

Redis

基础知识

数据分类

学习目标：

这一节，我们从 结构化、半结构化、非结构化、小结 四个方面来学习。

简介

常见的数据有以下几种：

- 结构化数据
- 半结构化数据
- 非结构化数据

结构化数据

基础知识

所谓的结构化数据，指的是数据的表现样式有一定的(横竖)结构，一般情况下，这种数据是以二维表的方式来实现数据的存储和逻辑关系的表达。

-- 数据以行为单位，一行数据表示一个实体的信息，每一行数据的属性是相同的。

这些数据在存储的时候，为了实现数据的统一存储，往往对数据存储的格式和长度规范都进行了一定程度的限制，这些数据的具体存储主要是以关系型数据库软件来实现。

结构化数据，是指由二维表结构来逻辑表达和实现的数据，严格地遵循数据格式与长度规范，主要通过关系型数据库进行存储和管理。

结构化数据的存储和排列是很有规律的，所以这些数据在查询或修改等操作的时候非常方便，但是由于数据在存储的时候，有一定的关联关系，所以在数据扩充属性或者收缩属性的时候不太方便 -- 扩展性不好。

样式

ID	姓名	性别	电话	籍贯
1	张三	男	13412345678	河南
2	李四	女	13512345678	北京

半结构化数据

基础知识

所谓的半结构化数据，它其实是结构化数据的一种特殊形式，这些数据无法通过普通的二维表样式来进行存储，它们是通过一些特殊的标记实现数据的逻辑关系和存储样式。有时候，这种数据类型也被称为自描述结构数据。

半结构化数据，通过专用的标记，将相关的数据或者属性信息关联在一起，由于结构标记有用户自己控制，所以这种数据的扩展性非常好。但是内容的更改或其他操作需要通过专用的方式来实现。

半结构化数据的存储一般是以文件的方式来实现的，比较常见的文件样式有：json、XML等。

XML数据

```
<?xml version="1.0" encoding="gb2312"?>
<namelist>
  <name1>
    <ID>01</ID>
    <name>张三</name>
    <sex>男</sex>
    <address>北京市市丰台区五里店</address>
  </name1>
</namelist>
```

数据关系

存储格式是以节点为主，一个节点衍生出另外的子节点

每个节点遵循html的风格，但是里面的标签属性是我们自定义的。

Json数据

```
{
  "status": 200,
  "message": {
    "person": [
      {
        "id": 1,
        "name": "张三",
        "gender": "男",
        "address": {
          "Country": "中国",
          "Province": "北京市",
          "city": "北京市",
          "district": "丰台区",
          "town": "五里店"
        }
      }
    ]
  }
}
```

数据关系

[]中括号代表的是一个数组或列表

{ }大括号代表的是一个数据对象

双引号“”表示的是属性值

冒号：代表的是前后之间的关系，冒号前面是属性的名称，后面是属性的值，

非结构化数据

基础知识

所谓的非结构化数据，其实就是没有固定结构的数据 -- 即结构化数据之外的一切数据。它们常以 图片、视频、音频等 样式存在。对于这类数据，我们一般直接整体进行存储，而且一般存储为二进制的数据格式。

非结构化数据一般有两种生成方式：

人为手工生成 - 文本文件、图片、视频、音频、业务应用程序等。

机器自动生成 - 卫星图形、科学数据、数据监控、传感数据等

一般情况下，非结构化数据存储在非关系数据库中，并使用NoSQL进行查询。工作生活，非结构化数据是越来越多，占比远远的超出结构化数据。



软件实现

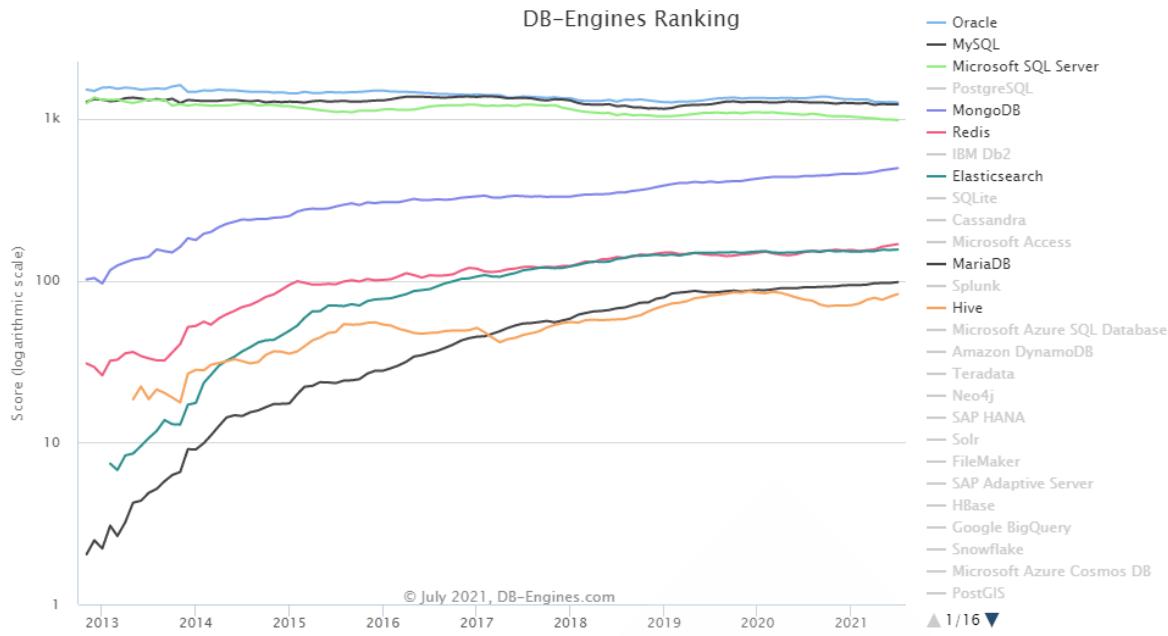
常见的软件实现

373 systems in ranking, July 2021

Rank	DBMS			Database Model	Score		
	Jul 2021	Jun 2021	Jul 2020		Jul 2021	Jun 2021	Jul 2020
1.	1.	1.	Oracle	Relational, Multi-model	1262.66	-8.28	-77.59
2.	2.	2.	MySQL	Relational, Multi-model	1228.38	+0.52	-40.13
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	981.95	-9.12	-77.77
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.15	+8.64	+50.15
5.	5.	5.	MongoDB	Document, Multi-model	496.16	+7.95	+52.68
6.	↑ 7.	↑ 8.	Redis	Key-value, Multi-model	168.31	+3.06	+18.26
7.	↓ 6.	↓ 6.	IBM Db2	Relational, Multi-model	165.15	-1.88	+1.99
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	155.76	+1.05	+4.17
9.	9.	9.	SQLite	Relational	130.20	-0.33	+2.75
10.	↑ 11.	10.	Cassandra	Wide column	114.00	-0.11	-7.08

资料来源: <https://db-engines.com/en/ranking>

数据库欢迎趋势



小结

NoSQL

基础知识

简介

NoSQL: 一类新出现的数据库(not only sql)

特点

泛指非关系型的数据库
不支持SQL语法
存储结构跟传统关系型数据库中的那种关系表完全不同，nosql中存储的数据都是KV形式
NoSQL的世界中没有一种通用的语言，每种nosql数据库都有自己的api和语法，以及擅长的业务场景

