指向错误的主节点。

13.3.1.3 安全性

对于数据比较重要的节点，主节点会通过设置**requirepass**参数进行密码验证，这时所有的客户端访问必须使用**auth**命令实行校验。从节点与主节点的复制连接是通过一个特殊标识的客户端来完成，因此需要配置从节点的**masterauth**参数与主节点密码保持一致，这样从节点才可以正确地连接到主节点并发起复制流程。

13.3.1.4 只读

默认情况下，从节点使用**replica-read-only yes** 配置为只读模式。由于复制只能从主节点到从节点，对于从节点的任何修改主节点都无法感知，修改从节点会造成主从数据不一致。因此建议线上不要修改从节点的只读模式。

13.3.1.5 传输延迟

主从节点一般部署在不同机器上，复制时的网络延迟就成为需要考虑的问题，BCS为我们提供了**repl-disable-tcp-nodelay**参数用于控制是否关闭TCP_NODELAY，默认关闭，说明如下：

- 当关闭时，主节点产生的命令数据无论大小都会及时地发送给从节点，这样主从之间延迟会变小，但增加了网络带宽的消耗。适用于主从之间的网络环境良好的场景，如同机架或同机房部署。
- 当开启时，主节点会合并较小的TCP数据包从而节省带宽。默认发送时间间隔取决于Linux的内核，一般默认为40毫秒。这种配置节省了带宽但增大主从之间的延迟。适用于主从网络环境复杂或带宽紧张的场景，如跨机房部署。

运维提示：部署主从节点时需要考虑网络延迟、带宽使用率、防灾级别等因素，如要求低延迟时，建议同机架或同机房部署并关闭repl-disable-tcp-nodelay；如果考虑高容灾性，可以同城跨机房部署并开启repl-disable-tcp-nodelay。

13.3.2 拓扑

BCS的主从复制拓扑结构可以支持单层或多层复制关系，根据拓扑复杂性可以分为以下两种：一主一从、一主多从，下面分别介绍：

13.3.2.1 一主一从结构

一主一从结构是最简单的复制拓扑结构，用于主节点出现宕机时从节点提供故障转移支持(如下图所示)。当应用的写命令并发量较高且需要持久化时，可以只在从节点上开启AOF，这样既保证数据安全性同时也避免了持久化对主节点的性能干扰。但需要注意的是，当主节点关闭

持久化功能时，如果主节点脱机要避免自动重启操作。因为主节点之前没有开启持久化功能自动重启后数据集为空，这时从节点如果继续复制主节点会导致从节点数据也被清空的情况，丧失了持久化的意义。安全的做法是在从节点上执行**replicaof no one**断开与主节点的复制关系，再重启主节点从而避免这一问题。

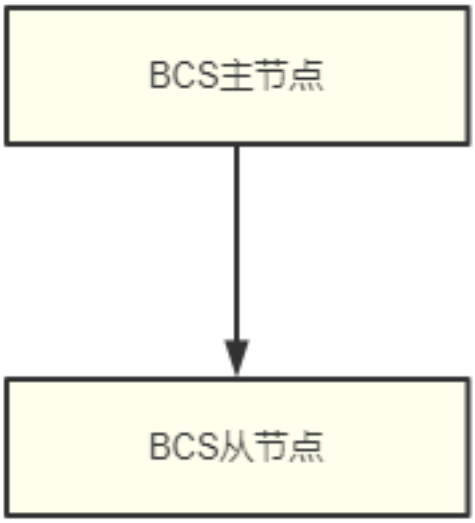

图 13-1 一主一从

13.3.2.2 一主多从结构

一主多从结构（又称为星形拓扑结构）使得应用端可以利用多个从节点实现读写分离（见下图）。对于读占比较大的场景，可以把读命令发送到从节点来分担主节点压力。同时在日常开发中如果需要执行一些比较耗时的读命令，如：keys、sort等，可以在其中一台从节点上执行，防止慢查询对主节点造成阻塞从而影响线上服务的稳定性。对于写并发量较高的场景，多个从节点会导致主节点写命令的多次发送从而过度消耗网络带宽，同时也加重了主节点的负载影响服务稳定性。

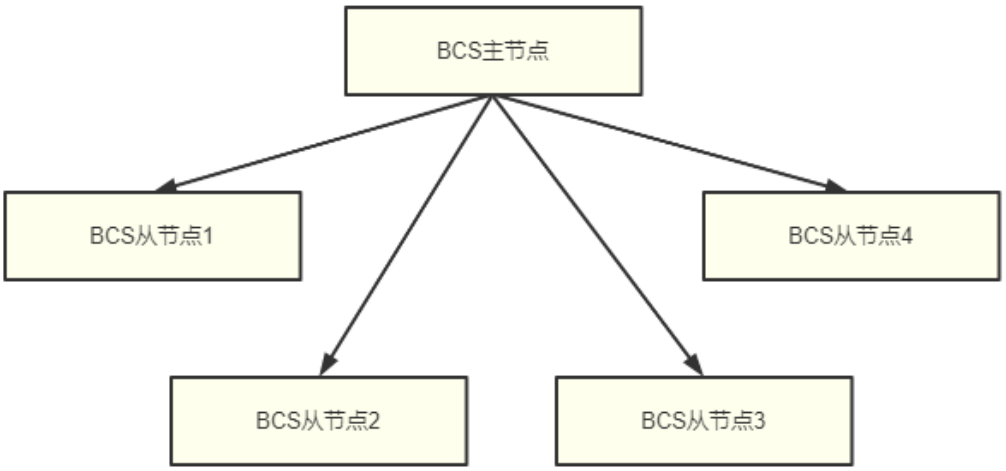

图 13-2 一主多从

13.3.3 工作流程

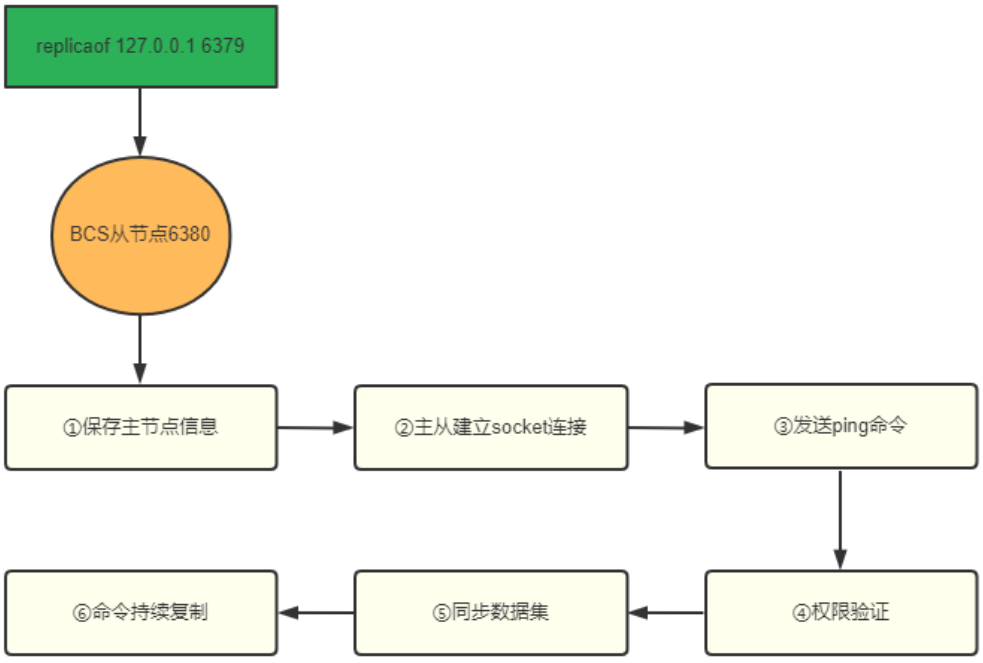

图 13-3 主从复制工作流程

13.4 哨兵

13.4.1 高可用性

BCS的主从复制模式下，一旦主节点由于故障不能提供服务，需要人工将从节点晋升为主节点，同时还要通知应用方更新主节点地址，对于很多应用场景这种故障处理的方式是无法接受的。因此BCS提供了Sentinel（哨兵）架构来解决这个问题，当主节点出现故障时，BCS

Sentinel能自动完成故障发现和故障转移，并通知应用方，从而实现真正的高可用。

BCS Sentinel是一个分布式架构，其中包含若干个Sentinel节点和BCS数据节点，每个Sentinel节点会对数据节点和其余Sentinel节点进行监控，当它发现节点不可达时，会对节点做下线标识。如果被标识的是主节点，它还会和其他Sentinel节点进行“协商”，当大多数Sentinel节点都认为主节点不可达时，它们会选举出一个Sentinel节点来完成自动故障转移的工作，同时会将这个变化实时通知给BCS应用方。整个过程完全是自动的，不需要人工来介入，所以这套方案很有效地解决了BCS的高可用问题。

