**用户操作手册**

**2021年 12 月**

**（ 版本号 1.0 ）**

文档信息

|  |  |  |  |
| --- | --- | --- | --- |
| 编写者 | 杨毅辉、赵鹏举、马彦峰、王志超 | 编写日期 | 2021-12 |
| 审核者 |  | 审核日期 |  |
| 批准人 |  | 批准日期 |  |

变更历史

|  |  |  |
| --- | --- | --- |
| 日期 | 变更描述 | 批准 |
| 2021-12-24 | 用户操作手册初步编写完成。 |  |
|  |  |  |
|  |  |  |

**目 录**

[文档信息 2](#_Toc91841153)

[变更历史 2](#_Toc91841154)

[第一章 引言 １](#_Toc91841155)

[1.1 编写目的 １](#_Toc91841156)

[1.2 面向读者 １](#_Toc91841157)

[1.3 编写原则 １](#_Toc91841158)

[1.4 参考资料 １](#_Toc91841159)

[第二章 系统简介 ２](#_Toc91841160)

[第三章 功能详述 ３](#_Toc91841161)

[3.1 缓存管理 ３](#_Toc91841162)

[3.1.1 集群管理 ３](#_Toc91841163)

[3.1.1.1 搜索 ３](#_Toc91841164)

[3.1.1.2 新增 ３](#_Toc91841165)

[3.1.1.3 初始化（集群模式） ４](#_Toc91841166)

[3.1.1.4 节点管理（集群模式） ６](#_Toc91841167)

[1.主节点 ７](#_Toc91841168)

[2.从节点 １２](#_Toc91841169)

[3.添加主节点 １７](#_Toc91841170)

[4.数据迁移 １８](#_Toc91841171)

[3.1.1.5 初始化（哨兵模式） １８](#_Toc91841172)

[3.1.1.6 节点管理（哨兵模式） １９](#_Toc91841173)

[1.哨兵节点 １９](#_Toc91841174)

[2.主节点（被监控的主从集群） ２１](#_Toc91841175)

[3.从节点（被监控的主从集群） ２５](#_Toc91841176)

[4.添加哨兵节点 ２６](#_Toc91841177)

[5.监控主从集群 ２７](#_Toc91841178)

[3.1.1.7 初始化（主从模式） ２８](#_Toc91841179)

[3.1.1.8 节点管理（主从模式） ２９](#_Toc91841180)

[1.主节点 ２９](#_Toc91841181)

[2.从节点 ３１](#_Toc91841182)

[3.1.1.9 复制 ３５](#_Toc91841183)

[3.1.1.10 编辑 ３６](#_Toc91841184)

[3.1.1.11 更多 ３６](#_Toc91841185)

[3.1.1.12 设置密码 ３７](#_Toc91841186)

[3.1.1.13 清空缓存 ３７](#_Toc91841187)

[3.1.1.14 集群的一键启动/停止 ３８](#_Toc91841188)

[3.1.1.15 删除集群 ３８](#_Toc91841189)

[3.1.2 持久化 ３９](#_Toc91841190)

[3.1.2.1 搜索 ３９](#_Toc91841191)

[3.1.2.2 设置 ４０](#_Toc91841192)

[3.1.3 淘汰策略 ４１](#_Toc91841193)

[3.1.3.1 搜索 ４１](#_Toc91841194)

[3.1.3.2 设置 ４２](#_Toc91841195)

[3.2 缓存审计 ４２](#_Toc91841196)

[3.2.1 搜索 ４２](#_Toc91841197)

[3.3 缓存分析 ４３](#_Toc91841198)

[3.3.1 大Key分析 ４３](#_Toc91841199)

[3.3.1.1 搜索 ４３](#_Toc91841200)

[3.3.1.2 编辑 ４３](#_Toc91841201)

[3.3.1.3 删除 ４５](#_Toc91841202)

[3.3.2 热Key分析 ４５](#_Toc91841203)

[3.3.2.1 搜索 ４５](#_Toc91841204)

[3.3.2.2 详情 ４５](#_Toc91841205)

[3.3.2.3 删除 ４６](#_Toc91841206)

[3.3.2.4 阈值设置 ４６](#_Toc91841207)

[3.3.3 慢日志查询 ４７](#_Toc91841208)

[3.3.3.1 搜索 ４７](#_Toc91841209)

[3.3.3.2 阈值设置 ４７](#_Toc91841210)

[3.4 缓存监控 ４８](#_Toc91841211)

[3.4.1 集群监控 ４８](#_Toc91841212)

[3.4.2 节点监控 ４８](#_Toc91841213)

[3.4.3 客户端监控 ４９](#_Toc91841214)

[3.5 组织多中心架构 ５０](#_Toc91841215)

[3.5.1 多中心架构设置 ５０](#_Toc91841216)

[3.5.1.1 搜索 ５０](#_Toc91841217)

[3.5.1.2 新增 ５０](#_Toc91841218)

[3.5.1.3 编辑 ５１](#_Toc91841219)

[3.5.1.4 删除 ５１](#_Toc91841220)

[3.5.1.5 中心管理 ５１](#_Toc91841221)

[1.添加主中心 ５１](#_Toc91841222)

[2.关联备中心 ５２](#_Toc91841223)

[3.启动/停止 ５３](#_Toc91841224)

[3.删除 ５３](#_Toc91841225)

[3.5.2 多中心监控 ５３](#_Toc91841226)

[3.5.3 多中心数据迁移 ５４](#_Toc91841227)

[3.5.3.1 创建任务 ５４](#_Toc91841228)

[3.5.3.2 启动 ５５](#_Toc91841229)

[3.5.3.3 详情 ５５](#_Toc91841230)

[3.5.3.4 删除 ５５](#_Toc91841231)

[3.6 ACL鉴权 ５６](#_Toc91841232)

[3.6.1 新增 ５６](#_Toc91841233)

[3.6.2 编辑 ５６](#_Toc91841234)

[3.6.3 密码修改 ５７](#_Toc91841235)

[3.6.4 删除 ５７](#_Toc91841236)

[3.7 控制台 ５８](#_Toc91841237)

[3.7.1 命令输入 ５８](#_Toc91841238)

[3.7.2 支持的命令 ５８](#_Toc91841239)

[3.8 SDK ５９](#_Toc91841240)

[3.8.1 引入依赖包 ５９](#_Toc91841241)

[3.8.1.1 方式一：通过私服（推荐） ５９](#_Toc91841242)

[3.8.1.2 方式二：直接覆盖本地仓库 ５９](#_Toc91841243)

[3.8.2 配置文件 ６０](#_Toc91841244)

[3.8.2.1 各种模式配置文件 ６０](#_Toc91841245)

[3.8.2.2 相关配置说明 ６１](#_Toc91841246)

[3.8.3 缓存使用 ６２](#_Toc91841247)

[3.8.4 其他接口说明 ６３](#_Toc91841248)

# 引言

## 编写目的

为了帮助用户更好地了解和使用该系统平台，特编写本手册。在本手册中，我们分别对本系统不同权限的用户分别进行详细讲解，包括各功能模块的详细操作步骤和配置方式。使用户能够快速了解本系统的所有功能以及使用方法。

## 面向读者

使用本系统的用户。

## 编写原则

* **一致性**：对于用户来说，同样的文字、状态、按钮，都应该触发相同的事情，遵从通用的平台惯例，也就是，同一用语、功能、操作保持一致。
* **完整性**：用户、进程或者硬件组件具有能力，能够验证所发送或传送的东西的准确性，并且进程或硬件组件不会被以任何方式改变。
* **无二义性**： 每个句子只存在一颗语法树。

## 参考资料

无

# 系统简介

本系统为PaaS平台下的缓存中心，主要功能提现在对缓存的管理与查看上。所涵盖的功能模块主要有缓存管理、缓存审计、缓存分析、缓存监控、组织多中心架构、ACL鉴权、控制台等，其中缓存管理下又分为集群管理、持久化、淘汰策略；缓存分析下又分为大Key分析、热Key分析、慢日志查询；缓存监控下又分为集群监控、节点监控、客户端监控；组织多中心架构下又分为多中心架构设置、多中心监控、多中心数据迁移。

本系统的主要功能涉及到对redis集群的操作，包括增删改查等，也有对用户权限的操作，包括授予或收回用户的权限。

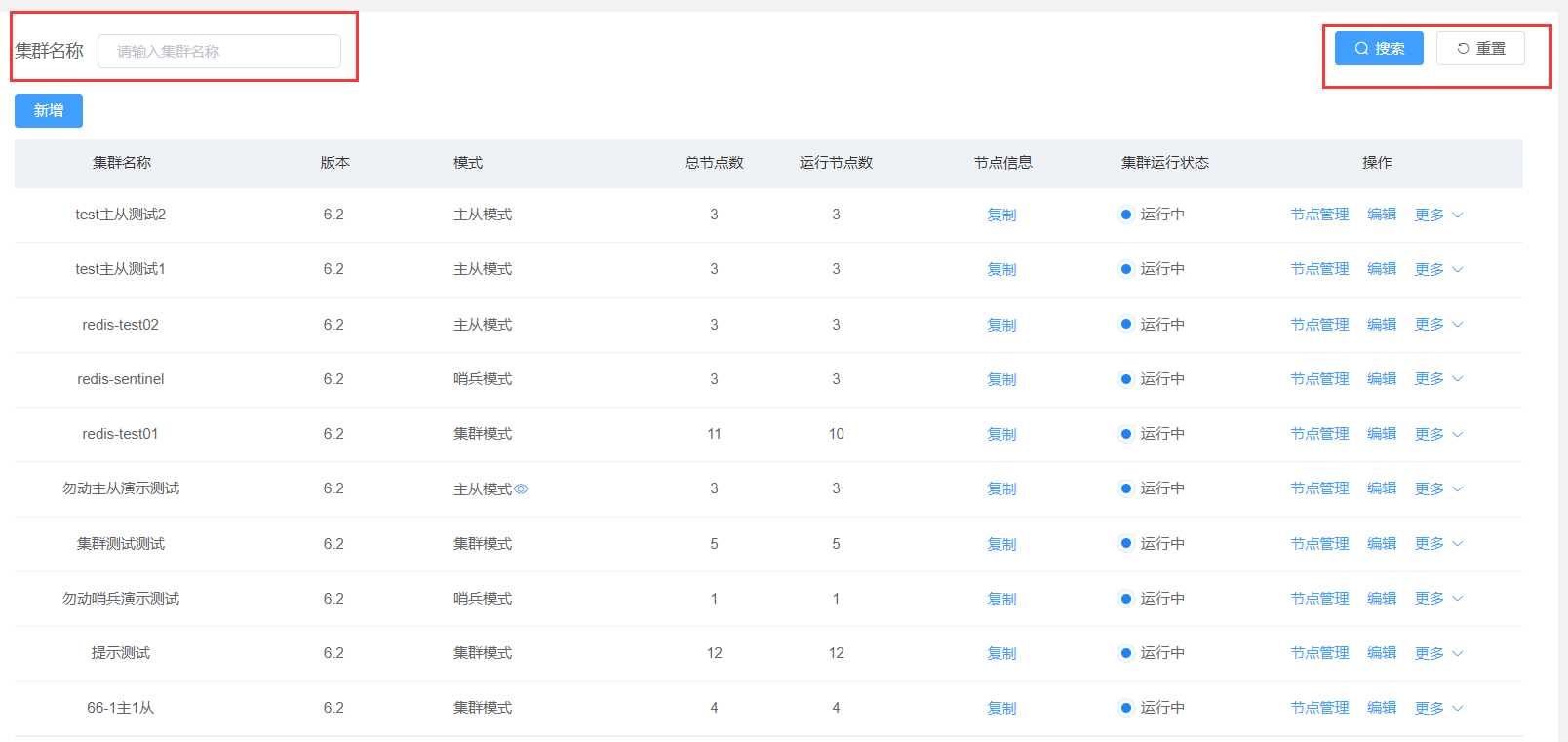
# 功能详述

## 缓存管理

### 集群管理

#### 搜索

上方搜索框可以输入集群名称来进行查询，该搜索框支持模糊查询，输入完成集群名称后点击“搜索”按钮即可展示出符合条件的集群，点击“重置”按钮会清空搜索框里的内容，同时会将所有的集群都展示出来。

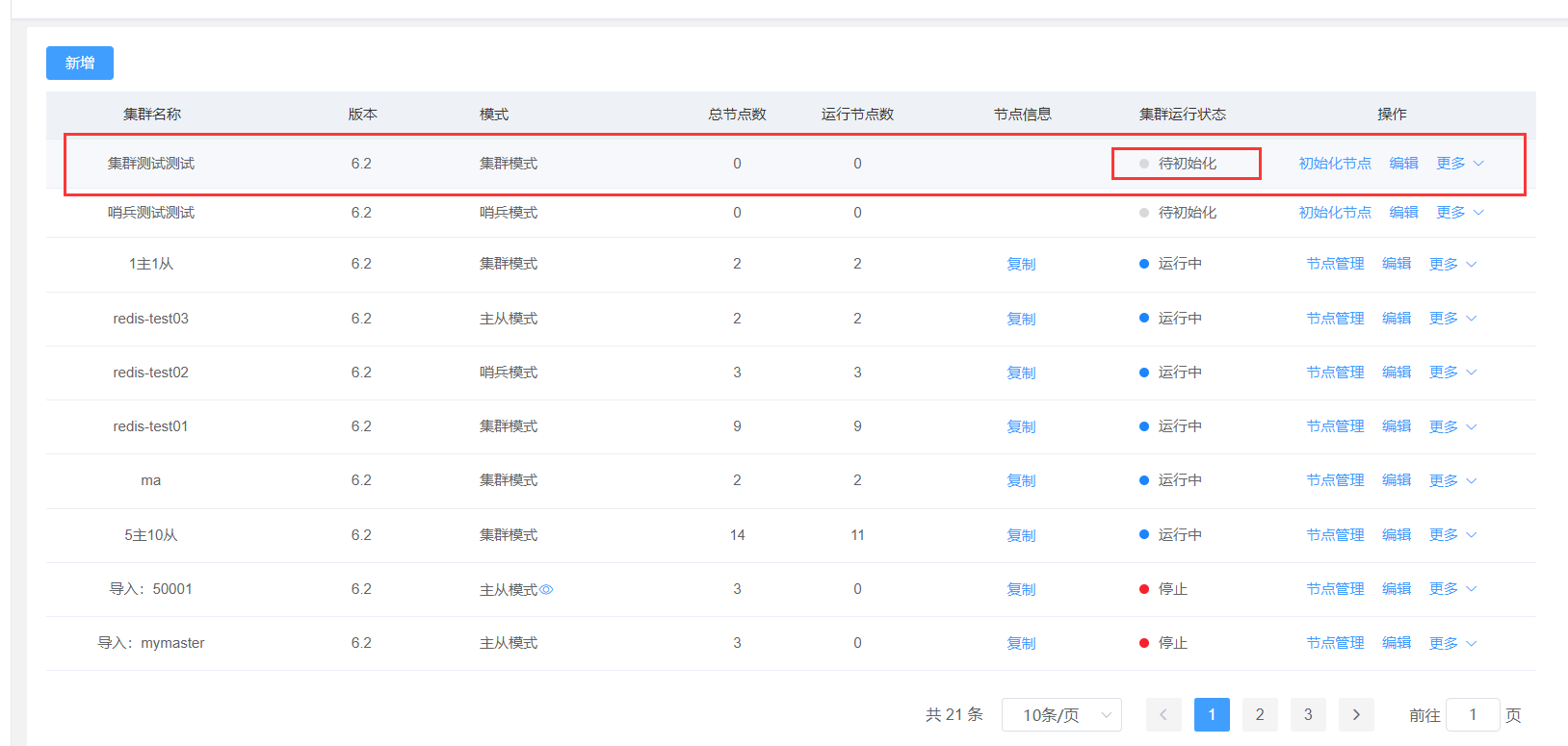


#### 新增

新增集群，点击 “新增”按钮，填入集群名称，选择其他信息后就可以点击“确定”来保存集群信息。集群名称不能输入已存在的集群名称，如果输入会提示名称重复；版本可以选择四种版本，根据需求进行选择；模式有三种模式，每种模式的结构都不相同，集群模式下是多主多从的结构，主从模式下是一主多从的结构，哨兵模式下会有一个或多个哨兵，用来监控主从模式的集群；组织，省、市与机房有二级联动，选择组织和省、市后机房才会有数据供选择。



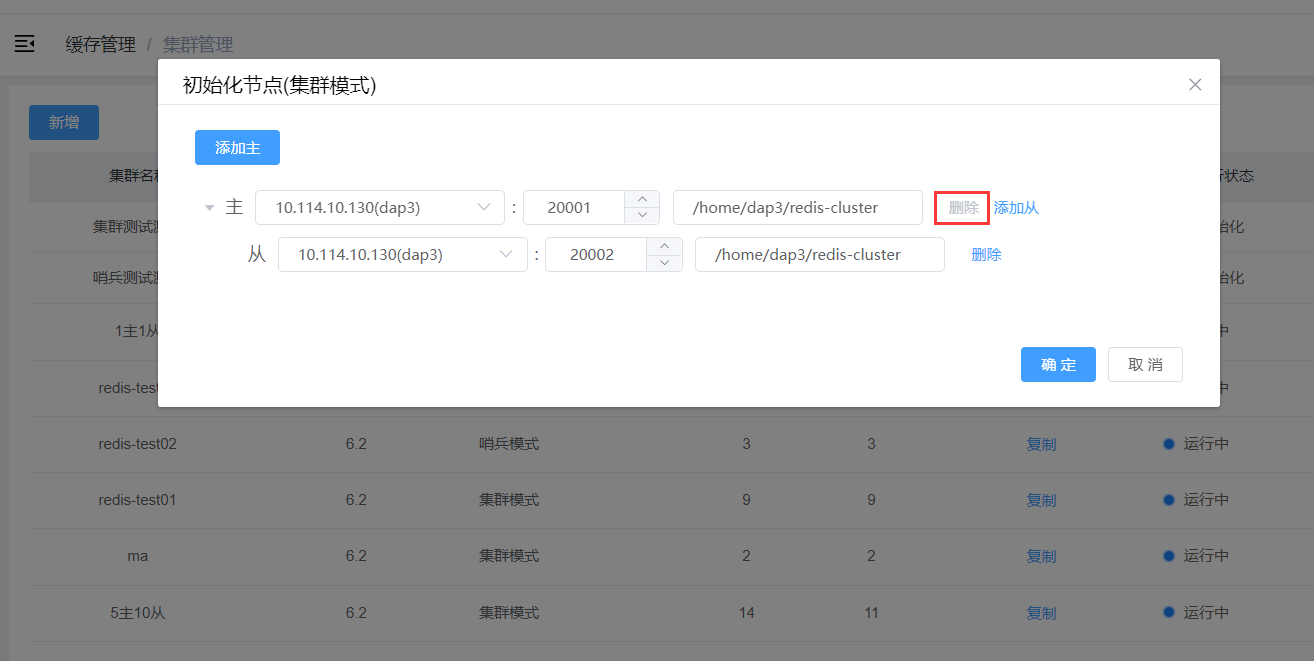
集群新增后处于待初始化的状态，此时集群下没有节点，也就不能进行节点管理。三种集群模式新增成功后都是待初始化状态，都需要先进行初始化才能有节点，才能进行节点管理。

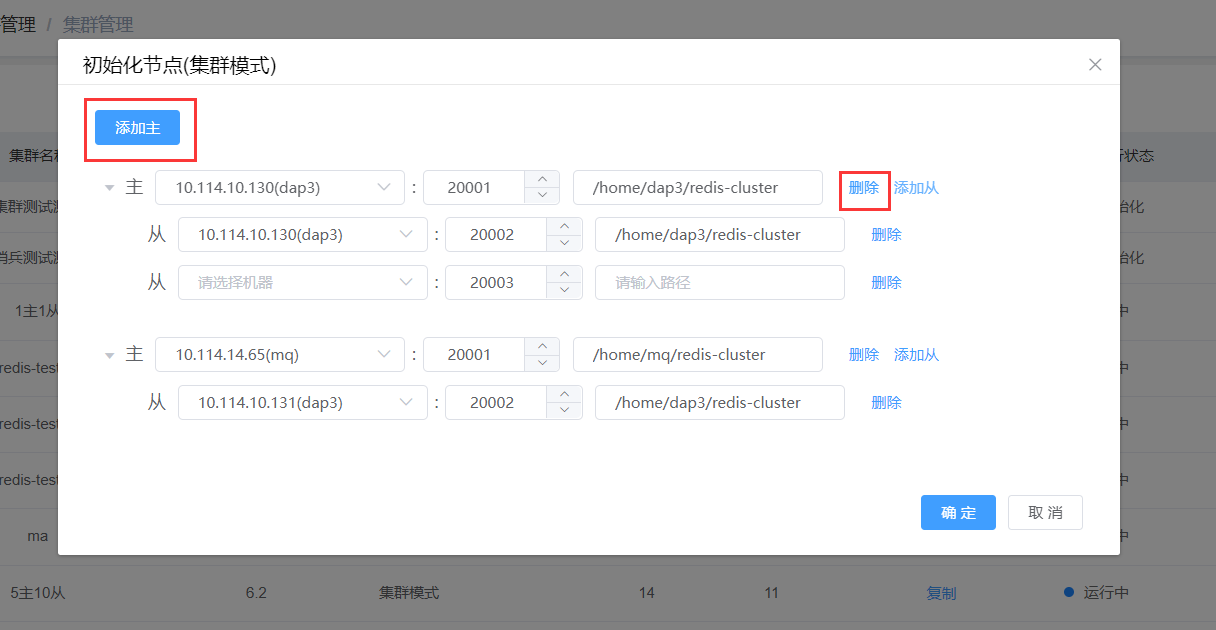


#### 初始化（集群模式）

点击“初始化节点”按钮会对节点进行初始化，初始化节点时根据不同的集群模式，界面会有所变化，下面是集群模式的集群初始化。

集群模式下的集群是多主多从的模式，只有一个主节点的情况下不允许删除主节点，可以添加和删除从节点。点击“添加主”按钮后会再添加一个主节点和从节点，如果不需要也可以删除从节点，有了两个或以上的主节点后，就可以删除主节点了，主节点后的删除二字由灰色变为蓝色，点击“删除”后会删除该主节以及其下面的所有从节点。