相关软件

Redis

简介: 开源的内存结构数据库
官网: <https://redis.io/>
最新版本: 6.2.5

Mongodb

简介: 分布式文档存储数据库，旨在为WEB应用提供可扩展的高性能数据存储解决方案。
官网: <https://www.mongodb.com/>
最新版本: 4.4

CouchDB

简介: 开源的面向文档的数据库管理系统，可以通过 RESTful API 方式访问。
官网: <https://couchdb.apache.org/>
版本: 3.1.1

Hbase hadoop

简介：HBase 是基于 Apache Hadoop 的面向列的 NoSQL 数据库，是 Google 的 BigTable 的开源实现。HBase 是一个针对半结构化数据的场景。

官网：<http://hbase.apache.org/downloads.html>

最新版本：2.4.1

Cassandra hadoop

简介：一套开源分布式 Key-value 存储系统，用于储存特别大的数据。是 Google 的 BigTable 的开源实现。

官网：<https://cassandra.apache.org/>

最新版本：3.11.10

NoSQL VS SQL

概念

SQL (Structured Query Language) 关系型数据库。

主要代表：SQL Server, Oracle, MySQL(开源), PostgreSQL(开源)。

NoSQL (Not Only SQL) 泛指非关系型数据库。

主要代表：MongoDB, Redis, CouchDB。

存储方式

SQL

SQL数据存在特定结构的表中，通常以数据库表形式存储数据。

NoSQL

NoSQL存储方式比较灵活，web场景中，通常以json样式来进行数据的承载。

存储理论

学习目标：

这一节，我们从 ACID、BASE、CAP、小结 四个方面来学习。

ACID

对于一个关系型数据库来说，有一个非常重要的基本属性：ACID

ACID，是指数据库管理系统（DBMS）在写入或更新资料的过程中，为保证事务（transaction）是正确可靠的，所必须具备的四个特性。

注意：

事务是由一些列对系统数据进行访问或者更新操作组成的一个程序执行单元，狭义的事务指的是数据库事务，这里主要来说分布式场景的事务

特性	解释	备注
原子性 (atomicity)	一个事务 (transaction) 中的所有操作 , 要么全部完成 , 要么全部不完成 , 不会结束在中间某个环节。	
一致性 (consistency)	在事务开始之前和事务结束以后 , 数据库的完整性没有被破坏 , 侧重于整体	
隔离性 (isolation)	数据库允许多个并发事务同时对其数据进行读写和修改的能力 , 隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。	
持久性 (durability)	事务处理结束后 , 对数据的修改就是永久的 , 即便系统故障也不会丢失。	

CAP

简介

对于数据库来说 , 因为受到主机资源、容量配置等限制 , 导致我们无法用一个数据库、一台主机来存储所有的数据 , 所以在实际的工作中 , 我们的所有信息都是分散的存储在不同的主机上 , 这就是 -- 分布式数据存储。对于分布式数据存储来说 , 传统的ACID就不太适合了 , 所以针对当前环境的特性就梳理出来一种为事务服务的理论 CAP -- 它是指在一个分布式系统中 , 一致性、可用性、容错性三者不可兼得。

特性	解释	备注
一致性 (Consistency)	更新操作成功后 , 所有节点在同一时间的数据完全一致。	
可用性 (Availability)	用户访问数据时 , 系统是否能在正常响应时间返回结果。	
容错性(Partition Tolerance)	分布式系统在遇到某节点或网络分区故障的时候 , 仍然能够对外提供满足一致性和可用性的服务。	

基于CAP三种特性的两两组合 , 可以将我们之前所说的各种数据库来进行简单的划分归类



CAP理论告诉我们 一个分布式系统不可能同时满足一致性可用性和分区容错性这三个基本需求，最多同时满足这三个当中的两项

一般来说：我们都是在一致性和分区容错性之间寻找所谓的平衡

BASE

简介

BASE 理论是针对 **NOSQL** 数据库而言的，它是对 **CAP** 理论中一致性（C）和可用性（A）进行权衡的结果，源于提出者自己在大规模分布式系统上实践的总结。其核心思想是无法做到强一致性，但每个应用都可以根据自身的特点，采用适当方式达到最终一致性。

BASE理论是由eBay的架构师提出。

特性	解释	备注
基本可用 (Basically Available)	分布式系统在出现不可预知故障时，系统允许损失部分可用性，即保证核心功能或者当前最重要功能可用。 比如说：访问的网页速度稍微降低一些，用户量大的时候，实时限流等	
软状态(Soft-state)	允许系统数据存在中间状态，但不会影响系统的整体可用性，即允许不同节点的副本之间存在暂时的不一致情况。 比如：集群系统的多个节点之间运行xx秒是数据同步延迟。	
最终一致 (Eventually Consistent)	这是三个特点中最重要的，它强调的是系统中所有主机的数据副本，在一段时间同步后，最终能够达到一致状态，而不需要实时保证数据副本一致。	

最终一致性是 **BASE** 原理的核心，也是 **NoSQL** 数据库的主要特点，通过弱化一致性，提高系统的可伸缩性、可靠性和可用性。而且对于大多数 **web** 应用，其实并不需要强一致性，因此牺牲一致性而换取高可用性，是多数分布式数据库产品的方向。

小结

Redis

学习目标：

这一节，我们从 基础知识、特点解析、小结 三个方面来学习。

基础知识

简单介绍

Redis 是 **Remote Dictionary Server**(远程数据服务)的缩写，由意大利人 **antirez(salvatore Sanfilippo)** 开发的一款 内存高速缓存数据库，该软件使用 **C** 语言编写，它的数据模型为 **key-value**。

官方介绍



Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions, and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster. [Learn more →](#)