13.4.2 拓扑

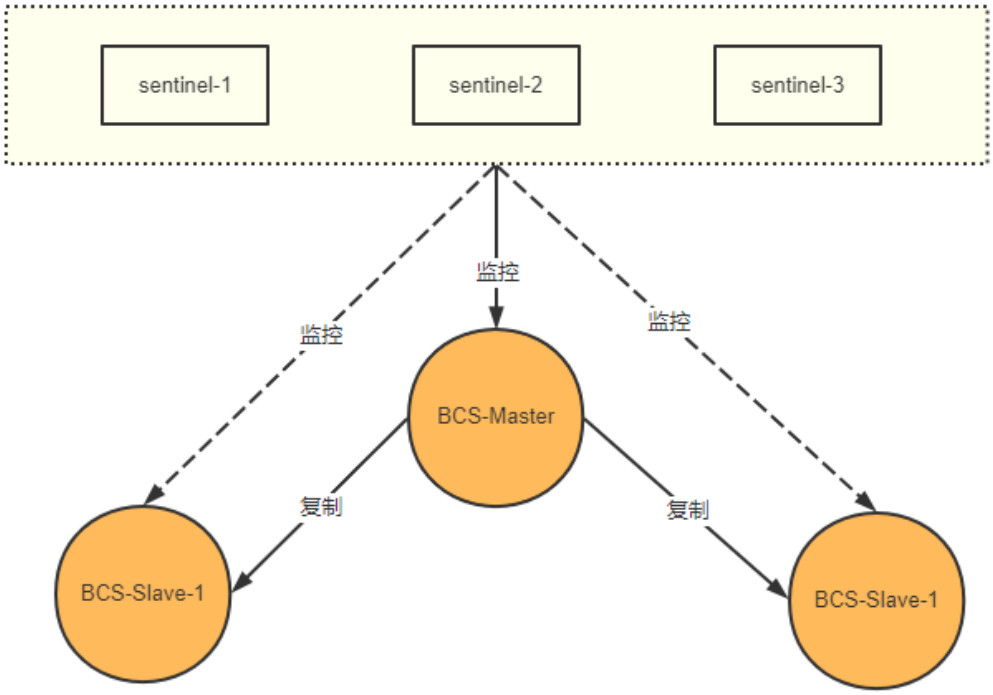

图 13-4 哨兵拓扑

从逻辑架构上看，Sentinel节点集合会定期对所有节点进行监控，特别是对主节点的故障实现自动转移。下面以1个主节点、2个从节点、3个Sentinel节点组成的BCS Sentinel为例子进行说明，拓扑结构如上图所示。

13.4.3 工作流程

整个故障转移的处理逻辑有下面4个步骤：

1）主节点出现故障，此时两个从节点与主节点失去连接，主从复制失败：

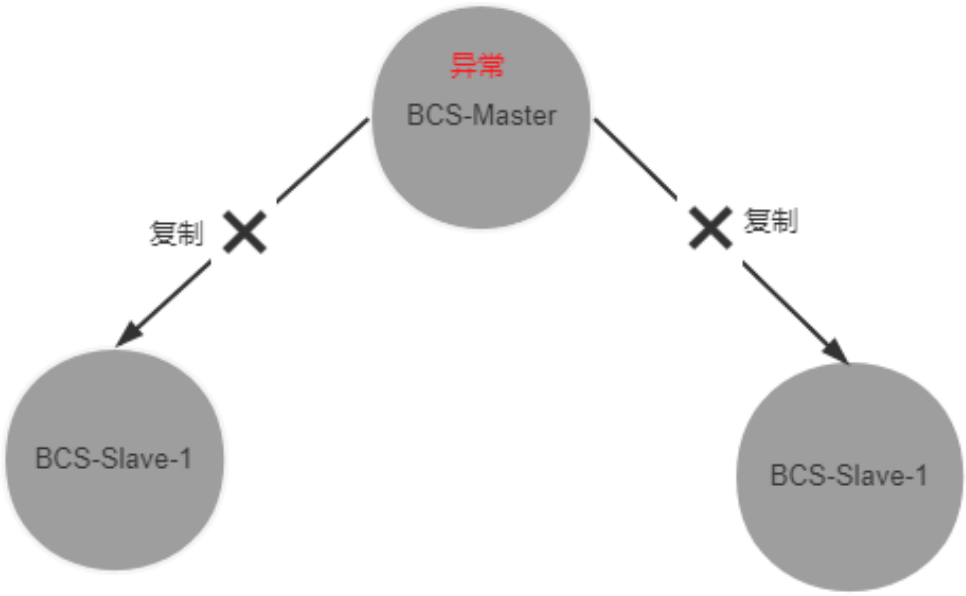

图 13-5 master异常

2）每个Sentinel节点通过定期监控发现主节点出现了故障：

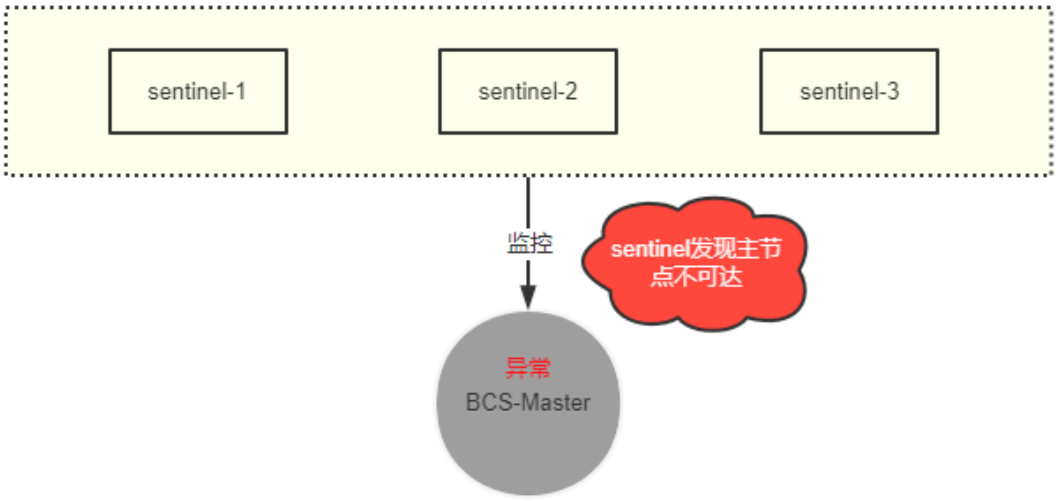

图 13-6 sentinel发现异常

3）多个Sentinel节点对主节点的故障达成一致，选举出sentinel-3节点作为领导者负责故障转移。

4）Sentinel领导者节点自动执行故障转移：

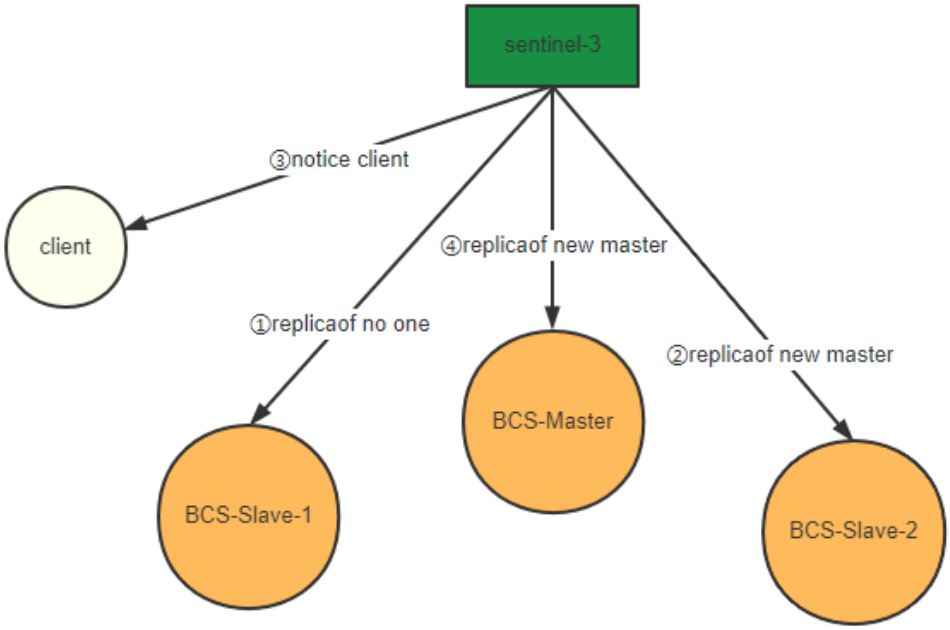

图 13-7 故障转移

通过上面介绍的BCS Sentinel逻辑架构以及故障转移的处理，可以看出BCS Sentinel具有以下几个功能：

- **监控**：Sentinel不断的去检查主从实例是否按照预期在工作、其余Sentinel节点是否可达。
- **通知**：Sentinel可以通过一个api来通知系统管理员或者另外的应用程序，被监控的Redis实例有一些问题。
- **自动故障转移**：如果一个主节点没有按照预期工作，Sentinel会开始故障转移过程，把一个从节点提升为主节点，并重新配置其他的从节点使用新的主节点，使用BCS服务的应用程序在连接的时候也被通知新的地址。
- **配置提供者**：Sentinel给客户端的服务发现提供来源：对于一个给定的服务，客户端连接到Sentinel 来寻找当前主节点的地址。当故障转移发生的时候，Sentinels将报告新的地址。