选择好主从节点后点击“确定”按钮会进行初始化。



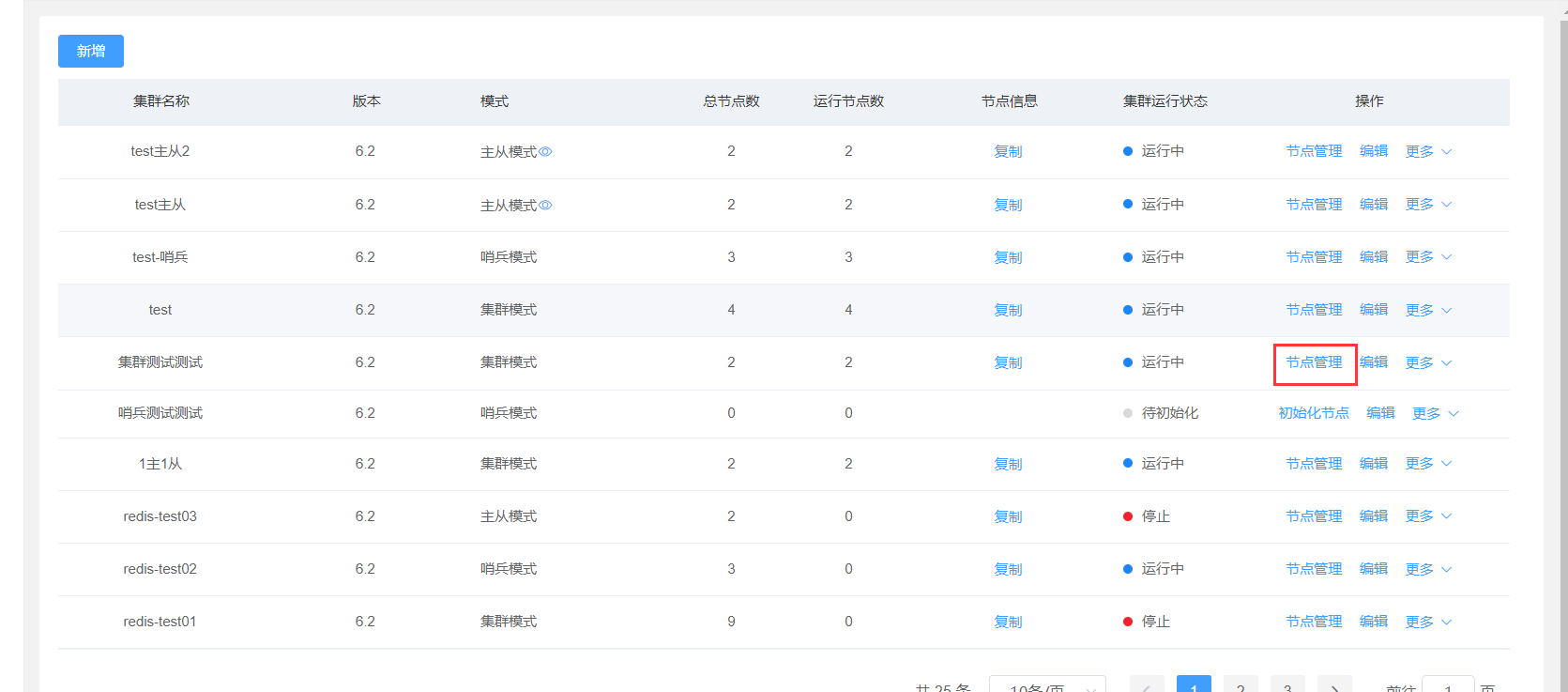


初始化成功后集群运行状态就为运行中。

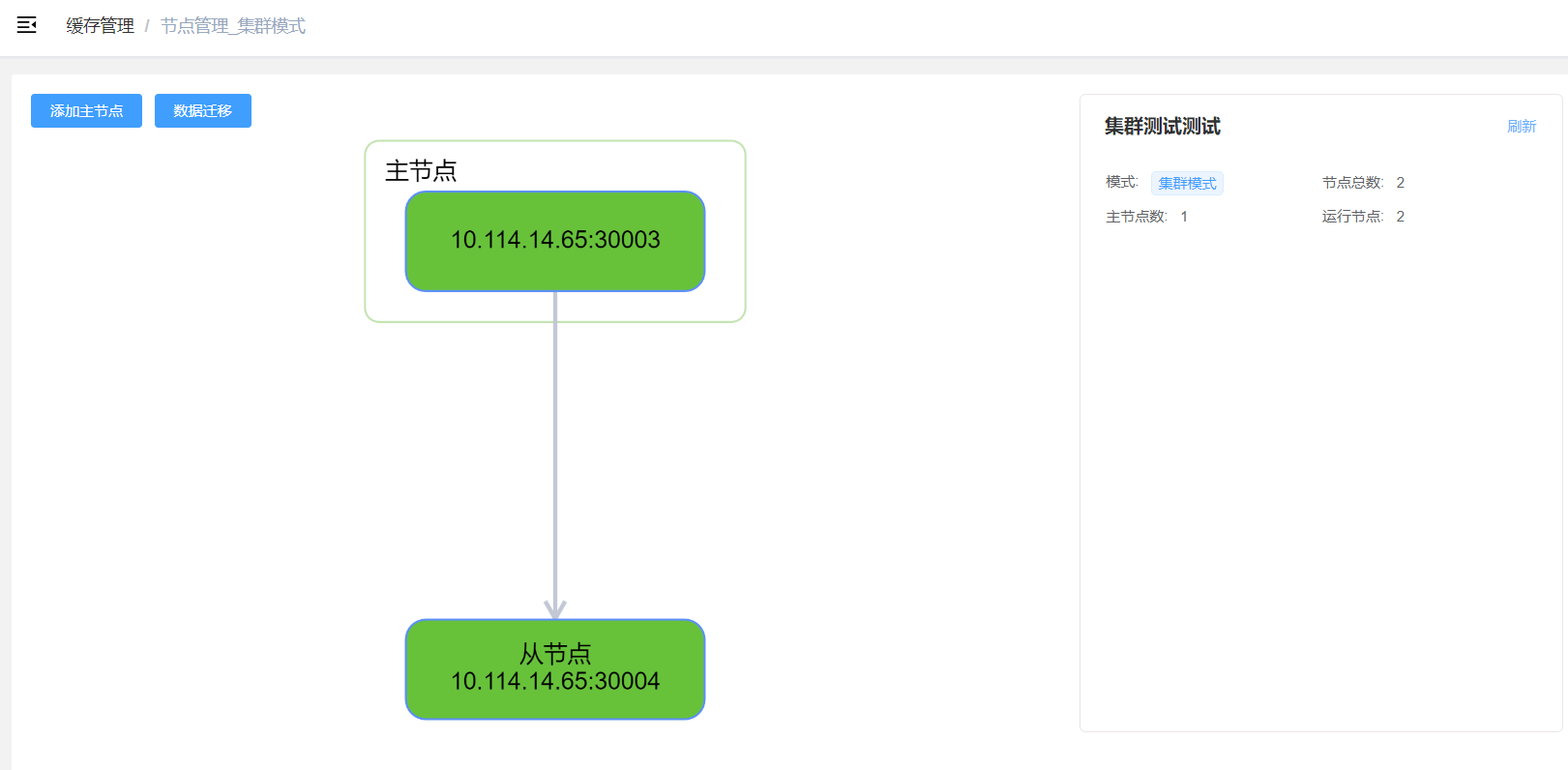


#### 节点管理（集群模式）

点击“节点管理”按钮会进入一个新页面



进入界面后节点的关联关系会通过图像展示出来。



##### 1.主节点

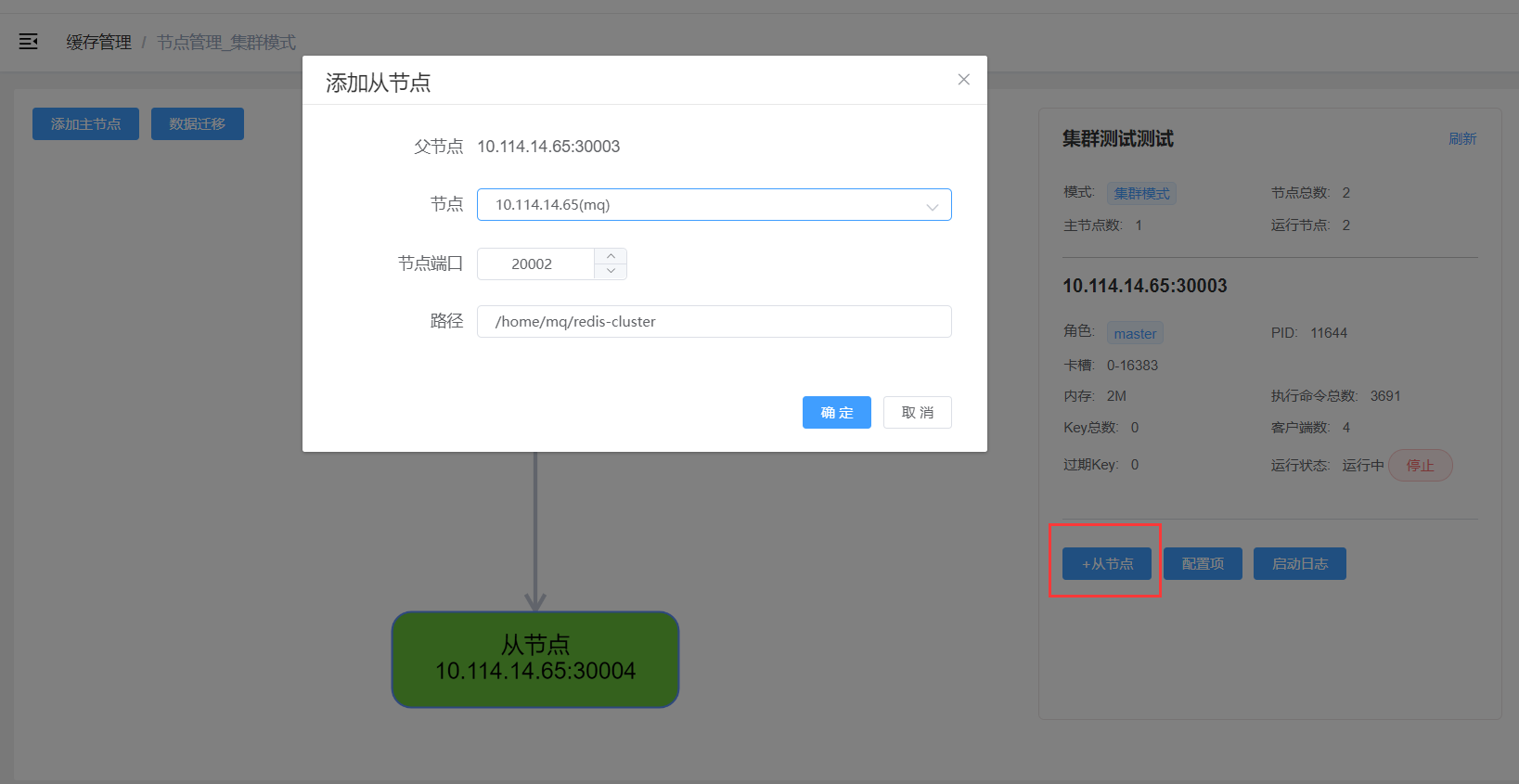
###### 主节点图像

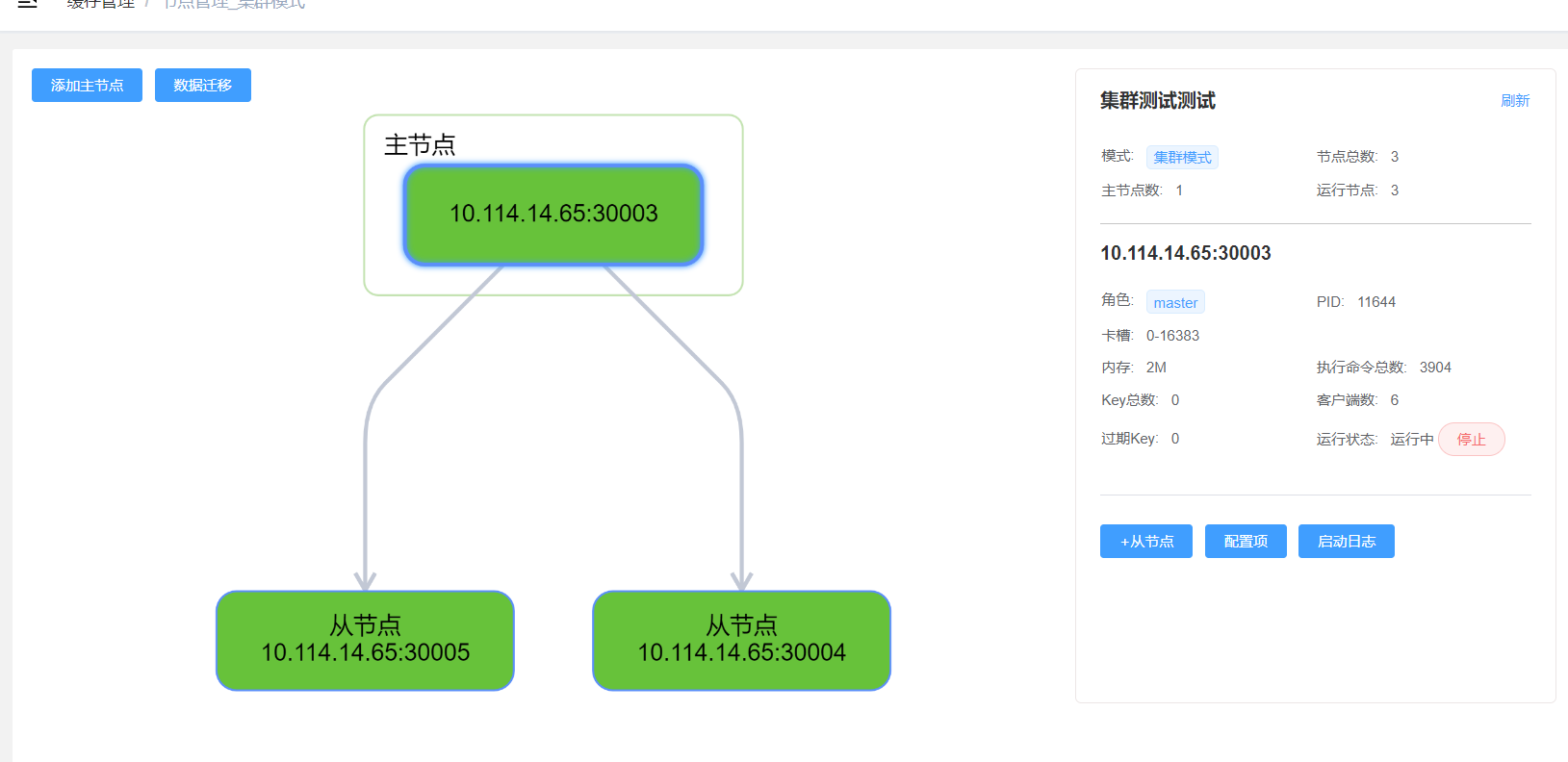
点击一个节点图像后会在右侧展示出该节点的详细信息。



###### +从节点

点击“+从节点”按钮，弹框中的父节点是选中的主节点，根据需求填入信息后点击“确定”按钮，成功后会在图像上显示出已经添加的从节点。



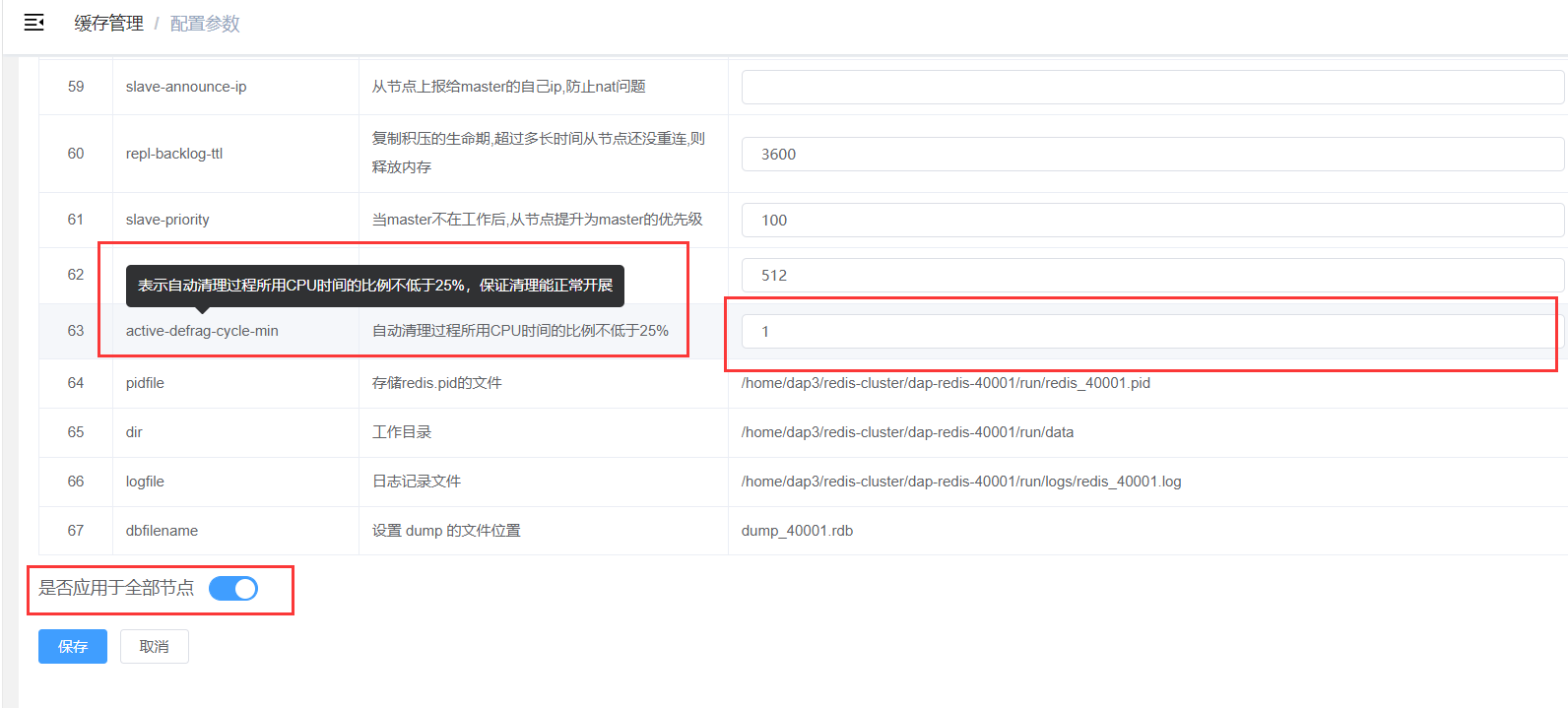


###### 配置项

在主节点图像上点击“配置项”按钮后会进入一个新界面，

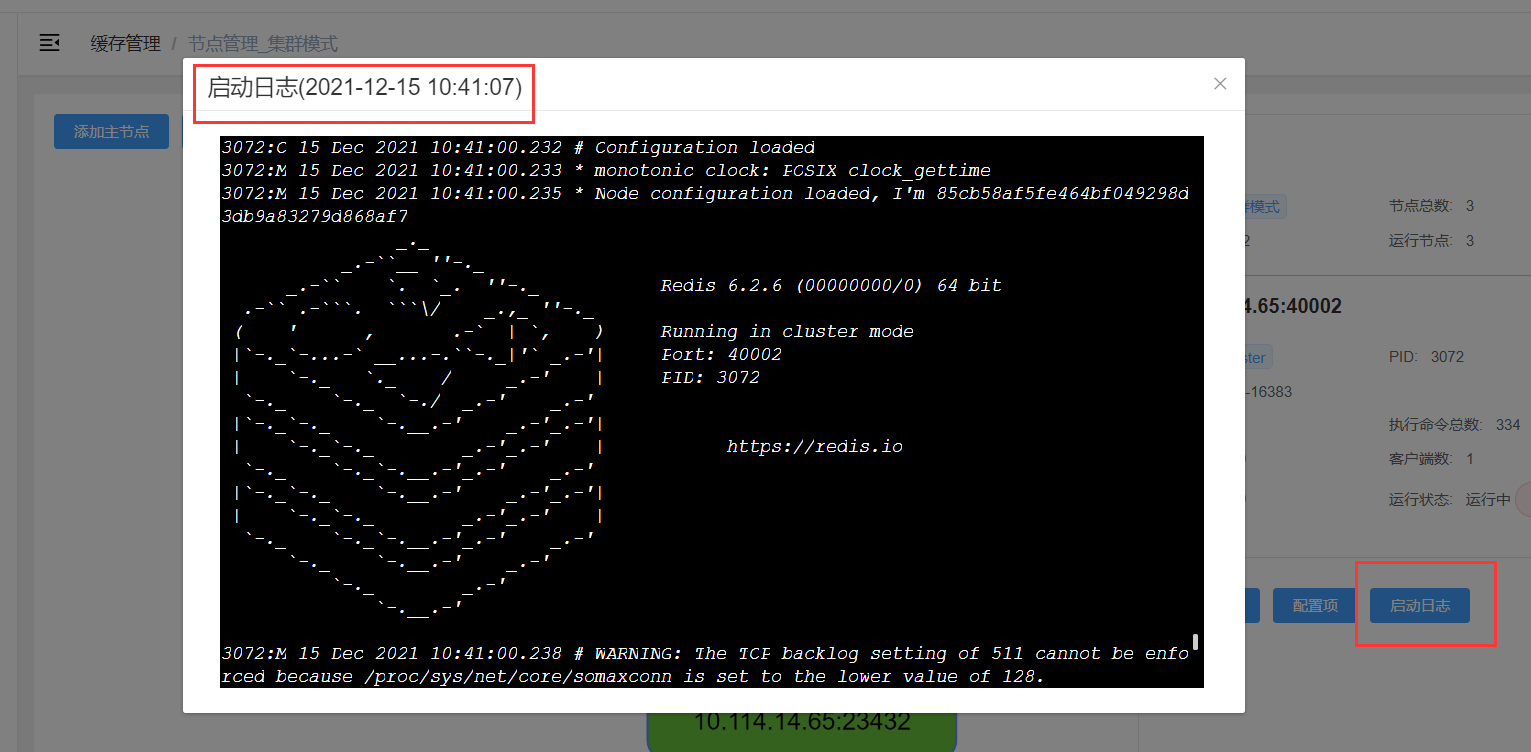


在配置项界面中，有几十条关于redis节点的配置，可以将鼠标移动到某个配置项上，就可以查看该配置项代表的含义，可以进行修改的配置项后会有文本框提供修改，不可修改的配置项则没有文本框。下方有“是否应用于全部节点”的开关，打开开关后再点击“保存”就可以将此处修改的配置项保存到该集群下的所有节点上。



###### 启动日志

点击“启动日志”按钮后，会弹出这个节点的启动日志，可以用来查看节点是否正常启动停止。



###### 主节点的启动/停止

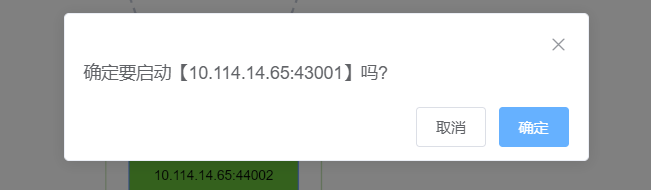
在右侧详细信息上有运行状态一列，当节点运行状态为“运行中”时，其右侧的按钮为“停止”，点击“停止”按钮后弹出弹框会提示是否停止该节点，点击“确定”后则会停止该节点，同时运行状态为“停止”，其右侧按钮变为“启动”。点击“启动”按钮后弹出弹框会提示是否启动该节点，点击“确定”后则会启动该节点，同时运行状态变为“运行中”。

当节点状态为停止时，“配置项”按钮会隐藏。









###### 移除节点

当主节点状态为停止时，“配置项”按钮会被隐藏，取而代之的是“移除节点”按钮，点击“移除节点”按钮后会有弹窗提示是否移除该节点，点击“确定”按钮后会再次弹窗提示是否删除本地文件，可根据需求进行删除，之后就会将该节点移除。存在卡槽的主节点不能进行移除操作。





##### 2.从节点

###### 从节点的启动/停止

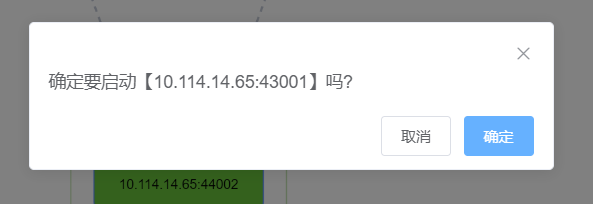
在右侧详细信息上有运行状态一列，当节点运行状态为“运行中”时，其右侧的按钮为“停止”，点击“停止”按钮后弹出弹框会提示是否停止该节点，点击“确定”后则会停止该节点，同时运行状态为“停止”，其右侧按钮变为“启动”。点击“启动”按钮后弹出弹框会提示是否启动该节点，点击“确定”后则会启动该节点，同时运行状态变为“运行中”。

当节点状态为停止时，“配置项”按钮会隐藏。





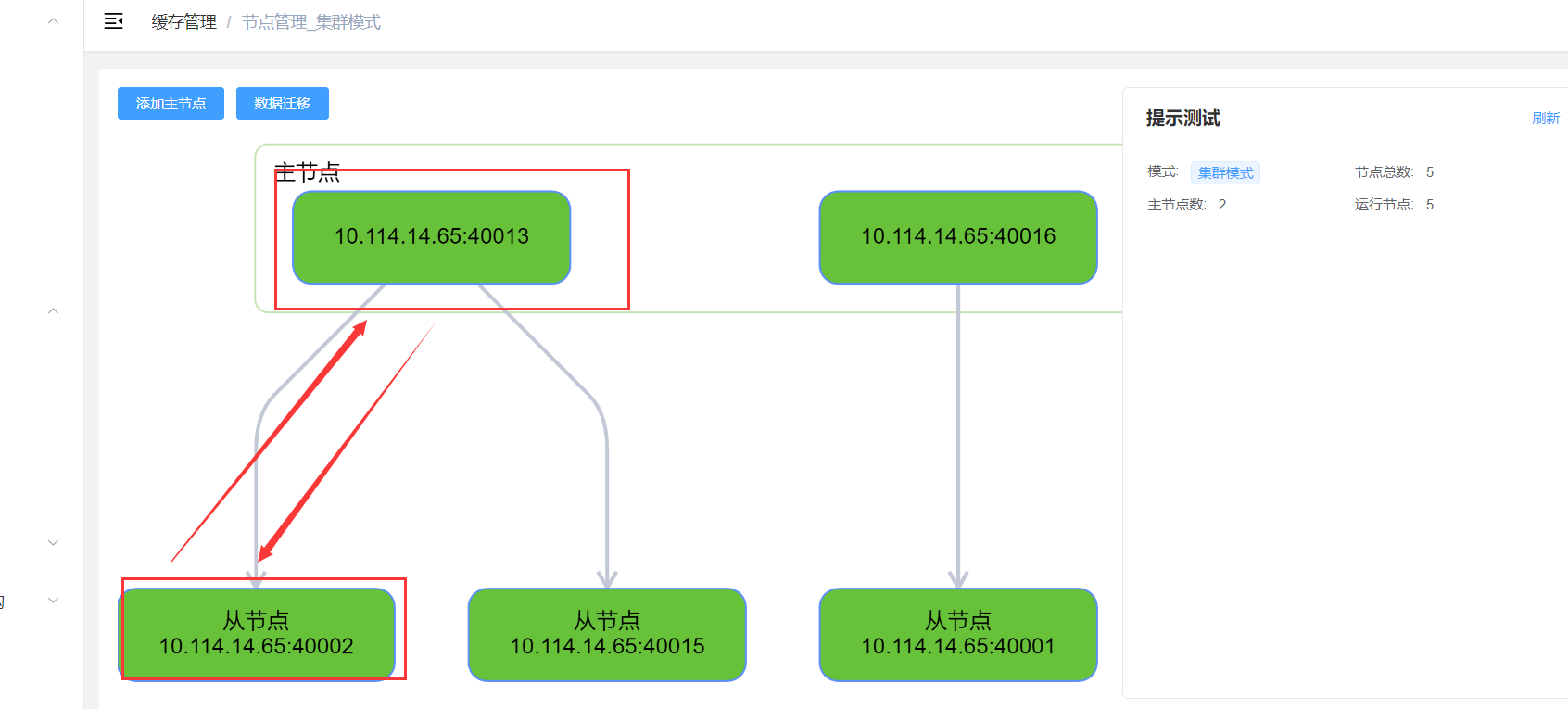




###### 故障切换

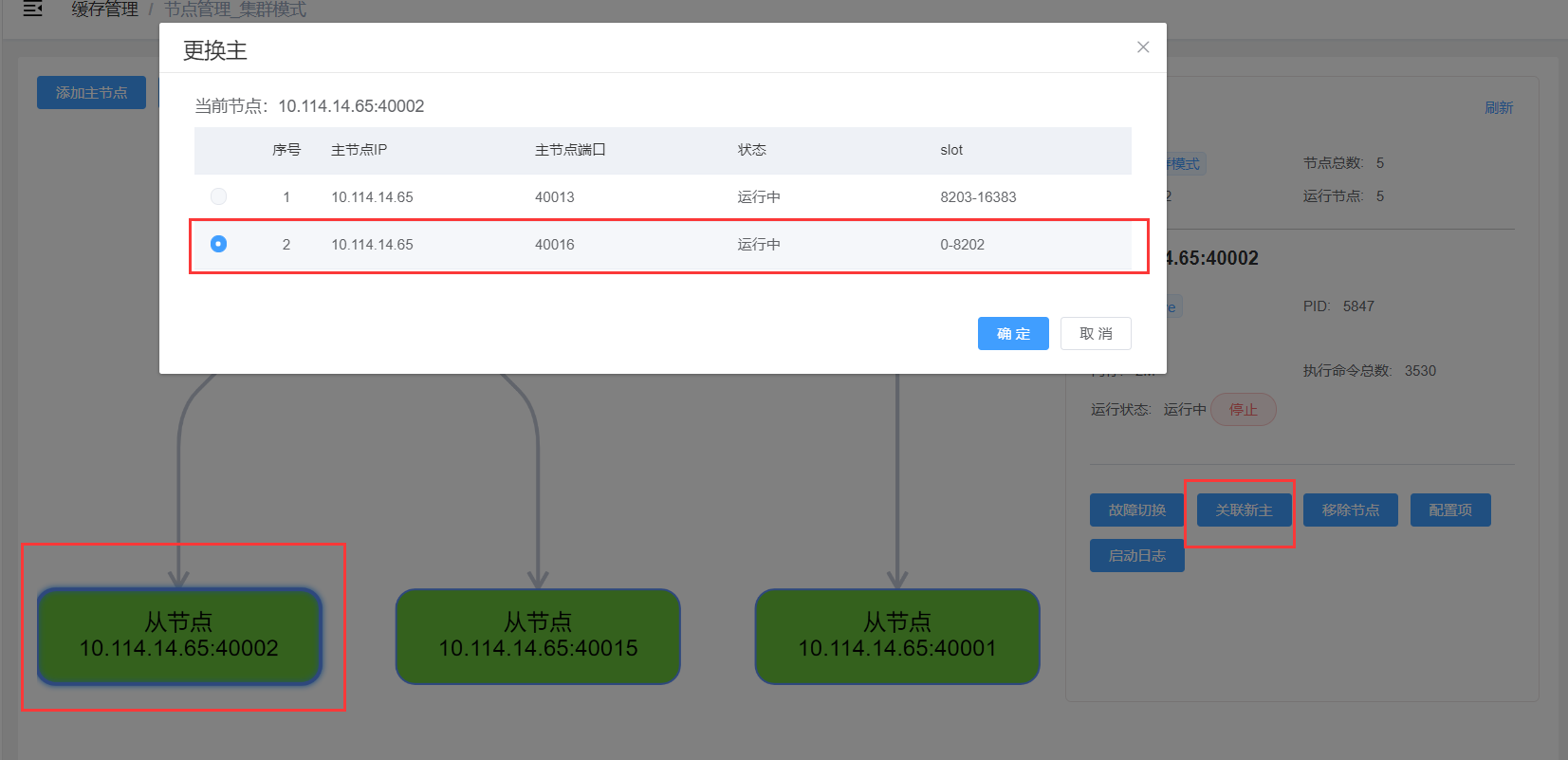
选择从节点，点击“故障切换”按钮后会有弹窗提示，点击“确定”按钮后，会将该从节点升为主节点，其意思是该从节点会取代其关联的主节点，原来的主节点就会成为该节点下的从节点。

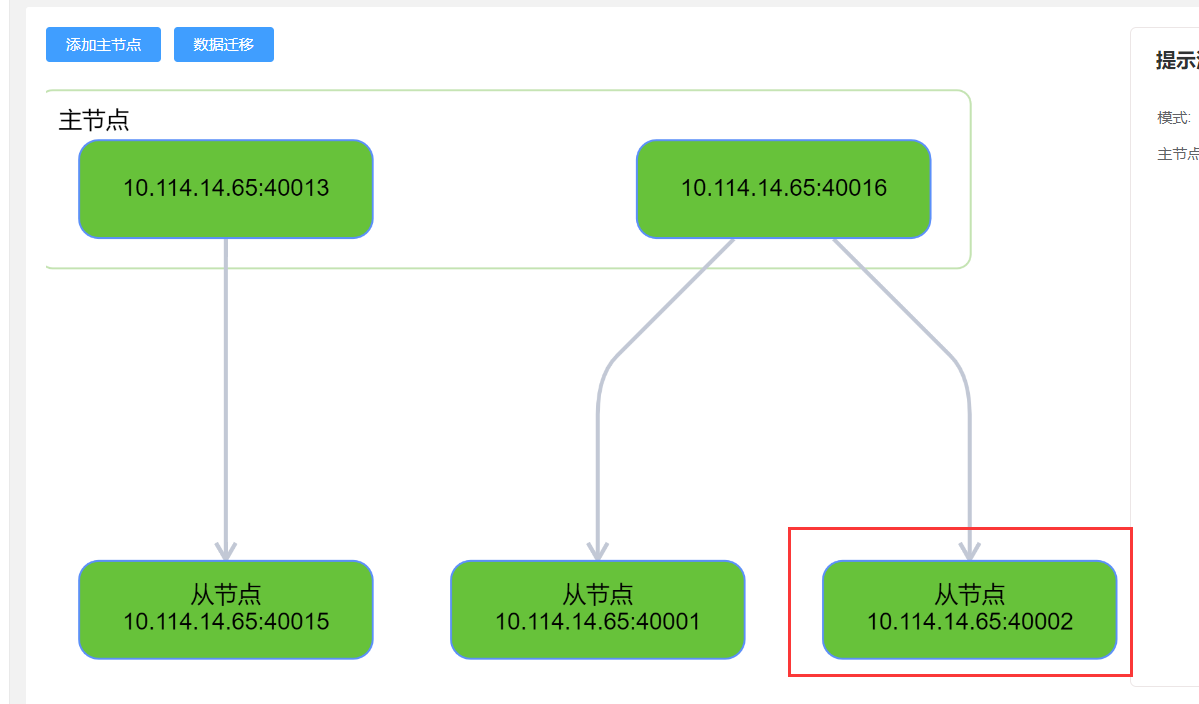




###### 关联新主

“关联新主”按钮是将该从节点关联到另一主节点下，可以在弹窗中进行选择，选择好想要关联的新主节点后点击“确定”就会将该从节点关联到选择的主节点下。

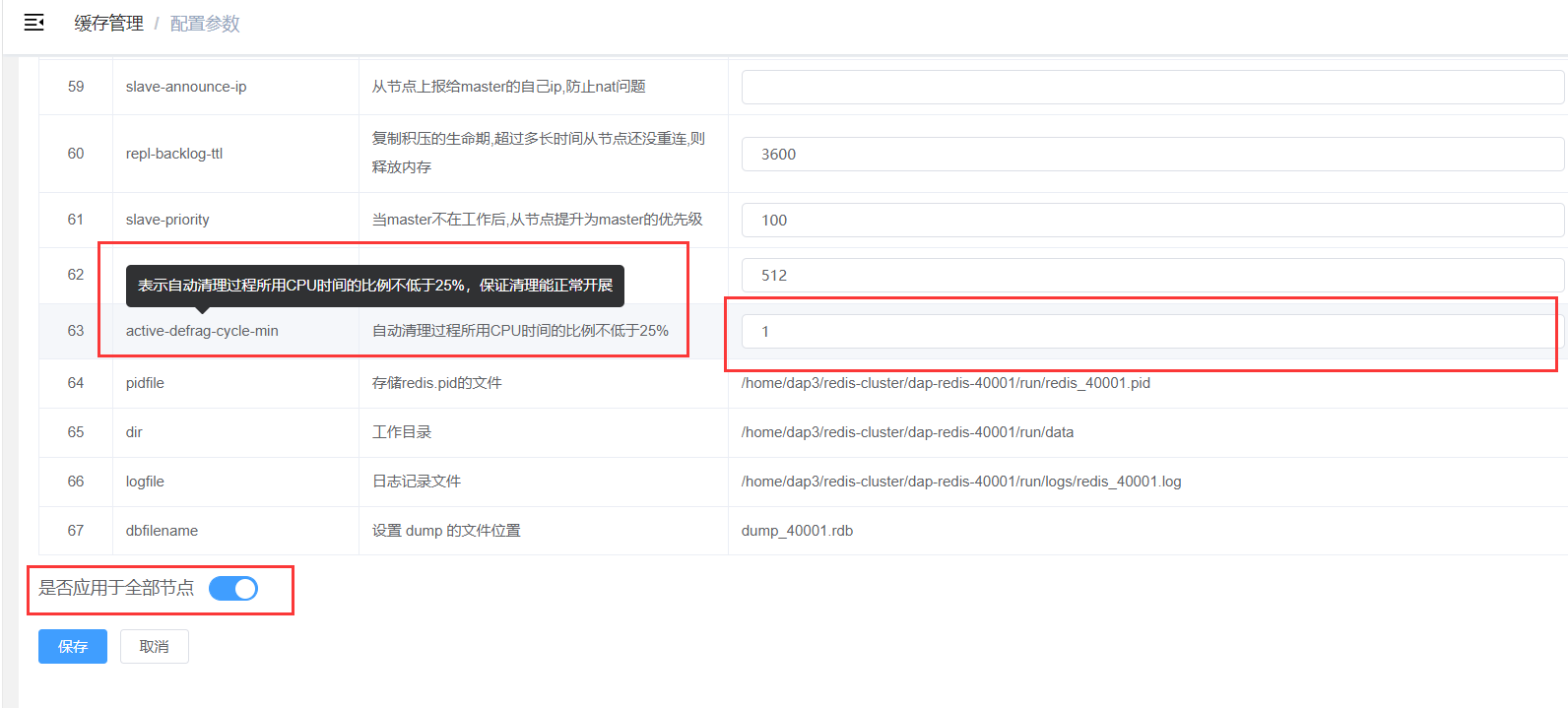




###### 配置项

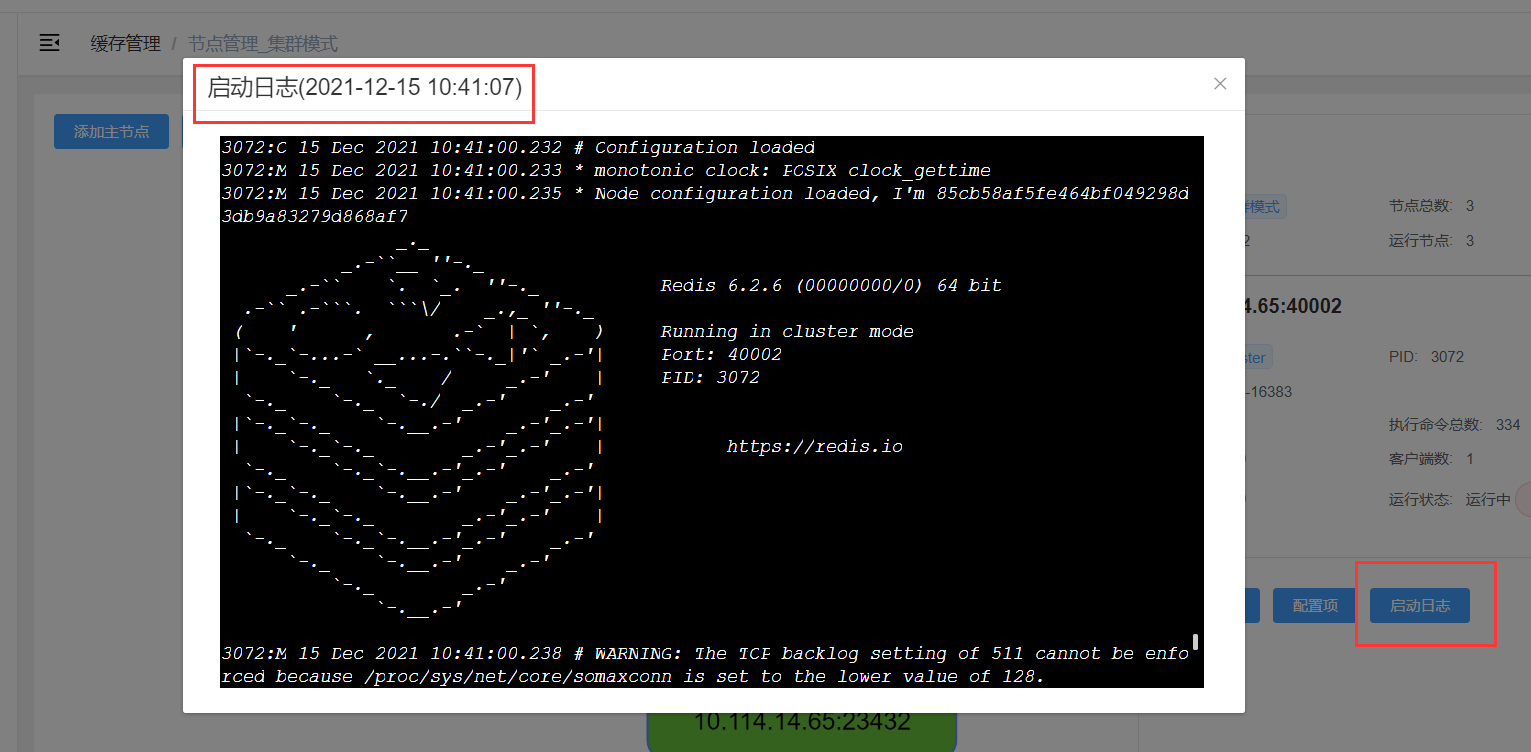
点击“配置项”按钮会进入一个新节面。

在配置项界面中，有几十条关于redis节点的配置，可以将鼠标移动到某个配置项上，就可以查看该配置项代表的含义，可以进行修改的配置项后会有文本框提供修改，不可修改的配置项则没有文本框。下方有“是否应用于全部节点”的开关，打开开关后再点击“保存”就可以将此处修改的配置项保存到该集群下的所有节点上。