关键点：

开源、基于内存的数据结构存储、可以作为数据库、缓存、消息代理
提供了 九种+ 的数据结构。

特点解析

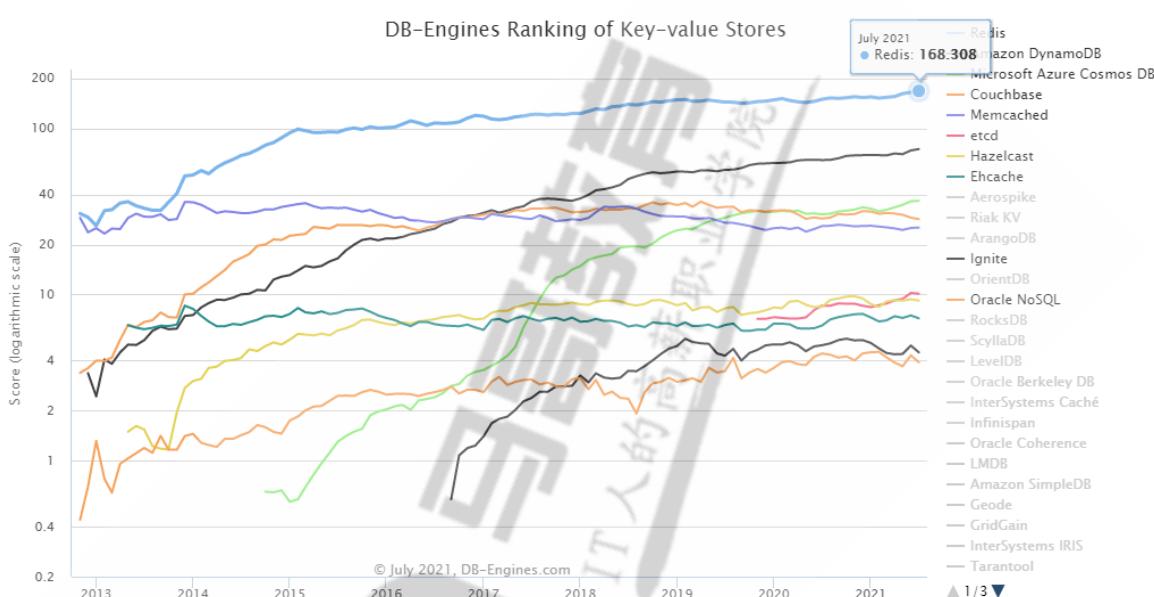
vs其他NoSQL软件

include secondary database models

63 systems in ranking, July 2021

Rank			DBMS	Database Model	Score		
Jul 2021	Jun 2021	Jul 2020			Jul 2021	Jun 2021	Jul 2020
1.	1.	1.	Redis	Key-value, Multi-model	168.31	+3.06	+18.26
2.	2.	2.	Amazon DynamoDB	Multi-model	75.20	+1.43	+10.62
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	36.70	+0.23	+6.30
4.	4.	4.	Memcached	Key-value	25.34	+0.16	-0.50
5.	5.	5.	etcd	Key-value	10.10	-0.12	+1.53
6.	6.	6.	Hazelcast	Key-value, Multi-model	9.20	-0.18	+0.75
7.	7.	↑ 8.	Ehcache	Key-value	7.20	-0.29	+0.69
8.	8.	↓ 7.	Aerospike	Key-value, Multi-model	5.30	-0.48	-1.52
9.	9.	↑ 10.	Riak KV	Key-value	5.10	-0.30	-0.31
10.	↑ 11.	↓ 9.	ArangoDB	Multi-model	4.73	-0.18	-1.12

资料来源: <https://db-engines.com/en/ranking/key-value+store/all>



技术特点

还用说么？你学不学

应用场景

我们直接从几种数据本身的应用特性来描述一下该软件的应用场景：

Sort Set (有序集合)

有序集合在普通集合的基础上做了分数比较的特性，所以主要用来做一些分类排序等功能

比如：排行榜应用，取 `top n` 操作

List (列表)

列表本身具有排序、切片等特性，因为redis的基于内存的分布式特性，它主要来做一些数据筛选、排序等功能

比如：获得最新 N 个数据 或 某个分类的最新数据等

String (字符串)

字符串的其实就是数据的临时存储，借助于redis的内存特性，主要做一些共享之类的功能。

比如：计数器应用、会话共享、购物车状态数据等

Set (集合)

集合主要是数据的统计，由于数据本身具有权重的特性，所以判断数据是否存在特性要比list好很多。

比如：获得共同数据、安全防御的ip判断、社交好友等

一句话：

只要你有丰富的想象力，redis你想着么用就怎么用。

小结

环境部署

学习目标：

这一节，我们从基础知识、简单实践、小结三个方面来学习。

基础知识

准备工作

```
add-apt-repository ppa:redislabs/redis  
apt-get update
```

快速安装

```
# 查看软件版本  
apt info redis  
apt info redis-server  
  
# 安装软件  
apt install redis  
注意：会自动安装 redis-server、redis-tools 依赖软件  
  
# 服务管理  
systemctl stop redis  
systemctl disable redis  
systemctl start redis
```

手工安装

```
# 下载软件  
mkdir /data/softs && cd /data/softs  
wget https://download.redis.io/releases/redis-6.2.5.tar.gz  
  
# 解压文件  
tar xf redis-6.2.5.tar.gz  
cd redis-6.2.5/
```

```
# 确认安装效果
grep 'make PREF' -B 2 README.md
% make install

You can use `make PREFIX=/some/other/directory install` if you wish to use a

# 编辑安装
make PREFIX=/data/server/redis install

# 查看效果
# tree /data/server/redis/
/data/server/redis/
└── bin
    ├── redis-benchmark
    ├── redis-check-aof -> redis-server
    ├── redis-check-rdb -> redis-server
    ├── redis-cli
    ├── redis-sentinel -> redis-server
    └── redis-server

1 directory, 6 files

# 配置环境变量
echo 'PATH=/data/server/redis/bin:$PATH' > /etc/profile.d/redis.sh
source /etc/profile.d/redis.sh

# 创建基本目录
mkdir /data/server/redis/{etc,log,data,run} -p
cp redis.conf /data/server/redis/etc/
```

启动命令

```
root@python-auto:~# redis-server --help
Usage: ./redis-server [/path/to/redis.conf] [options] [-]
  ./redis-server - (read config from stdin)
  ./redis-server -v or --version
  ./redis-server -h or --help
  ./redis-server --test-memory <megabytes>

Examples:
  ./redis-server (run the server with default conf)
  ./redis-server /etc/redis/6379.conf
  ./redis-server --port 7777
  ./redis-server --port 7777 --replicaof 127.0.0.1 8888
  ./redis-server /etc/myredis.conf --loglevel verbose -
  ./redis-server /etc/myredis.conf --loglevel verbose

Sentinel mode:
  ./redis-server /etc/sentinel.conf --sentinel
```

前台启动redis

```
redis-server /data/server/redis/etc/redis.conf
```

```
root@python-auto:~# redis-server /data/server/redis/etc/redis.conf
99516:C 28 Jul 2021 11:19:04.035 # o000o000o000 Redis is starting o000o000o000
99516:C 28 Jul 2021 11:19:04.035 # Redis version=6.2.5, bits=64, commit=00000000, modified=0, pid=99516, just started
99516:C 28 Jul 2021 11:19:04.035 # Configuration loaded
99516:M 28 Jul 2021 11:19:04.035 * Increased maximum number of open files to 10032 (it was originally set to 1024).
99516:M 28 Jul 2021 11:19:04.035 * monotonic clock: POSIX clock_gettime

                               Redis 6.2.5 (00000000/0) 64 bit
                               Running in standalone mode
                               Port: 6379
                               PID: 99516

                               https://redis.io

99516:M 28 Jul 2021 11:19:04.036 # Server initialized
99516:M 28 Jul 2021 11:19:04.036 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' f
```

检查效果

```
root@python-auto:~# netstat -tnulp | grep redis
tcp        0      0 127.0.0.1:6379          0.0.0.0:*
              LISTEN      99516/redis-server
tcp6       0      0 ::1:6379            :::*
              LISTEN      99516/redis-server
root@python-auto:~#
```

redis可以在同一个配置文件启动多个程序

```
# 启动多个实例
redis-server /data/server/redis/etc/redis.conf --port 6666
redis-server /data/server/redis/etc/redis.conf --port 7777
redis-server /data/server/redis/etc/redis.conf --port 8888
redis-server /data/server/redis/etc/redis.conf --port 9999

# 查看效果
netstat -tnulp | grep redis
```

```
root@python-auto:~# netstat -tnulp | grep redis
tcp        0      0 127.0.0.1:7777          0.0.0.0:*
              LISTEN      113469/redis-server
tcp        0      0 127.0.0.1:6666          0.0.0.0:*
              LISTEN      113321/redis-server
tcp        0      0 127.0.0.1:9999          0.0.0.0:*
              LISTEN      110726/redis-server
tcp        0      0 127.0.0.1:8888          0.0.0.0:*
              LISTEN      109870/redis-server
tcp6       0      0 ::1:7777            :::*
              LISTEN      113469/redis-server
tcp6       0      0 ::1:6666            :::*
              LISTEN      113321/redis-server
tcp6       0      0 ::1:9999            :::*
              LISTEN      110726/redis-server
tcp6       0      0 ::1:8888            :::*
              LISTEN      109870/redis-server
root@python-auto:~#
```