同时看到，BCS Sentinel包含了若个Sentinel节点，这样做也带来了两个好处：

- 对于节点的故障判断是由多个Sentinel节点共同完成，这样可以有效地防止误判。
- Sentinel节点集合是由若干个Sentinel节点组成的，这样即使个别Sentinel节点不可用，整个Sentinel节点集合依然是健壮的。但是Sentinel节点本身就是独立的BCS节点，只不过它们有一些特殊，它们不存储数据，只支持部分命令。

13.4.4 配置

了解每个配置的含义有助于更加合理地使用BCS Sentinel，因此本节将对每个配置的使用和优化进行详细介绍。

通过BCS管理控制中心可以配置的主要参数如下：

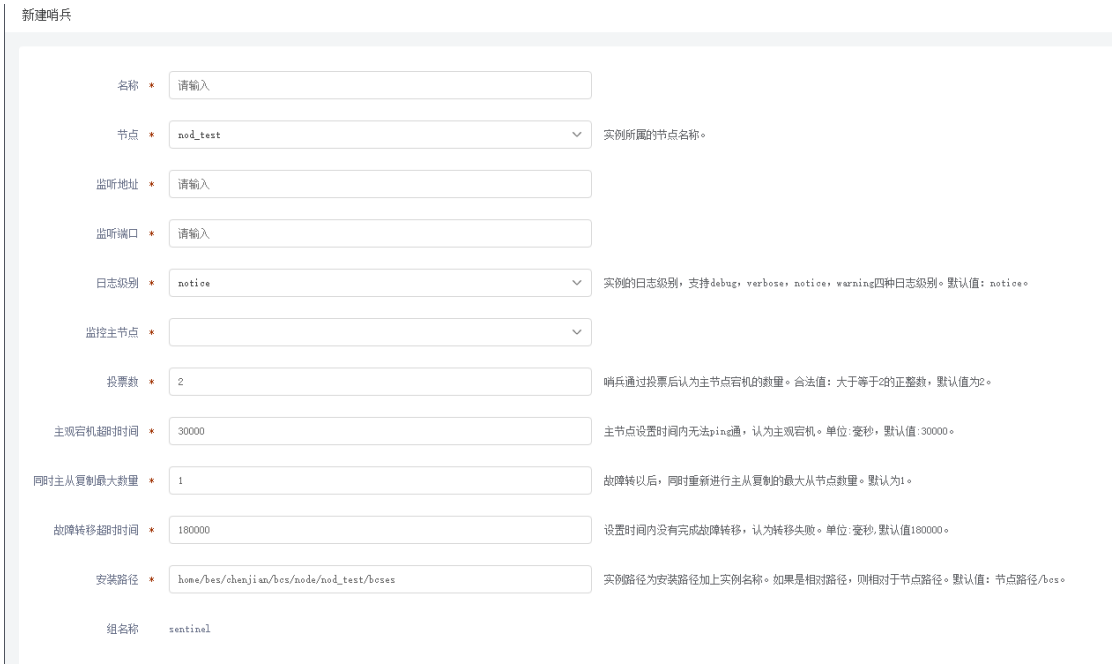

图 13-8 新建哨兵

BCS还支持通过CLI命令在启动时指定当前节点为sentinel节点

```
# 提前配置好sentinel的相关配置到sentinel.conf
./bes-cache-server /xx/sentinel.conf --sentinel
```

13.4.4.1 监控主节点

```
sentinel monitor <master-name> <ip> <port> <quorum>
```

master-name：监控的BCS主节点。

13.4.4.2 投票数

quorum 是需要同意主节点不可用的Sentinels的数量。

一般建议将其设置为Sentinel节点的一半加1。

quorum 仅仅只是用来检测失败。为了实际的执行故障转移，会选举Sentinel中的一个为leader并且被授权进行操作

如果**quorum** 配置不合理，比如如果有五个Sentinel进程，对于一个主节点quorum被设置为2，下面是发生的事情：

- 同时有两个Sentinels同意主节点不可用，其中的一个将会尝试开始故障转移。

- 如果至少有三个Sentinel是可用的，故障转移将会被授权并且开始。

实际中，这意味着在失败时，如果大多数的Sentinel进程没有同意，Sentinel永远不会开始故障转移。仅仅只需要指定要监控的主节点，并给每个单独的主节点一个不同的名称（master-name）。不需要指定从节点，从节点会被自动发现。Sentinel将会根据从节点额外的信息自动更新配置（为了在重启时保留信息）。在故障转移中每当一个从节点被提升为主节点或者当一个新的Sentinel被发现的时候，配置信息也被重新写入。

一个sentinel集群可以监控多个主节点，目前BCS3.0.2版本只支持通过命令行或手动编辑配置文件进行配置，BCS310版本将支持控制台操作。

13.4.4.3 主观宕机超时时间

```
sentinel down-after-milliseconds <master-name> <times>
```

每个Sentinel节点都要通过定期发送ping命令来判断BCS数据节点和其余Sentinel节点是否可达，如果超过了down-after-milliseconds配置的时间且没有有效的回复，则判定节点不可达，（单位为毫秒）就是超时时间。这个配置是对节点失败判定的重要依据。

优化说明：down-after-milliseconds越大，代表Sentinel节点对于节点不可达的条件越宽松，反之越严格。条件宽松有可能带来的问题是节点确实不可达了，那么应用方需要等待故障转移的时间越长，也就意味着应用方故障时间可能越长。条件严格虽然可以及时发现故障完成故障转移，但是也存在一定的误判率。

down-after-milliseconds虽然以为参数，但实际上对Sentinel节点、主节点、从节点的失败判定同时有效。

13.4.4.4 同时主从复制最大数量

```
sentinel parallel-syncs <master-name> <nums>
```

parallel-syncs：当Sentinel节点集合对主节点故障判定达成一致时，Sentinel领导者节点会做故障转移操作，选出新的主节点，原来的从节点会向新的主节点发起复制操作，parallel-syncs就是用来限制在一次故障转移之后，每次向新的主节点发起复制操作的从节点个数。如果这个参数配置的比较大，那么多个从节点会向新的主节点同时发起复制操作，尽管复制操作通常不会阻塞主节点，但是同时向主节点发起复制，必然会对主节点所在的机器造成一定的网络和磁盘IO开销。

对应页面的 **同时主从复制最大数量** 参数。

优化说明：通常设置值为1。

13.4.4.5 故障转移超时时间

```
sentinel failover-timeout <master-name> <times>
```

failover-timeout通常被解释成故障转移超时时间，但实际上它作用于故障转移的各个阶段。

所有的配置参数都可以在运行时使用SENTINEL SET命令进行更改。

13.5 集群

13.5.1 虚拟槽分区

虚拟槽分区巧妙地使用了哈希空间，使用分散度良好的哈希函数把所有数据映射到一个固定范围的整数集合中，整数定义为槽（slot）。这个范围 一般远远大于节点数，比如BCS Cluster槽范围是0~16383。槽是集群内数据管理和迁移的基本单位，采用大范围槽的主要目的是为了方便数据拆分和集群扩展。