###### 启动日志

点击“启动日志”按钮后，会弹出这个节点的启动日志，可以用来查看节点是否正常启动停止。



###### 移除节点

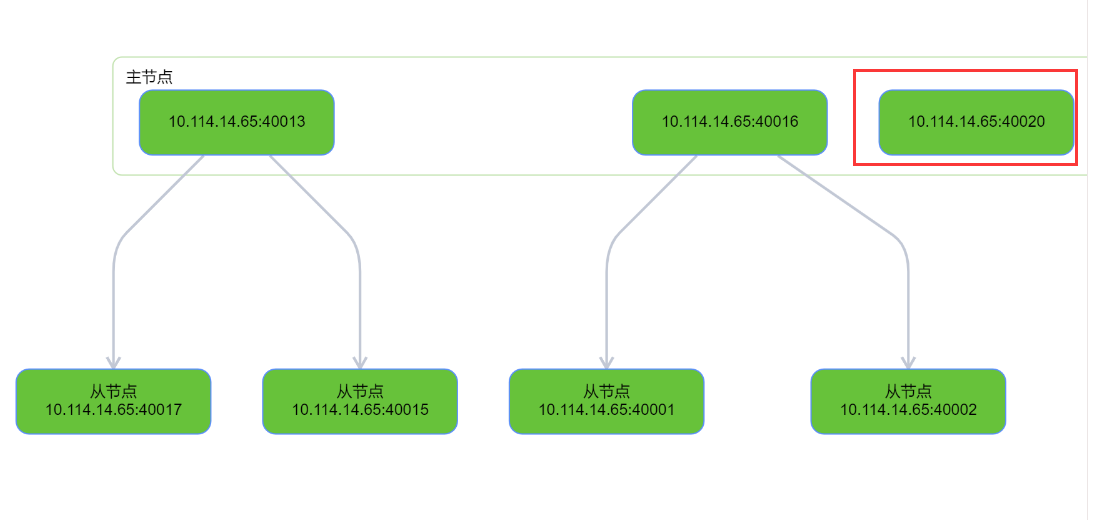
点击“移除节点”按钮后会有弹窗提示是否删除该节点，点击“确定”后会再次提示是否删除本地文件，可以根据需求进行删除。



##### 3.添加主节点

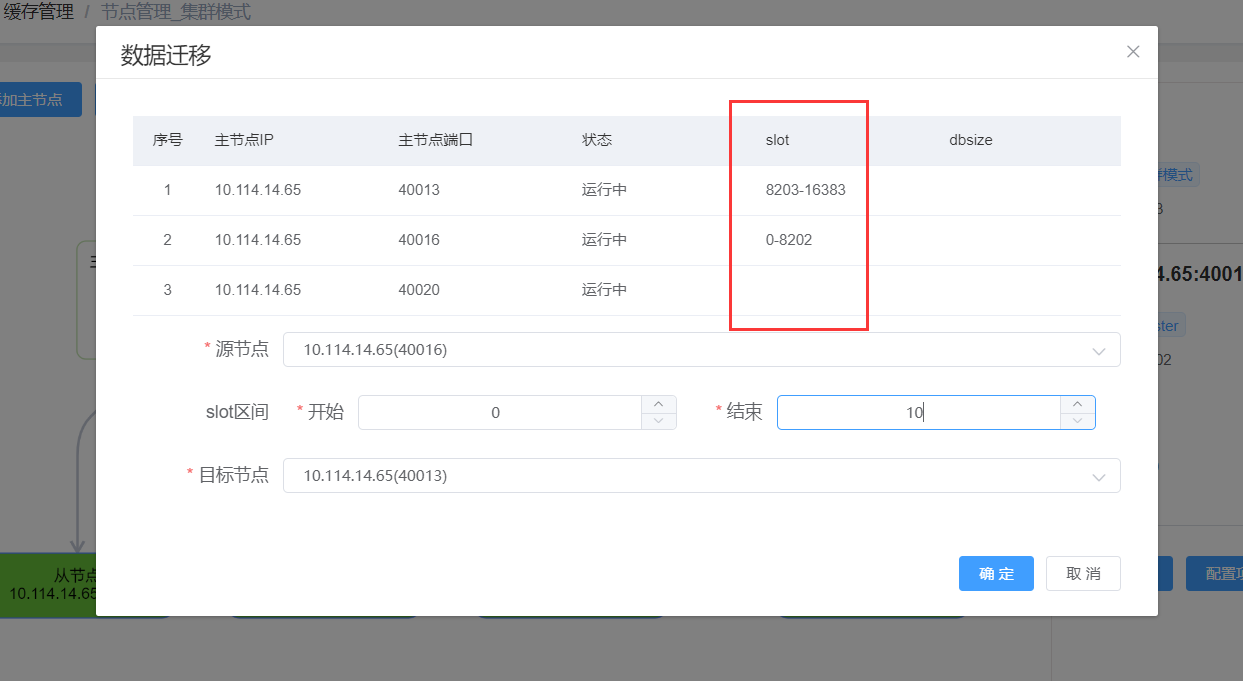
“添加主节点”按钮会在该集群下添加一个主节点，选择好节点与端口后点击“确定”按钮就可以添加成功。





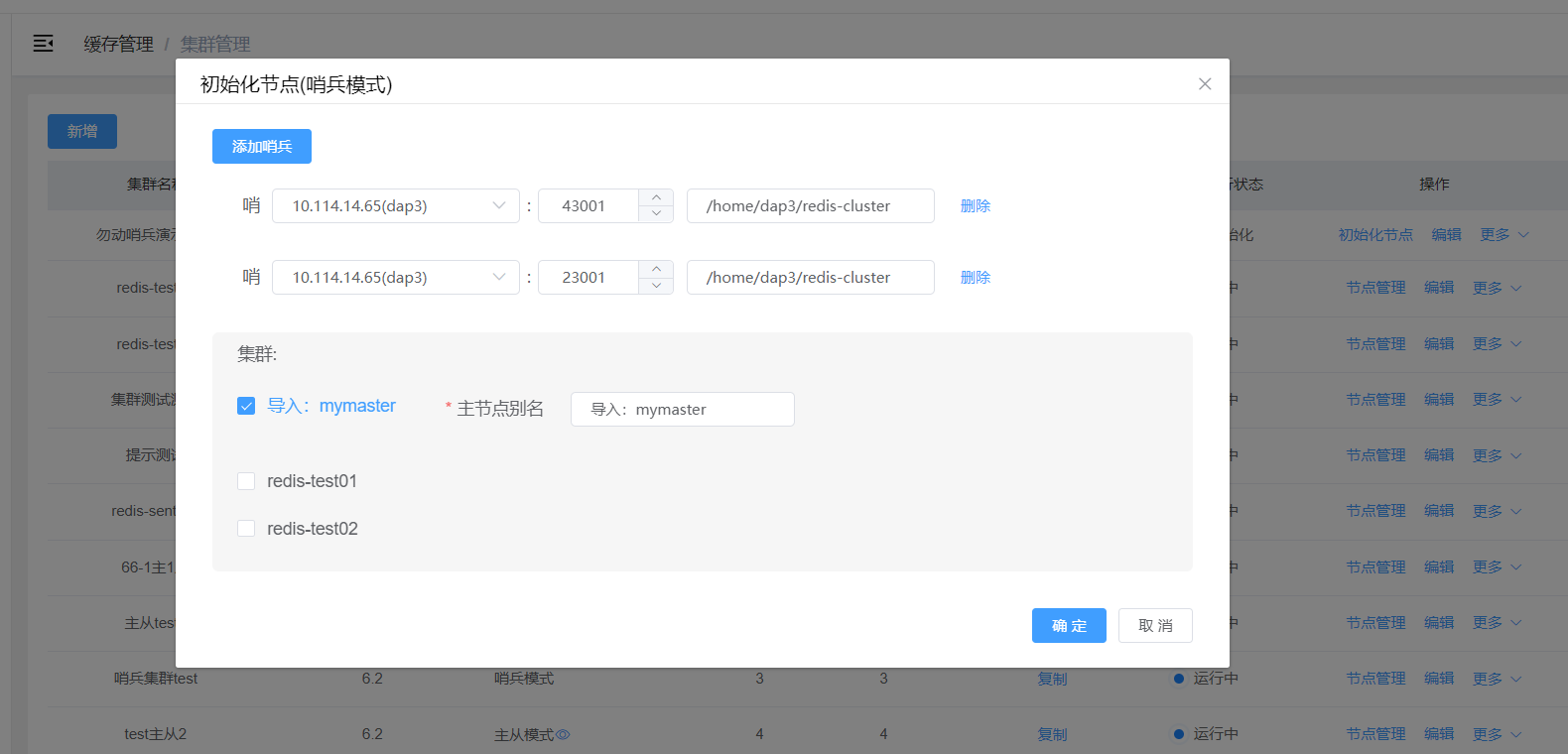
##### 4.数据迁移

“数据迁移”按钮会将源节点的数据迁移到目标节点上，点击按钮后弹出弹框，根据需求进行选择，其中slot区间的开始与结束可以根据上方列表的slot字段下的大小来进行输入，且源节点与目标节点都需要处于“运行中”的状态才可以。点击“确定”按钮就可以进行数据迁移了。



#### 初始化（哨兵模式）

哨兵模式下，redis集群没有主从节点，有的只是哨兵节点，哨兵节点可以有一个或多个，点击“添加哨兵”按钮就可以添加一个哨兵节点，当只有一个哨兵节点时“删除”按钮为灰色，不能删除，当有多个哨兵节点时，“删除”按钮变为蓝色，就可以进行删除了。哨兵模式是一种特殊的模式，首先Redis提供了哨兵的命令，哨兵是一个独立的进程，作为进程，它会独立运行。其原理是哨兵通过发送命令，等待Redis服务器响应，从而监控运行的多个Redis实例。下方的集群就是该哨兵模式可监控的集群，这些集群的模式为主从模式，没有被其他哨兵集群监控且redis版本必须与该哨兵模式的版本一致，选择一个集群后可以输入主节点别名，主节点别名只能输入数字、字母的组合。



#### 节点管理（哨兵模式）

哨兵模式下的图像不止有自己本身的哨兵节点，也有已监控的主从集群的主节点，同时右侧也会有节点的详细信息。

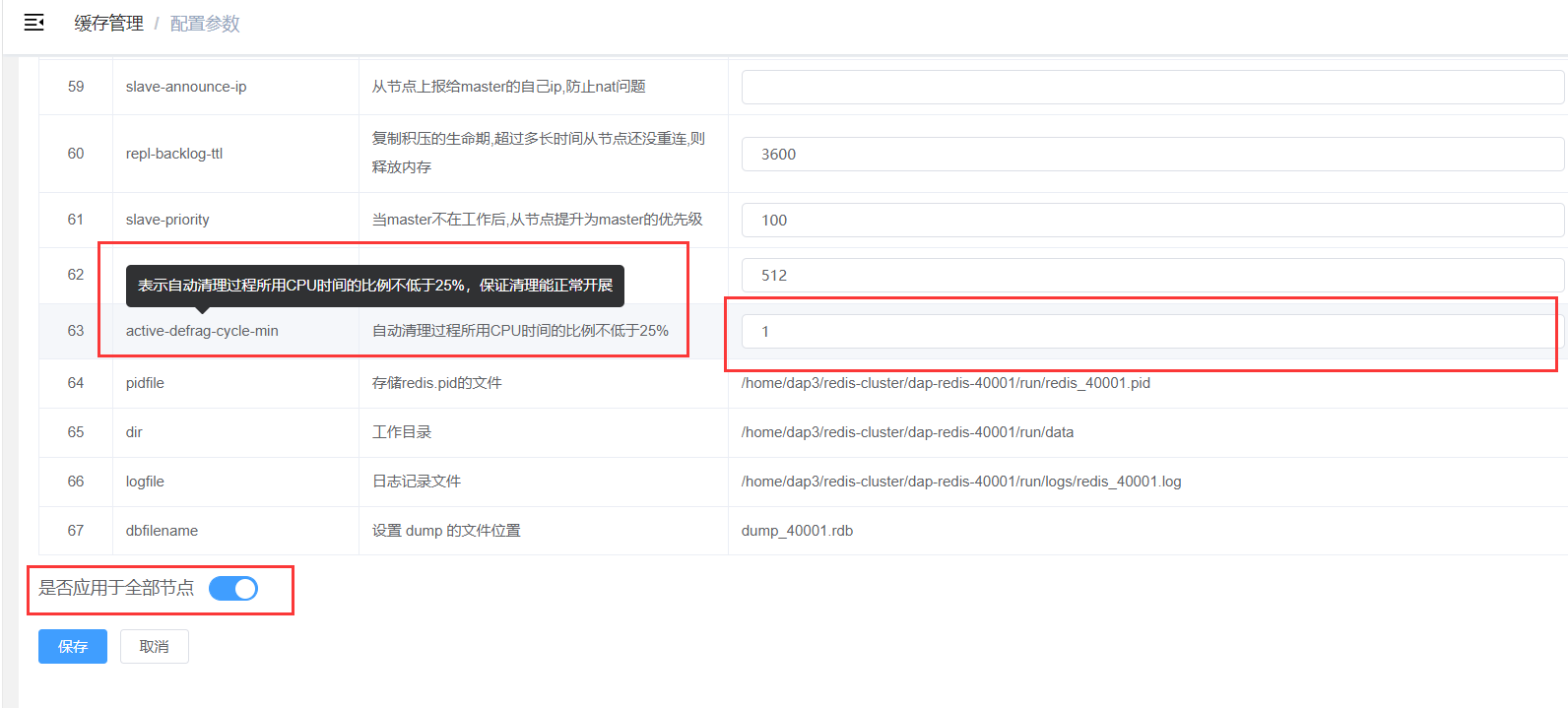
##### 1.哨兵节点

###### 配置项



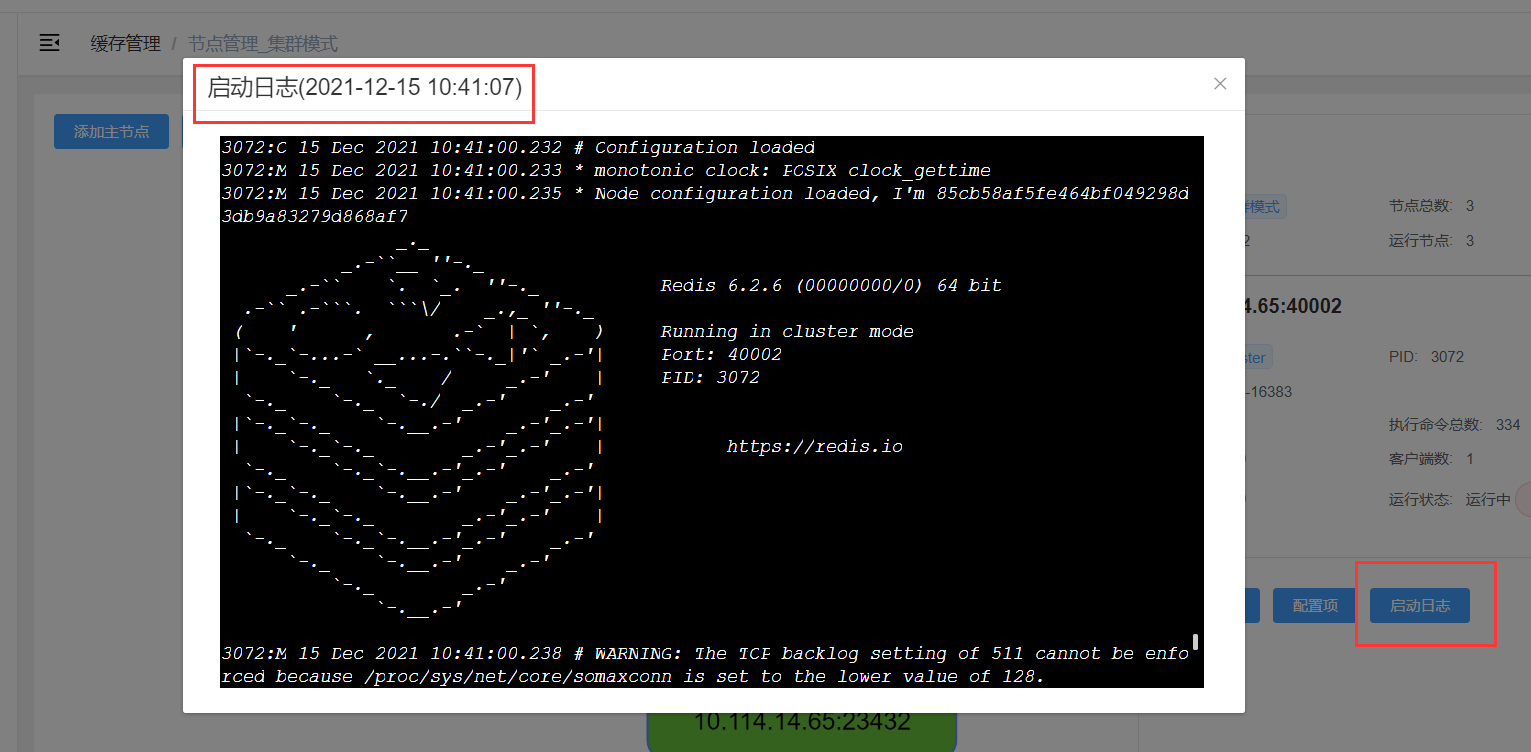
点击“配置项”按钮会进入一个新界面

在配置项界面中，有几十条关于redis节点的配置，可以将鼠标移动到某个配置项上，就可以查看该配置项代表的含义，可以进行修改的配置项后会有文本框提供修改，不可修改的配置项则没有文本框。下方有“是否应用于全部节点”的开关，打开开关后再点击“保存”就可以将此处修改的配置项保存到该集群下的所有节点上。



###### 启动日志

点击“启动日志”按钮后，会弹出这个节点的启动日志，可以用来查看节点是否正常启动停止。



###### 移除节点

点击“移除节点”按钮后会有弹窗提示是否删除该节点，点击“确定”后会再次提示是否删除本地文件，可以根据需求进行删除。

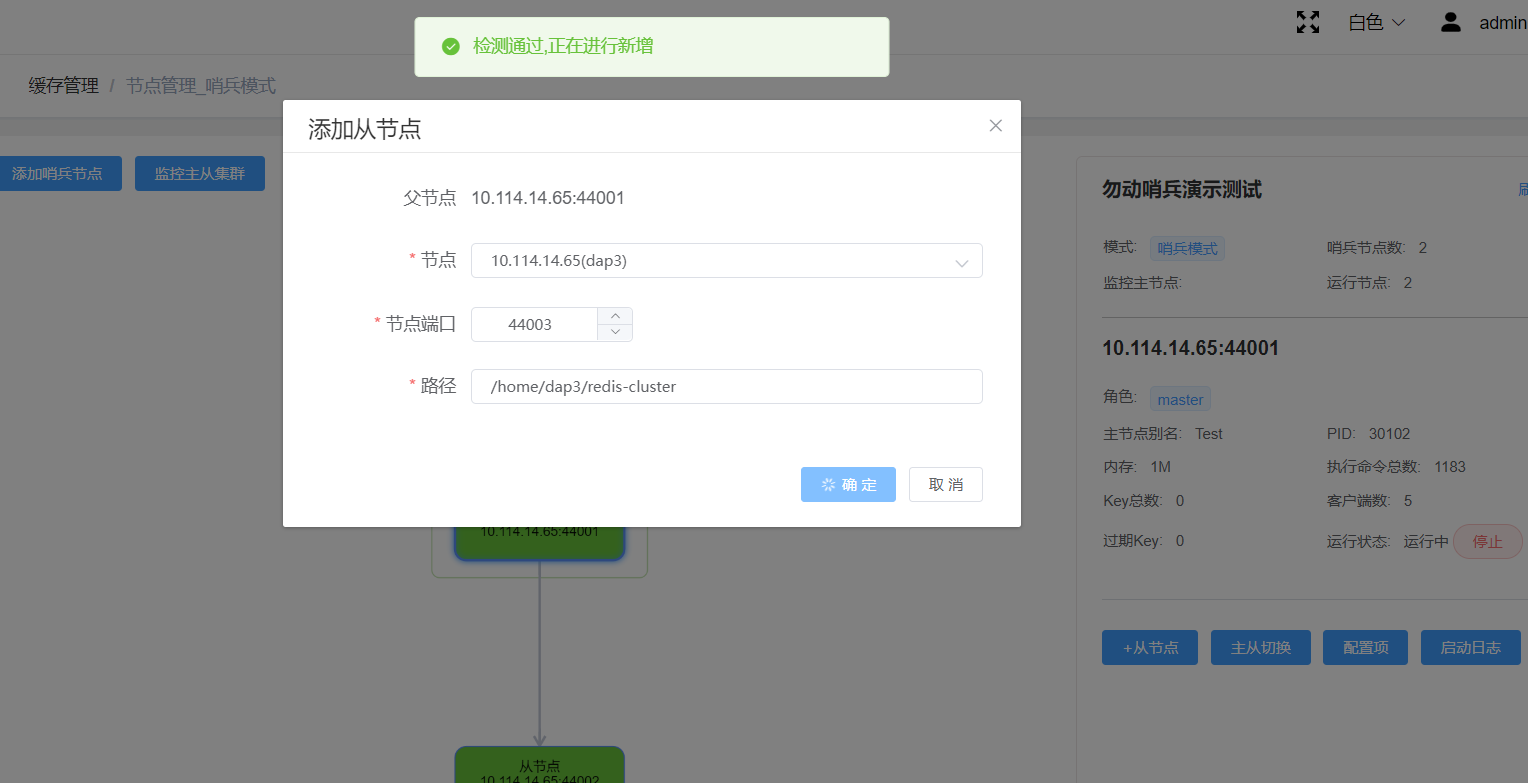


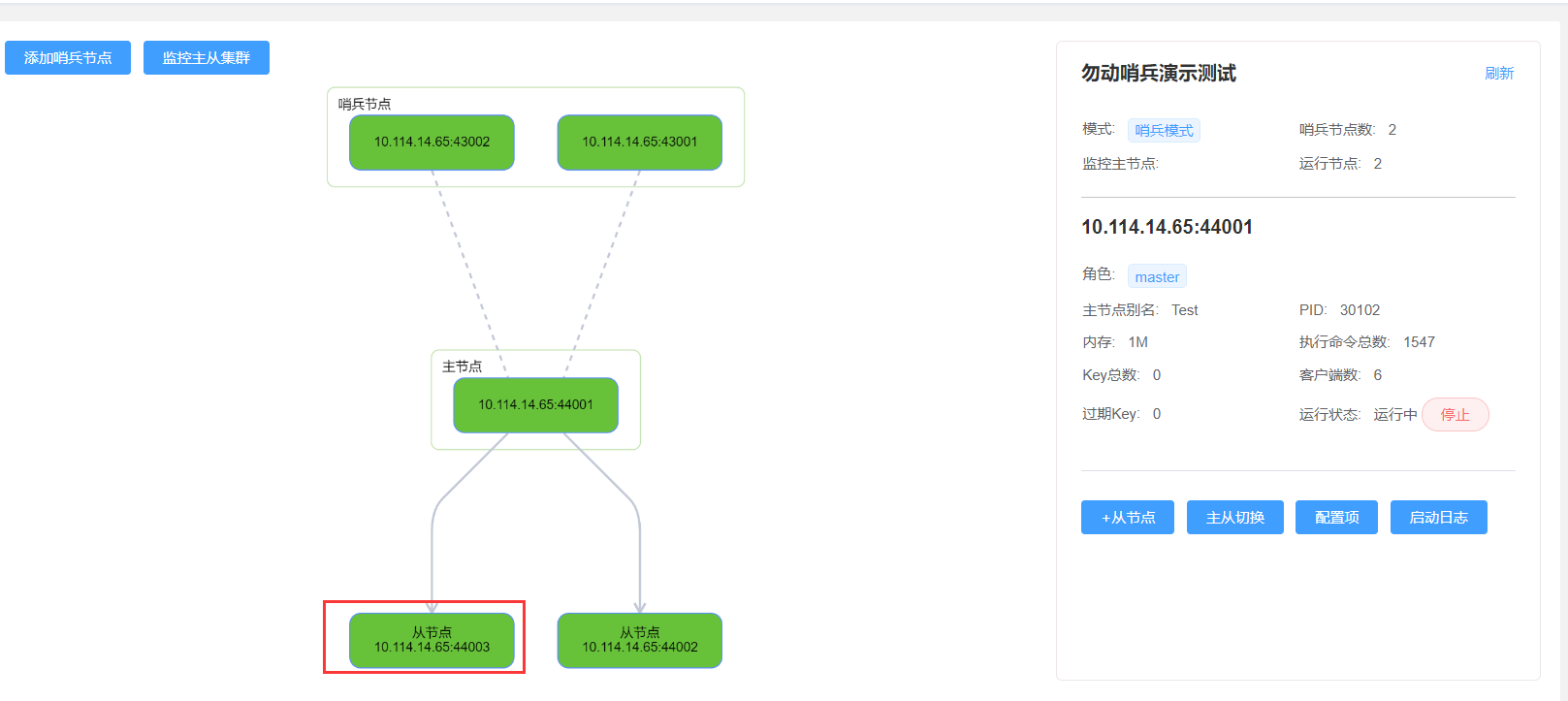
##### 2.主节点（被监控的主从集群）

被监控的主从集群的主节点图像也可以点击。

###### +从节点

点击“+从节点”按钮，弹框中的父节点是选中的主节点，根据需求填入信息后点击“确定”按钮，成功后会在该哨兵集群下的图像显示已经添加的从节点，在被监控的主从集群下也会显示刚刚添加的从节点。