后台启动

```
# 定制redis配置文件
root@python-auto:~# vim /data/server/redis/etc/redis.conf
# daemonize no    将redis的启动设定为后台启动
daemonize yes

# 启动redis服务
/data/server/redis/bin/redis-server /data/server/redis/etc/redis.conf

# 查看效果
netstat -tnulp | grep redis
```

```
# 关闭服务  
redis-cli shutdown  
redis-cli -h 127.0.0.1 -p 6666  
kill -9 $(lsof -Pti :6379)
```

简单实践

查看配置

```
root@python-auto:~# dpkg -L redis-server  
/.  
...  
/etc/redis  
/etc/redis/redis.conf  
/lib  
/lib/systemd  
/lib/systemd/system  
/lib/systemd/system/redis-server.service  
...  
配置文件  
服务启动文件
```

查看配置

```
root@python-auto:/etc/redis# grep -Env '#|^$' redis.conf  
75:bind 127.0.0.1 -::1  
94:protected-mode yes  
98:port 6379  
107:tcp-backlog 511  
119:timeout 0  
136:tcp-keepalive 300  
257:daemonize yes  
275:supervised auto  
289:pidfile /run/redis/redis-server.pid  
297:loglevel notice  
302:logfile /var/log/redis/redis-server.log  
327:databases 16  
336:always-show-logo no  
341:set-proc-title yes  
358:proc-title-template "{title} {listen-addr} {server-mode}"  
398:stop-writes-on-bgsave-error yes  
404:rdbcompression yes  
413:rdbchecksum yes  
431:dbfilename dump.rdb  
444:rdb-del-sync-files no  
454:dir /var/lib/redis  
...  
绑定地址  
暴露端口  
连接队列  
后台启动  
默认16个数据库  
数据文件名称  
数据文件所在目录
```

端口效果

```
root@python-auto:/etc/redis# netstat -tnulp | grep redis  
tcp      0      0 127.0.0.1:6379          0.0.0.0:*          LISTEN      11020/redis-server  
tcp6     0      0 ::1:6379              :::*          LISTEN      11020/redis-server  
root@python-auto:/etc/redis#
```

操作命令

```
# 连接数据库  
root@python-auto:/etc/redis# redis-cli  
127.0.0.1:6379>  
  
root@python-auto:/etc/redis# redis-cli -h localhost -p 6379  
localhost:6379>  
  
root@python-auto:~# redis-cli --raw  
127.0.0.1:6379>  
  
# 测试效果  
127.0.0.1:6379> ping  
PONG  
  
# 退出效果  
127.0.0.1:6379[5]> quit
```

小结

基本命令

基本操作

学习目标：

这一节，我们从基础知识、简单实践、小结三个方面来学习。

基础知识

帮助信息

```
127.0.0.1:6379> help  
redis-cli 6.2.5  
To get help about Redis commands type:  
    "help @<group>" to get a list of commands in <group>  
    "help <command>" for help on <command>  
    "help <tab>" to get a list of possible help topics  
    "quit" to exit  
  
To set redis-cli preferences:  
    ":set hints" enable online hints  
    ":set nohints" disable online hints  
Set your preferences in ~/.redisclirc
```

注意：

历史操作命令在 `~/.rediscli_history` 文件中

命令解析

redis将大量的命令进行了简单的分组操作，对于6.2.5来说，他有 15个命令组

<code>@generic</code>	通用的命令组
<code>@string</code>	字符相关命令组

@list	列表相关命令组
@set	集合相关命令组
@sorted_set	有序集合相关命令组
@hash	hash相关命令组
@pubsub	发布订阅相关命令组
@transactions	事务相关命令组
@connection	连接相关命令组
@server	服务相关命令组
@scripting	脚本相关命令组
@hyperloglog	超级日志相关命令组
@cluster	集群相关命令组
@geo	基因类数据相关命令组
@stream	流数据相关命令组

帮助操作

```
查看命令组  
127.0.0.1:6379> help @generic  
  
查看命令帮助  
127.0.0.1:6379> help ECHO  
  
ECHO message  
summary: Echo the given string  
since: 1.0.0  
group: connection
```

简单实践

基本操作

```
# 选择数据库  
127.0.0.1:6379> select 5  
OK  
127.0.0.1:6379[5]>  
  
# 查看所有属性信息  
127.0.0.1:6379[5]> info  
...  
  
# 查看部分属性信息  
127.0.0.1:6379[5]> info cpu  
# CPU  
used_cpu_sys:1.222606  
used_cpu_user:0.905046  
used_cpu_sys_children:0.000000  
used_cpu_user_children:0.000000  
used_cpu_sys_main_thread:1.221889  
used_cpu_user_main_thread:0.904515  
  
# 获取配置属性  
127.0.0.1:6379[5]> CONFIG GET bind  
1) "bind"  
2) "127.0.0.1 -::1"
```

统用操作

```
# 获取所有的key信息
127.0.0.1:6379> help KEYS

KEYS pattern
summary: Find all keys matching the given pattern
since: 1.0.0
group: generic

# 判断一个key是否存在
127.0.0.1:6379> help EXISTS

EXISTS key [key ...]
summary: Determine if a key exists
since: 1.0.0
group: generic

# 设置一个key
127.0.0.1:6379> help set

SET key value [EX seconds|PX milliseconds|EXAT timestamp|PXAT milliseconds-
timestamp|KEEPTTL] [NX|XX] [GET]
summary: Set the string value of a key
since: 1.0.0
group: string

# 获取一个key
127.0.0.1:6379> help get

GET key
summary: Get the value of a key
since: 1.0.0
group: string

# 删除一个key
127.0.0.1:6379> help DEL

DEL key [key ...]
summary: Delete a key
since: 1.0.0
group: generic

# 查看key的类型
127.0.0.1:6379> help TYPE

TYPE key
summary: Determine the type stored at key
since: 1.0.0
group: generic

# 设置一个有过期期限的key
127.0.0.1:6379> help EXPIRE

EXPIRE key seconds
summary: Set a key's time to live in seconds
```

```
since: 1.0.0
group: generic

# 查看一个key的有效时间
127.0.0.1:6379> help TTL

TTL key
summary: Get the time to live for a key
since: 1.0.0
group: generic

# 删除当前库的所有key
127.0.0.1:6379> help FLUSHDB

FLUSHDB [ASYNC|SYNC]
summary: Remove all keys from the current database
since: 1.0.0
group: server

# 删除当前数据库所有的数据
127.0.0.1:6379> help FLUSHALL

FLUSHALL [ASYNC|SYNC]
summary: Remove all keys from all databases
since: 1.0.0
group: server
```