13.5.2 架构设计

BCS Cluser采用虚拟槽分区，所有的键根据哈希函数映射到0~16383整数槽内，计算公式：

slot=CRC16（key）&16383

每一个节点负责维护一部分槽以及槽所映射的键值数据，如下图所示：

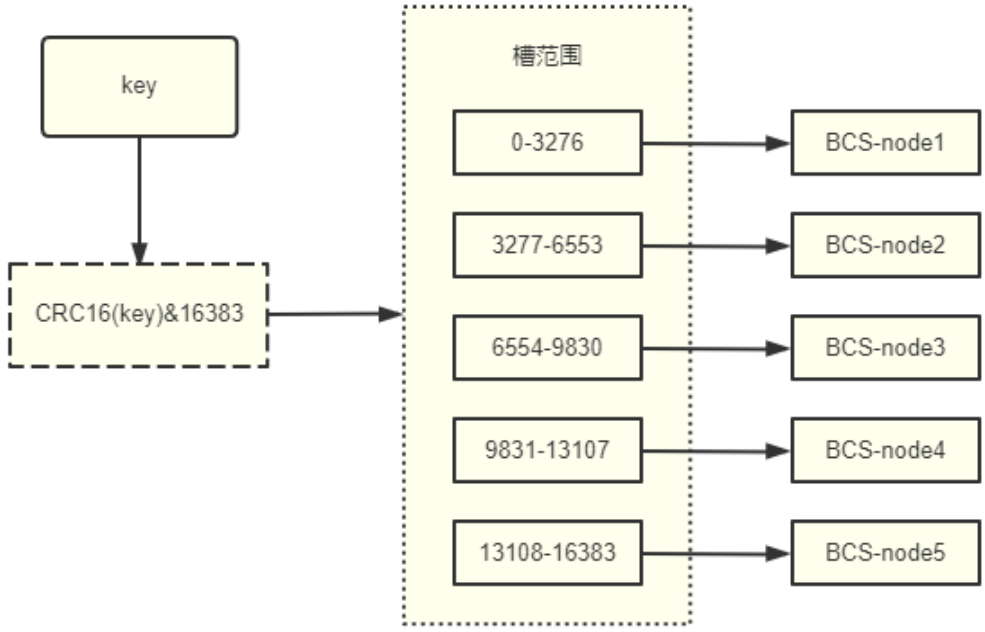

图 13-9 集群拓扑

BCS虚拟槽分区的特点：

1. 解耦数据和节点之间的关系，简化了节点扩容和收缩难度。
2. 节点自身维护槽的映射关系，不需要客户端或者代理服务维护槽分区元数据。
3. 支持节点、槽、键之间的映射查询，用于数据路由、在线伸缩等场景。

集群功能限制:

BCS集群相对单机在功能上存在一些限制，需要开发人员提前了解，在使用时做好规避。限制如下：

1. key批量操作支持有限。如mset、mget，目前只支持具有相同slot值的key执行批量操作。对于映射为不同slot值的key由于执行mget、mget等操作可能存在于多个节点上因此不被支持。
2. key事务操作支持有限。同理只支持多key在同一节点上的事务操作，当多个key分布在不同的节点上时无法使用事务功能。
3. key作为数据分区的最小粒度，因此不能将一个大的键值对象如 hash、list等映射到不同的节点。
4. 不支持多数据库空间。单机下的Redis可以支持16个数据库，集群模式下只能使用一个数据库空间，即db0。
5. 复制结构只支持一层，从节点只能复制主节点，不支持嵌套树状复制结构。

13.5.3 搭建集群

BCS集群一般由多个节点组成，节点数量**至少为6个**才能保证组成完整 高可用的集群。每个节点需要开启配置**cluster-enabled yes**，让BCS运行在集群模式下。建议为集群内所有节点统一目录，一般划分三个目录：conf、data、log，分别存放配置、数据和日志相关文件。

集群相关配置如下：

```
#节点端口
port 7878
# 开启集群模式
cluster-enabled yes
# 节点超时时间，单位毫秒
cluster-node-timeout 15000
# 集群内部配置文件
cluster-config-file "bcs-7878.conf"
```

集群模式的BCS除了原有的配置文件之外又加了一份集群配置文件。当集群内节点信息发生变化，如添加节点、节点下线、故障转移等。节点会自动保存集群状态到配置文件中。需要注意的是，BCS自动维护集群配置文件，不要手动修改，防止节点重启时产生集群信息错乱。

#cat data/bcs-7878.conf

cfb28ef1deee4e0fa78da86abe5d24566744411e 127.0.0.1:7878 myself,master - 0 0 0 connected

vars currentEpoch 0 lastVoteEpoch 0

文件内容记录了集群初始状态，这里最重要的是节点ID，它是一个40位 16进制字符串，用于唯一标识集群内一个节点，之后很多集群操作都要借助于节点ID来完成。需要注意是，节点ID不同于运行ID。节点ID在集群初始化 时只创建一次，节点重启时会加载集群配置文件进行重用，而BCS的运行ID每次重启都会变化。

启动过程:

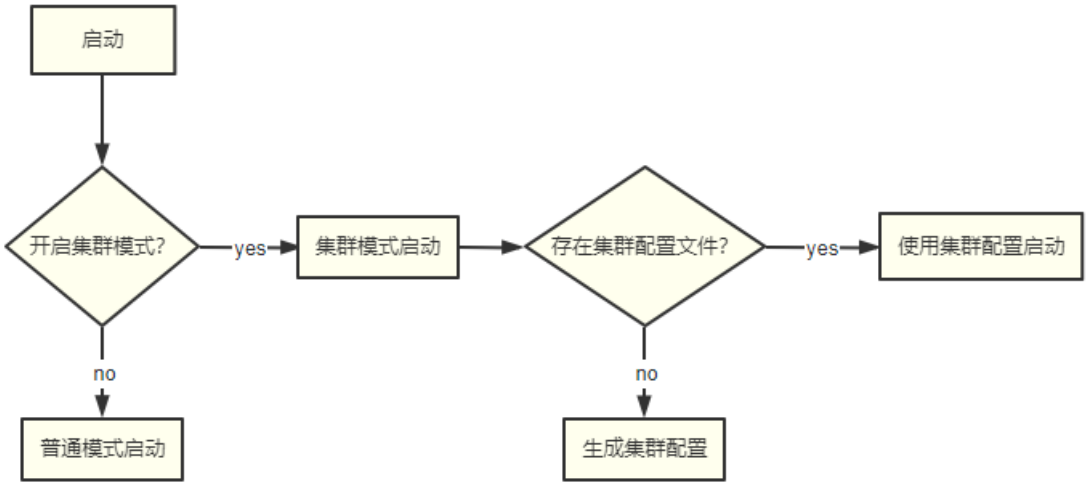

图 13-10 集群启动流程

下面介绍如何搭建一个下图所示的三主三从的BCS集群:

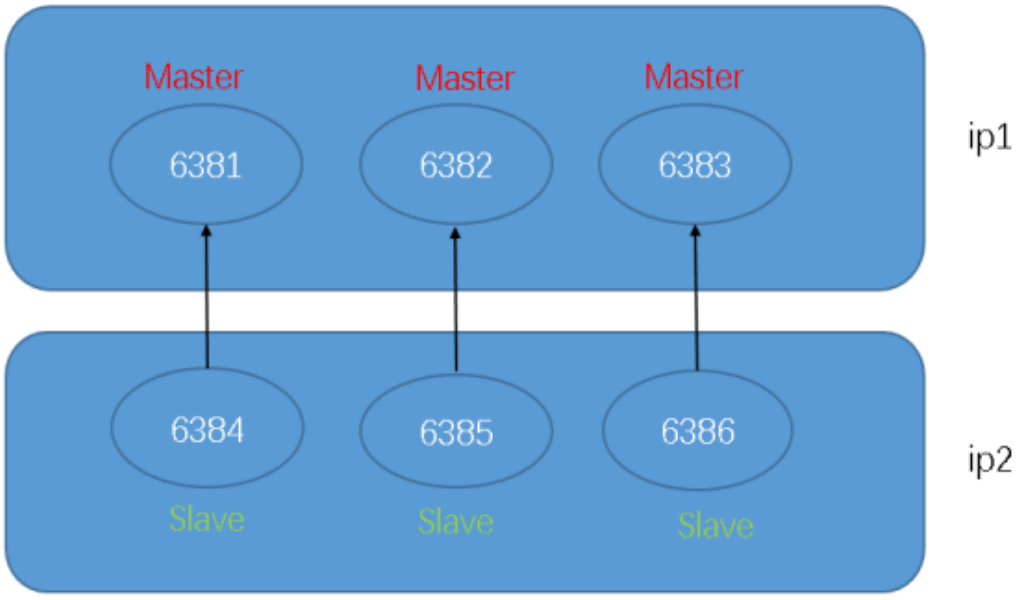

图 13-11 构建集群说明

1. 创建实例，使用真实ip:port,并启动:

   ① ip1：cluster_6381(主)、cluster_6382(主)、cluster_6383(主)。

   ② ip2：cluster_6384 (从)、cluster_6385(从)、cluster_6386(从)。

2. 执行./bes-cache-cli –cluster create ip1:6381 ip1:6382 ip1:6383

等待一会，输入yes继续等待集群的主节点槽位分配，直至集群创建完成。

3. 客户端登录一个端口，执行 cluster nodes，查看集群节点clusterId。

4. 将6384加入到集群成为6381的slave，执行

```
./bes-cache-cli --cluster add-node ip2:6384 ip1:6381 --cluster-slave 」
↪  --cluster-master-id cfb28ef1deee4e0fa78da86abe5d24566744411e
```

5. 采用4的步骤，依次将6385、6386加入集群。然后通过**cluster info**或者**cluster nodes**可以发现指定主从集群搭建完毕。

BCS节点角色分为主节点和从节点。首次启动的节点和被分配槽的节点都是主节点，从节点负责复制主节点槽信息和相关的数据。使用**cluster replicate** {nodeId} 命令让一个节点成为从节点。其中命令执行必须在对应的从节点上执行，nodeId是要复制主节点的节点ID，命令如下:

```
127.0.0.1:6384>cluster replicate cfb28ef1deee4e0fa78da86abe5d24566744411e
OK
```

13.6 配置说明

13.6.1 文件引入配置

参数	默认值	说明
include		引入其他的配置文件

13.6.2 模块加载

参数	默认值	说明
loadmodule		启动时加载模块

13.6.3 网络配置

参数	默认值	说明
bind	0.0.0.0	指定 BCS只接收来自于该IP地址的请求，如果不进行设置，那么将处理所有请求
port	无	BCS监听的端口号
protected-mode	yes	是否开启保护模式。如果没有指定bind和密码，BCS只会本地进行访问，拒绝外部访问。

参数	默认值	说明
tcp-backlog	511	TCP连接中已完成队列(完成三次握手之后)的长度
timeout	0	客户端空闲时间超过timeout，服务端会断开连接，为0则服务端不会主动断开连接，不能小于0。
tcp-keepalive	300（秒）	客户端发送的最后一个数据包与BCS发送的第一个保活探测报文之间的时间间隔，用于检测tcp连接是否还存活,建议设置300(秒),如果小于0启动失败

13.6.4 基本配置

参数	默认值	说明
daemonize	yes	是否后台启动
supervised	no	可以通过upstart和systemd管理Redis守护进程no：没有监督互动 upstart :通过将Redis置于SIGSTOP模式来启动信号 systemd：signal systemd将READY = 1写入$ NOTIFY_SOCKET auto：检测upstart或systemd方法基于UPSTART_JOB或NOTIFY_SOCKET环境变量
pidfile		配置PID文件路径
loglevel		日志级别：de-bug/verbose/notice/warning
logfile		日志文件
database	16	数据库的数量，集群环境默认只有DB 0
always-show-logo	yes	是否一直显示logo

13.6.5 RDB配置

参数	默认值	说明
save	save 900 1save 300 10save 60 10000	保存数据到磁盘的触发频率
stop-writes-on-bgsave-error	yes	持久化出现错误后，是否依然继续进行工作
rdbcompression	yes	是否压缩rdb文件，rdb文件压缩使用LZF压缩算法
rdbchecksum	yes	是否校验rdb文件
dbfilename	dump.rdb	rdb文件名称
dir		使用上面的dbfilename配置指令的文件名保存到这个目录

13.6.6 主从复制

参数	默认值	说明
replicaof		同步复制的主节点信息
masterauth		主节点的密码
replica-serve-stale-data	yes	当一个slave失去和master的连接，或者同步正在进行中，slave的行为有两种可能：如果 replica-serve-stale-data 设置为“yes”(默认值)，slave会继续响应客户端请求，可能是正常数据，也可能是还没获得值的空数据。如果 replica-serve-stale-data 设置为“no”，slave会回复”正在从master同步（SYNC with master in progress）“来处理各种请求，除了 info和 replicaof命令。
replica-read-only	yes	配置从是否为只读，开启后从节点则不能写入数据
repl-diskless-sync	no	同步策略: 磁盘或socket，默认磁盘方式

参数	默认值	说明
repl-diskless-sync-delay	5（秒）	如果非磁盘同步方式开启，可以配置同步延迟时间，以等待master产生子进程通过socket传输RDB数据给slave。默认值为5秒，设置为0秒则每次传输无延迟。
repl-ping-replica-period	10（秒）	slave根据指定的时间间隔向master发送ping请求
repl-disable-tcp-nodelay	no	是否在slave套接字发送SYNC之后禁用TCP_NODELAY。如果选择yes，BCS将使用更少的TCP包和带宽来向slave发送数据。但是这将使数据传输到slave上有延迟，Linux内核的默认配置会达到40毫秒。如果选择no，数据传输到salve的延迟将会减少但要使用更多的带宽。默认我们会为低延迟做优化，但高流量情况或主从之间的跳数过多时，可以设置为“yes”。
replica-priority	100	当master不可用，Sentinel会根据slave的优先级选举一个master。最低的优先级的slave，当选master。而配置成0，永远不会被选举

13.6.7 安全配置

参数	默认值	说明
requirepass		节点密码

13.6.8 限制配置

参数	默认值	说明
maxclients	10000	设置最多同时连接的客户端数量
maxmemory		内存限制

参数	默认值	说明
maxmemory-policy	noeviction	如果达到上面最大的内存限制，BCS如何选择删除key: volatile-lru -> 根据LRU算法删除设置过期时间的key allkeys-lru -> 根据LRU算法删除任何key volatile-random -> 随机移除设置过过期时间的keyallkeys-random -> 随机移除任何key volatile-ttl -> 移除即将过期的key noeviction -> 不移除任何key，只返回一个写错误
maxmemory-samples	5	LRU、LFU和最小TTL算法的样本个数

13.6.9 懒删除配置

参数	默认值	说明
lazyfree-lazy-eviction	no	内存使用达到maxmeory，并设置有淘汰策略时，在被动淘汰键时，是否采用lazy free机制。
lazyfree-lazy-expire	no	针对设置有TTL的键，达到过期后，清理删除时是否采用lazy free机制。
lazyfree-lazy-server-del	no	内部删除，比如rename oldkey newkey时，如果newkey存在需要删除时，是否采用lazy free机制。
replica-lazy-flush	no	slave进行全量数据同步，slave在加载master的RDB文件前，会运行flushall来清理自己的数据场景，参数设置决定是否采用flush机制。如果内存变动不大，建议可开启。可减少全量同步耗时，从而减少主库因输出缓冲区爆涨引起的内存使用增长。

13.6.10 AOF配置

参数	默认值	说明
appendonly	no	是否开启AOF持久化
appendfilename	appendonly.aof	AOF文件名称

参数	默认值	说明
appendfsync	everysec	支持三种不同的模式：no：不要立刻刷，只有在操作系统需要刷的时候再刷。比较快。always：每次写操作都立刻写入到aof文件。慢，但是最安全。everysec：每秒写一次。折中方案。
no-appendfsync-on-rewrite	no	如果AOF的同步策略设置成“always”或者“everysec”，并且后台的存储进程会产生很多磁盘I/O开销。某些Linux的配置下会使BCS因为 fsync()系统调用而阻塞很久。为了缓解这个问题，可以用这个选项。它可以在 BGSAVE 或 BGREWRITEAOF 处理时阻止 fsync()。这就意味着如果有子进程在进行保存操作，那么BCS就处于”不可同步”的状态。也就是说在最差的情况下可能会丢掉30秒钟的日志数据。（默认Linux设定）如果把这个设置成”yes”带来了延迟问题，就保持”no”，从耐久性的角度来看，这是最安全的选择。
auto-aof-rewrite-percentage	100	自动重写AOF文件。如果AOF日志文件增大到指定百分比，BCS能够通过 BGREWRITEAOF 自动重写AOF日志文件。指定百分比为0会禁用AOF自动重写特性。
auto-aof-rewrite-min-size	64mb	文件达到大小阈值的时候进行重写
aof-load-truncated	yes	如果设置为yes，如果一个因异常被截断的AOF文件被BCS启动时加载进内存，redis将会发送日志通知用户。如果设置为no，BCS将会拒绝启动。此时需要用工具修复文件。

参数	默认值	说明
aof-use-rdb-preamble	yes	加载时BCS识别出AOF文件以“REDIS”开头字符串，并加载带此前缀的RDB文件，然后继续加载AOF