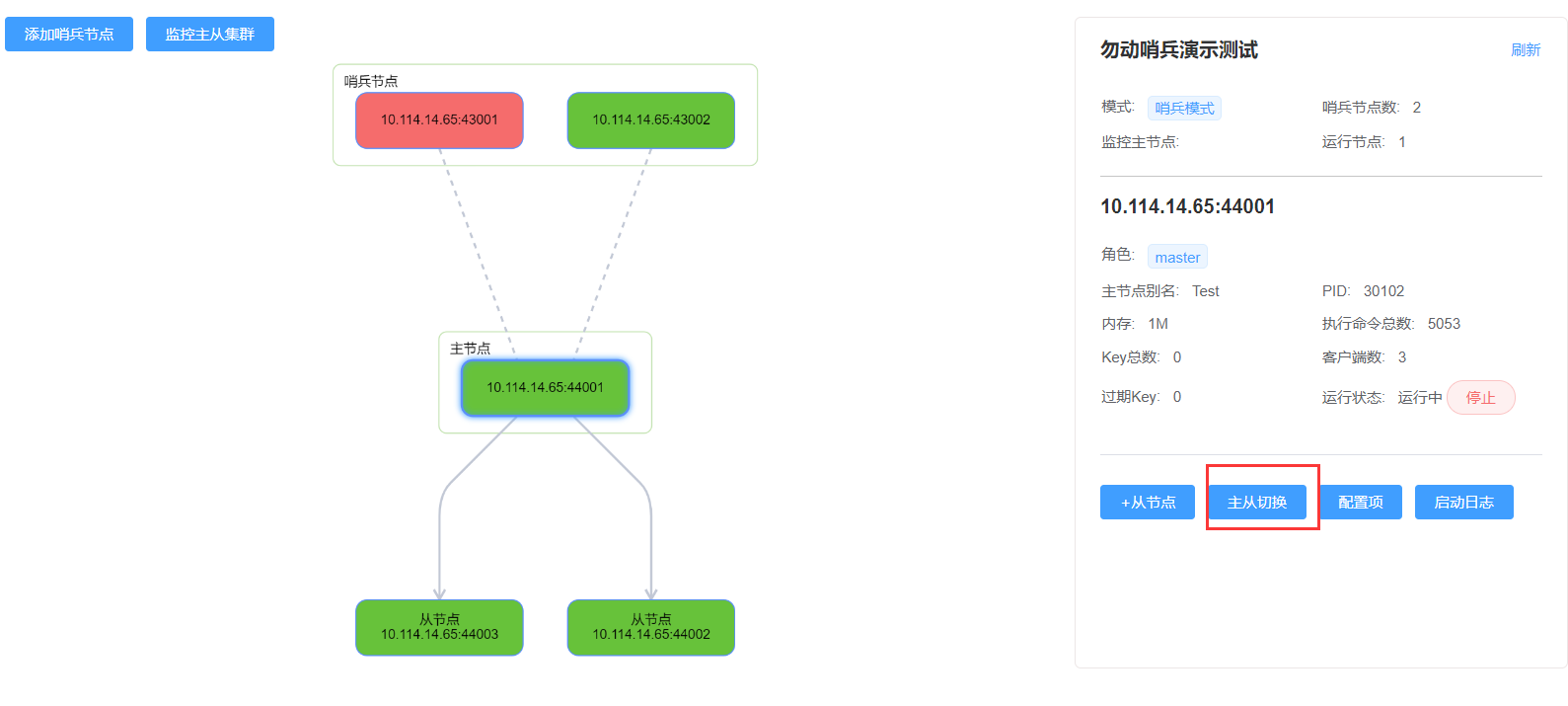




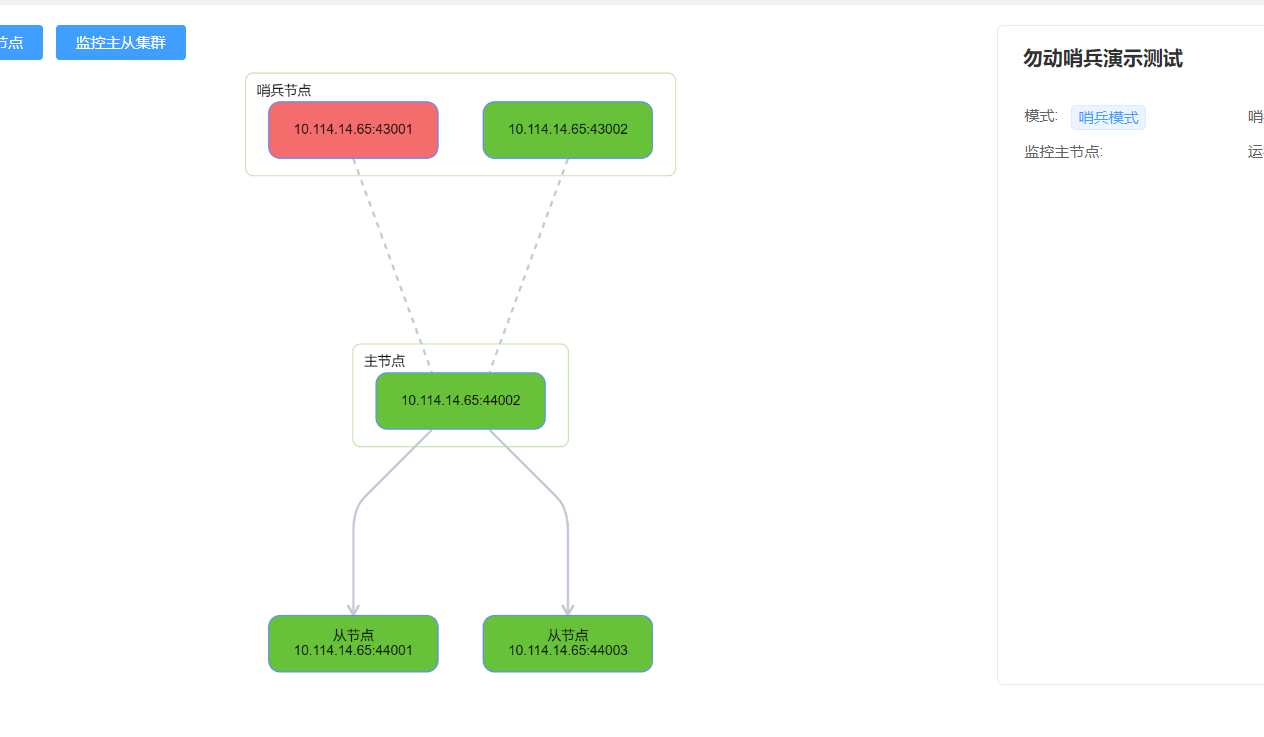


###### 主从切换

点击“主从切换”按钮后会弹出提示是否切换新主节点。点击“确定”按钮后会将改集群的主节点进行切换，随机将一个从节点切换为主节点，将主节点切换为从节点。



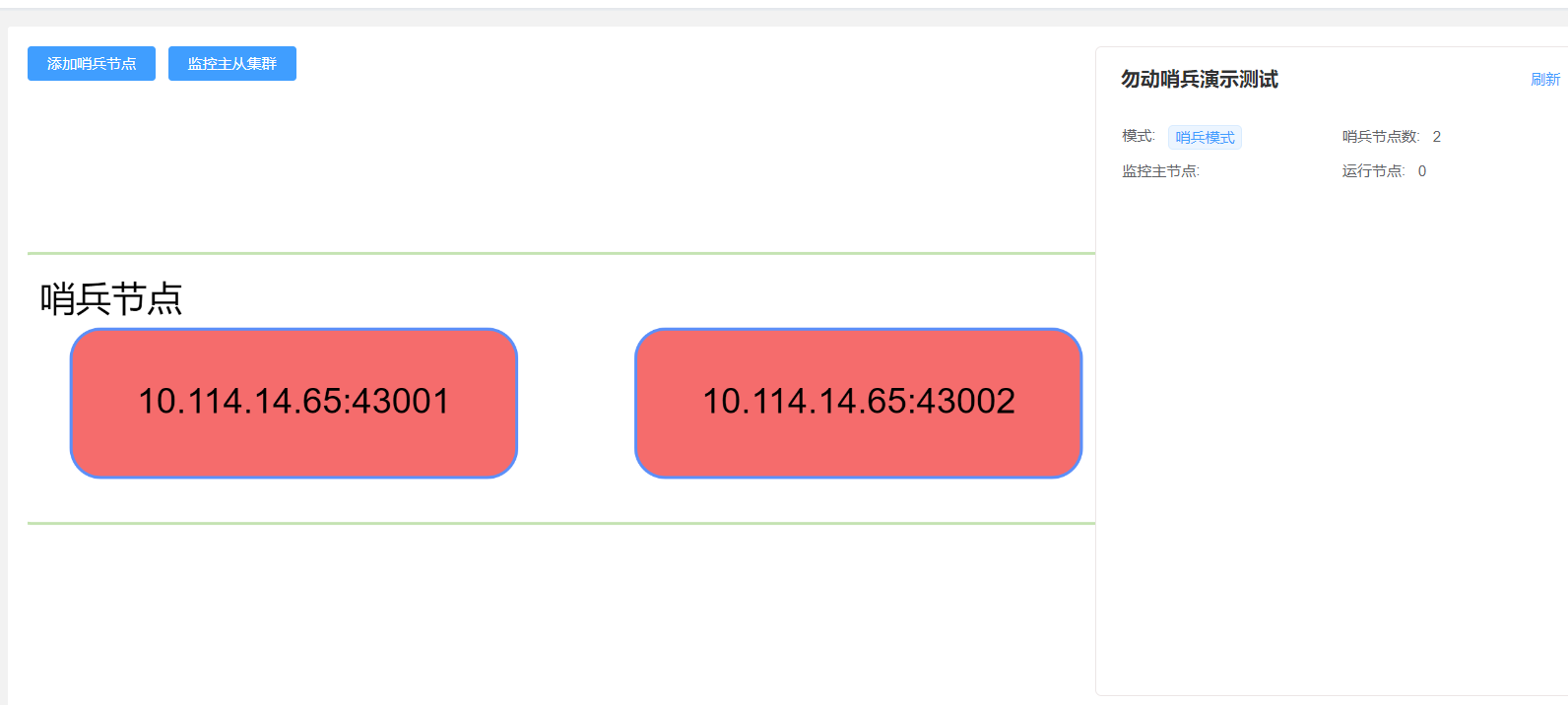




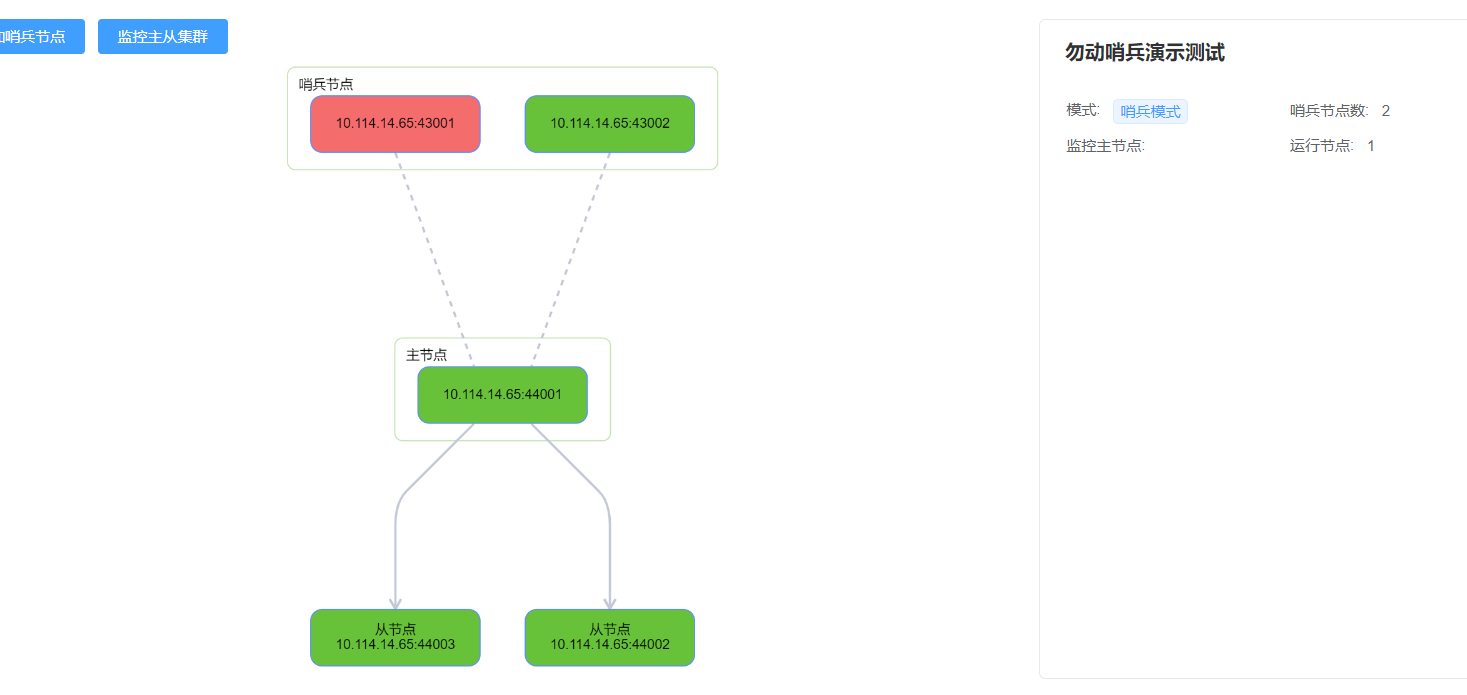
###### 主节点的启动/停止

在右侧详细信息上有运行状态一列，当节点运行状态为“运行中”时，其右侧的按钮为“停止”，点击“停止”按钮后弹出弹框会提示是否停止该节点，点击“确定”后则会停止该节点，同时运行状态为“停止”，其右侧按钮变为“启动”。点击“启动”按钮后弹出弹框会提示是否启动该节点，点击“确定”后则会启动该节点，同时运行状态变为“运行中”。

当节点状态为停止时，“配置项”按钮会隐藏，同时已经被监控的主从集群的主节点也会被隐藏。当有多个哨兵节点时，只要有一个哨兵节点是运行状态，就会显示被监控的主从集群下的节点。



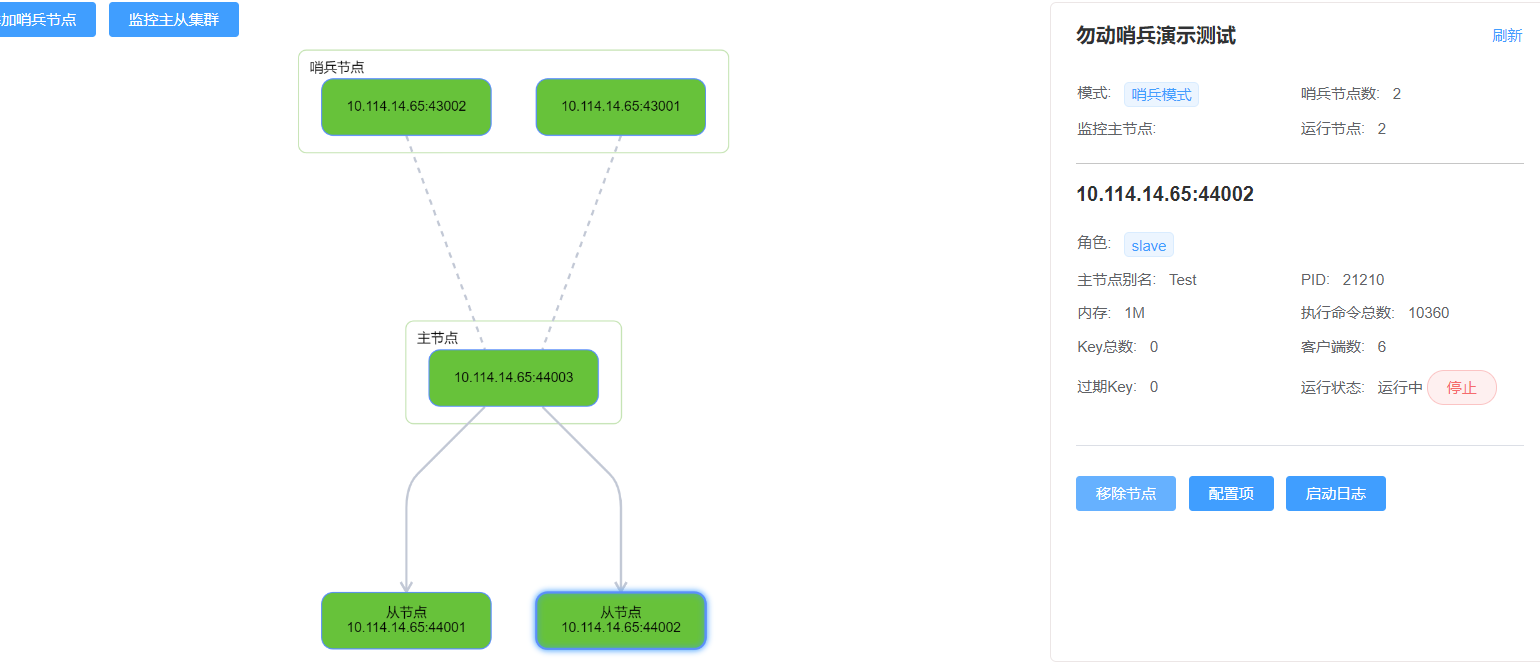






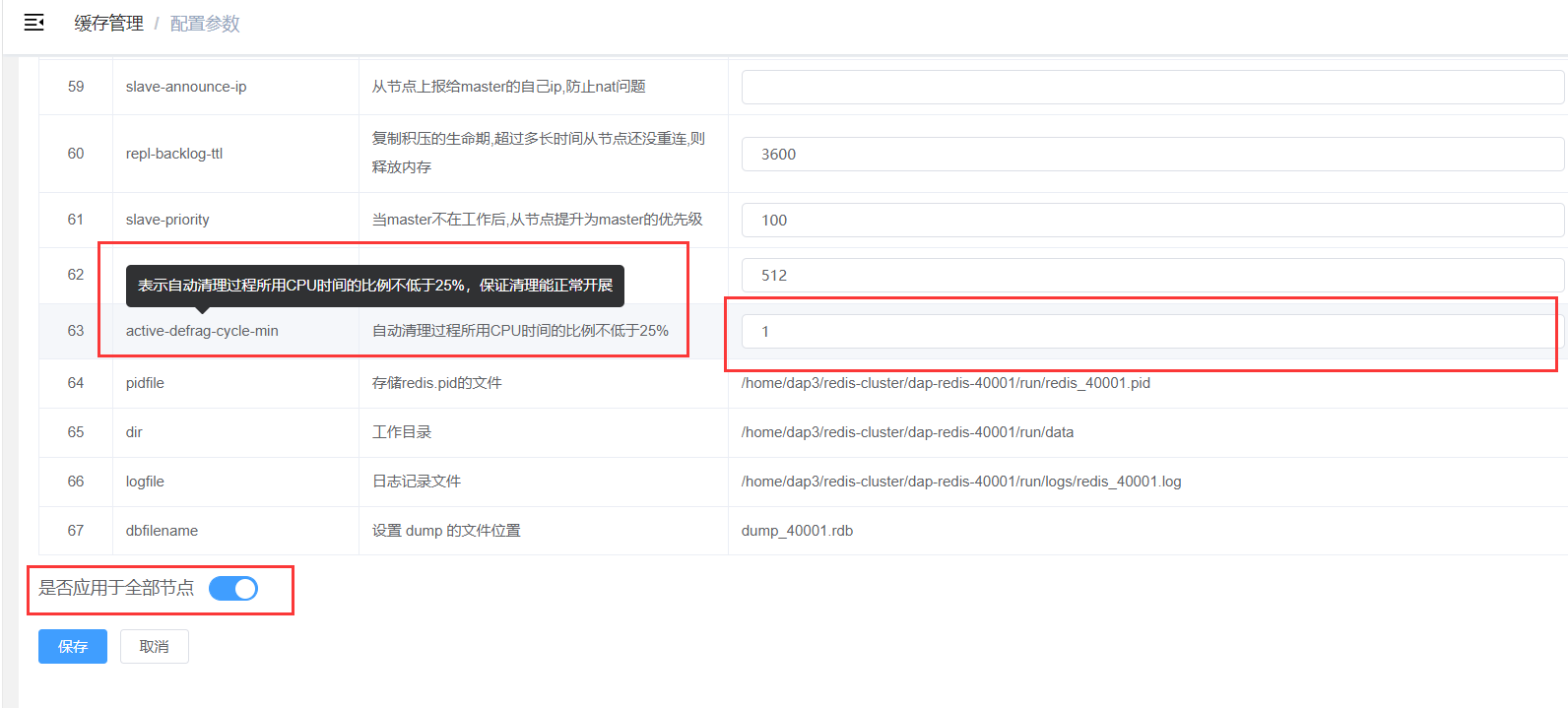
##### 3.从节点（被监控的主从集群）

###### 配置项



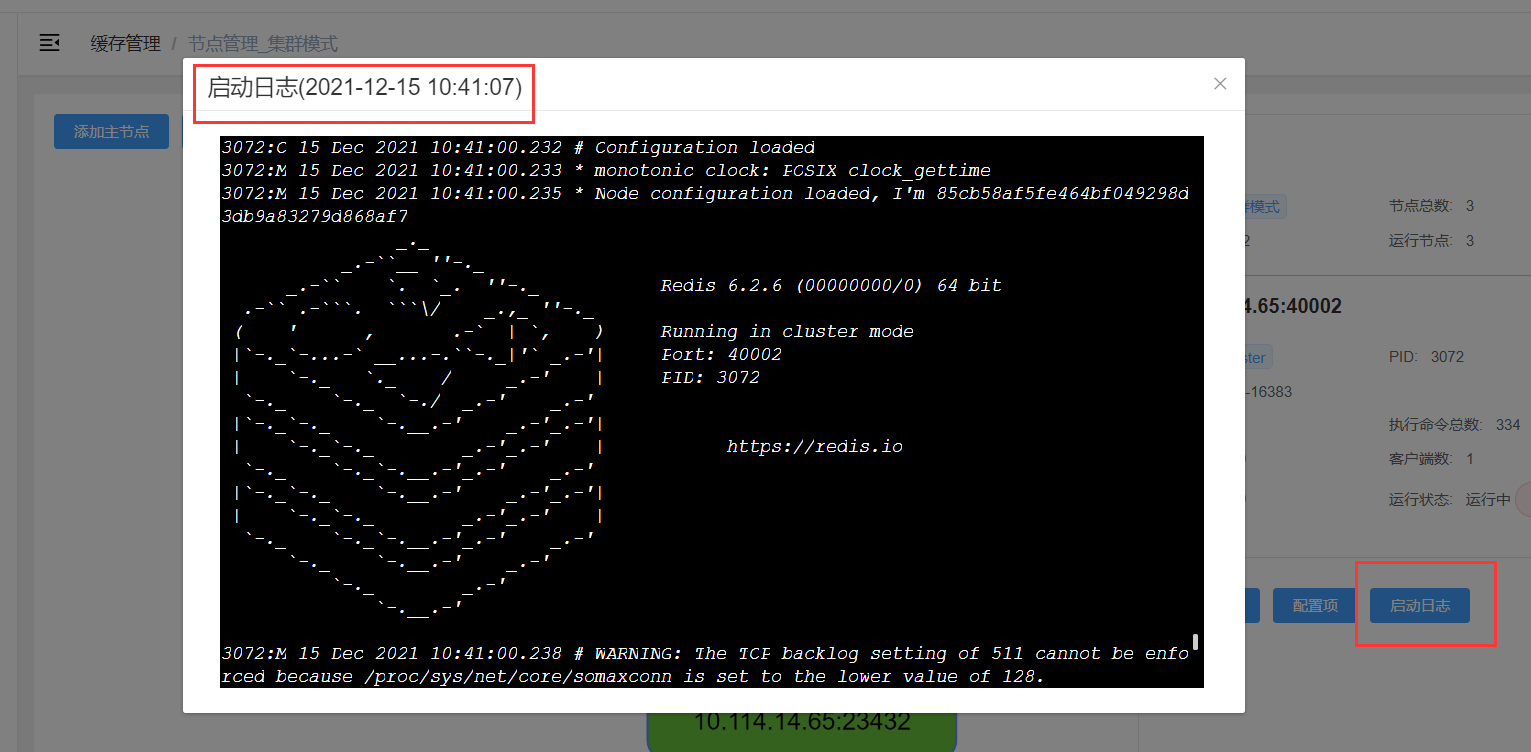
点击“配置项”按钮会进入一个新界面

在配置项界面中，有几十条关于redis节点的配置，可以将鼠标移动到某个配置项上，就可以查看该配置项代表的含义，可以进行修改的配置项后会有文本框提供修改，不可修改的配置项则没有文本框。下方有“是否应用于全部节点”的开关，打开开关后再点击“保存”就可以将此处修改的配置项保存到该集群下的所有节点上。



###### 启动日志

点击“启动日志”按钮后，会弹出这个节点的启动日志，可以用来查看节点是否正常启动停止。



###### 移除节点

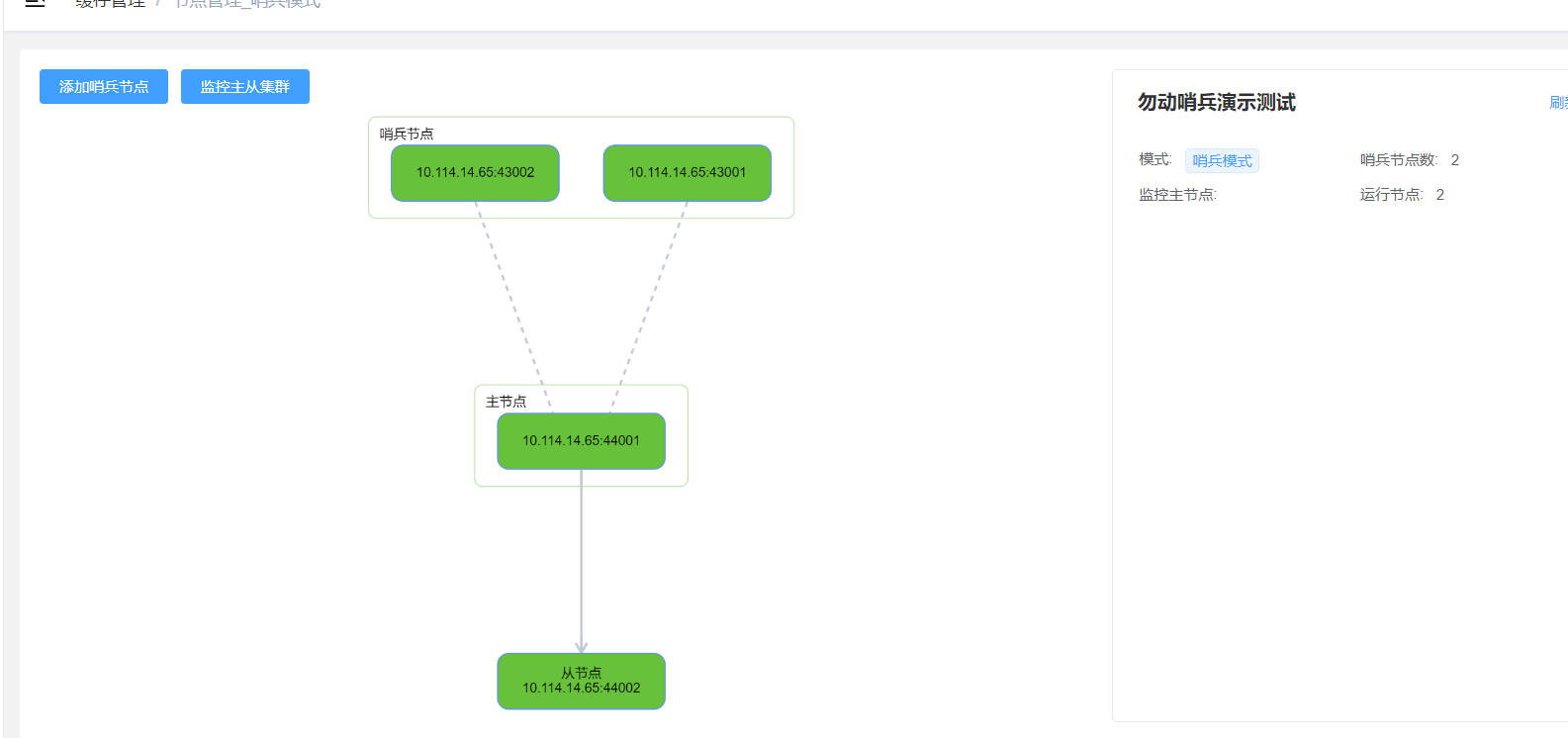
点击“移除节点”按钮后会有弹窗提示是否删除该节点，点击“确定”后会再次提示是否删除本地文件，可以根据需求进行删除。移除被监控的主从集群的从节点时该哨兵集群的哨兵节点的运行状态必须全部都为“运行中”状态。



##### 4.添加哨兵节点

点击“添加哨兵节点”按钮后，会弹出弹框，选择好节点与端口后点击“确定”按钮，会添加一个哨兵节点。



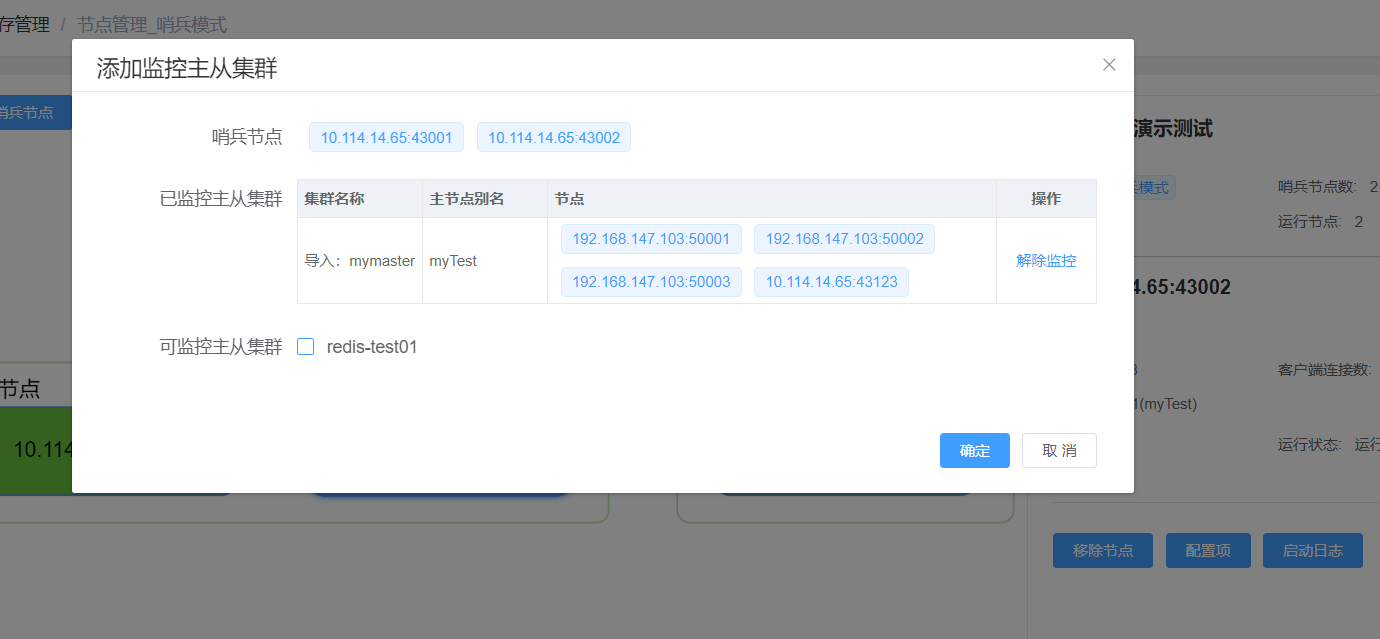


##### 5.监控主从集群

“监控主从集群”按钮可以添加被哨兵节点监控的主从集群，最下方是可监控的主从集群，被监控的主从集群必须是没有被其他哨兵集群监控的且自身版本需要与该哨兵节点一致，选择一个集群后可以输入主节点别名，主节点别名只能输入数字、字母的组合。

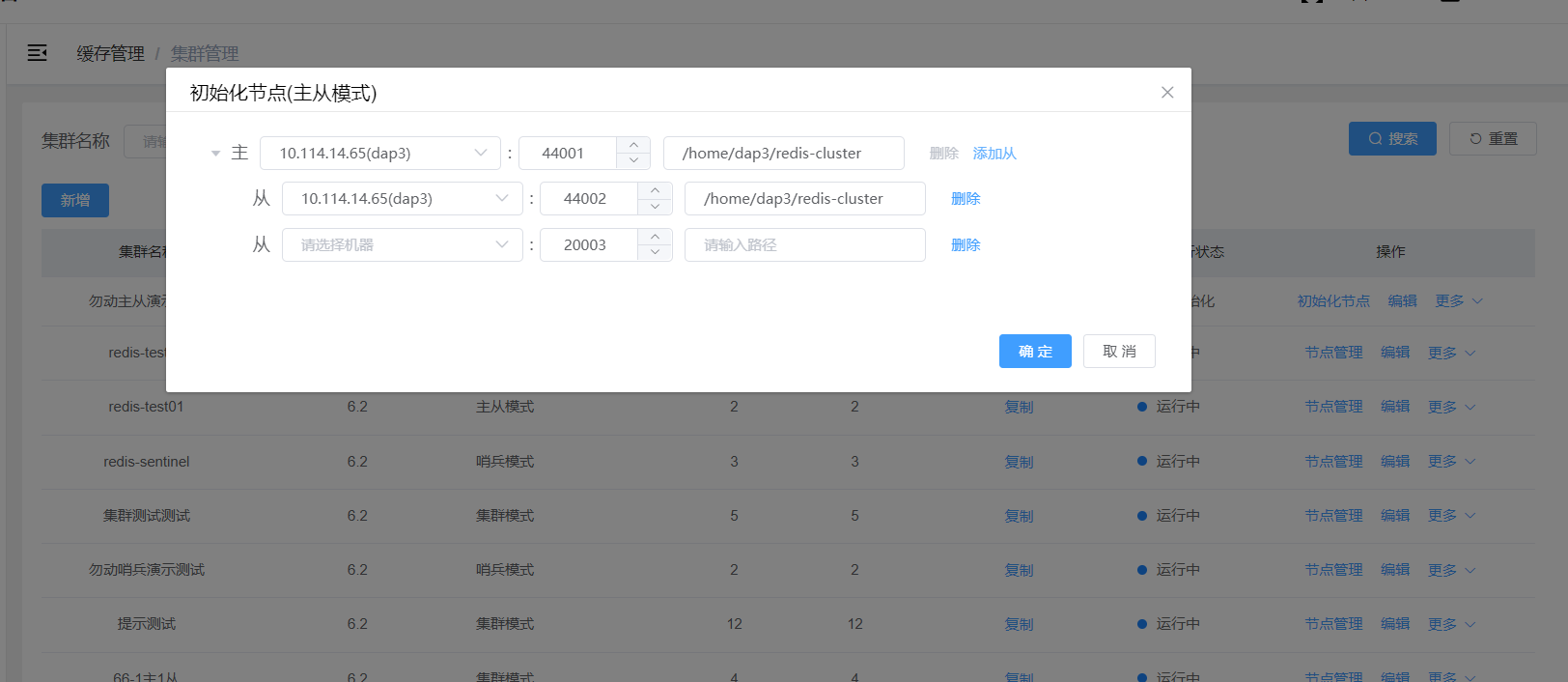


已经添加监控完成的主从集群会在已监控主从集群一栏显示出来，也会在图像上显示出来，且在已监控主从集群这一栏最后方有“解除监控”操作按钮，点击后可以解除该哨兵节点对这一主从集群的监控，之后就可以再次被改哨兵集群或者其他同版本的哨兵集群监控。