String

学习目标：

这一节，我们从基础知识、简单实践、小结两个方面来学习。

基础知识

简介

string类型是实战中应用最多的数据类型，可以用于各种其他类型数据的值的存储，非常方便

简单实践

设定key

```
设定一个普通的key
set key value

设定一个有过期时间的key
setex key seconds value

同时设定多个值
mset key1 value1 key2 value2 ...
```

获取key

获取一个key

```
get key
```

获取多个key

```
mget key1 key2 ...
```

删除key

删除一个key

```
del key1
```

删除多个key

```
del key1 key2 ...
```

List

学习目标：

这一节，我们从 基础知识、简单实践、小结 两个方面来学习。

基础知识

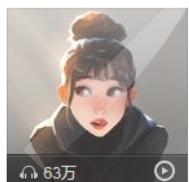
list 是一个string类型的 列表，redis的每个list都可以存储 $2^{32}-1$ 个元素，列表的元素方便排序、获取、统计等相关操作。

各种各样的列表场景都可以

全部

选择分类 ▾

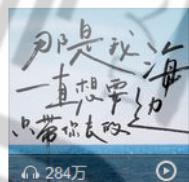
热门



晨间空腹跳绳40min...
by SuperDan1013



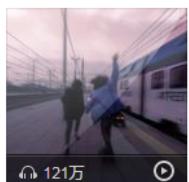
控球 节奏 打球时听...
by 琴师浊酒



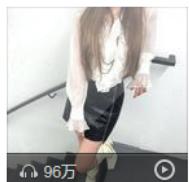
摇滚夏日祭° | 少年...
by 十三逆旅Cobain



酸爵士 | 迷幻骑士绝...
by 点燃煌



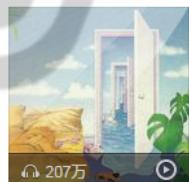
很chill的R&B歌单 ...
by alwaysbeROMA...



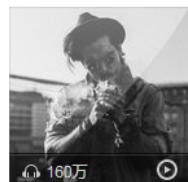
96万



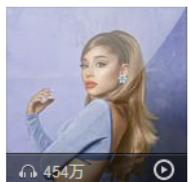
116万



207万



160万



454万

简单实践

设定key

左侧添加数据

```
lpush key value1 value2
```

右侧添加数据

```
rpush key value1 value2
```

插入指定元素

```
linsert key before|after 现有元素 新元素
```

获取key

获取列表数据

lrange key start stop

注意: **start** 是从 0开始、**stop**如果为 -1的话, 代表最后一个。

获取**key**指定位置的值

LINDEX key index

获取**key**列表的值的数量

LLEN key

删除key

从**key**中删除指定的**value**

lrem key count value

注意:

count > 0: 从头往尾移除指定数量个 **value**

count < 0: 从尾往头移除指定数量个 **value**

count = 0: 移除所有的 **value**

从**key**的左侧删除指定个数的 **value**

LPOP key [count]

从**key**的右侧删除指定个数的 **value**

RPOP key [count]

保留范围数据, 范围之外的都删除

LTRIM key 起始索引 结束索引

set

学习目标 :

这一节 , 我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

简介

set 是一个**string**类型的 集合, **redis**的每个**list**都可以存储 $2^{32} - 1$ 个元素, 集合元素无序且不重复, 可以进行各种排序统计场景。

今日打擂歌曲

换一批

所有歌曲 (29)



约定 |

打擂3596人 / 评委4.9万人 >

打擂



我的梦 |

打擂595人 / 评委8766人 >

打擂



体面 |

打擂4275人 / 评委5.5万人 >

打擂



再度重相逢 |

打擂989人 / 评委1.3万人 >

打擂



相思风雨中 |

打擂3970人 / 评委5.4万人 >

打擂

打擂歌单预告 >

昨日打擂金曲

更多 >



做评委
1109406人



光辉岁月



站着等你三千年



最远的你是我最近的爱

无序集合

设定key

添加数据

SADD key member [member ...]

注意：

因为是无序的，所以查看的时候，没有顺序

如果key中已经存在 member，那么不会重复增加

合并多个key

SUNION key [key ...]

将多个key的内容合并在一起，相同的member只会存在一个

获取key

获取set数据

SMEMBERS key

获取set中的member个数

SCARD key

获取多个key相同的内容 -- 取交集

SINTER key [key ...]

获取多个key不相同的内容 -- 取差集

SDIFF key [key ...]

删除key

从key中删除指定的member

SREM key member [member ...]

从key的随机删除指定个数的 member

SPOP key [count]

Sort set

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

简介

set 是一个string类型的 集合，redis的每个list都可以存储 $2^{32} - 1$ 个元素，集合元素无序且不重复，可以进行各种排序统计场景。

云音乐特色榜

- 飙升榜** 刚刚更新
- 新歌榜** 刚刚更新
- 原创榜** 每周四更新
- 热歌榜** 刚刚更新

全球媒体榜

- 黑胶VIP爱听榜** 每周四更新
- 云音乐说唱榜** 每周五更新

飙升榜

最近更新：07月28日 (刚刚更新)

播放 + (3800802) (10749) 下载 (206977)

歌曲列表 99首歌 播放：4428655616次

标题	时长	歌手
1 NEW How You Like That (JP Ver.)	03:01	
2 NEW 你就是我的唯一		+ 回 正

简单实践

设定key

添加数据

```
ZADD key score member [score member ...]
```

注意：

因为每个member有score，所以查看的时候，会按照score的值进行排序
如果key中已经存在 member，那么不会重复增加

获取key

获取有序集合数据

```
ZRANGE key min max [REV]
```

注意：

min 是从 0开始、max如果为 -1的话，代表最后一个。

rev 代表反序

获取有序集合中的指定分数范围的元素

```
ZRANGEBYSCORE key min max
```

获取有序集合元素的权重

```
ZSCORE key member
```

获取有序集合元素个数

```
ZCARD key
```

删除key

从key中删除指定的member

```
ZREM key member [member ...]
```

从key的随机删除指定个数的 member

```
ZREMRANGEBYSCORE key min max
```

小结

Hash

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

适用场景

hash 是一个string类型的 字段和值 的关联表，redis的每个hash都可以存储 $2^{32}-1$ 个键值对，非常适合于存储对象场景



简单实践

设定key

添加数据

HSET key field value [field value ...]

注意：在实践的时候、**hset** 也可以实现添加多个数据对的效果

添加多个数据

HMSET key field value [field value ...]

获取key

获取所有属性

HKEYS key

获取多个属性的值

HMGET key field [field ...]

删除key

从key中删除指定的value

HDEL key field [field ...]