13.6.11 LUA脚本配置

参数	默认值	说明
lua-time-limit	5000	Lua 脚本的最大执行毫秒数

13.6.12 集群配置

参数	默认值	说明
cluster-enabled	yes	是否开启redis集群
cluster-config-file		配置BCS自动生成的集群配置文件名。确保同一系统中运行的各BCS实例该配置文件不要重名。
cluster-node-timeout	15000（毫秒）	集群节点超时毫秒数

13.6.13 Docker集群配置

默认情况下，BCS会自动检测自己的IP和从配置中获取绑定的PORT，告诉客户端或者是其他节点。而在Docker环境中，如果使用的不是host网络模式，在容器内部的IP和PORT都是隔离的，那么客户端和其他节点无法通过节点公布的IP和PORT建立连接。如果开启以下配置，BCS节点会将配置中的这些IP和PORT告知客户端或其他节点。而这些IP和PORT是通过Docker转发到容器内的临时IP和PORT的。

参数	默认值	说明
cluster-announce-ip		
cluster-announce-port		
cluster-announce-bus-port		集群总线端口

13.6.14 慢查询日志配置

参数	默认值	说明
slowlog-log-slower-than	10000	记录超过多少微秒的查询命令。1000000等于1秒，设置为0则记录所有命令。
slowlog-max-len	128	记录大小，可通过SLOWLOG RESET命令重置

13.6.15 延时监控系统配置

参数	默认值	说明
latency-monitor-threshold	0	记录执行时间大于或等于预定时间（毫秒）的操作,为0时不记录

13.6.16 事件通知配置

参数	默认值	说明
notify-keyspace-events		BCS能通知 Pub/Sub 客户端关于键空间发生的事件，默认关闭。

13.6.17 内部数据结构相关配置

参数	默认值	说明
hash-max-ziplist-entries	512	当hash只有少量的entry时，并且最大的entry所占空间没有超过指定的限制时，会用一种节省内存的数据结构来编码。
hash-max-ziplist-value	64	

参数	默认值	说明
list-max-ziplist-size	-2	当取正值的时候，表示按照数据项个数来限定每个quicklist节点上的ziplist长度。比如，当这个参数配置成5的时候，表示每个quicklist节点的ziplist最多包含5个数据项。当取负值的时候，表示按照占用字节数来限定每个quicklist节点上的ziplist长度。这时，它只能取-1到-5 这五个值，每个值含义如下：-5: 每个quicklist节点上的ziplist大小不能超过64 Kb。（注：1kb => 1024 bytes）-4: 每个quicklist节点上的ziplist大小不能超过32 Kb。-3: 每个quicklist节点上的ziplist大小不能超过16 Kb。-2: 每个quicklist节点上的ziplist大小不能超过8 Kb。-1: 每个quicklist节点上的ziplist大小不能超过4 Kb。

参数	默认值	说明
list-compress-depth	0	这个参数表示一个quicklist两端不被压缩的节点个数。注：这里的节点个数是指quicklist双向链表的节点个数，而不是指ziplist里面的数据项个数。实际上，一个quicklist节点上的ziplist，如果被压缩，就是整体被压缩的。参数list-compress-depth的取值含义如下：0: 是个特殊值，表示都不压缩。这是Redis的默认值。1: 表示quicklist两端各有1个节点不压缩，中间的节点压缩。2: 表示quicklist两端各有2个节点不压缩，中间的节点压缩。3: 表示quicklist两端各有3个节点不压缩，中间的节点压缩。依此类推... 由于0是个特殊值，很容易看出quicklist的头节点和尾节点总是不被压缩的，以便于在表的两端进行快速存取。
set-max-intset-entries	512	set有一种特殊编码的情况：当set数据全是十进制64位有符号整型数字构成的字符串时。这个配置项就是用来设置set使用这种编码来节省内存的最大长度。
zset-max-ziplist-entries	128	与hash和list相似，有序集合也可以用一种特别的编码方式来节省大量空间。这种编码只适合长度和元素都小于下面限制的有序集合。
zset-max-ziplist-value	64	

参数	默认值	说明
hll-sparse-max-bytes	3000	HyperLogLog稀疏结构表示字节的限制。该限制包括16个字节的头。当HyperLogLog使用稀疏结构表示这些限制，它会被转换成密度表示。值大于16000是完全没用的，因为在该点密集的表示是更多的内存效率。建议值是3000左右，以便具有的内存好处, 减少内存的消耗。
stream-node-max-bytes	4096	Streams宏节点最大大小/项目。流数据结构是基数编码内部多个项目的大节点树。使用此配置可以配置单个节点的字节数, 以及切换到新节点之前可能包含的最大项目数追加新的流条目。如果以下任何设置设置为0, 忽略限制，因此例如可以设置一个 大入口限制将max-bytes设置为0，将max-entries设置为所需的值。
stream-node-max-entries	100	
activerehashing	yes	指定是否激活重置哈希，默认为开启，每100个CPU毫秒会拿出1个毫秒来刷新BCS的主哈希表

参数	默认值	说明
client-output-buffer-limit		客户端的输出缓冲区的限制，可用于强制断开那些因为某种原因从服务器读取数据的速度不够快的客户端：normal 0 0 0：对于普通客户端来说，限制为0，也就是不限制。因为普通客户端通常采用阻塞式的消息应答模式，何谓阻塞式呢？如：发送请求，等待返回，再发送请求，再等待返回。这种模式下，通常不会导致Redis服务器输出缓冲区的堆积膨胀；replica 256mb 64mb 60 ：对于slave客户端来说，大小限制是256M，持续性限制是当客户端缓冲区大小持续60秒超过64M，则关闭客户端连接。pubsub 32mb 8mb 60：对于Pub/Sub客户端（也就是发布/订阅模式），大小限制是8M，当输出缓冲区超过8M时，会关闭连接。持续性限制是，当客户端缓冲区大小持续60秒超过2M，则关闭客户端连接；
hz	10	通过修改hz参数的值，您可以调整BCS执行定期任务的频率，从而改变BCS清除过期key、清理超时连接的效率。hz的取值范围为1~500。增大hz参数的值会提升各项定期任务的执行频率，但也会提高BCS服务的CPU使用率。默认值10在一般情况下已经可以满足需求，如果业务场景对于某些定期任务的执行频率有很高的要求，您可以尝试在100以内调整参数值。将hz的值增加到100以上对CPU使用率有相对较大的影响，请谨慎操作。
dynamic-hz	yes	开启动态hz

参数	默认值	说明
aof-rewrite-incremental-fsync	yes	当一个子进程重写AOF文件时，如果启用该选项，则文件每生成32M数据会被同步。
rdb-save-incremental-fsync	yes	当BCS保存RDB文件时，如果启用了该选项，每生成32MB数据，文件将被fsync-ed。这很有用，以便以递增方式将文件提交到磁盘并避免大延迟峰值。

13.6.18 碎片整理配置

参数	默认值	说明
activedefrag yes	yes	启用主动碎片整理
active-defrag-ignore-bytes	100mb	启动活动碎片整理的最小碎片浪费量
active-defrag-threshold-lower	10	启动碎片整理的最小碎片百分比
active-defrag-threshold-upper	100	使用最大消耗时的最大碎片百分比
active-defrag-cycle-min	5	在CPU百分比中进行碎片整理的最小消耗
active-defrag-cycle-max	75	磁盘碎片整理的最大消耗
active-defrag-max-scan-fields	1000	将从主字典扫描处理的最大set / hash / zset / list字段数

第14章 实例调优指南

14.1 Linux配置调优

vm.overcommit_memory参数建议设置为1，避免主从复制fork子进程生成rdb文件申请内存不够导致的主从复制失败。

```
编辑/etc/sysctl.conf，修改vm.overcommit_memory=1（如果没有该键值对则添加），
↪ 然后sysctl -p使配置文件生效。
```

BCS实例建议将Transparent Huge Pages（THP）关闭，Linux kernel在2.6.38内核增加了THP特性，支持大内存页（2MB）分配，默认开启，当开启时会降低fork子进程的速度，fork操作之后，每个内存页从原来4KB变为2MB，会大幅增加重写期间父进程内存消耗。关闭方法为：

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

BCS的默认的tcp-backlog值为511，可以通过修改配置tcp-backlog进行调整，如果Linux的tcp-backlog小于bcs实例设置的tcp-backlog，可以修改linux配置值。修改方法为：

```
echo 511 > /proc/sys/net/core/somaxconn
```

14.2 主从复制调优

如果主从节点部署在异地不同机房，复制时的网络延迟就成为需要考虑的问题，BCS实例的repl-disable-tcp-nodelay参数来控制是否关闭TCP_NODELAY，默认为no。配置在从实例上，如果主从会故障转移，主和从实例都要配置该参数。