#### 初始化（主从模式）

主从模式的集群下只能有一个主节点，但可以有很多个从节点，主节点旁“的删除”按钮始终为灰色，不能点击。可以通过“添加从”来添加从节点，而且从节点旁只有“删除”按钮，没有“添加从”按钮，主从模式下从节点下不能再添加从节点。选择好ip与端口后点击“确定”就可以进行主从模式集群的初始化。



#### 节点管理（主从模式）

##### 1.主节点

###### 主节点的启动/停止

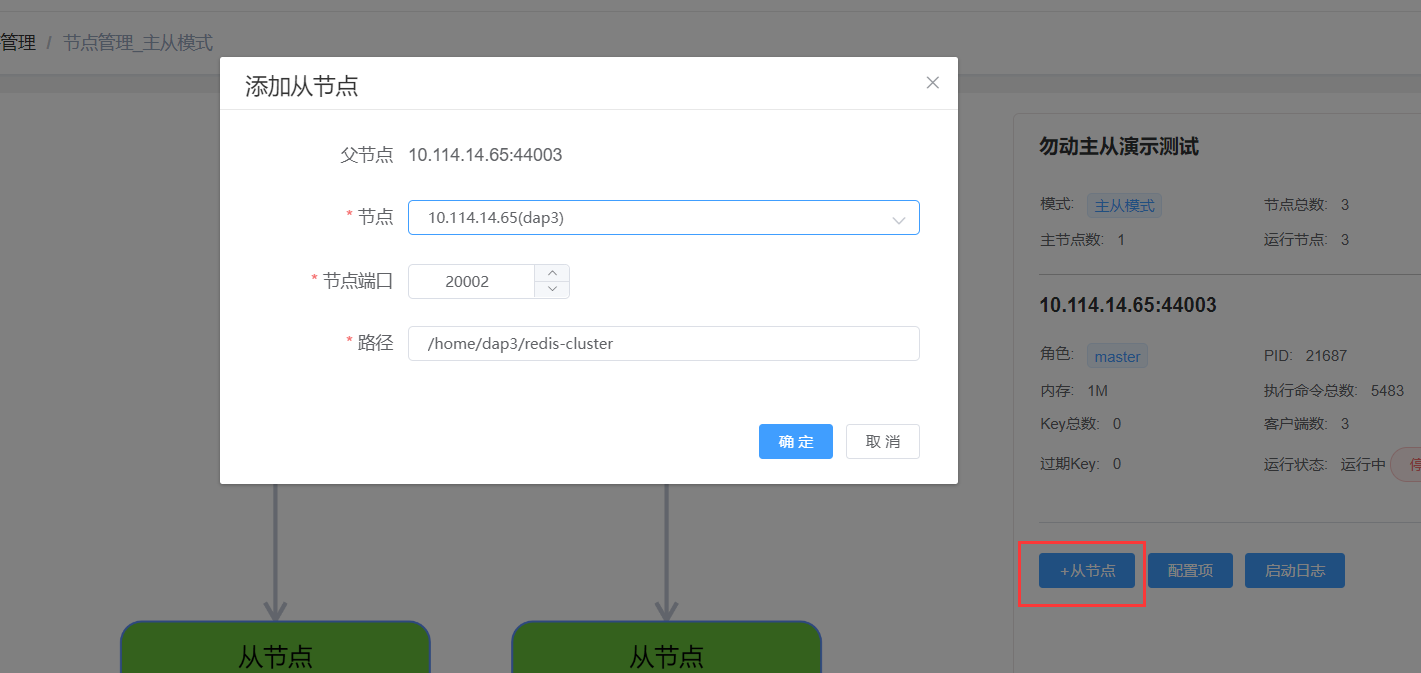
在右侧详细信息上有运行状态一列，当节点运行状态为“运行中”时，其右侧的按钮为“停止”，点击“停止”按钮后弹出弹框会提示是否停止该节点，点击“确定”后则会停止该节点，同时运行状态为“停止”，其右侧按钮变为“启动”。点击“启动”按钮后弹出弹框会提示是否启动该节点，点击“确定”后则会启动该节点，同时运行状态变为“运行中”。

当节点状态为停止时，“配置项”按钮会隐藏。



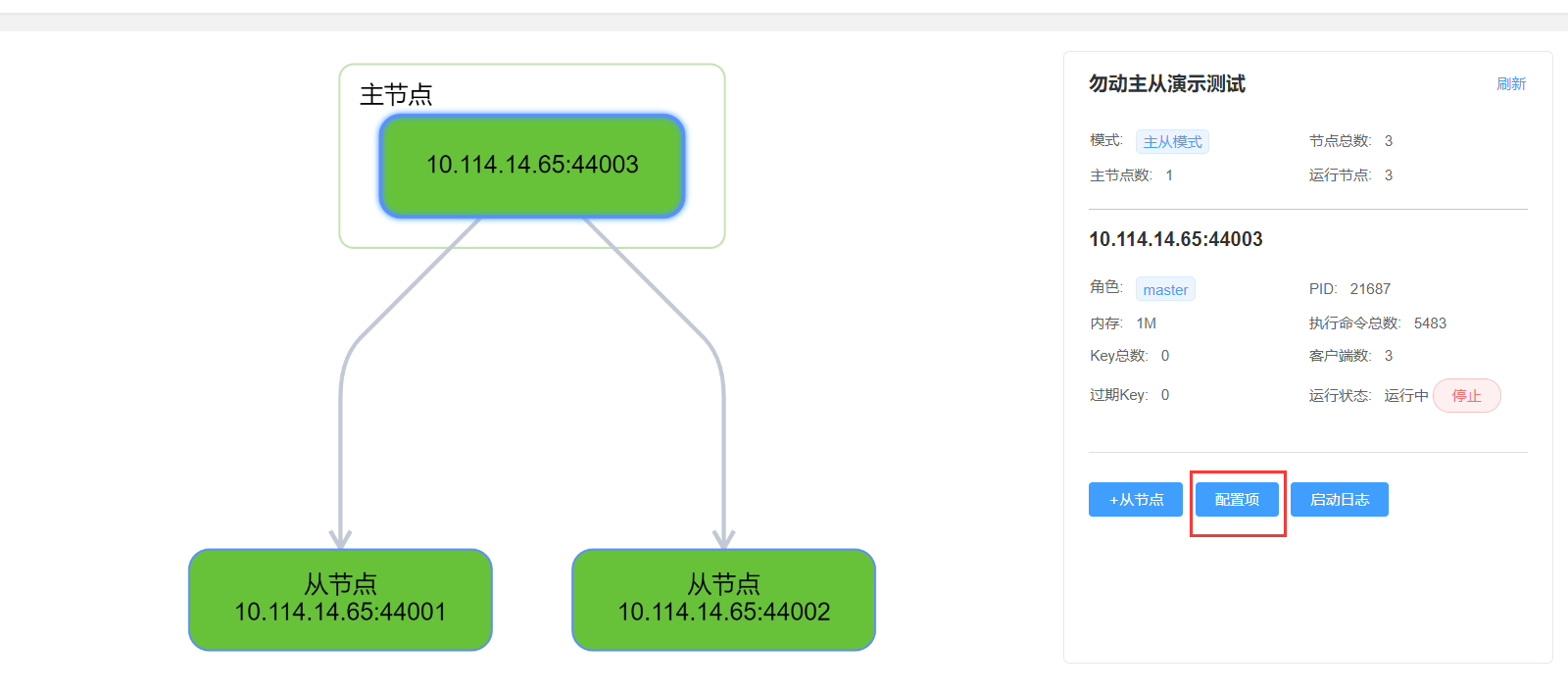
###### +从节点

点击“+从节点”按钮，弹框中的父节点是选中的主节点，根据需求填入信息后点击“确定”按钮，成功后会在图像上显示出已经添加的从节点。

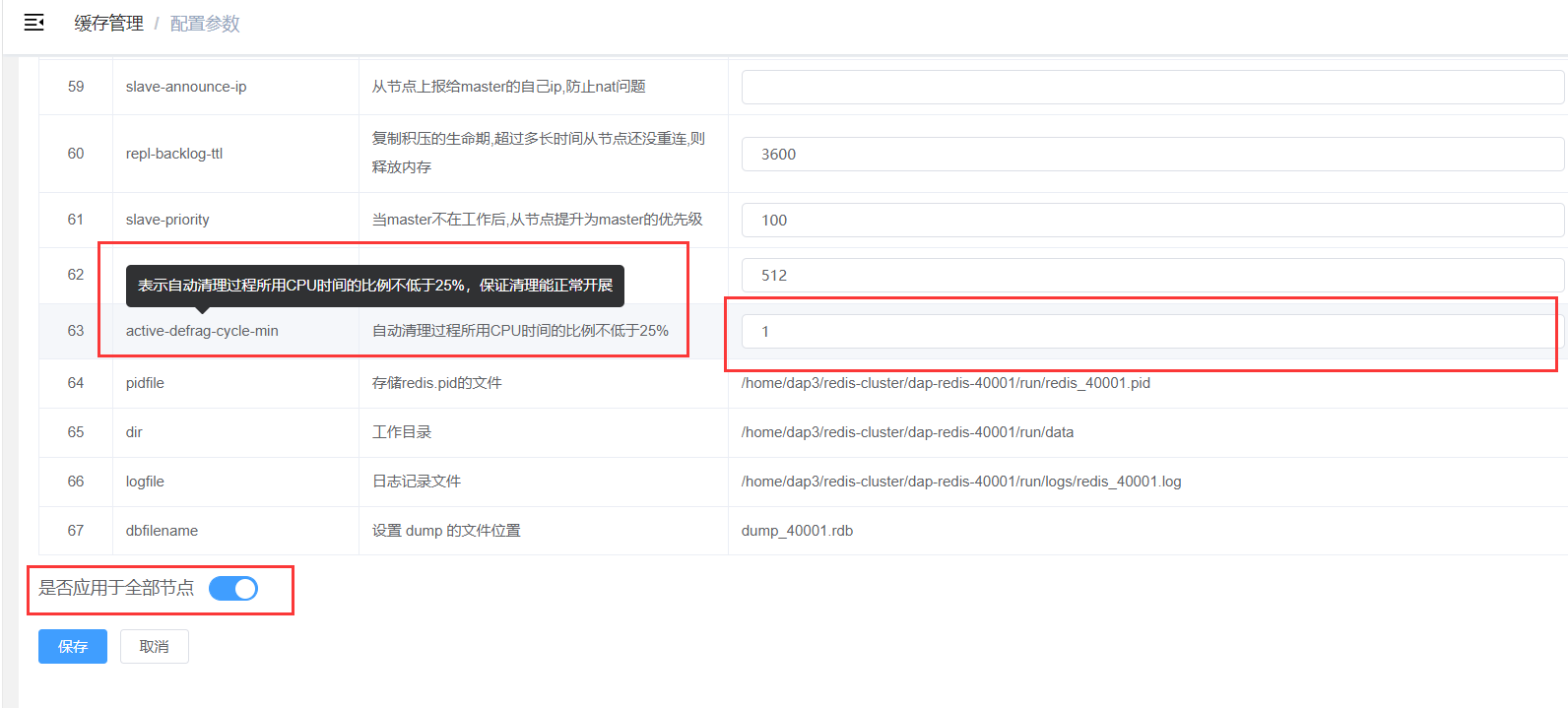


###### 配置项

在主节点图像上点击“配置项”按钮后会进入一个新界面，

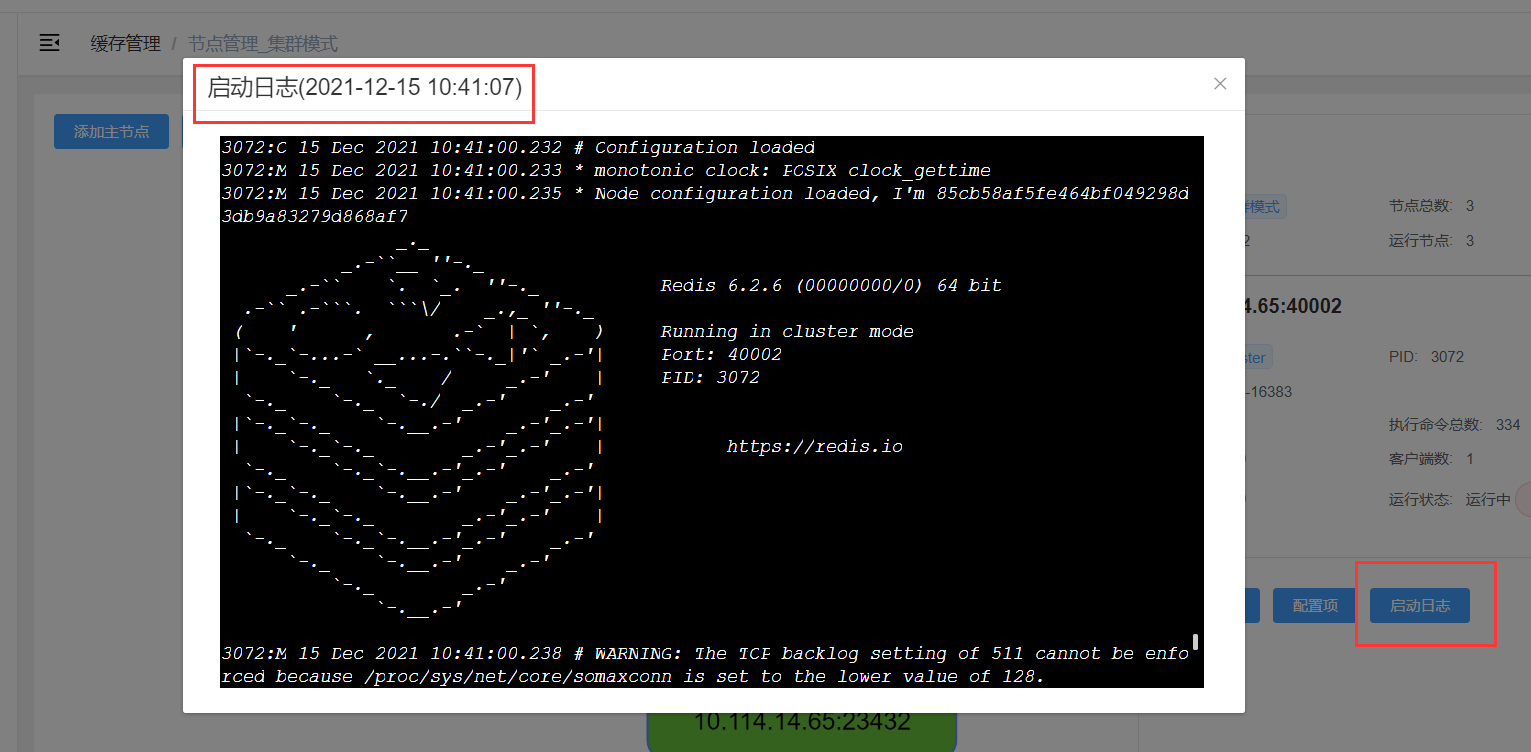


在配置项界面中，有几十条关于redis节点的配置，可以将鼠标移动到某个配置项上，就可以查看该配置项代表的含义，可以进行修改的配置项后会有文本框提供修改，不可修改的配置项则没有文本框。下方有“是否应用于全部节点”的开关，打开开关后再点击“保存”就可以将此处修改的配置项保存到该集群下的所有节点上。



###### 启动日志

点击“启动日志”按钮后，会弹出这个节点的启动日志，可以用来查看节点是否正常启动停止。



##### 2.从节点

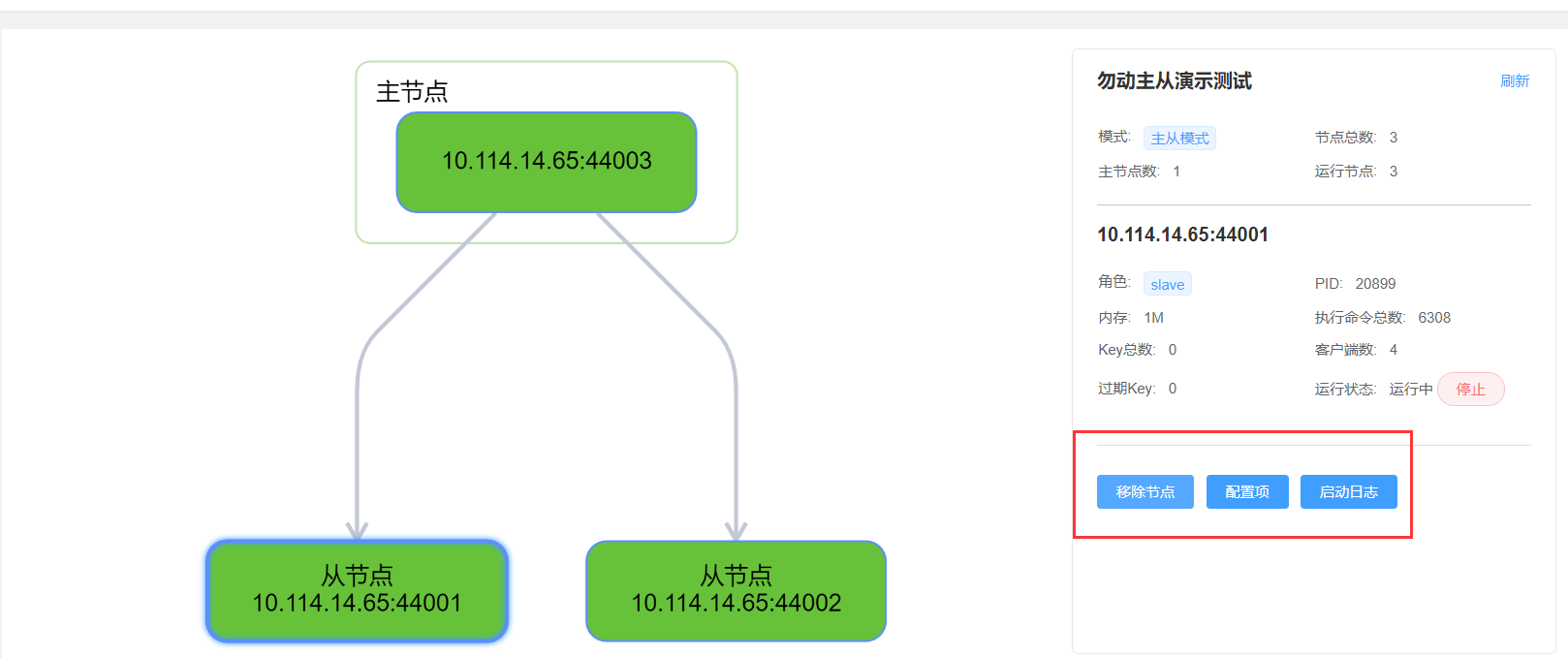
从节点右侧只有三个按钮“移除节点”、“配置项”、“启动日志”。

###### 从节点的启动/停止

在右侧详细信息上有运行状态一列，当节点运行状态为“运行中”时，其右侧的按钮为“停止”，点击“停止”按钮后弹出弹框会提示是否停止该节点，点击“确定”后则会停止该节点，同时运行状态为“停止”，其右侧按钮变为“启动”。点击“启动”按钮后弹出弹框会提示是否启动该节点，点击“确定”后则会启动该节点，同时运行状态变为“运行中”。 当从节点的运行状态为“停止”时，他将会默认升为与主节点同一级别的节点，不过当他的运行状态变为“运行中”时就会恢复为主节点下的从节点。

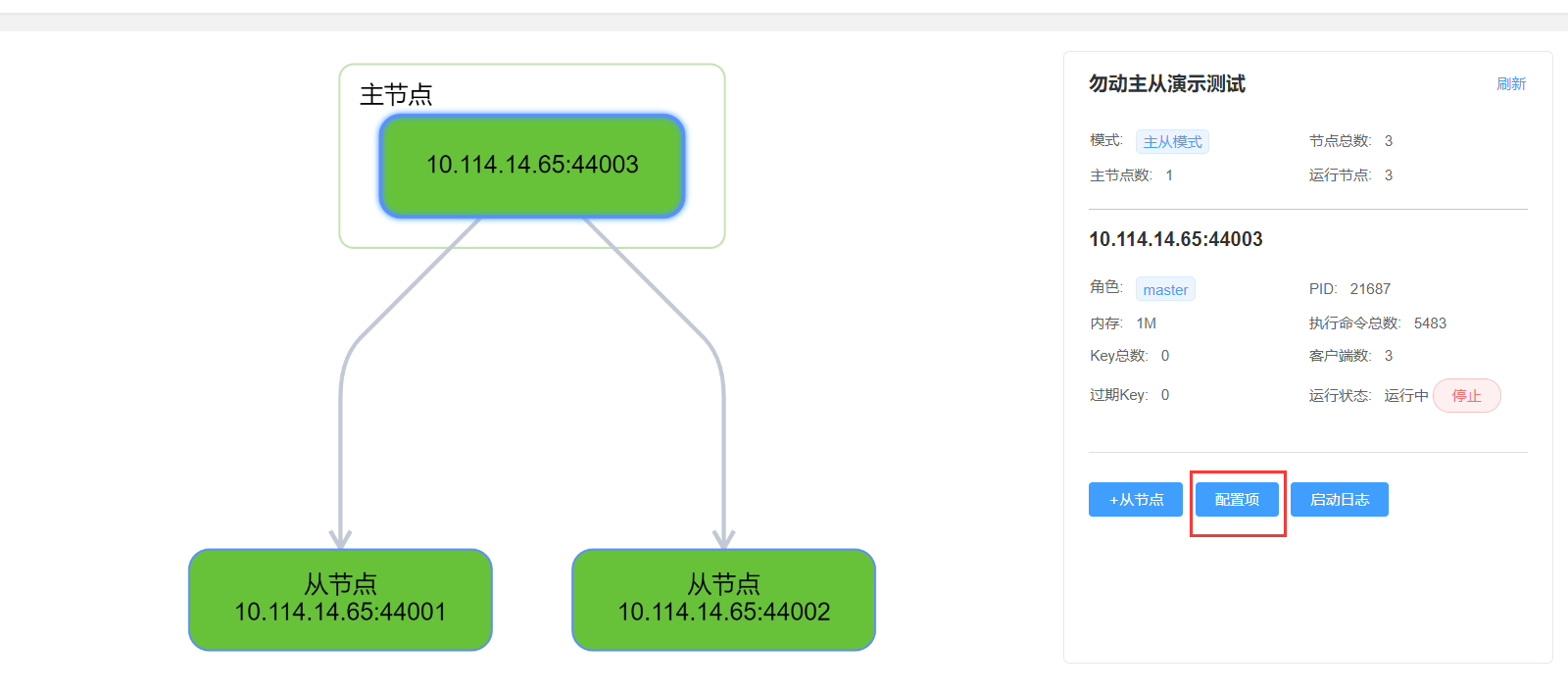
当节点状态为停止时，“配置项”按钮会隐藏。



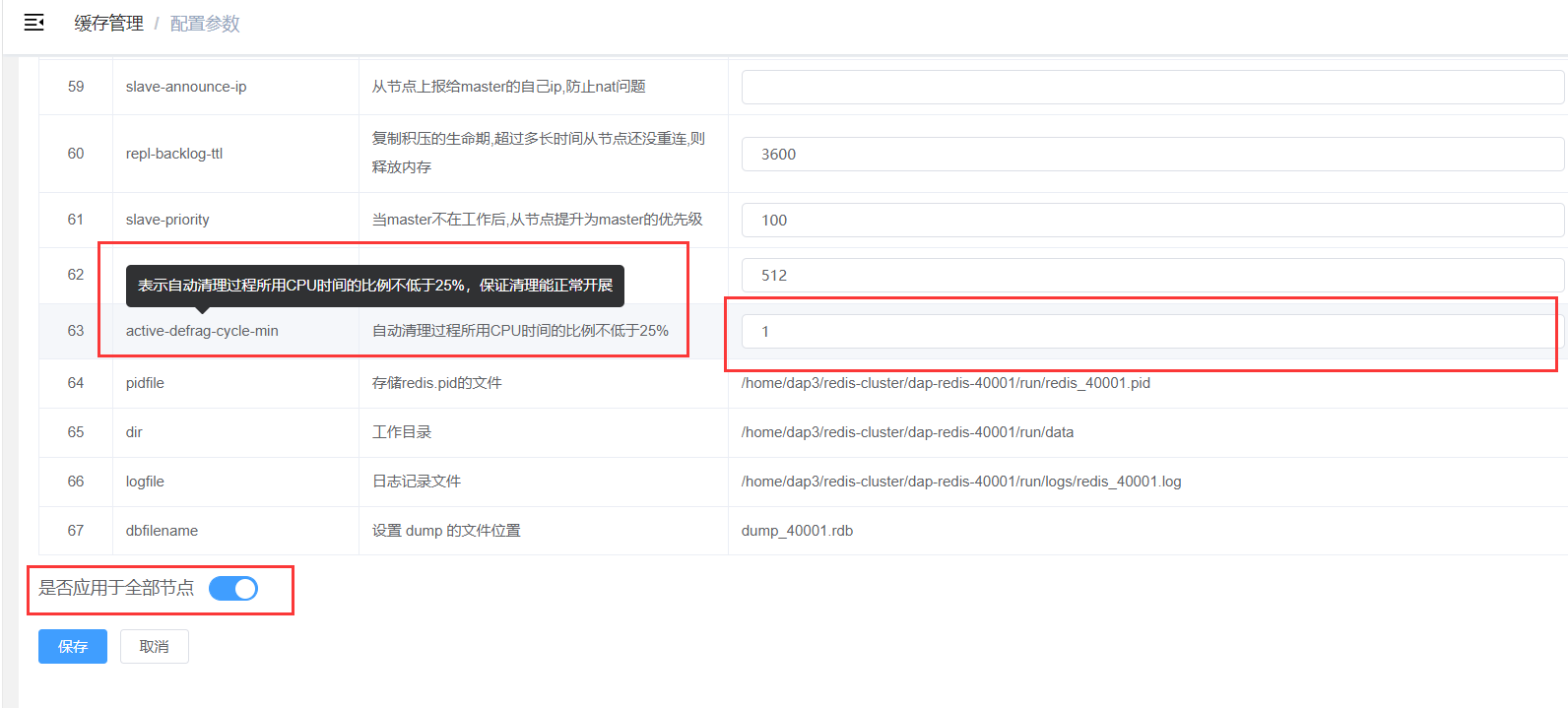


###### 配置项

点击“配置项”按钮后会进入一个新界面，

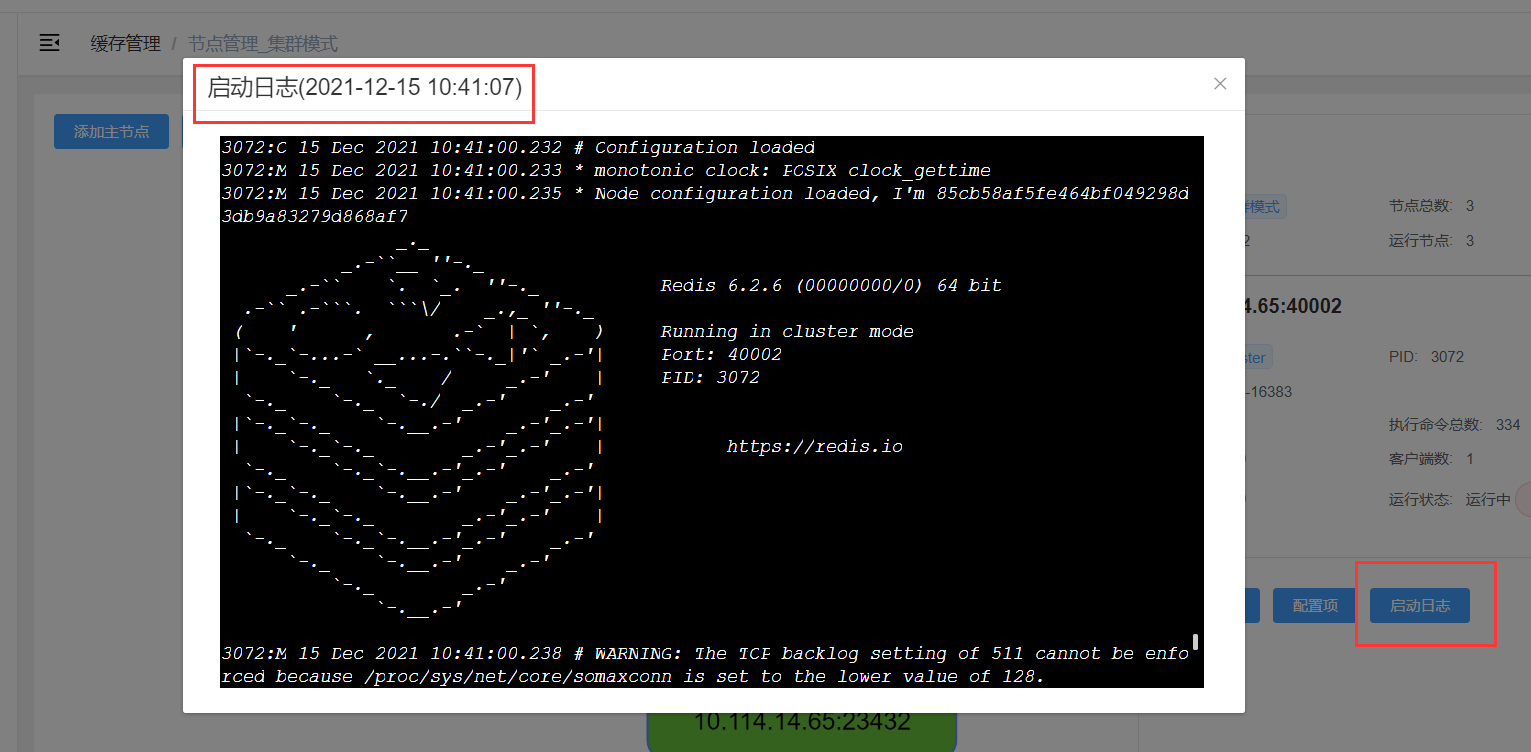


在配置项界面中，有几十条关于redis节点的配置，可以将鼠标移动到某个配置项上，就可以查看该配置项代表的含义，可以进行修改的配置项后会有文本框提供修改，不可修改的配置项则没有文本框。下方有“是否应用于全部节点”的开关，打开开关后再点击“保存”就可以将此处修改的配置项保存到该集群下的所有节点上。



###### 启动日志

点击“启动日志”按钮后，会弹出这个节点的启动日志，可以用来查看节点是否正常启动停止。



###### 移除节点

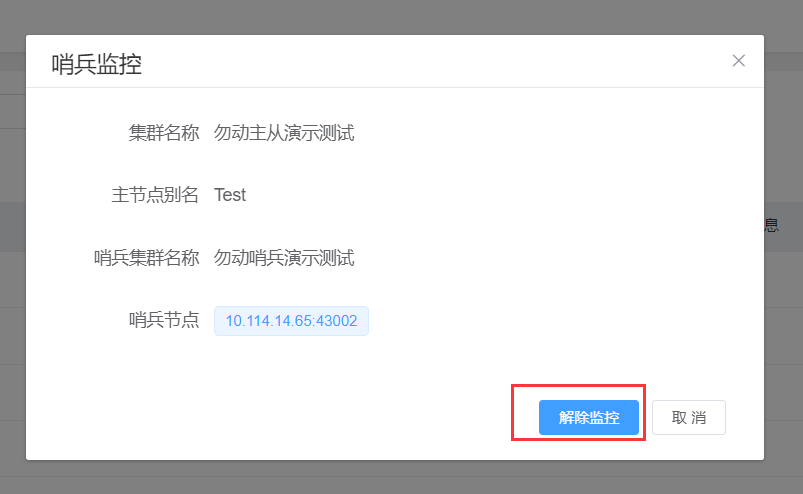
点击“移除节点”按钮后会有弹窗提示是否删除该节点，点击“确定”后会再次提示是否删除本地文件，可以根据需求进行删除。移除被监控的主从集群的从节点时该哨兵集群的哨兵节点的运行状态必须全部都为“运行中”状态。