直接删除key所有的内容

del key

python实践

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

简介

我们在后面是准备在python web项目中应用redis，所以我们需要在python虚拟环境中安装redis的模块插件，然后才可以正常的应用。

redis-py提供两个类**Redis**和**StrictRedis**用于实现**Redis**的命令，**StrictRedis**用于实现大部分官方的命令，并使用官方的语法和命令，**Redis**是**StrictRedis**的子类，用于向后兼容旧版本的**redis-py**。

模块安装

```
pip install redispy
```

模块操作

```

# 导入模块
import redis

# 方法1
r = redis.Redis(host='127.0.0.1', port=6379, db=2)
# 方法2
r = redis.StrictRedis(host='127.0.0.1', port=6379, db=2)

redis-py使用connection pool来管理对一个redis server的所有连接，避免每次建立、释放连接的开销。默认，每个Redis实例都会维护一个自己的连接池。当然，我们还可以直接建立一个连接池，然后作为参数Redis，这样就可以实现多个Redis实例共享一个连接池
# 方法3
pool = redis.ConnectionPool(host='127.0.0.1', port=6379)
r = redis.Redis(connection_pool=pool)

```

简单操作 - 以String为例

```

# 单值实践
r.set('key', 'value', ex=5)
或
r.setex("key1", 5, "value1")

r.get('key')

# 多值实践
r.mset(k1="v1", k2="v2")
r.mset({'k3':"v3", 'k4':"v4"})

r.mget('k1', 'k2')
r.mget(['k3', 'k4'])

# 自增自减
r.set('num', 4)
r.get('num')
r.incr('num')
r.incr('num', 6)
r.incrby('num', 6)
r.decr('num')
r.decr('num', 3)    注意：没有decrby

# 删除操作
r.delete('num')

# 判断存在
r.exists('num')

# 模糊匹配
r.keys()
r.keys('k*')
r.keys('*2')

# 查询数据量
r.dbsize()

```

简单实践

需求

对于各种web框架来说，只要涉及到redis，基本上都提供了相关的 属性配置，我们这里以简单的 Flask web框架为例。

安装模块

```
pip install Flask  
pip install flask-session
```

代码效果

```
]# cat flask_redis.py  
from flask import Flask, session  
from flask_session import Session  
import redis  
  
app = Flask(__name__)  
app.debug = True  
app.secret_key = 'xxxx'  
  
app.config['SESSION_TYPE'] = 'redis'  
app.config['SESSION_PERMANENT'] = True  
app.config['SESSION_USE_SIGNER'] = False  
app.config['SESSION_KEY_PREFIX'] = 'session:'  
app.config['SESSION_REDIS'] = redis.Redis(host='127.0.0.1', port='6379', db=4)  
  
Session(app)  
  
@app.route('/')  
def index():  
    return 'session storage to redis'  
  
@app.route('/set_name/')  
def set():  
    session['user_name'] = "zhangsan"  
    return 'ok'  
  
@app.route('/get_name/')  
def get():  
    return session.get('user_name', 'not set')  
  
@app.route('/pop_name/')  
def pop():  
    session.pop('user_name')  
    return session.get('user_name', 'pop key')  
  
@app.route('/clear_name/')  
def clear():
```

```
session.clear()
return session.get('user_name', 'clear key')

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

启动flask

```
]# python flask_redis.py
* Serving Flask app "flask_redis" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production
  deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 170-146-674
...
```

浏览器刷新 <http://0.0.0.0:5000/index> , 在memcache中查看效果

```
127.0.0.1:6379> select 4
OK
127.0.0.1:6379[4]> keys *
1) "session:305a95ff-8e39-4bb3-ba0a-2a2915f74436"
127.0.0.1:6379[4]> get session:305a95ff-8e39-4bb3-ba0a-2a2915f74436
"\x80\x03}q\x00(\x\n\x00\x00\x00_permanentq\x01\x88X\t\x00\x00\x00user_nameq\x02X
\b\x00\x00\x00zhangsanq\x03u."
```

小结

持久复制

持久化

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

简介

Redis虽然是一个内存级别的缓存程序，但是其可以将内存的数据按照一定的策略保存到硬盘上，从而实现数据持久保存的目的。

目前，**redis**支持两种不同方式的数据持久化保存机制：

RDB

- 基于时间，生成某个时间点的快照文件，默认只保留最近的一次快照。
- 恢复速度非常快，但是可能丢失之前的快照数据，非实时同步。

AOF

- **AppendOnlyFile**(日志追加模式)，基于**Redis**协议格式保存信息到指定日志文件的末尾
- 基于写时复制的机制，每隔x秒将新执行的命令同步到对应的文件中
- 默认是禁用的，需要开启
- 数据保存全，时间过长导致文件过大，恢复时候速度比**RDB**慢。

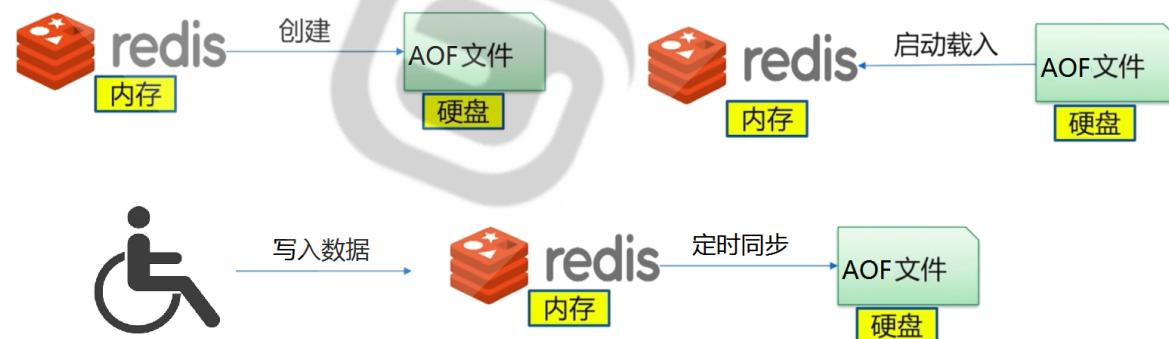
RDB原理



Redis从**master**主进程中创建一个子进程，基于写时复制机制，子进程将内存的数据保存到.rdb文件中，数据保存完毕后，再将上次保存的rdb文件覆盖替换掉，最后关闭子进程。

Redis提供了手工的机制，我们可以执行命令实现文件的保存

AOF原理



RDB实践

RDB配置解析

```
#####
##### SNAPSHOTTING #####
# Save the DB to disk.
#
# save <seconds> <changes>
#
# Redis will save the DB if both the given number of seconds and the given
# number of write operations against the DB occurred.
#
# Snapshotting can be completely disabled with a single empty string argument
# as in following example:
#
# save ""
#
# Unless specified otherwise, by default Redis will save the DB:
#   * After 3600 seconds (an hour) if at least 1 key changed
#   * After 300 seconds (5 minutes) if at least 100 keys changed
#   * After 60 seconds if at least 10000 keys changed
#
# You can set these explicitly by uncommenting the three following lines.
#
# save 3600 1
```

```
root@python-auto:~# grep -Env '#|^$' /etc/redis/redis.conf
...
381:save 3600 1
382:save 300 100
383:save 60 10000
398:stop-writes-on-bgsave-error yes
404:rdbcompression yes
413:rdbchecksum yes
431:dbfilename dump.rdb
444:rdb-del-sync-files no
454:dir /var/lib/redis
```

持久化命令

```
# 数据同步操作，执行时候，会导致其他命令无法执行
127.0.0.1:6379> help SAVE

SAVE -
summary: Synchronously save the dataset to disk
since: 1.0.0
group: server
```