当设置为no，表示开启TCP-NODELAY，禁用Nagle算法，主从复制延迟性降低，网络负担增加。

当设置为yes，表示关闭TCP-NODELAY，使用Nagle算法，节省带宽，主从复制延迟性增加。

```
# BCS实例参数配置
# 选项为no： 开启tcp-nodelay，禁用Nagle算法
# 选项为yes：关闭tcp-nodelay，使用Nagle算法
repl-disable-tcp-nodelay yes
```

当主实例通过requirepass参数设置密码，从实例都要配置masterauth参数与主实例密码保持一致，这样从实例才能连接发起主从复制。

1）如果主从会故障转移，不管什么模式，主从都要配上masterauth，以防故障转移，主变成从，未配置masterauth就会疯狂输出日志。

2）如有要设置密码，所有实例都配置requirepass和masterauth参数，密码要保持一致。

```
# BCS实例参数配置
# 主从模式：主和从实例都要配置下列两个选项
# 集群模式：主和从实例都要配置下列两个选项
# 哨兵模式：主和从实例都要配置下列两个选项，
↪  哨兵要配置sentinel auth-pass <master-name> 123456
requirepass 123456
masterauth 123456
```

如果主实例写入磁盘rdb文件比较慢，网络带宽很大时，可以采用无磁盘模式复制repl-diskless-sync,设置为yes时，不写入到磁盘rdb，直接写入到从节点。生产环境不建议使用，测试环境可以使用。

```
# BCS实例参数配置
repl-diskless-sync yes
```

repl-ping-replica-period参数表示主实例检测从实例心跳时间间隔，默认10秒。repl-timeout参数表示主从复制超时时间，默认为60s。repl-timeout不能配置比repl-ping-replica-period小。

1）对于数据量较大的主实例，比如生成的RDB文件超过6GB，在千兆网卡的机器，理论每秒传输100MB，在不考虑其他进程消耗带宽的情况下，6GB的RDB文件至少需要60秒传输时间，默认配置下，极易出现主从数据同步超时。尤其对于异地部署的主从，要考虑下带宽是否适合用默认的repl-timeout值。

```
# BCS实例参数配置
repl-ping-replica-period 10
repl-timeout 60
```

replica-serve-stale-data参数默认为yes，表示主从复制时，从节点继续接收客户端的请求，但是返回的数据可能是旧的。如果设置为no,将对于所有的命令都返回错误 “SYNC with master in progress”,但以下命令除外：INFO、REPLICAOF、AUTH、PING、SHUTDOWN、REPLCONF、ROLE、CONFIG、SUBSCRIBE、UNSUBSCRIBE、PSUBSCRIBE、PUNSUBSCRIBE、PUBLISH、PUBSUB、COMMAND、POST、HOST、LATENCY。

```
# BCS实例参数配置
replica-serve-stale-data yes
```

replica-priority参数用于哨兵模式下主从发生故障转移，从实例晋升为主实例的优先级，该值越低表示优先级越高，比如在异地主从环境下（有哨兵的情况下），当主有故障，想让同机房的从实例优先晋升为主，异地的从次之，可以设置该值。

```
# BCS实例参数配置
# 默认值100，值越小优先级越高
replica-priority 50
```

14.3 集群配置调优

当集群部署到一定规模，gossip消息会占用很多的带宽，如果带宽无法提升，可以调大cluster-node-timeout参数能够有效的带宽。但是并不是越大越好，增大会增加集群故障转移的时间。

```
# BCS实例参数配置
# 默认值15秒，建议生产上吞吐量不够，调大该值。
cluster-node-timeout 15000
```

第15章 实例部署规划

15.1 主从模式

适合于数据量不大、写入量少、扩展性要求不高。该模式下可以创建选择单个主实例或者一主多从，生产环境下如果对于主挂掉，业务层面可以容忍无法自动故障转移，可以手动故障转移前提下可以选择该模式。

业务总数据量	每台实例资源配置	部署规模
4G	4C8G	一主两从
8G	8C16G	一主两从
参考下面估算公式	参考下面估算公式	一主两从

- 由于主从模式下，业务的数据量取决于主实例所在机器的内存，建议BCS实例存储的数据为机器内存的1/2，比如16G内存，那么bcs实例存储的数据控制在8G左右。
- **估算公式**：业务总数据量 = 每台实例内存大小 / 2，建议主从数量比例1：2。
- 需要根据带宽大小和数据量，调优repl-timeout参数。否则主从数据复制全量复制可能失败。（其他BCS实例参数以及linux内核参数建议参考”实例调优指南-Linux配置调优、主从复制调优”）

15.2 哨兵模式

适合于数据量不大、写入量少、扩展性要求不高、自动故障转移。该模式下可以创建主从+哨兵来实现高可用，生产环境下主挂掉，能够自动故障转移，具备高可用性，推荐生产环境下使用。

业务总数据量	每台实例资源配置	部署规模
4G	4C8G	一主两从三哨兵
8G	8C16G	一主两从三哨兵
参考下面估算公式	参考下面估算公式	一主两从三哨兵

- 该模式实际上跟主从模式类似，业务的数据量取决于主实例所在机器的内存，建议BCS实例存储的数据为机器内存的1/2，比如16G内存，那么BCS实例存储的数据控制在8G左右。
- **估算公式**：业务总数据量 = 每台实例内存大小 / 2，建议主从数量比例1：2。
- 需要根据带宽大小和数据量，调优repl-timeout参数。否则主从数据复制全量复制可能失败。（其他BCS实例参数以及linux内核参数建议参考”实例调优指南-Linux配置调优、主从复制调优”）

15.3 集群模式

适合于数据量大、写入量多、支持扩容缩容。集群模式推荐生产环境下使用。

业务总数据量	每台实例资源配置	部署规模
12G	4C8G	三主三从
24G	8C16G	三主三从
24G	4C8G	六主六从
48G	8C16G	六主六从
参考下面估算公式	参考下面估算公式	参考下面估算公式

- 跟其他模式有区别的是，数据存在分片，并不是其他模式的单点写入，估算公式自然也存在差异。
- **估算公式**：业务总数据量 =（每台实例内存大小 * 主实例个数）/2 ,建议主从数量比例1:1。
- **建议最大规模1000个实例，实例中通常不会超过500个。如果超过一定数量，集群的吞吐量降低，可以适当调大cluster-node-timeout参数，集群间消息通信开销的带宽是极其可观的**。集群模式用的是gossip协议，每秒通信消息体至少包含1/10的节点数，比如说100个实例，每个节点状态大小数据约为104byte，这部分消息的大小约为10*104,大约为1KB，消息头大概2KB，所以每个gossip消息大约为3KB，携带消息体的大小与集群规模相关，规模越大消息体越大，通信成本越高。达到一定程度后整体集群性能会下降。
- 需要根据带宽大小和数据量，调优repl-timeout参数。否则主从数据复制全量复制可能失败（其他bcs实例参数以及linux内核参数建议参考”实例调优指南-Linux配置调优、主从复制调优、集群配置调优”）

第16章 示例

BES CacheServer全面兼容Redis和Jedis，可以使用JAVA和PHP客户端来访问BCS服务

在集群管理控制台创建一个单实例模式的实例，监听地址为0.0.0.0，监听端口为7501，启动实例。

16.0.1 使用BCS客户端连接

- 客户端直接连接

```
package com.test;

import com.bes.bcs.clients.java.BCSClient;

public class TestBCSClient {
    public static void main(String[] args) {
        //连接 BCS 服务器
        BCSClient bcsClient = new BCSClient("127.0.0.1", 7501);
        //如果设置了密码，需要进行权限认证
        //bcsClient.auth("111111");
        //添加数据
        bcsClient.set("name", " BCS"); //向 key-->name 中放入了 value-->BCS
        //获取并输出数据
        System.out.println(bcsClient.get("name")); //执行结果: BCS
    }
}
```