###### 主从集群被监控

当该主从模式集群被另一哨兵模式集群监控时，该主从模式集群名称后会出现一个“小眼睛”样式的按钮，点击该按钮会展示出监控该集群的哨兵模式集群的信息，同时下方会有“解除监控”按钮，点击该按钮会解除哨兵模式集群对于该主从模式集群的监控，试该主从模式集群可以被其他哨兵模式集群监控。



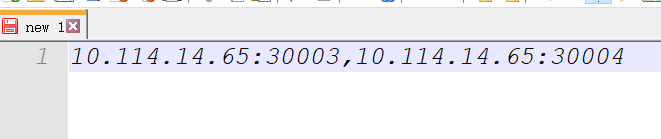


三种模式的集群初始化已经展示完毕，并且初始化完成只后的节点管理也展示完毕。

#### 复制

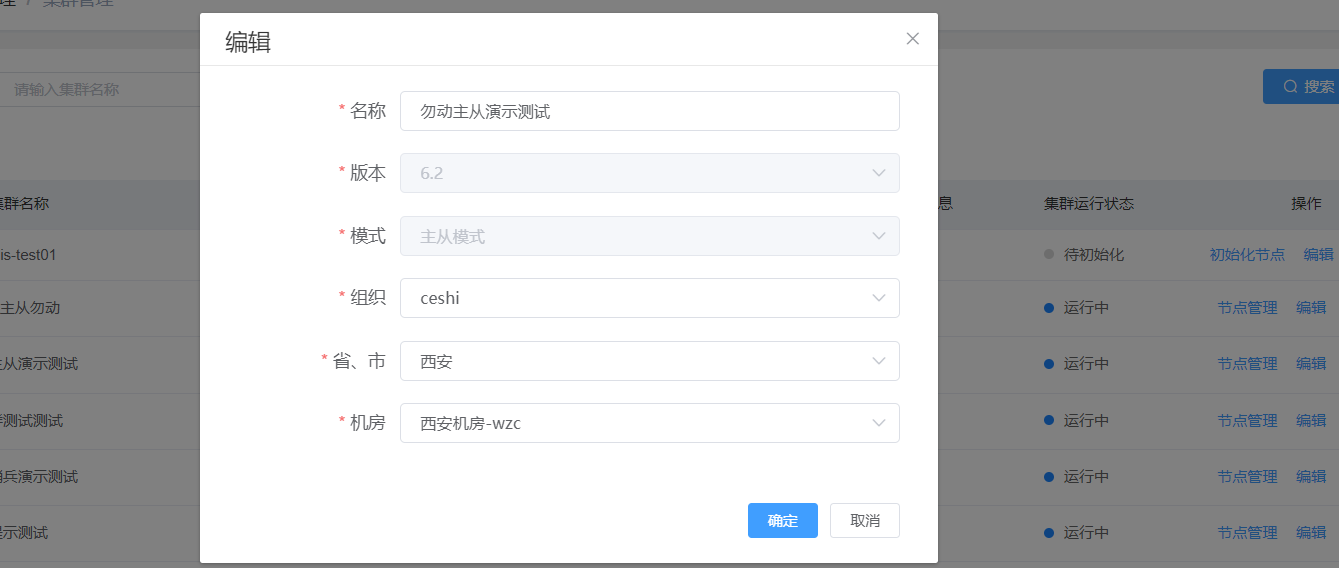
鼠标移动到“复制”按钮上会显示出该集群下的节点ip与端口，点击“复制”按钮后会复制节点ip与端口，可以在其他地方进行粘贴





#### 编辑

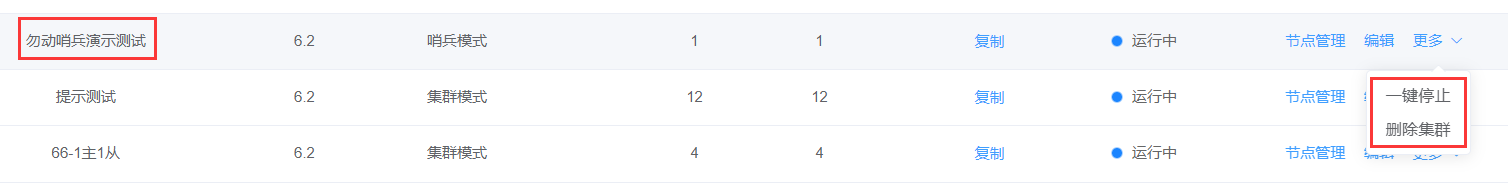
点击“编辑”按钮会弹出编辑弹窗，其中版本与模式不可修改，其余可以修改，根据需求修改后点击“确定”会保存修改后的数据，点击“取消”会取消修改。



#### 更多

鼠标移动到“更多”按钮上后会下拉展示更多操作，其中主从模式和集群模式集群的更多操作一致，哨兵模式的更多操作只有“一键停止”与“删除集群”。



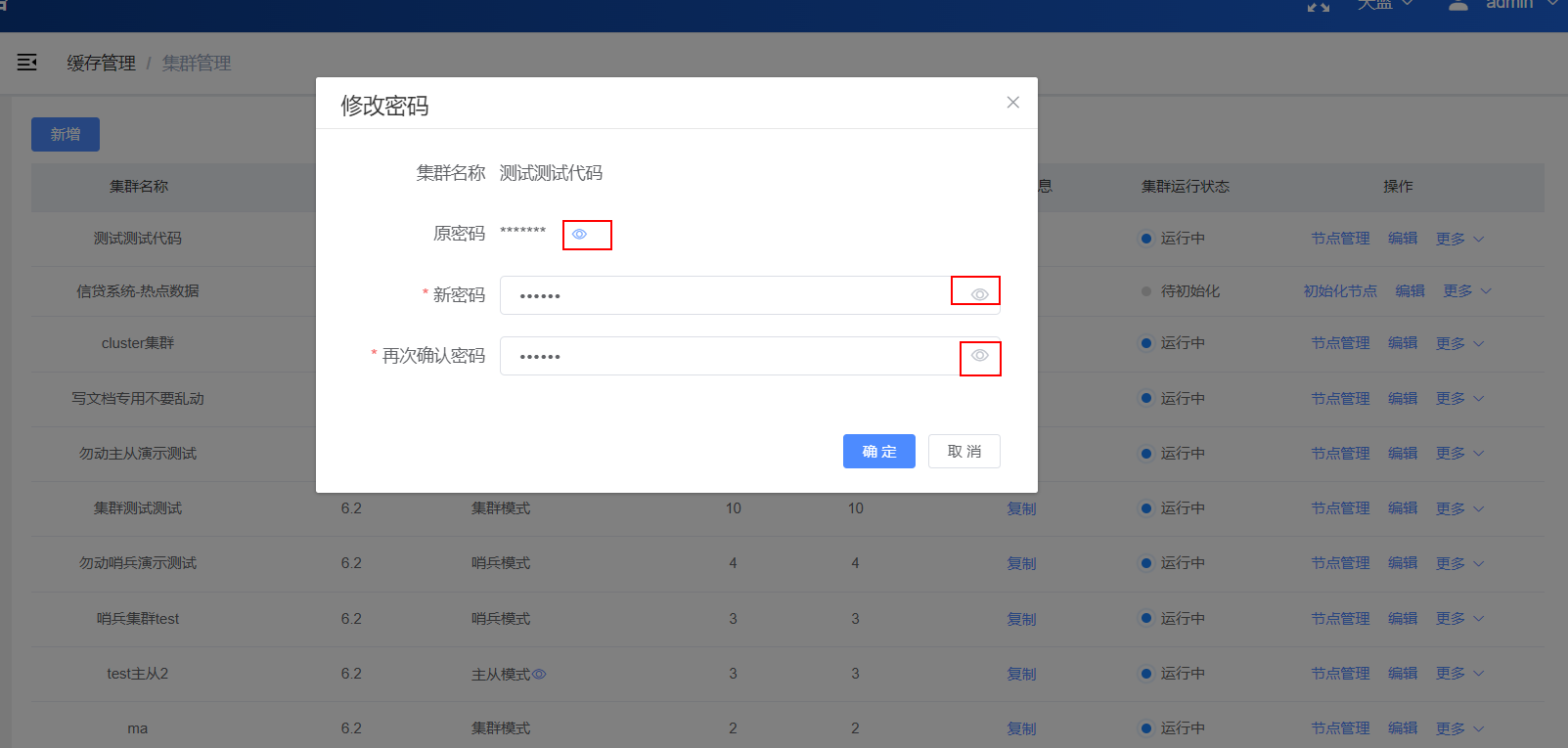


#### 设置密码

“设置密码”按钮是对redis集群进行密码设置，只能对主从模式和集群模式集群进行密码设置。

输入新密码和再次确认密码后点击“确定”按钮就会将密码设置进redis集群中，如果有原密码的话就会在原密码一栏以密文的形式展示出来。

点击密文密码后面的“小眼睛”图标就会将密码明文显示。



#### 清空缓存

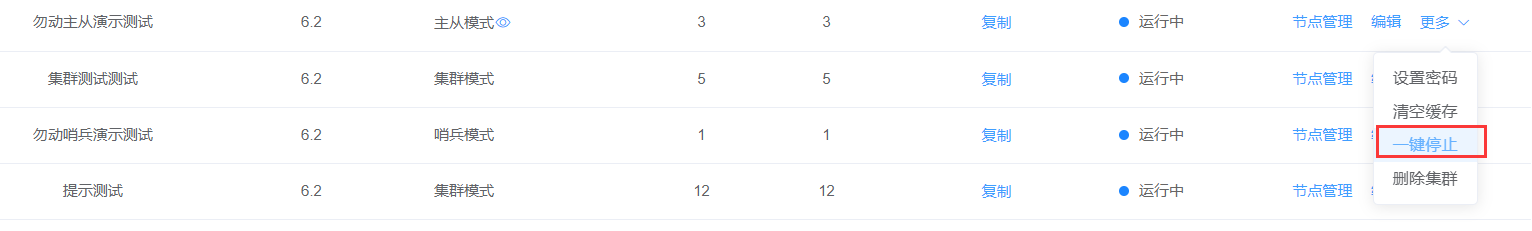
“清空缓存”按钮会将redis集群下所有节点的数据全部清空，此操作无法恢复，请谨慎操作，点击“确定清空”按钮后会再次弹窗进行提示是否清空，点击“确定”即可清空该集群下所有节点的数据。





#### 集群的一键启动/停止

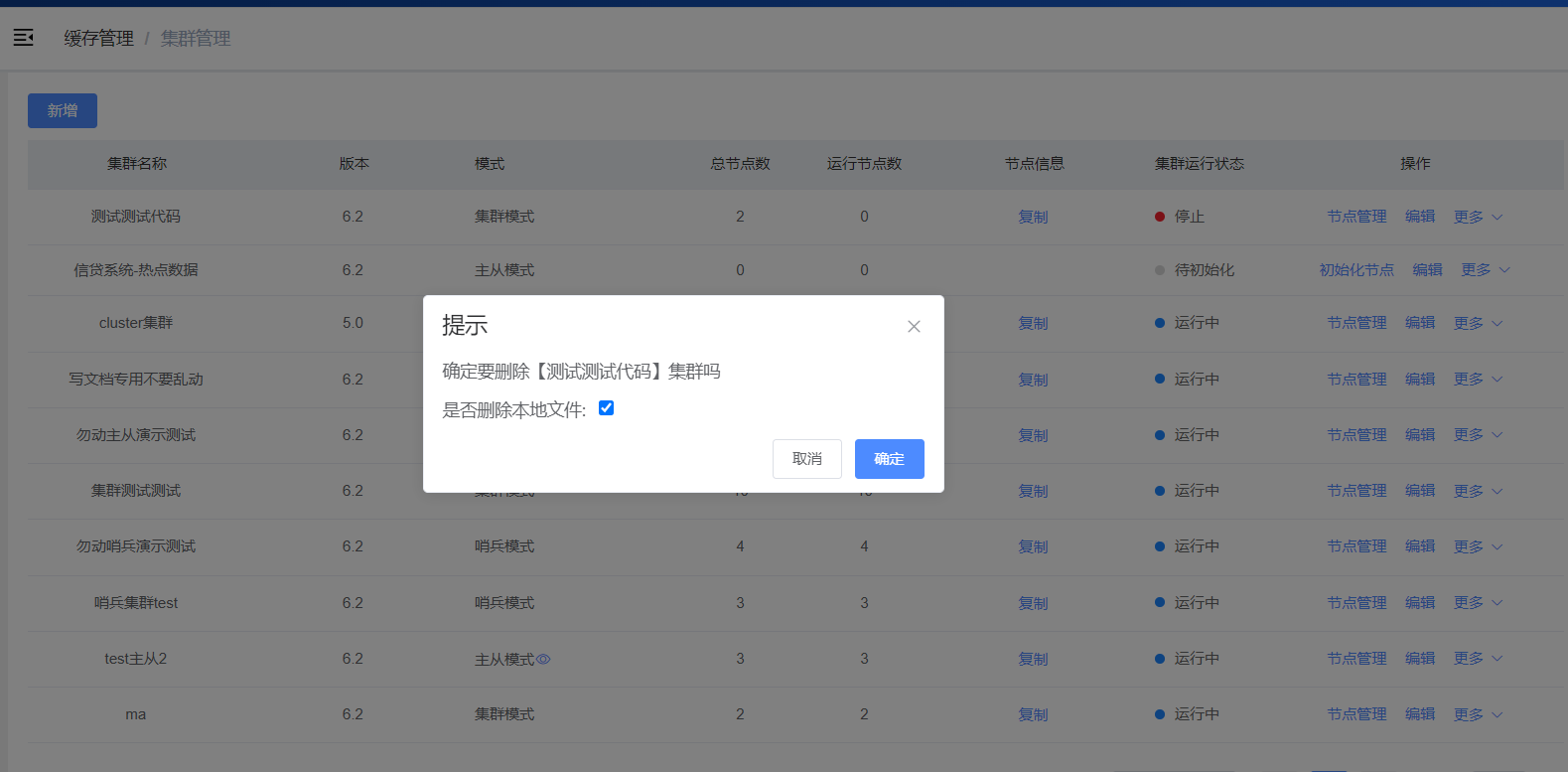
当集群运行状态为“运行中”时该操作为“一键停止”，当集群运行状态为“停止”时，该操作为“一键启动”。





#### 删除集群

点击“删除集群”后会弹出弹窗再次提醒是否删除该集群，点击“确定”按钮就会删除该集群，但是前提是该集群没有在多中心架构设置中作为主中心或备中心所使用，否则就不能进行删除。





### 持久化

redis持久化是对主从模式集群和集群模式集群生效的，所以持久化设置操作对这两种模式的集群没有区别，下面展示如何进行持久化设置操作。

本界面只会展示主从模式和集群模式的集群。

#### 搜索

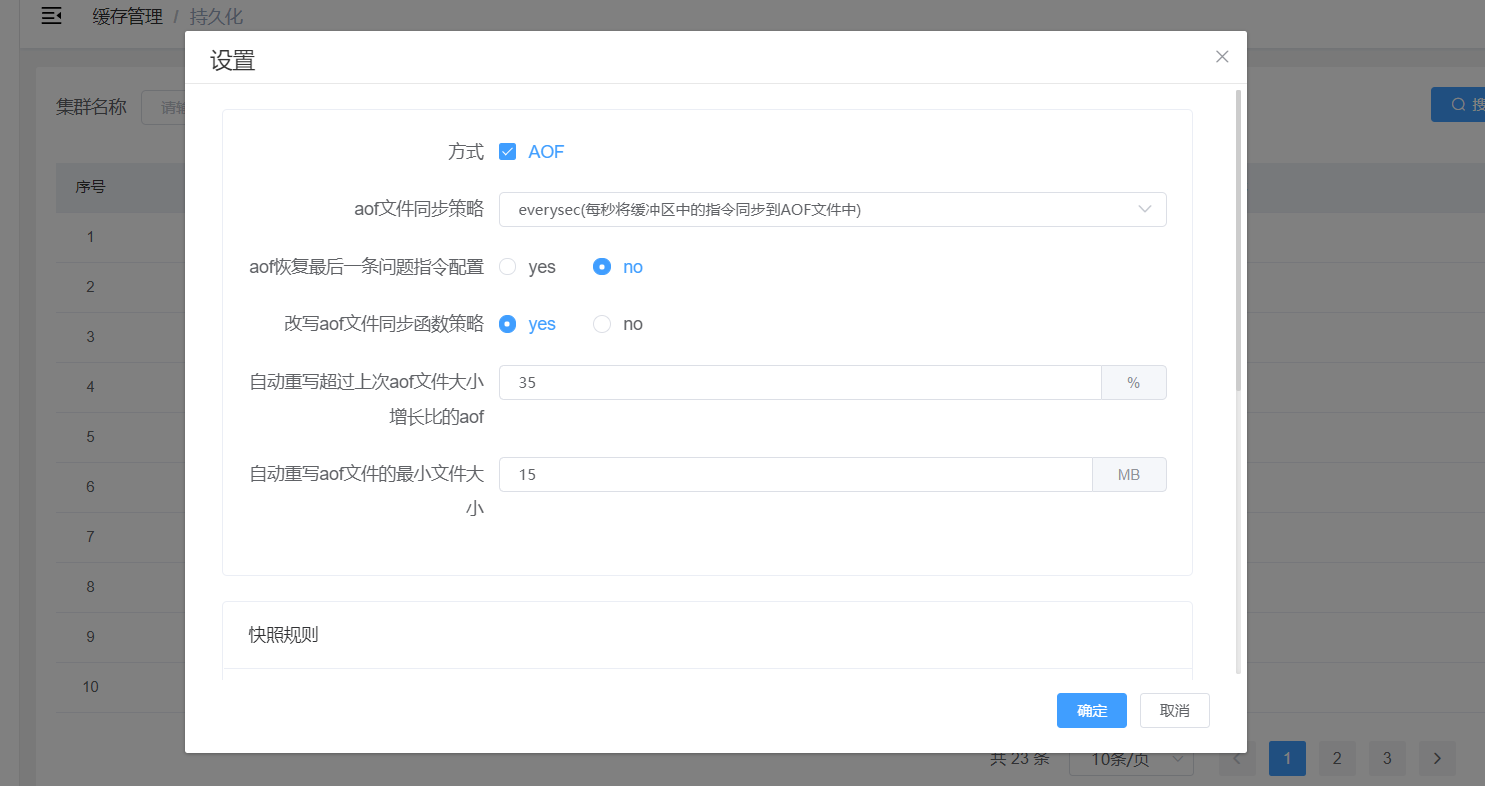
上方搜索框可以输入集群名称来进行查询，该搜索框支持模糊查询，输入完成集群名称后点击“搜索”按钮即可展示出符合条件的集群，点击“重置”按钮会清空搜索框里的内容，同时会将所有的集群都展示出来。



#### 设置

在下方列表中点击“设置”按钮即可在弹出的弹框中对redis集群进行持久化设置。

下方是redis持久化的两种方式，这两种方式都是在同一弹窗内的，并且这两种方式可以同时进行设置，也就是说一个redis集群可以设置两种不同的持久化方式，具体设置可以根据自己的实际需求进行设置。



点击下方的“+”号图案可以增加一行“save”输入框，点击“垃圾桶”图案可以删除该行。



### 淘汰策略

redis淘汰策略是对主从模式集群和集群模式集群生效的，所以淘汰策略设置操作对这两种模式的集群没有区别，下面展示如何进行淘汰策略设置操作。

本界面只会展示主从模式和集群模式的集群。

#### 搜索

上方搜索框可以输入集群名称来进行查询，该搜索框支持模糊查询，输入完成集群名称后点击“搜索”按钮即可展示出符合条件的集群，点击“重置”按钮会清空搜索框里的内容，同时会将所有的集群都展示出来。



#### 设置

在下方列表中点击“设置”按钮即可在弹出的弹框中对redis集群进行淘汰策略设置。



可以根据自己的实际需求进行设置。

## 缓存审计

### 搜索

上方搜索框有两个，分别是“用户”与“功能”，两个搜索框均支持模糊查询，在搜索框输入信息后点击“搜索”，会将符合条件的信息展示在下方列表中，点击“重置”后会将搜索框中的内容清除，同时会将所有信息展示在下方列表。

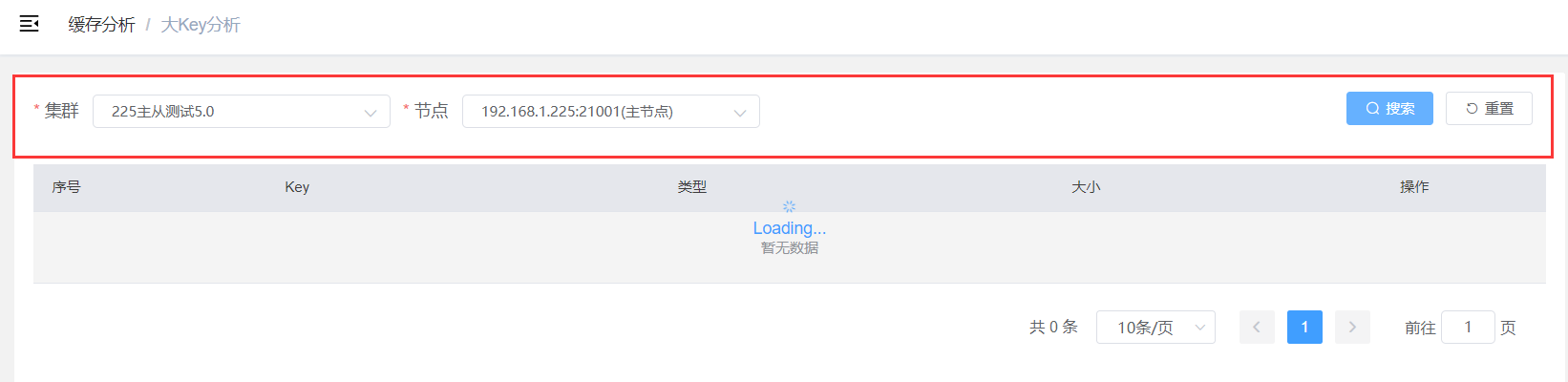


## 缓存分析

### 大Key分析

#### 搜索

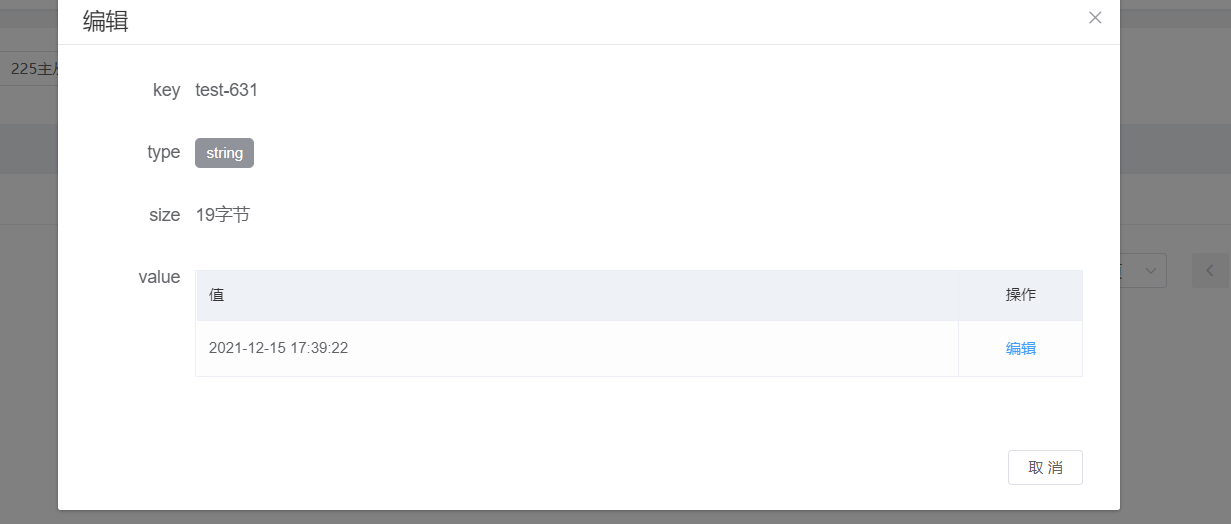
首先需要查询出大key后才能对大key进行操作分析，需要在上方选择集群与节点后点击“搜索”按钮，经过等待下方会展示出该集群和该节点下的每个数据类型中最大的一个key。



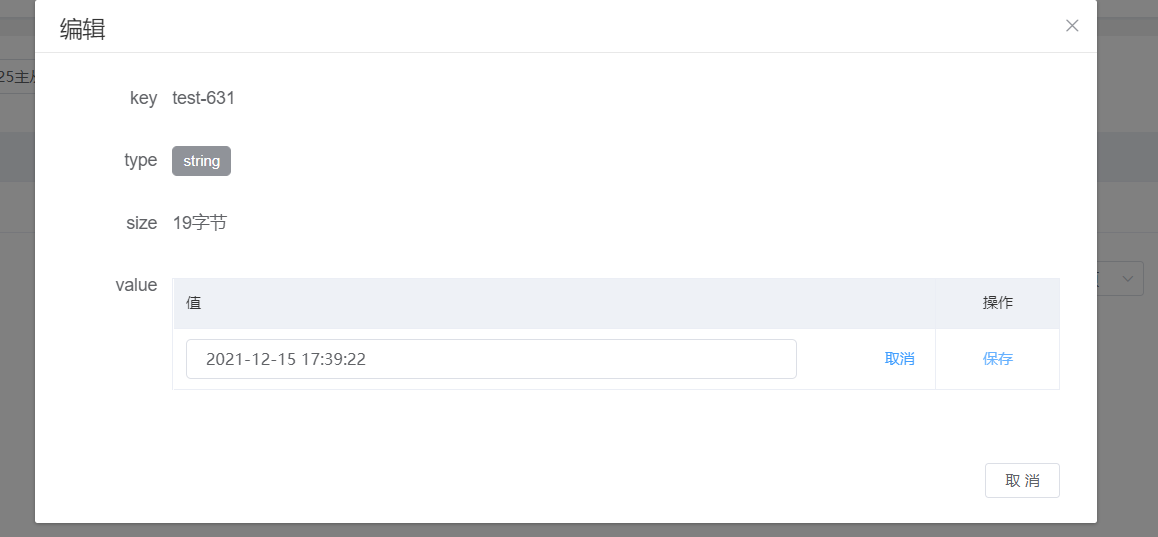
#### 编辑

点击编辑会弹出弹窗展示该key的详细数据。

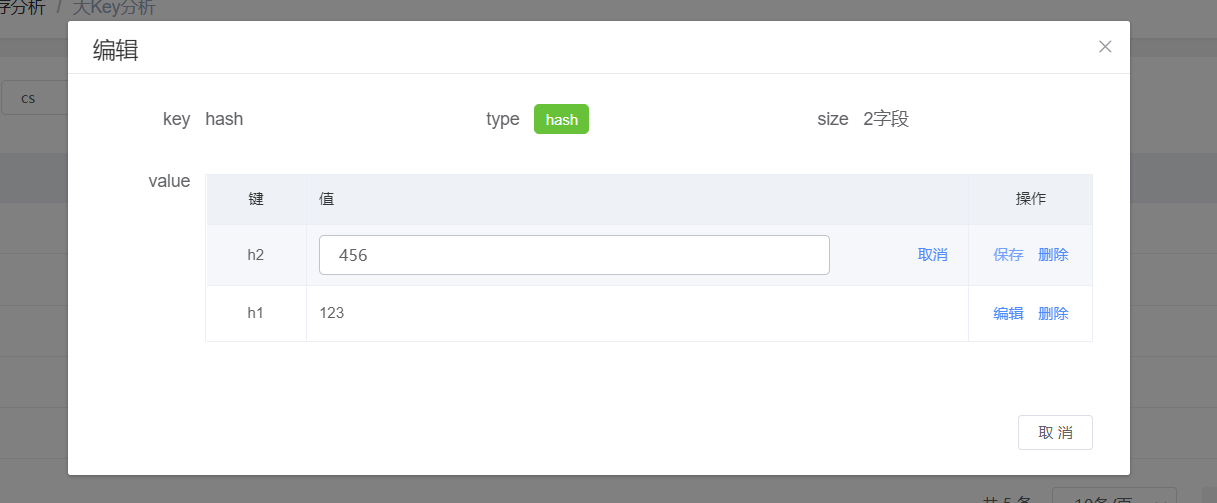




点击弹窗中的编辑会将值那一列变为可输入的文本框，输入想修改的数据后点击“保存”会弹出弹窗，再次点击“确定”后即可保存修改后的值，点击“取消”就会取消编辑。







#### 删除

在外面列表中点击“删除”按钮会弹出弹窗提示是否删除该key，确定后会删除该key。



### 热Key分析

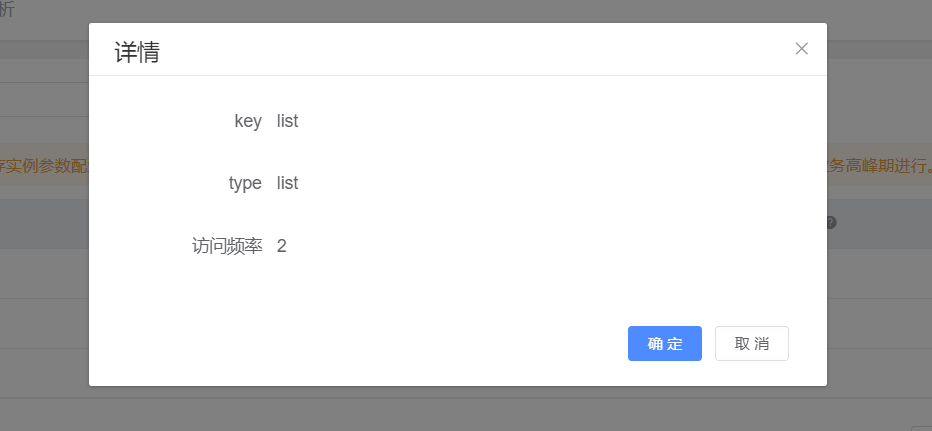
#### 搜索

首先需要查询出热key后才能对热key进行操作分析，需要在上方选择集群与节点后点击“搜索”按钮，经过等待下方会展示出该集群和该节点下的热key。



#### 详情

点击“详情”按钮会展示出该热key的类型和访问频率。



#### 删除

在外面列表中点击“删除”按钮会弹出弹窗提示是否删除该key，确定后会删除该key。



#### 阈值设置

点击“阈值设置”按钮会设置该集群节点的衰减因子和概率因子，前提是选择了集群和节点。



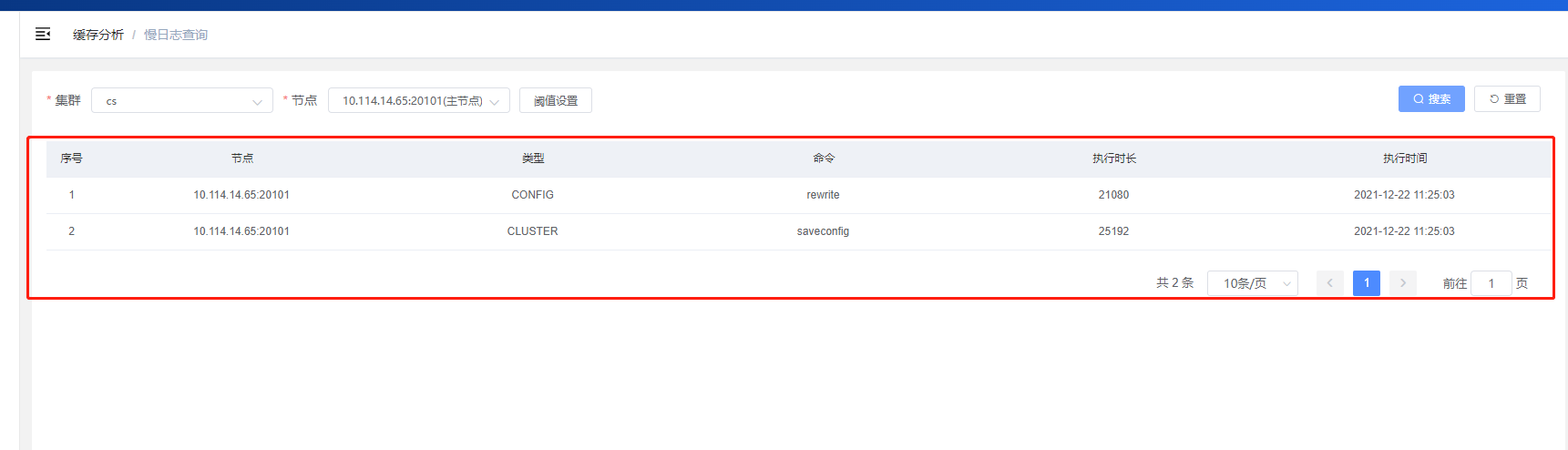
下方有“同步修改集群其他节点”的开关，打开开关后会将修改后的内容同步到该集群下的其他节点上。关于衰减因子与概率因子的解释，将鼠标移动到“！”号上就会显示出其含义。