```
# 异步方式后台执行数据的同步，不影响其他命令的执行
127.0.0.1:6379> help BGSAVE
```

```
BGSAVE [SCHEDULE]
summary: Asynchronously save the dataset to disk
since: 1.0.0
group: server
```

简单测试

```
# 执行备份前查看效果
root@python-auto:~# ll -h /var/lib/redis/*.rdb
-rw-rw---- 1 redis redis 268 7月 28 18:31 /var/lib/redis/dump.rdb

# 执行备份
127.0.0.1:6379> bgsave
```

```
Background saving started
```

```
# 执行备份后查看效果  
root@python-auto:~# ll -h /var/lib/redis/*.rdb  
-rw-rw---- 1 redis redis 268 7月 29 09:38 /var/lib/redis/dump.rdb
```

```
# 备份文件  
root@python-auto:/var/lib/redis# cp dump.rdb /tmp
```

```
# 在做一些操作  
127.0.0.1:6379> set xxx xxx  
...
```

```
# 关闭redis  
systemctl stop redis
```

查看效果

```
root@python-auto:~# ll -h /var/lib/redis/*.rdb  
-rw-rw---- 1 redis redis 488 7月 29 09:41 /var/lib/redis/dump.rdb
```

还原配置文件

```
cp /tmp/dump.rdb ./
```

```
启动redis  
systemctl start redis
```

查看效果

```
127.0.0.1:6379> keys *
```

AOF实践

配置解析

```
#####
# APPEND ONLY MODE #####
#####

# By default Redis asynchronously dumps the dataset on disk. This mode is
# good enough in many applications, but an issue with the Redis process or
# a power outage may result into a few minutes of writes lost (depending on
# the configured save points).
#
# The Append Only File is an alternative persistence mode that provides
# much better durability. For instance using the default data fsync policy
# (see later in the config file) Redis can lose just one second of writes in a
# dramatic event like a server power outage, or a single write if something
# wrong with the Redis process itself happens, but the operating system is
# still running correctly.
#
# AOF and RDB persistence can be enabled at the same time without problems.
# If the AOF is enabled on startup Redis will load the AOF, that is the file
# with the better durability guarantees.
#
# Please check https://redis.io/topics/persistence for more information.

appendonly no

# The name of the append only file (default: "appendonly.aof")
appendfilename "appendonly.aof"
```

```
root@python-auto:~# grep -Env '#|^$' /etc/redis/redis.conf
...
1252:appendonly no
1256:appendfilename "appendonly.aof"
1282:appendfsync everysec
1304:no-appendfsync-on-rewrite no
1323:auto-aof-rewrite-percentage 100
1324:auto-aof-rewrite-min-size 64mb
1348:aof-load-truncated yes
1359:aof-use-rdb-preamble yes
...
```

持久化命令

```
# 数据同步操作，执行时候，会导致其他命令无法执行
127.0.0.1:6379> help BGREWRITEAOF

BGREWRITEAOF -
summary: Asynchronously rewrite the append-only file
since: 1.0.0
group: server
```

简单测试

```
# 检查现状
127.0.0.1:6379> CONFIG GET appendonly
1) "appendonly"
2) "no"

root@python-auto:~# ll -h /var/lib/redis/*.rdb
-rw-rw---- 1 redis redis 92 7月 29 10:06 /var/lib/redis/dump.rdb

# 修改持久化模式
127.0.0.1:6379> CONFIG SET appendonly yes
OK

# 确认效果
root@python-auto:~# ll -h /var/lib/redis/
总用量 16K
-rw-rw---- 1 redis redis 92 7月 29 10:09 appendonly.aof
-rw-rw---- 1 redis redis 92 7月 29 10:06 dump.rdb

# 开始备份
127.0.0.1:6379> MSET a1 v1 a2 v2 a3 v3
OK
127.0.0.1:6379> BGREWRITEAOF
Background append only file rewriting started

# 检查效果
root@python-auto:~# ll -h /var/lib/redis/*.aof
-rw-rw---- 1 redis redis 118 7月 29 10:11 /var/lib/redis/appendonly.aof
```

小结

复制

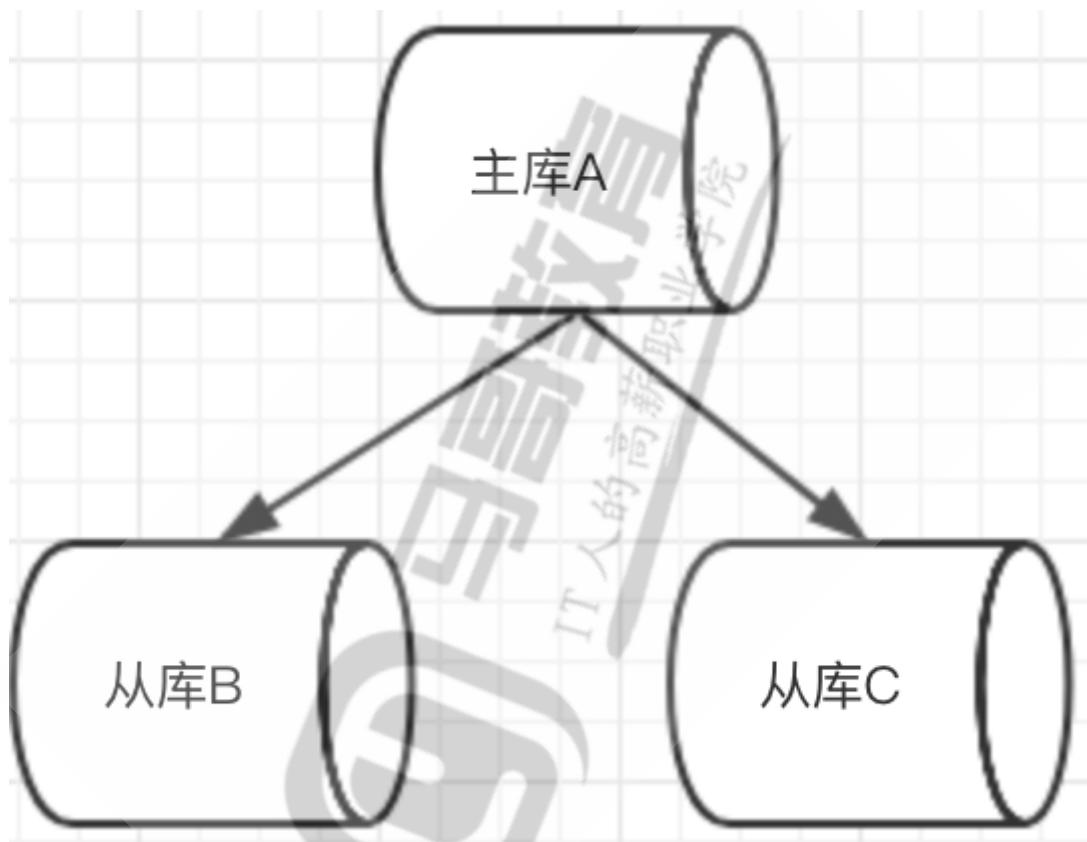
学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

简介

redis 作为一个分布式的数据缓存平台，我们可以借助于redis的多实例机制，让多个实例间的数据，达成一个同步的效果，这样即使某个实例出现异常，也不影响数据整体的使用。



复制特性

- 1 redis 如果想要实现主从复制的效果，我们需要为它划分 主角色和从角色，实现数据 由主向从 的单向传递。
- 2 对于 从redis，一旦发现 主redis 更换了，那么将本地数据清空，从新主上同步数据。
- 3 如果 从redis仅仅是断开了 主redis，那么不会删除已经同步过的数据。

实践要点

- 1 主角色redis 必须开启持久化功能
- 2 从角色redis 指定谁是主，以及自己作为从的唯一标识。
redis4.0之前用 slaveof
redis4.0之后用 replicaof

配置属性