- 连接池连接

bcs 连接资源的创建和销毁时很消耗程序性能的，所以建议使用连接池。bcs 连接池在创建时初始化一些连接资源存储到连接池中，使用 bcs 连接资源时不需要创建，而是从连接池中获取一个资源进行 bcs 的操作，使用完毕后，不需要销毁该 bcs 连接资源，而是将该资源归还给连接池，供其他请求使用。

```
package com.test;

import com.bes.bcs.clients.java.BCSClient;
import com.bes.bcs.clients.java.BCSClientPool;
import com.bes.bcs.clients.java.BCSClientPoolConfig;

public class TestBCSClientPool {
    public static void main(String[] args) {
        //1、获取连接池配置对象，设置配置项
        BCSClientPoolConfig bcsClientPoolConfig = new BCSClientPoolConfig();
        bcsClientPoolConfig.setMaxTotal(20);//最大连接数
        bcsClientPoolConfig.setMaxIdle(10); //最大空闲连接数
        bcsClientPoolConfig.setMaxWaitMillis(3000); //最大等待时间
        //2、初始化连接池
        BCSClientPool bcsClientPool = new BCSClientPool(bcsClientPoolConfig, 」
        ↪  "127.0.0.1", 7501);
        //3、从连接池中获取到一个连接
        BCSClient bcsClient = bcsClientPool.getResource();
```

```
        //4、操作函数
        bcsClient.set("name", "BCS");//向 key-->name 中放入了 value-->BCS
        System.out.println(bcsClient.get("name"));//执行结果: BCS
        //5、关闭当前连接，返回连接池
        bcsClient.close();
    }
}
```

16.0.2 使用JAVA客户端连接

- 直接连接:

```
package com.test;

import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import org.junit.Before;
import org.junit.Test;

import redis.clients.jedis.Jedis;

public class TestBCS {
    private Jedis jedis;

    @Before
    public void setup() {
        //连接redis服务器
        jedis = new Jedis("127.0.0.1", 7501);
        //权限认证
//        jedis.auth("admin");
    }

    /**
     * redis存储字符串
     */
    @Test
    public void testString() {
        //-----添加数据----------
        jedis.set("name","xinxin");//向key-->name中放入了value-->xinxin
        System.out.println(jedis.get("name"));//执行结果: xinxin

        jedis.append("name", " is my lover"); //拼接
        System.out.println(jedis.get("name"));

        jedis.del("name");  //删除某个键
        System.out.println(jedis.get("name"));
        //设置多个键值对
        jedis.mset("name","liuling","age","23","qq","476777XXX");
        jedis.incr("age"); //进行加1操作
        System.out.println(jedis.get("name") + "-" + jedis.get("age") + "-" +
        ↪  jedis.get("qq"));

    }
```

```
/**
 * redis操作Map
 */
@Test
public void testMap() {
    //-----添加数据----------
    Map<String, String> map = new HashMap<String, String>();
    map.put("name", "xinxin");
    map.put("age", "22");
    map.put("qq", "123456");
    jedis.hmset("user",map);
    //取出user中的name，执行结果:[minxr]-->注意结果是一个泛型的List
    //第一个参数是存入redis中map对象的key，后面跟的是放入map中的对象的key,
    ↪ 后面的key可以跟多个，是可变参数
    List<String> rsmap = jedis.hmget("user", "name", "age", "qq");
    System.out.println(rsmap);

    //删除map中的某个键值
    jedis.hdel("user","age");
    System.out.println(jedis.hmget("user", "age")); //因为删除了,
    ↪ 所以返回的是null
    System.out.println(jedis.hlen("user")); //返回key为user的键中存放的值的个」
    ↪ 数2
    System.out.println(jedis.exists("user"));//是否存在key为user的记录 返回t」
    ↪ rue
    System.out.println(jedis.hkeys("user"));//返回map对象中的所有key
    System.out.println(jedis.hvals("user"));//返回map对象中的所有value

    Iterator<String> iter=jedis.hkeys("user").iterator();
    while (iter.hasNext()){
        String key = iter.next();
        System.out.println(key+":"+jedis.hmget("user",key));
    }
}

/**
 * jedis操作List
 */
@Test
public void testList(){
    //开始前，先移除所有的内容
    jedis.del("java framework");
    System.out.println(jedis.lrange("java framework",0,-1));
    //先向key java framework中存放三条数据
    jedis.lpush("java framework","spring");
    jedis.lpush("java framework","struts");
    jedis.lpush("java framework","hibernate");
    //再取出所有数据jedis.lrange是按范围取出,
    // 第一个是key，第二个是起始位置，第三个是结束位置,
    ↪ jedis.llen获取长度 -1表示取得所有
    System.out.println(jedis.lrange("java framework",0,-1));

    jedis.del("java framework");
    jedis.rpush("java framework","spring");
    jedis.rpush("java framework","struts");
    jedis.rpush("java framework","hibernate");
    System.out.println(jedis.lrange("java framework",0,-1));
}
```

```
    /**
     * jedis操作Set
     */
    @Test
    public void testSet(){
        //添加
        jedis.sadd("users","liulin");
        jedis.sadd("users","xinxin");
        jedis.sadd("users","ling");
        jedis.sadd("users","zhangxinxin");
        jedis.sadd("users","who");
        //移除noname
        jedis.srem("users","who");
        System.out.println(jedis.smembers("users"));//获取所有加入的value
        System.out.println(jedis.sismember("users", "who"));//判断 who 是否是u」
        ↪  ser集合的元素
        System.out.println(jedis.srandmember("users"));
        System.out.println(jedis.scard("users"));//返回集合的元素个数
    }

    @Test
    public void testSort() throws InterruptedException {
        //jedis 排序
        //注意，此处的rpush和lpush是List的操作。是一个双向链表（但从表现来看的）
        jedis.del("a");//先清除数据，再加入数据进行测试
        jedis.rpush("a", "1");
        jedis.lpush("a","6");
        jedis.lpush("a","3");
        jedis.lpush("a","9");
        System.out.println(jedis.lrange("a",0,-1));// [9, 3, 6, 1]
        System.out.println(jedis.sort("a")); //[1, 3, 6, 9]  //输入排序后结果
        System.out.println(jedis.lrange("a",0,-1));
    }

}
```

- 连接池连接

```
package com.test;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;

public class TestJedisPool {

    public static void main(String[] args) {
        //1、获取连接池配置对象，设置配置项
        JedisPoolConfig config = new JedisPoolConfig();
        config.setMaxTotal(20); //最大连接数
        config.setMaxIdle(10); //最大空闲连接数
        config.setMaxWaitMillis(3000); //最大等待时间

        //2、初始化连接池
        JedisPool jedisPool = new JedisPool(config,"127.0.0.1",7501);
```

```
        //3、从连接池中获取到一个连接
        Jedis jedis = jedisPool.getResource();

        //4、操作函数
        jedis.set("name","BCS");//向key-->name中放入了value-->BCS
        System.out.println(jedis.get("name"));//执行结果：BCS

        //5、关闭当前连接，返回连接池
        jedis.close();
    }
}
```

16.0.3 使用php客户端连接

```
<?php
    //连接本地的 BES CacheServer服务
   $redis = new Redis();
   $redis->connect('127.0.0.1', 7501);
   echo "Connection to server successfully";
         //查看服务是否运行
   echo "Server is running: " . $redis->ping();
?>


执行脚本，输出结果为:
Connection to server sucessfully
Server is running: PONG
```

- Redis PHP String(字符串) 实例

```
<?php
   //连接本地的 BES  CacheServer 服务
   $redis = new Redis();
   $redis->connect('127.0.0.1', 7501);
   echo "Connection to server successfully";
   //设置  BES  CacheServer 字符串数据
   $redis->set("tutorial-name", "BES CacheServer");
   // 获取存储的数据并输出
   echo "Stored string in BCS:: " . $redis->get("tutorial-name");
?>

执行脚本，输出结果为:
Connection to server sucessfully
Stored string in BCS:: BES CacheServer
```

- Redis PHP List(列表) 实例

```
<?php
   //连接本地的 BES  CacheServer 服务
   $redis = new Redis();
   $redis->connect('127.0.0.1', 7501);
   echo "Connection to server successfully";
```

```
   //存储数据到列表中
   $redis->lpush("tutorial-list", "BCS");
   $redis->lpush("tutorial-list", "Mongodb");
   $redis->lpush("tutorial-list", "Mysql");
   // 获取存储的数据并输出
   $arList = $redis->lrange("tutorial-list", 0 ,5);
   echo "Stored string in BES CacheServer";
   print_r($arList);
?>

执行脚本，输出结果为:
Connection to server sucessfully
Stored string in BES CacheServer
Mysql
Mongodb
BCS
```

- Redis PHP Keys 实例

```
<?php
   //连接本地的 BES  CacheServer 服务
   $redis = new Redis();
   $redis->connect('127.0.0.1', 7501);
   echo "Connection to server successfully";
   // 获取数据并输出
   $arList = $redis->keys("*");
   echo "Stored keys in BES CacheServer:: ";
   print_r($arList);
?>

执行脚本，输出结果为:
Connection to server sucessfully
Stored string in BES CacheServer::
tutorial-name
tutorial-list
```