### 慢日志查询

#### 搜索

需要在上方选择集群与节点后点击“搜索”按钮，经过等待下方会展示出该集群和该节点下的慢日志。



#### 阈值设置

点击“阈值设置”按钮会设置该集群节点的慢日志配置参数，前提是选择了集群和节点。下方有“同步修改集群其他节点”的开关，打开开关后会将修改后的内容同步到该集群下的其他节点上。关于配置参数的解释，将鼠标移动到“！”号上就会显示出其含义。



## 缓存监控

### 集群监控

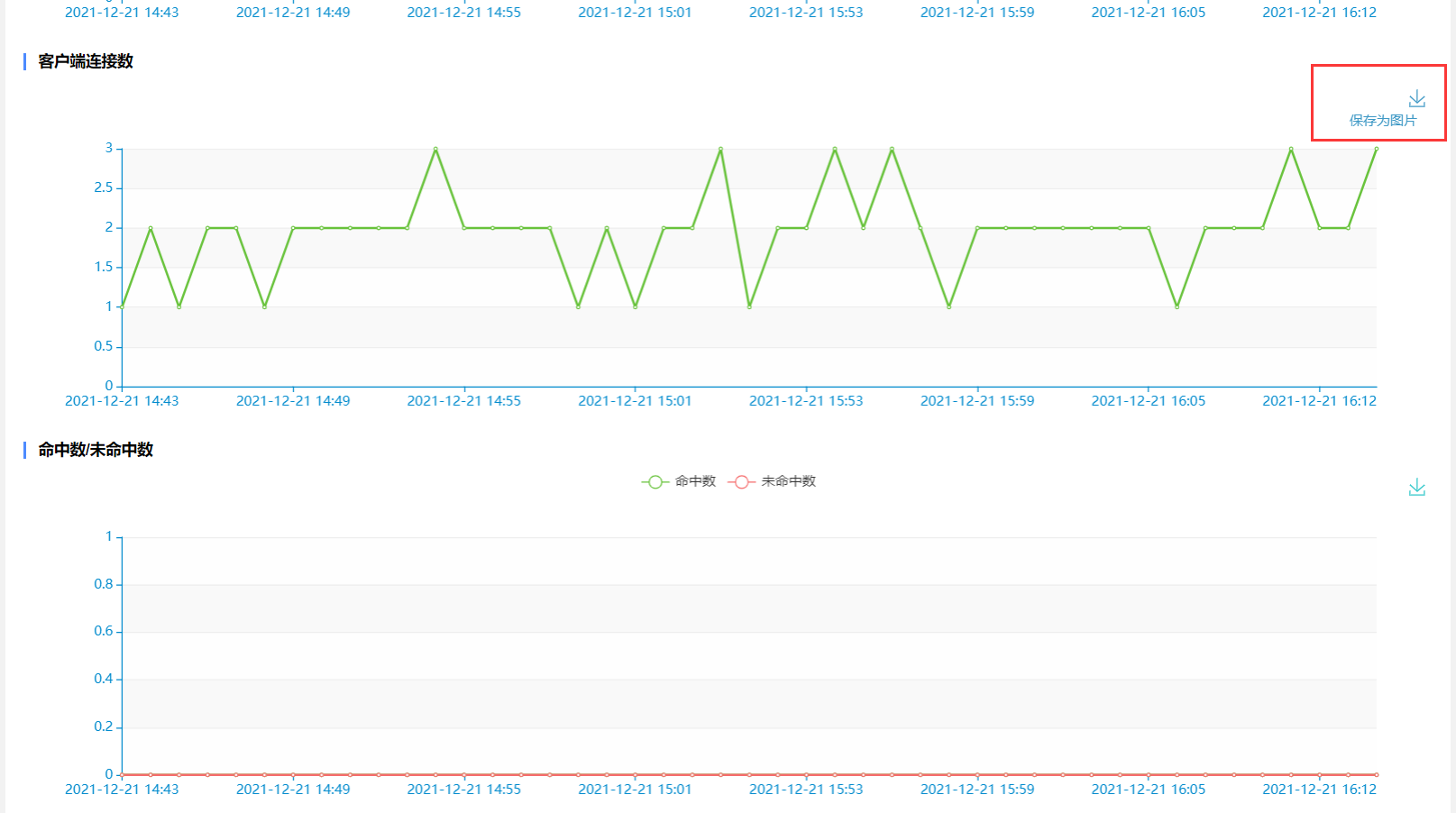
集群监控界面点进去后会将所有redis集群都展示出来，最上方是集群统计，下面是集群的详细信息，每个集群都会是一块一块的图像。



### 节点监控

节点监控主要是监控redis集群下的节点的各种状态和信息，在上方下拉框中选择集群和节点后，下面就会展示出这个节点的详细信息，包括各种数据的统计以及通过图像展示出来的信息。





### 客户端监控

客户端监控是监控集群下的节点所链接的客户端，在上方下拉框中选择集群节点后，就会在下方展示出该节点下所链接的所有客户端，关于每个字段的解释都可以将鼠标移动到字段后面的“？”上，就可以显示出该字段的含义。





## 组织多中心架构

### 多中心架构设置

#### 搜索

上方搜索框可以输入名称来进行查询，该搜索框支持模糊查询，输入完成名称后点击“搜索”按钮即可展示出符合条件的架构，点击“重置”按钮会清空搜索框里的内容，同时会将所有的架构都展示出来。



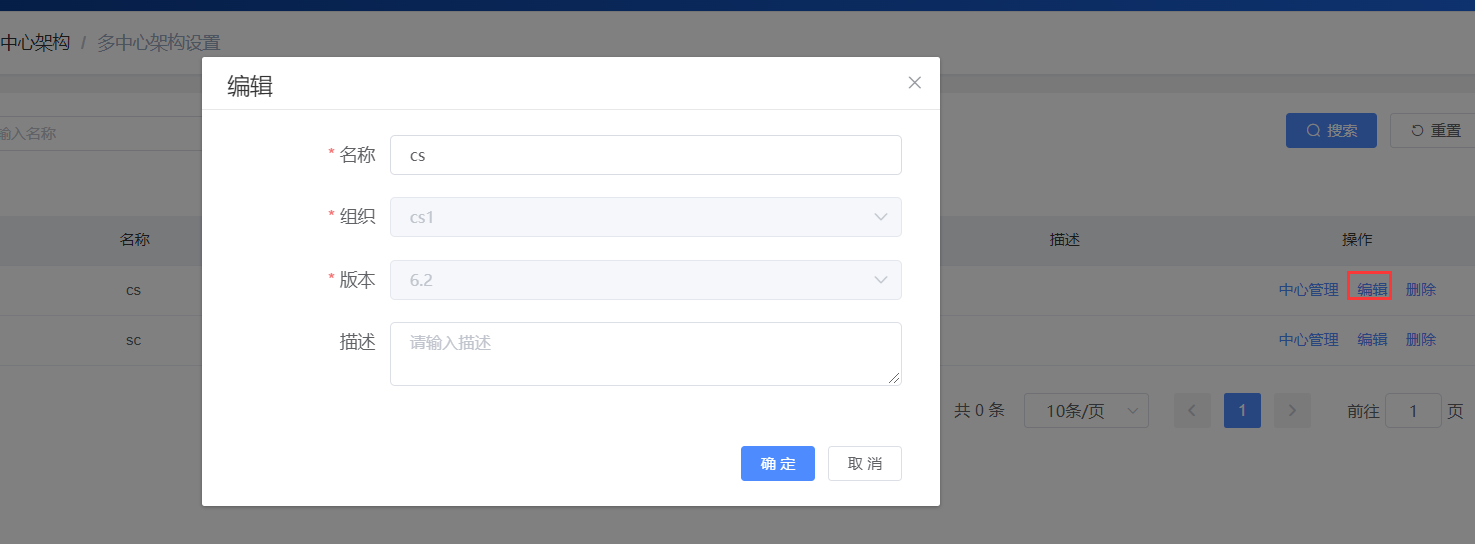
#### 新增

点击“新增”按钮后弹出弹框，输入信息后点击“确定”按钮，就会保存该架构信息。



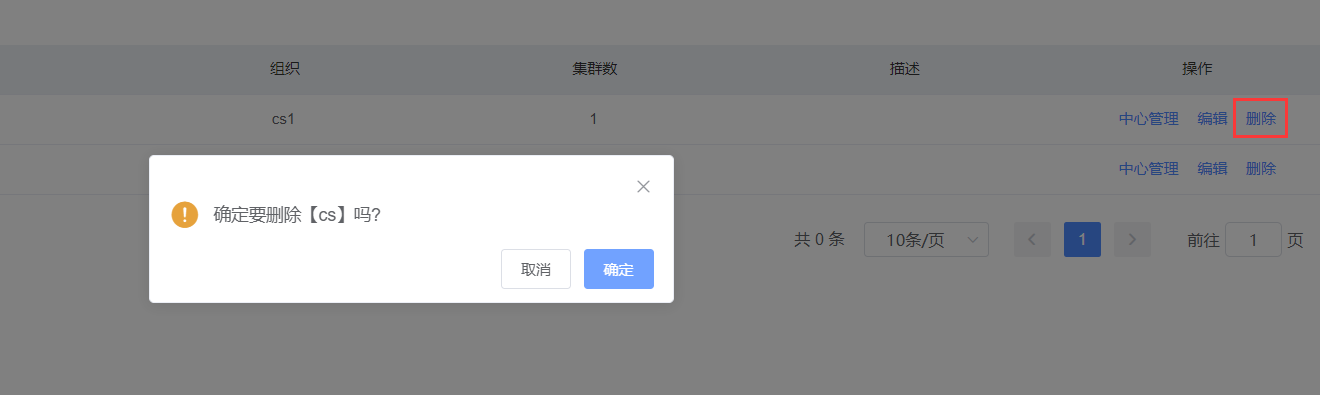
#### 编辑

点击“编辑”按钮会弹出编辑弹窗，其中组织与版本不可修改，其余可以修改，根据需求修改后点击“确定”会保存修改后的数据，点击“取消”会取消修改。



#### 删除

点击“删除”按钮会弹出弹窗提醒再次确认删除，点击“确定”后就会删除该架构，在此提醒，删除一个架构时必须保证该架构下没有集群才可删除。

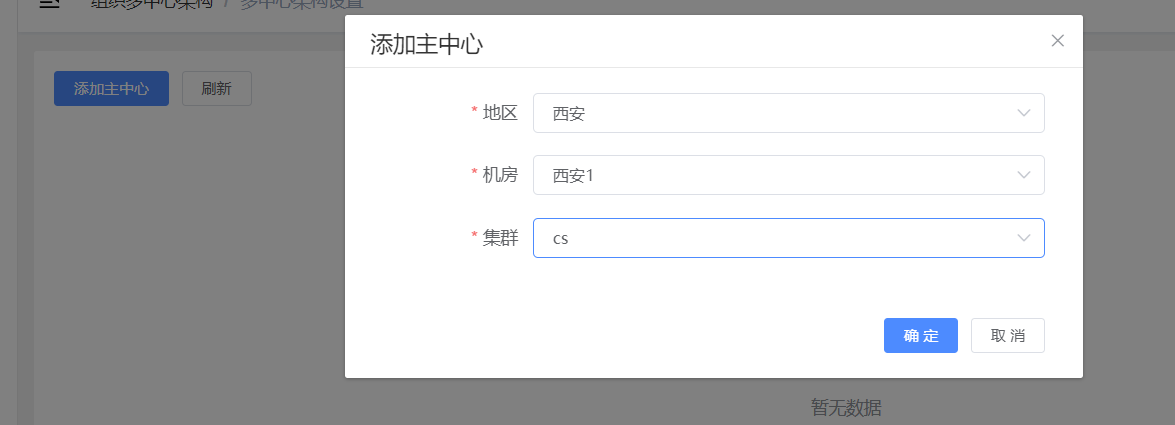


#### 中心管理

点击“中心管理”后会进去一个新页面。

##### 1.添加主中心

进入页面后点击“添加主中心”后弹出弹框，选择信息后点击“确定”会保存该主中心，在添加完一个主中心后，“添加主中心”按钮就会消失，是因为一个架构下只能有一个主中心。





##### 2.关联备中心

点击“关联备中心”后弹出弹窗，选择信息后点击“确定”按钮后会保存该关联后的备中心。



##### 3.启动/停止



备中心有“启动”、“停止”按钮，备中心初始状态为停止，点击“启动”按钮会将状态变为“增量进行中”，点击“停止”会将状态变为“停止”。

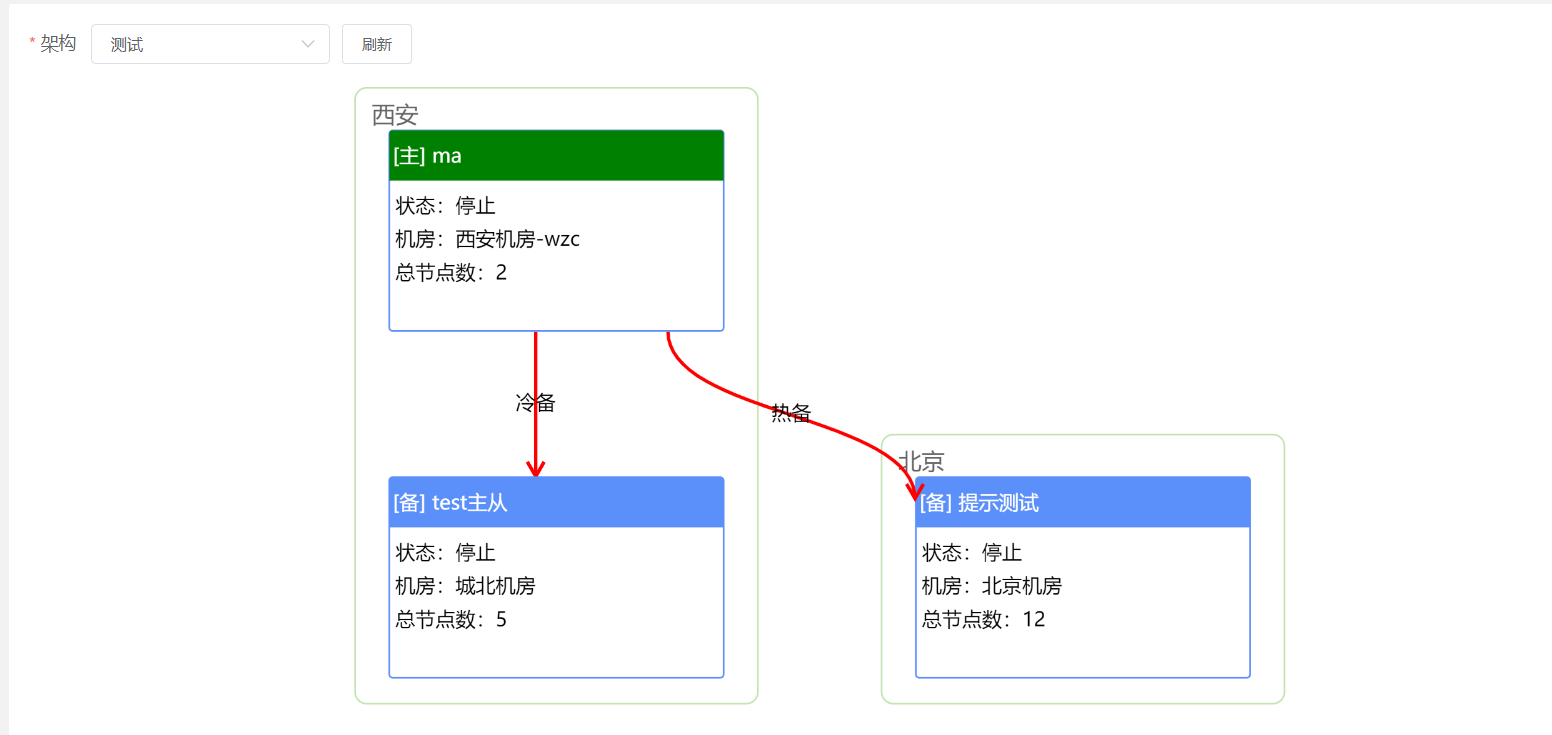
##### 3.删除

点击“删除”按钮会弹出弹窗提醒再次确认删除，点击“确定”后就会删除该架构，在此提醒，不论是删除备中心还是主中心时，都要保证该中心下没有关联其他的备中心，这样才能删除。



### 多中心监控

在上方下拉框中选择架构后，会将该架构下的所有中心按照地区的划分来进行展示。



### 多中心数据迁移

#### 创建任务

点击“创建任务”，输入名称，选择源集群与目标集群以及迁移类型后点击“确定”就会创建成功该任务。





#### 启动

点击“启动”按钮会弹出弹窗再次提醒是否启动该任务，点击“确定”就会启动该任务。





#### 详情

点击“详情”会展示出该任务的详细信息。



#### 删除

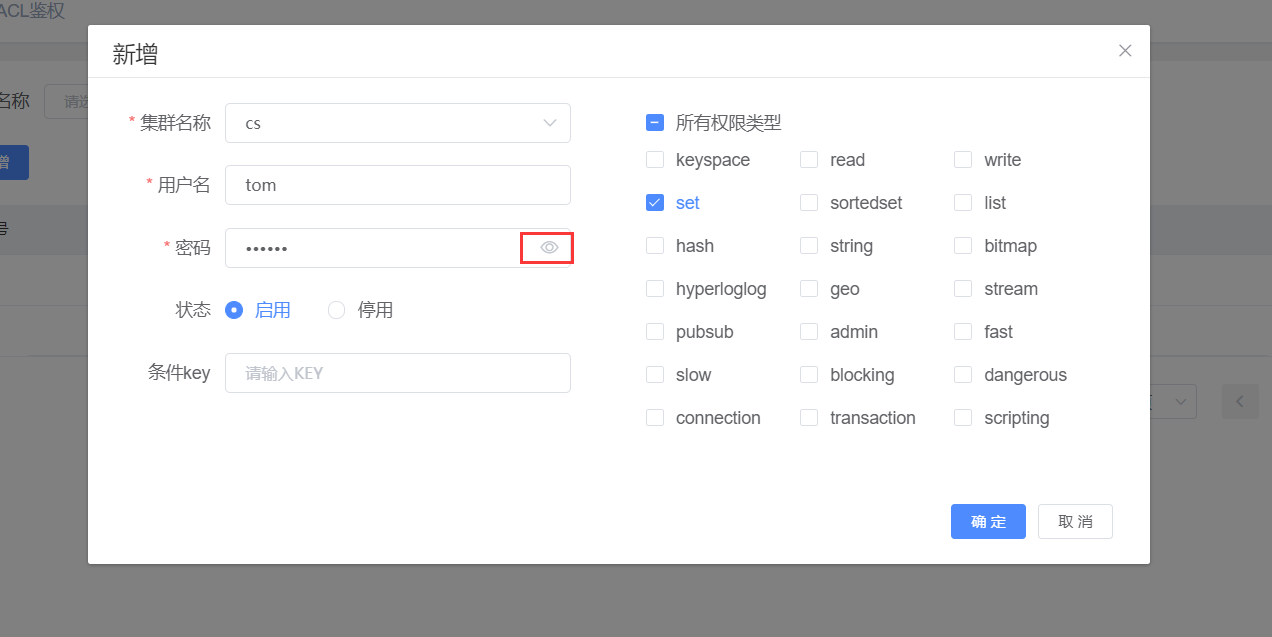
点击“删除”会弹出弹窗再次提醒是否删除该任务，点击“确定”后就会删除该任务。



## ACL鉴权

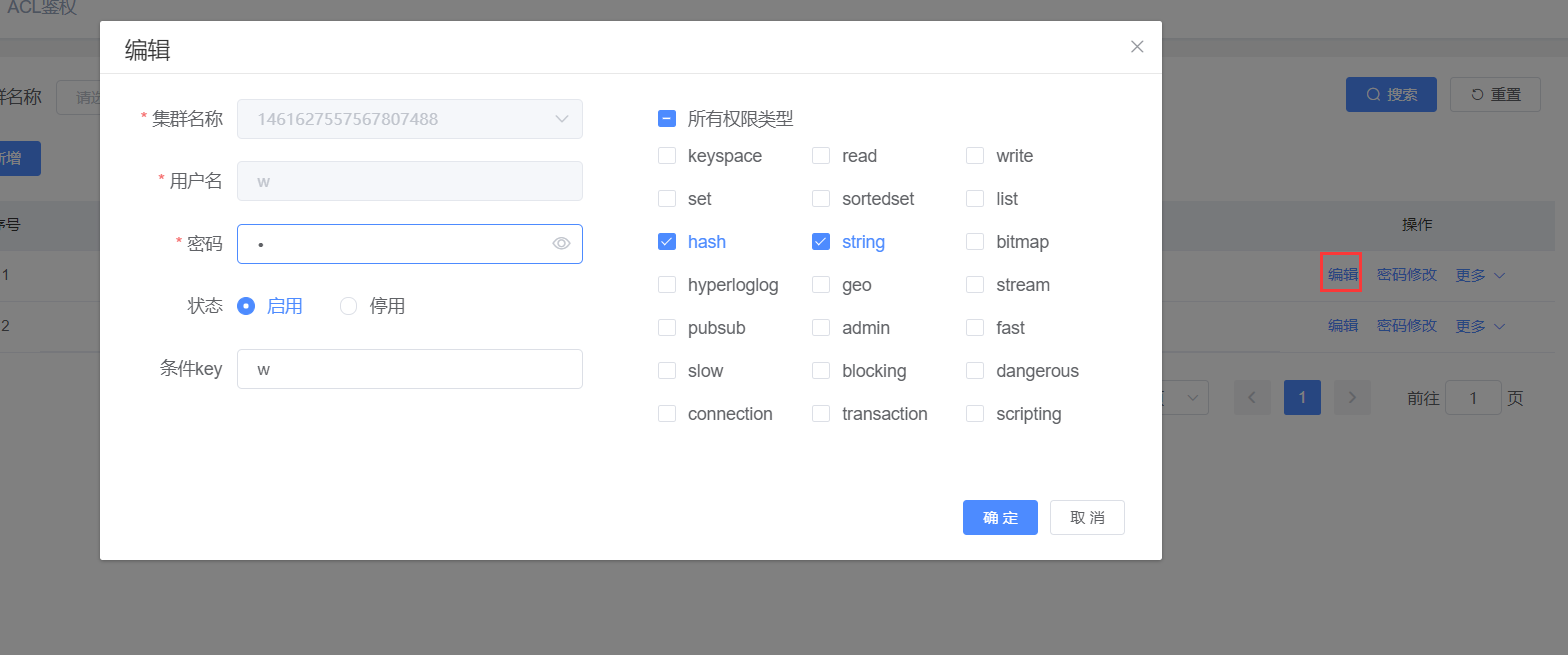
### 新增

点击“新增”按钮，弹出弹窗后输入信息，右侧是可以多选的权限类型，输入完毕后点击“确定”按钮将会保存该条的信息。点击密码右侧的小眼睛图标会将密码明文显示。



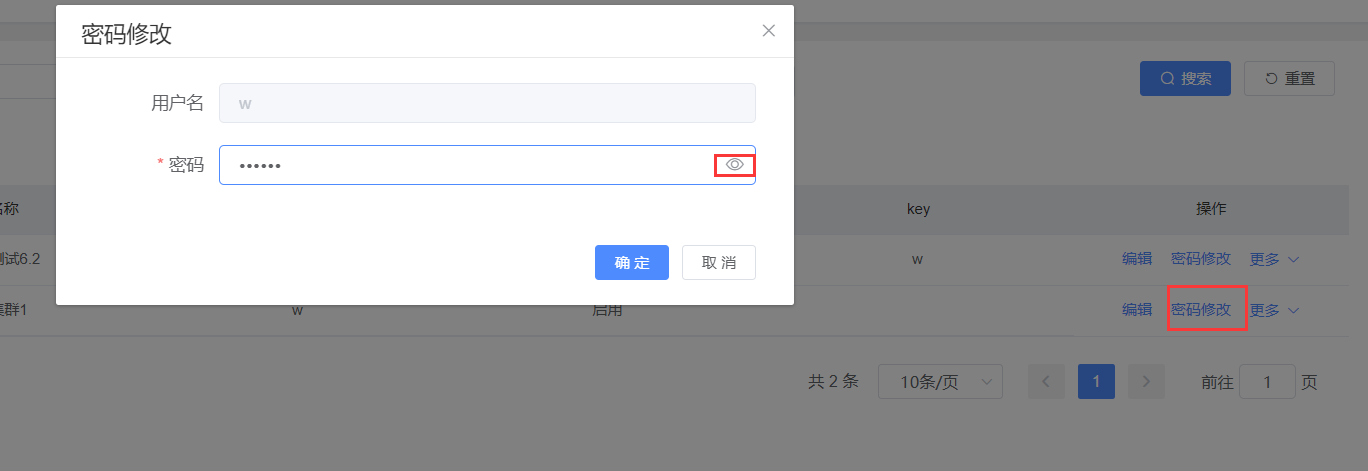
### 编辑

点击“编辑”按钮会弹出编辑弹窗，其中集群名称和用户名不可修改，其余可以修改，根据需求修改后点击“确定”会保存修改后的数据，点击“取消”会取消修改。



### 密码修改

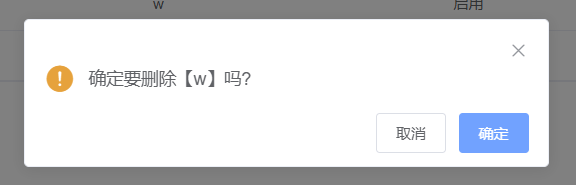
点击“密码修改”按钮，弹出弹窗，用户名不可修改。点击密码右侧的小眼睛图标会将密码明文显示。



### 删除

点击“删除”按钮会弹出弹窗提醒再次确认删除，点击“确定”后就会删除该条数据。

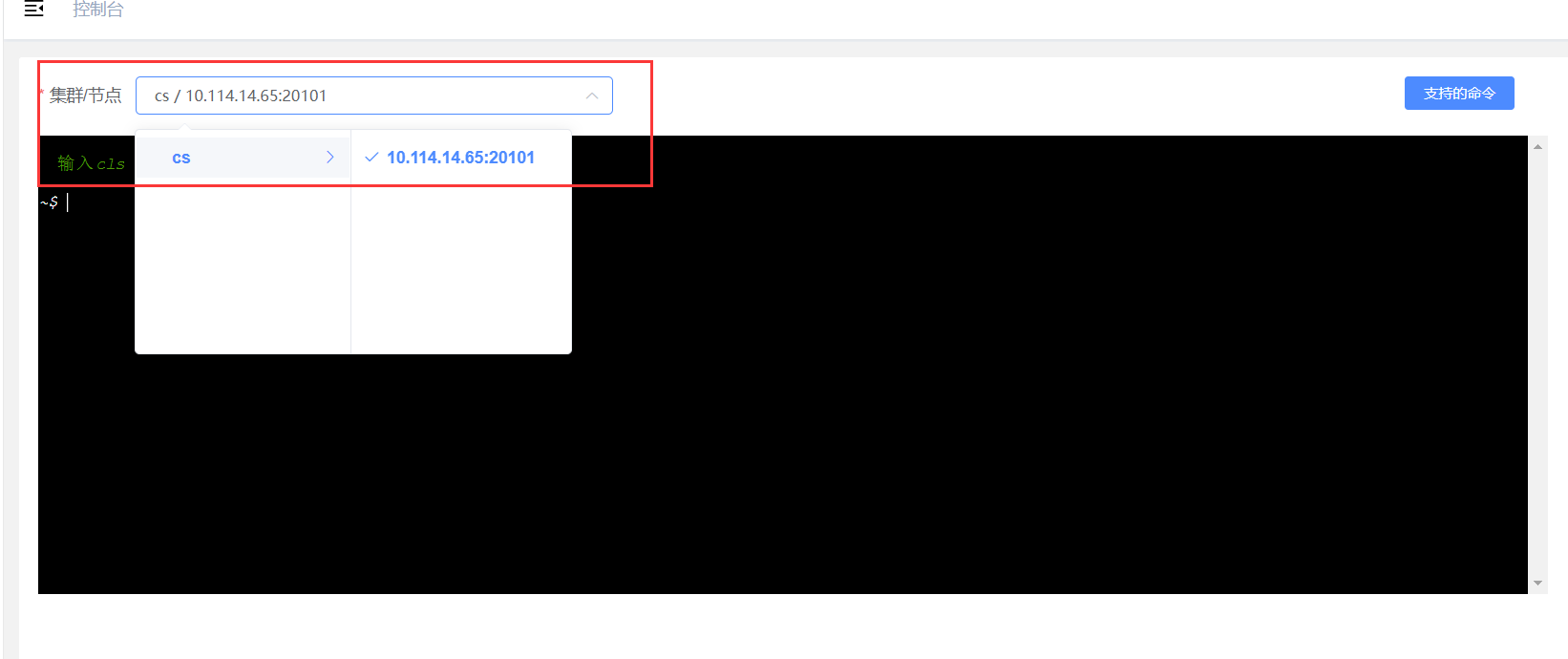




## 控制台

### 命令输入

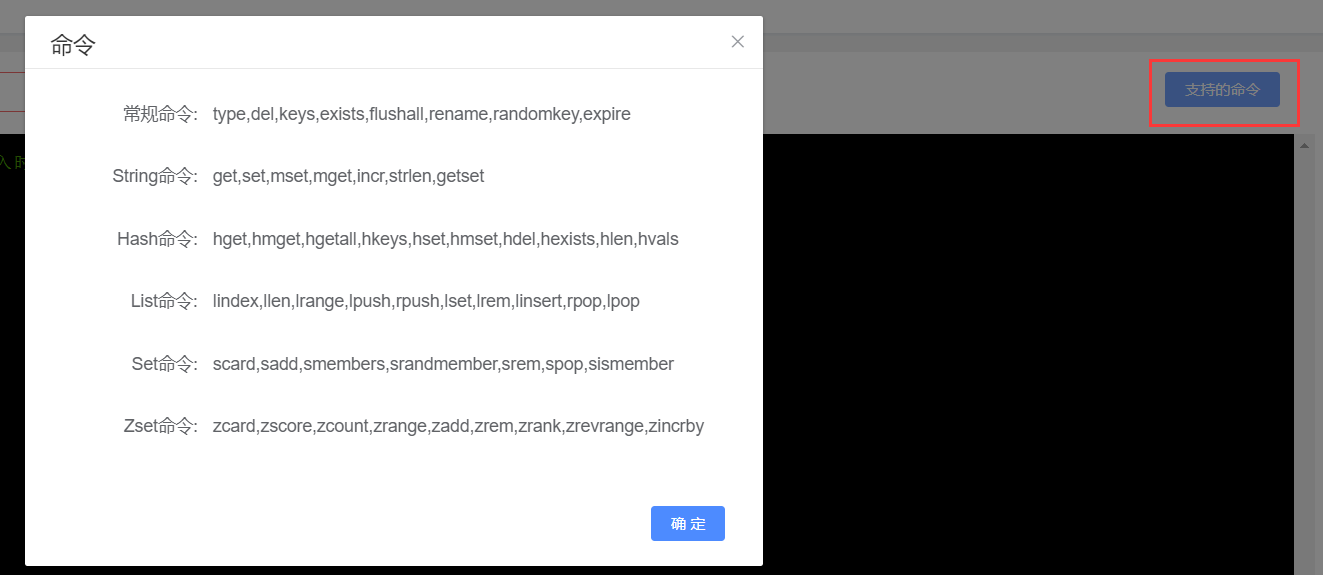
控制台提供了可供输入操作集群redis的命令，在上方下拉框中选择节点后在下方输入命令后点击回车键才能执行命令，否则会提示选择集群节点。





### 支持的命令

点击“支持的命令”会展示出该控制台支持的redsi命令。



## SDK

### 引入依赖包

如果缓存相关包及pom文件已发布到私服可使用以下依赖：

<dependency>

<groupId>dap.paas</groupId>

<artifactId>dap-pass-spring-cache-redis</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

### 配置文件

#### 各种模式配置文件

（1）cluster模式

dap.cache.redis[0].valueSerializer=jackson

dap.cache.redis[0].groupName=cluster-1

dap.cache.redis[0].type=cluster

dap.cache.redis[0].hosts=10.114.10.x1:20001,10.114.10.x2:20002

dap.cache.redis[0].namespace=cluster1

dap.cache.redis[0].timeout=6000

dap.cache.redis[0].password=

dap.cache.redis[0].pool.maxTotal=10

dap.cache.redis[0].pool.minIdle=10

dap.cache.redis[0].pool.maxIdle=10

（2）哨兵模式

dap.cache.redis[0].valueSerializer=jackson

dap.cache.redis[0].groupName=sentinel0

dap.cache.redis[0].type=sentinel

dap.cache.redis[0].hosts=192.168.1.158:26379

dap.cache.redis[0].namespace=fop

dap.cache.redis[0].masters=mymaster

dap.cache.redis[0].password=foobared

dap.cache.redis[0]=192.168.1.158:26379

dap.cache.redis[0].timeout=6000

dap.cache.redis[0].pool.maxTotal=2

dap.cache.redis[0].pool.minIdle=0

dap.cache.redis[0].pool.maxIdle=2

（3）单机版

dap.cache.redis[0].valueSerializer=jackson

dap.cache.redis[0].groupName=standalone0

dap.cache.redis[0].type=standalone

dap.cache.redis[0].hosts=127.0.0.1:6379

dap.cache.redis[0].namespace=cluster2

dap.cache.redis[0].timeout=6000

dap.cache.redis[0].pool.maxTotal=100

dap.cache.redis[0].pool.minIdle=100

dap.cache.redis[0].pool.maxIdle=100

#### 相关配置说明

valueSerializer：value的序列化方式 支持jackson ，fastjson，kryo，string 四种方式。

groupName：分组名称，ICacheClient 启动后会设置为bean的名称。

type：standalone单机版 ，sentinel哨兵版 ，cluster集群版。

hosts：redis实例IP：端口 多个用英文逗号隔开。

namespace：缓存key的命名空间。

useNamespace：是否使namespace生效， 默认false。

timeout ：连接超时时间，单位毫秒，默认6000。

pool：连接池相关参数。

maxTotal：在获取当时通过pool能够获取到的最大的连接的jedis个数（已经被客户端 连接上或者正在闲置等待客户端连接）。

maxIdle：资源池允许的最大空闲连接数。

minIdle：资源池确保的最少空闲连接数。

blockWhenExhausted：当资源池用尽后，调用者是否要等待。只有当值为true时，下面的maxWaitMillis才会生效。

maxWaitMillis：当资源池连接用尽后，调用者的最大等待时间（单位为毫秒）。

testOnBorrow：向资源池借用连接时是否做连接有效性检测（ping）。检测到的无效连接将会被移除。

testOnReturn：向资源池归还连接时是否做连接有效性检测（ping）。检测到无效连接将会被移除。

jmxEnabled：是否开启JMX监控。

minEvictableIdleTimeMillis：连接空闲的最小时间，达到此值后空闲连接将可能会被移除。负值(-1)表示不移除 , 默认30分钟。

softMinEvictableIdleTimeMillis：连接空闲的最小时间，达到此值后空闲链接将会被移除，且保留“minIdle”个空闲连接数。默认为-1。

numTestsPerEvictionRun：对于“空闲链接”检测线程而言，每次检测的链接资源的个数。默认为3。

timeBetweenEvictionRunsMillis：“空闲链接”检测线程，检测的周期，毫秒数。如果为负值，表示不运行“检测线程”。默认为-1。

### 缓存使用

**可通过以下方式直接注入依赖对象，调用缓存组件api**

**@Autowired**

**private ICacheClient cacheClient;**

**如果配置中使用的是多数据源可使用以下配置：**

**dap.cache.redis[0]**

**dap.cache.redis[1]**

**dap.cache.redis[2]**

**@Resource(name="cluster1")**

**private ICacheClient cacheClient1;**

**@Resource(name="cluster2")**

**private ICacheClient cacheClient2;**

**cluster1，cluster2 是配置文件中配置的groupName**

**常用接口：**

**将指定类型数据对象写入缓存**

**cacheClient.setData(key,value);**

**通过KEY查询一个缓存值**

**cacheClient.get (key);**

**删除缓存数据**

**cacheClient.delCachedData(a);**

### 其他接口说明

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 常规命令 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

/\*\*

\* 根据key模糊查询所有key

\*

\* @param key prefixKey

\* @return 所有key集合

\*/

public Set<String> keys(String key);

/\*\*

\* 判断key是否存在

\*

\* @param key 缓存key

\* @return true缓存存在，false

\* @throws PlatformException 缓存异常

\*/

boolean exist(String key) throws PlatformException;

/\*\*

\* 获取key 的类型

\*

\* @param key

\* @return

\* @throws PlatformException

\*/

String type(String key) throws PlatformException;

/\*\*

\* 删除多个缓存数据

\*

\* @param keys 字符串数组,多个缓存的key

\* @return 返回数字为删除的key数目，如果为零说明执行失败

\* @throws PlatformException 缓存异常

\*/

long delCachedData(String... keys) throws PlatformException;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* String \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

/\*\*

\* 从缓存中获取一个数据

\*

\* @param <T> 泛型

\* @param key key

\* @return 返回类型

\* @throws PlatformException 缓存异常

\*/

<T> T get(String key) throws PlatformException;

/\*\*

\* 同时设置一个或多个 key-value 对

\*

\* @param map 对应多组 key value

\* @return 写入是否成功

\*/

boolean mset(Map<String, Object> map);

/\*\*

\* 给定多个key，返回对应的缓存数据

\*

\* @param keys 缓存的key

\* @return 返回多个key的缓存数据

\* @throws PlatformException 缓存异常

\*/

<T> List<T> mget(List<String> keys);

/\*\*

\* 将字符串类型数据写入缓存,并指定过期时间

\*

\* @param key 缓存key

\* @param data 缓存数据

\* @param timeout 有效时长

\* @param unit 时间单位

\* @return 是否成功写入

\* @throws PlatformException 缓存异常

\*/

boolean setData(String key, Object data, long timeout, TimeUnit unit) throws PlatformException;

/\*\*

\* 将任意类型数据写入缓存

\* 默认缓存数据无过期时间,即永久有效

\*

\* @param key 缓存key

\* @param data 缓存数据

\* @return 是否成功写入

\* @throws PlatformException 缓存异常

\*/

boolean setData(String key, Object data) throws PlatformException;

/\*\*

\* 向缓存中写入Object类型数据,如果缓存中已存在该key,则不做任何操作直接返回false

\* 无过期时间

\*

\* @param key 缓存key

\* @param data 缓存数据

\* @return 是否成功写入

\* @throws PlatformException 缓存异常

\*/

boolean setDataNx(String key, Object data) throws PlatformException;

/\*\*

\* 向缓存中写入Object类型数据,并设置有效时长,如果缓存中已存在该key,则不做任何操作直接返回false

\*

\* @param key 缓存key

\* @param data 缓存数据

\* @param timeout 有效时长

\* @param unit 时间单位

\* @return 是否成功写入

\* @throws PlatformException 缓存异常

\*/

boolean setDataNx(String key, Object data, long timeout, TimeUnit unit)

throws PlatformException;

/\*\*

\* 读取缓存数据

\*

\* @param key 缓存的key

\* @return 返回一个缓存数据

\* @throws PlatformException 缓存异常

\*/

String getCachedString(String key) throws PlatformException;

/\*\*

\* 设置缓存有效期

\*

\* @param key 缓存的key

\* @param timeout 有效期

\* @param unit 有效期时间单位

\* @return 返回有效时间（秒）

\* @throws PlatformException 缓存异常

\*/

long setExpire(String key, long timeout, TimeUnit unit) throws PlatformException;

/\*\*

\* 获取缓存有效期

\*

\* @param key 缓存的key

\* @return 返回有效期时间（秒）

\* @throws PlatformException 缓存异常

\*/

long getExpire(String key) throws PlatformException;

/\*\*

\* 自增操作,可设置步长

\*

\* @param key 缓存的key

\* @param step 步长

\* @return 返回自增之后的值

\* @throws PlatformException 缓存异常

\*/

long incr(String key, long step) throws PlatformException;

/\*\*

\* 自减操作,可设置步长

\*

\* @param key 缓存的key

\* @param step 步长

\* @return 返回自减之后的值

\* @throws PlatformException 缓存异常

\*/

long decr(String key, long step) throws PlatformException;

/\*\*

\* @param oldKey 旧Key名

\* @param newKey 新Key名

\* @return 是否修改成功

\* @throws PlatformException 缓存异常

\*/

boolean reName(String oldKey, String newKey) throws PlatformException;

/\*\*

\* 锁在给定的等待时间内空闲，则获取锁成功 返回true， 否则返回false，作为阻塞式锁使用

\*

\* @param key 锁键

\* @param value 被谁锁定

\* @param trytime 尝试获取锁时长，建议传递500,结合时间单位，则可表示500毫秒

\* @param timeout 超时时间

\* @param unit 超时时间单位

\* @return 是否成功锁住

\* @throws PlatformException 缓存异常

\*/

boolean tryLock(String key, String value, long trytime, long timeout, TimeUnit unit)

throws PlatformException;

/\*\*

\* 获取锁 如果锁可用 立即返回true， 否则立即返回false，作为非阻塞式锁使用

\*

\* @param key 缓存的key

\* @param value 缓存的数据

\* @return 返回是否成功占用锁

\* @throws PlatformException 缓存异常

\*/

boolean tryLock(String key, String value) throws PlatformException;

/\*\*

\* 释放单个锁

\*

\* @param key 锁键

\* @param value 被谁释放

\* @return 返回是否成功解锁

\* @throws PlatformException 缓存异常

\*/

boolean unLock(String key, String value) throws PlatformException;

/\*\*

\* 删除hash(散列)表字段

\*

\* @param key hash表的key

\* @param field hash表的field

\* @return 删除的数量

\* @throws PlatformException 缓存异常

\*/

long delCachedDataH(String key, String field) throws PlatformException;

/\*\*

\* 将hash(散列)表 key 中的域 field 的值设为 value 。

\* 如果 key 不存在，一个新的hash(散列)表被创建并进行 HSET 操作。

\* 如果域 field 已经存在于hash(散列)表中，旧值将被覆盖

\*

\* @param key hash(散列)表的键

\* @param field hash(散列)表的域

\* @param value hash(散列)表的值

\* @return 是否成功写入

\* @throws PlatformException 缓存异常

\*/

boolean hset(String key, String field, Object value) throws PlatformException;

/\*\*

\* 将hash(散列)表 key 中的域 field 的值设置为 value ，当且仅当域 field 不存在

\* 若域 field 已经存在，该操作无效。

\* 如果 key 不存在，一个新hash(散列)表被创建并执行 HSETNX 命令。

\*

\* @param key hash(散列)表的键

\* @param field hash(散列)表的域

\* @param value hash(散列)表的值

\* @return 是否成功写入

\* @throws PlatformException 缓存异常

\*/

boolean hsetNx(String key, String field, Object value) throws PlatformException;

/\*\*

\* 同时将多个 field-value (域-值)对设置到hash(散列)表 key 中

\* 此命令会覆盖hash(散列)表中已存在的域

\*

\* @param key hash(散列)表的键

\* @param key\_value 参数Map中的key为hash(散列)表的域，Map中的value为hash(散列)表的值

\* @return 是否成功写入

\* @throws PlatformException 缓存异常

\*/

boolean hmset(String key, Map<String, Object> key\_value) throws PlatformException;

/\*\*

\* 查看hash(散列)表 key 中，给定域 field 是否存在

\* 如果hash(散列)表含有给定域,返回true

\* 如果hash(散列)表不含有给定域,或 key 不存在,返回 false

\*

\* @param key hash(散列)表的键

\* @param field hash(散列)表的域

\* @return 是否存在

\* @throws PlatformException 缓存异常

\*/

boolean hexist(String key, String field) throws PlatformException;

/\*\*

\* 删除hash(散列)表 key 中的一个或多个指定域，不存在的域将被忽略

\*

\* @param key hash(散列)表的键

\* @param field hash(散列)表的域

\* @return 被成功移除的域的数量，不包括被忽略的域

\* @throws PlatformException 缓存异常

\*/

long hdel(String key, String... field) throws PlatformException;

/\*\*

\* 获取hash(散列)表 key 中给定域 field 的值

\*

\* @param <T> 泛型

\* @param key hash(散列)表的键

\* @param field hash(散列)表的域

\* @return 给定域的值, 当给定域不存在或是给定 key 不存在时，返回null

\* @throws PlatformException 缓存异常

\*/

<T> T hget(String key, String field) throws PlatformException;

/\*\*

\* 返回hash(散列)表 key 中，所有的域和值

\* Map中的key为hash(散列)表中的域（field）,Map中key对应的value为hash(散列)表中域的值

\*

\* @param <T> 泛型

\* @param key hash(散列)表的键

\* @return 以Map形式返回hash(散列)表的域和域的值

\* @throws PlatformException 缓存异常

\*/

public <T> Map<String, T> hgetall(String key) throws PlatformException;

/\*\*

\* 返回hash(散列)表 key 中所有域的值

\*

\* @param <T> 泛型

\* @param key hash(散列)表的键

\* @return 返回hash(散列)表 key 中所有域的值

\* @throws PlatformException 缓存异常

\*/

<T> List<T> hvals(String key) throws PlatformException;

/\*\*

\* 获取连接池信息

\*

\* @return 连接池信息

\* @throws PlatformException 缓存异常

\*/

Map<String, String> getRedisPoolInfo() throws PlatformException;

/\*\*

\* 获取hash(散列)表所有的字段KEYS

\*

\* @param key hash表的key

\* @return 所有域

\* @throws PlatformException 缓存异常

\*/

Set<String> getHAllKeys(String key) throws PlatformException;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Set \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

/\*\*

\* 向redis Set集合 中添加元素

\*

\* @param key 缓存的key

\* @param members 写入set集合的元素

\* @return 是否成功写入

\* @throws PlatformException 缓存异常

\*/

public long sadd(String key, Object... members) throws PlatformException;

/\*\*

\* 移除并返回set集合中的一个随机元素

\*

\* @param key 缓存的key

\* @return set集合随机元素

\* @throws PlatformException 缓存异常

\*/

String spop(String key) throws PlatformException;

/\*\*

\* 返回set集合中的一个随机元素,但不从set集合中移除该元素

\*

\* @param key 缓存的key

\* @return set集合随机元素

\* @throws PlatformException 缓存异常

\*/

String srandmember(String key) throws PlatformException;

/\*\*

\* 获取 reids Set集合中所有元素

\*

\* @param <T> 泛型

\* @param key 缓存的key

\* @return String Set 集合

\* @throws PlatformException 缓存异常

\*/

<T> Set<T> smembers(String key) throws PlatformException;

/\*\*

\* 判断元素是否为集合key的成员

\*

\* @param key 缓存的key

\* @param member String 元素

\* @return 是否是key的成员

\* @throws PlatformException 缓存异常

\*/

boolean sismember(String key, Object member) throws PlatformException;

/\*\*

\* 获取集合key的元素数量

\*

\* @param key 缓存的key

\* @return 集合key的数量

\* @throws PlatformException 缓存异常

\*/

long scard(String key) throws PlatformException;

/\*\*

\* 从集合key中删除一个或多个元素

\*

\* @param key 缓存的key

\* @param members 集合key的元素

\* @return 删除的数量

\* @throws PlatformException 缓存异常

\*/

Long srem(String key, Object... members) throws PlatformException;

/\*\*

\* 从集合key中删除一个或多个元素

\*

\* @param key 缓存的key

\* @param members 集合key的元素

\* @return 删除的数量

\* @throws PlatformException 缓存异常

\*/

Long srem(String key, List<Object> members) throws PlatformException;

/\*\*

\* 通过pipeline管道的方式批量将set写入redis

\* 如果redis是集群部署，通过此接口写入redis的set集合不能做sdiff等算法操作

\*

\* @param key set的key

\* @param members set的元素

\* @return 返回写入set元素的数量

\* @throws PlatformException PlatformException

\*/

long saddPipeline(String key, List<Set<String>> members) throws PlatformException;

/\*\*

\* 遍历set集合获取符合match表达式的元素

\*

\* @param key set集合的key

\* @param stepSize 遍历步长

\* @param match 匹配表达式

\* @return 所有符合match表达式的元素

\* @throws PlatformException PlatformException

\*/

<T> Set<T> sscan(String key, int stepSize, String match) throws PlatformException;

/\*\*

\* 获取指定set集合的差集

\* 该接口使用于单节点redis模式，不适用哨兵、cluster模式

\*

\* @param key set集合的key

\* @return 返回差集

\* @throws PlatformException PlatformException

\*/

Set<String> sdiff(String... keys) throws PlatformException;

/\*\*

\* 获取指定set集合的差集,并将差集写入指定的集合

\* 该接口使用于单节点redis模式，不适用哨兵、cluster模式

\*

\* @param destinationKey 指定差集写入的集合

\* @param key set集合的key

\* @return 返回差集的元素数量

\* @throws PlatformException PlatformException

\*/

long sdiffStore(String destinationKey, String... key) throws PlatformException;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* List \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* 获取list列表中下标为index的元素

\*

\* @param <T> 泛型

\* @param key 缓存的key

\* @param index list列表的下标

\* @return 集合中下标为index的元素，如果下标越界，返回null

\* @throws PlatformException 缓存异常

\*/

<T> T lindex(String key, long index) throws PlatformException;

/\*\*

\* 获取list列表的长度

\*

\* @param key 缓存的key

\* @return 如果key不存在，返回0，如果key不是list，抛出异常

\* @throws PlatformException 缓存异常

\*/

long llen(String key) throws PlatformException;

/\*\*

\* 将一个或多个String类型的value插入list的表头

\*

\* @param key 如果key不存在，一个空列表会被创建并执行lpush操作

\* @param values 写入list列表的元素

\* @return 返回列表长度, 当key存在，但不是一个list，抛出异常

\* @throws PlatformException 缓存异常

\*/

long lpush(String key, Object... values) throws PlatformException;

/\*\*

\* 将一个或多个value插入list的表尾

\*

\* @param key 如果key不存在，一个空列表会被创建并执行rpush操作

\* @param values 写入list列表的元素

\* @return 当key存在，但不是一个list，抛出异常

\* @throws PlatformException 缓存异常

\*/

long rpush(String key, Object... values) throws PlatformException;

/\*\*

\* 移除并返回列表(list)的头元素

\*

\* @param <T> 泛型

\* @param key 缓存的key

\* @return 移除的list列表的元素

\* @throws PlatformException 缓存异常

\*/

<T> T lpop(String key) throws PlatformException;

/\*\*

\* 移除并返回列表(list)的尾元素

\*

\* @param <T> 泛型

\* @param key 缓存的key

\* @return 移除的list列表的元素

\* @throws PlatformException 缓存异常

\*/

<T> T rpop(String key) throws PlatformException;

/\*\*

\* 阻塞列表的移除原语。从给的列表参数中按顺序检查第一个不空的列表，然后从该列表的尾部移除元素。 BRPOP 是 RPOP 的阻塞版本，因为当没有元素从给定的列表中移除的时候，BRPOP 阻塞连接。

\*

\* @param key

\* @param timeout

\* @return

\*/

<T> List<T> brpop(String key, int timeout);

/\*\*

\* 获取list中指定区间内的元素

\*

\* @param <T> 泛型

\* @param key 缓存的key

\* @param start list列表起始下标

\* @param stop list列表结束下标

\* @return list中指定区间内的元素

\* @throws PlatformException 缓存异常

\*/

<T> List<T> lrange(String key, int start, int stop) throws PlatformException;

/\*\*

\* 获取list中所有的元素

\*

\* @param <T> 泛型

\* @param key 缓存的key

\* @return list中所有的元素

\* @throws PlatformException 缓存异常

\*/

<T> List<T> lrange(String key) throws PlatformException;

/\*\*

\* 命令从列表中取出最后一个元素，并插入到另外一个列表的头部； 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。

\*

\* @param <T>

\* @param key 源list key

\* @param dstkey 目标list key

\* @param timeout 超时时间 毫秒

\* @return

\*/

public <T> T brpoplpush(String key, String dstkey, int timeout);

/\*\*

\* 通过pipeline管道向缓存中存放数据

\* 该接口支持的Redis数据类型为String

\* 入参Map集合中的key-value 映射 Redis 中的key-value,入参Map中的key为java.lang.String 类型,value为java.io.Serializable类型

\*

\* @param data 写入的数据

\* @return 存入缓存的数据数量

\* @throws PlatformException 当入参Map集合长度大于100000时，抛出异常

\*/

long setDataPipeline(Map<String, Object> data) throws PlatformException;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 有序集合(sorted set) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

/\*\*

\* 将一个或多个 member 元素及其 score 值加入到有序集 key 当中。

\* 如果某个 member 已经是有序集的成员，那么更新这个 member 的 score 值，并通过重新插入这个 member 元素，来保证该 member 在正确的位置上。

\*

\* @param key

\* @param score

\* @param member

\* @return

\*/

Long zadd(String key, double score, Object member);

/\*\*

\* 返回有序集的成员个数，当 key 不存在时，返回 0

\*

\* @param key

\* @return

\*/

Long zcard(String key);

/\*\*

\* 返回有序集中，指定区间内的成员，其中成员的按分数值递增(从小到大)来排序，具有相同分数值的成员按字典序(lexicographical order )来排列。

\*

\* @param key

\* @param start

\* @param stop

\* @return

\*/

<T> List<T> zrange(String key, long start, long stop);

/\*\*

\* 返回有序集 key 中，所有 score 值介于 min 和 max 之间(包括等于 min 或 max )的成员。有序集成员按 score 值递增(从小到大)次序排列。

\*

\* @param <T>

\* @param key

\* @param start

\* @param stop

\* @return

\*/

<T> List<T> zrangeByScore(String key, double start, double stop);

/\*\*

\* 用于从有序集合key中删除指定的成员member

\*

\* @param key

\* @param member

\* @return

\*/

long zrem(String key, Object... member);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* lua 脚本 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

/\*\*

\* 执行一个不需要参数的lua脚本

\*

\* @param script lua 脚本文本,该脚本不需要参数

\* 例："return redis.call('set','k\_01','v\_01')" ,该脚本把一个字符串写入缓存中, key为k\_01,value为v\_01,redis数据类型为string

\* @return lua 脚本中的定义的返回值

\* @throws PlatformException 缓存异常

\*/

Object eval(String script) throws PlatformException;

/\*\*

\* 执行一个lua脚本

\*

\* @param script lua脚本文本

\* @param keyCount 脚本中需传入的key的数量

\* @param param 脚本中所需所有参数

\* @return 脚本中定义的返回值

\* @throws PlatformException 缓存异常

\*/

Object eval(String script, int keyCount, String... param) throws PlatformException;

/\*\*

\* 执行一个需要参数的lua脚本

\*

\* @param script lua脚本文本 ,

\* 例："return redis.call('set',KEYS[1],ARGV[1])" ,该脚本把一个字符串写入缓存中,key和value需要传入

\* @param keys 脚本需要的keys

\* @param args 脚本需要的args

\* //例1：把一个字符串写入缓存,key为key\_01,value为value\_01,返回值为lua脚本中定义的redis的set命令的返回值

\* public void evalTest(){

\* String script = "return redis.call('set',KEYS[1],ARGV[1])";

\* List<String> keys = new ArrayList<String>();

\* List<String> args = new ArrayList<String>();

\* keys.add("key\_01");

\* args.add("value\_01");

\* Object result = cacheClient.eval(script, keys, args);

\* }

\* <p>

\* //例2：把多个field-value写入hash(散列)表中,hash(散列)表的key为key\_01,域1为field\_01,域1的值为value\_1,

\* //域2为field\_02,域2的值为value\_2,返回值为redis命令hmset的返回值

\* public void evalTest(){

\* String script = "return redis.call('hmset',KEYS[1],ARGV[1],ARGV[2],ARGV[3],ARGV[4])";

\* List<String> keys = new ArrayList<String>();

\* List<String> args = new ArrayList<String>();

\* keys.add("key\_02");

\* args.add("field\_01");

\* args.add("value\_01");

\* args.add("field\_02");

\* args.add("value\_02");

\* Object result = cacheClient.eval(script, keys, args);

\* }

\* @return lua脚本中定义的返回值

\* @throws PlatformException 缓存异常

\*/

Object eval(String script, List<String> keys, List<String> args) throws PlatformException;

/\*\*

\* 通过给定的sha1校验码,执行已经缓存在服务器中的lua脚本,该lua脚本不需要传递参数

\* 将lua脚本缓存到服务器中使用srciptLoad方法

\* <p>

\* //例：将一个字符串写入缓存中

\* public void evalshaTest(){

\* //编写一个lua脚本,该脚本的功能是将字符串"value\_01"写入"key\_01"中

\* String script = "return redis.call('set','key\_01','value\_01')";

\* //将lua脚本缓存在redis服务器中,得到该lua脚本的摘要值sha1

\* String sha1 = cacheClient.scriptLoad(script);

\* //执行lua脚本

\* Object result = cacheClient.evalsha(sha1);

\* }

\*

\* @param scriptSha lua脚本在redis中的sha1摘要值

\* @return 返回lua脚本中定义的返回值

\* @throws PlatformException 缓存异常

\*/

Object evalsha(String scriptSha) throws PlatformException;

/\*\*

\* 通过给定的sha1校验码,执行已经缓存在服务器中的lua脚本

\*

\* @param scriptSha lua脚本在redis中的sha1摘要值

\* @param keyCount 脚本中需要传入的key的数量

\* @param param 脚本中所需要的所有参数

\* @return 脚本中定义的返回值

\* @throws PlatformException 缓存异常

\*/

Object evalsha(String scriptSha, int keyCount, String... param) throws PlatformException;

/\*\*

\* 通过给定的sha1校验码,执行已经缓存在服务器中的lua脚本.这个命令的其他地方如参数传递等与eval命令一致

\* 将lua脚本缓存到服务器中使用srciptLoad方法

\* //例1：把多个字符串写入set集合中,key为key\_set\_01,返回值为lua脚本中定义的redis的sadd命令的返回值

\* public void evalshaTest(){

\* //编写lua脚本

\* String script = "return redis.call('sadd',KEYS[1],ARGV[1],ARGV[2],ARGV[3])";

\* //将lua脚本缓存到服务器中,返回一个lua脚本的摘要值sha1,以后每次执行该脚本只需传递该摘要值sha1和参数即可,不必每次都传递整个lua脚本，

\* //适用于lua脚本比较大的情况下

\* String sha1 = cacheClient.scriptLoad(script);

\* List<String> keys = new ArrayList<String>();

\* List<String> args = new ArrayList<String>();

\* keys.add("key\_set\_01");

\* args.add("value\_01");

\* args.add("value\_02");

\* args.add("value\_03");

\* Object result = cacheClient.evalsha(sha1, keys, args);

\* //再次将多个值追写入该set集合中

\* args.clear();

\* args.add("value\_04");

\* args.add("value\_05");

\* args.add("value\_06");

\* result = cacheClient.evalsha(sha1, keys, args);

\* }

\*

\* @param scriptSha lua脚本在redis中的sha1摘要值

\* @param keys lua脚本中所需的keys

\* @param args lua脚本中所需的args

\* @return 返回执行的lua脚本中定义的返回值

\* @throws PlatformException PlatformException

\*/

Object evalsha(String scriptSha, List<String> keys, List<String> args) throws PlatformException;

/\*\*

\* 将一个lua脚本缓存到redis服务器中

\* 例：

\* public void scripLoadTest(){

\* //编写lua脚本

\* String script = "return redis.call('incr','key\_int')";

\* String sha1 = cacheClient.scriptLoad(script);

\* }

\*

\* @param script lua脚本文本

\* @return lua脚本在redis中的sha1摘要值

\* @throws PlatformException 缓存异常

\*/

String scriptLoad(String script) throws PlatformException;

/\*\*

\* 将文件中的lua脚本缓存到redis服务器中

\* <p>

\* 例：

\* <div>

\* public void scriptLoadFromFileTest(){

\* //首先编写一个lua脚本文件

\* String path = "D:\\temp\\test.lua";

\* String sha1 = cacheClient.scriptLoad(path);

\* }

\* </div>

\*

\* @param path lua脚本文件的路径

\* @return lua脚本在redis中的sha1摘要值

\* @throws PlatformException 缓存异常

\*/

String scriptLoadFromFile(String path) throws PlatformException;

/\*\*

\* 判断一个lua脚本的sha1校验码是否存在

\*

\* @param sha1 lua脚本的sha1摘要值

\* @return 是否存在

\* @throws PlatformException 缓存异常

\*/

boolean scriptExists(String sha1) throws PlatformException;

/\*\*

\* 释放lua脚本所占用的内存

\*

\* @return 是否释放成功

\* @throws PlatformException 缓存异常

\*/

boolean scriptFlush() throws PlatformException;

/\*\*

\* 杀死当前正在执行的lua脚本,此方法只能kill如下情况下的lua脚本

\* 1、当正在执行的lua脚本中无写入redis的操作时

\* 2、当正在执行的lua脚本中有写入redis的操作，但尚未执行时

\* 注意:正在执行的lua脚本中执行过写入redis的操作,该脚本无法kill.

\*

\* @return 执行结果

\* @throws PlatformException 缓存异常

\*/

String scriptKill() throws PlatformException;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Redis 发布订阅命令 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

/\*\*

\* redis 发布

\*

\* @param channel 主题名

\* @param message 消息

\* @return Long

\*/

Long publish(String channel, String message);

/\*\*

\* redis 发布

\*

\* @param channel 主题名

\* @param message 消息

\* @return Long

\*/

Long publish(byte[] channel, byte[] message);

/\*\*

\* redis 订阅

\*

\* @param jps 订阅类型

\* @param channels 主题名

\*/

void subscribe(Object jps, String... channels);