```
#####
##### REPLICATION #####
#
# Master-Replica replication. Use replicaof to make a Redis instance a copy of
# another Redis server. A few things to understand ASAP about Redis replication.
#
# +-----+     +-----+
# |      Master   | --> |      Replica   |
# | (receive writes) |          | (exact copy) |
# +-----+     +-----+
#
# 1) Redis replication is asynchronous, but you can configure a master to
# stop accepting writes if it appears to be not connected with at least
# a given number of replicas.
# 2) Redis replicas are able to perform a partial resynchronization with the
# master if the replication link is lost for a relatively small amount of
# time. You may want to configure the replication backlog size (see the next
# sections of this file) with a sensible value depending on your needs.
# 3) Replication is automatic and does not need user intervention. After a
# network partition replicas automatically try to reconnect to masters
# and resynchronize with them.
#
# replicaof <masterip> <masterport>
```

复制命令

```
127.0.0.1:6379> help SLAVEOF

SLAVEOF host port
summary: Make the server a replica of another instance, or promote it as
master. Deprecated starting with Redis 5. Use REPLICAOF instead.
since: 1.0.0
group: server

127.0.0.1:6379> help REPLICAOF

REPLICAOF host port
summary: Make the server a replica of another instance, or promote it as
master.
since: 5.0.0
group: server
```

注意：

关闭复制关系可以通过 `replicaof no one` 命令

默认情况下，任何一个redis实例启动时候，会自动将自己作为主角色而存在

简单实践

开启一个redis实例

```
redis-server /data/server/redis/etc/redis.conf --port 6666 --daemonize yes
```

连接新实例查看效果

```
# redis-cli -h 127.0.0.1 -p 6666
> info replication
# Replication
role:master
connected_slaves:0
...
```

新实例同步主角色

```
# 设置主角色
127.0.0.1:6666> REPLICATEOF 127.0.0.1 6379
OK

# 查看状态
127.0.0.1:6666> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:3
master_sync_in_progress:0
slave_repl_offset:0
slave_priority:100
...

# 查看同步效果
127.0.0.1:6666> KEYS *
1) "a1"
2) "a3"
3) "a2"
```

结果显示：

数据同步成功

主从校验

```
# 从角色只能查看数据，不能修改数据
127.0.0.1:6666> FLUSHALL
(error) READONLY You can't write against a read only replica.

# 主角色删除数据
127.0.0.1:6379> KEYS *
1) "a2"
2) "a1"
3) "a3"
```

```
127.0.0.1:6379> DEL a1  
(integer) 1
```

```
# 从角色查看效果  
127.0.0.1:6666> keys *  
1) "a3"  
2) "a2"
```

结果显示：

自动同步成功。

同步状态查看

```
# 同步后主角色查看效果
127.0.0.1:6379> info Replication
# Replication
role:master
connected_slaves:1
slave0:ip=127.0.0.1,port=6666,state=online,offset=1038,lag=0
master_failover_state:no-failover
master_replid:571bcaecc7eeb590326fc5a9262df569f3623b36
master_replid2:0000000000000000000000000000000000000000000000000000000000000000
master_repl_offset:1038 # 同步的偏移量
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:1038
```

小结

集群方案

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

哨兵模式

需求

对于主从同步来说：

- 1 我们需要手工进行相关的主从角色的配置，即时我们采用配置文件的方式，也需要通过属性进行角色的关联配置。
- 2 主从角色的架构无法自动实现 主和从 的角色切换，一旦主发送故障，整个集群就崩溃了。

所以我们需要一种能够自动实现主从集群的角色无缝切换，让业务无感知从而不影响业务使用，最好还能够动态扩展redis服务器，从而实现多台服务器实现并行写入以实现更高并发的目的。

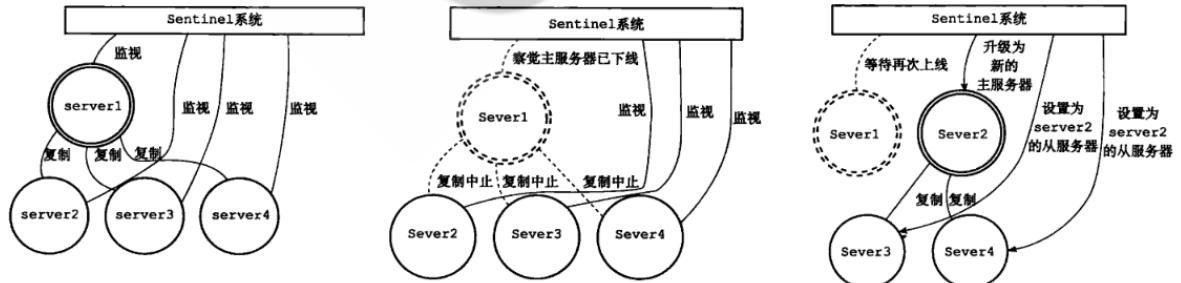
高可用方案

对于redis来说，它提供了一个非常好的高可用性解决方案 -- **Sentinel**(哨兵)：由一个或多个 **Sentinel** 实例 组成的**Sentinel** 系统可以监视任意多个主服务器，以及这些主服务器属下的所有从服务器，并在被监视的主服务器进入下线状态时，自动将下线主服务器属下的某个从服务器升级为新的主服务器。

原则上，我们需要配置 3台+ 的**sentinel**实例，实现哨兵本身的集群效果

- 单**sentinel**节点监控比较容易产生误判
- 三个哨兵本身的配置都是一样的。

哨兵原理



sentinel组件

组件	解析	备注
Monitoring	监控Master节点、Slave节点以及其他Sentinel节点是否正常工作	
Automatic failover	当Master故障时，Sentinel会选择一个Slave节点来替换当前Master，并配置其他的Slave节点成为新的Master节点的从节点。同时Sentinel会通知客户端新的Master的地址。	
Notification	当监控的集群节点出故障时，Sentinel可通过执行特定脚本或者订阅来告知系统管理员或者其他应用程序来通知相应信息。	
Configuration provider	当集群中发生failover时，应用方可从Sentinel获得新的Master节点和Slave节点的地址。	

核心配置

```
# sentinel monitor <master-name> <ip> <redis-port> <quorum>
#
# Tells Sentinel to monitor this master, and to consider it in O_DOWN
# (Objectively Down) state only if at least <quorum> sentinels agree.
#
# Note that whatever is the ODOWN quorum, a Sentinel will require to
# be elected by the majority of the known Sentinels in order to
# start a failover, so no failover can be performed in minority.
#
# Replicas are auto-discovered, so you don't need to specify replicas in
# any way. Sentinel itself will rewrite this configuration file adding
# the replicas using additional configuration options.
# Also note that the configuration file is rewritten when a
# replica is promoted to master.
#
# Note: master name should not include special characters or spaces.
# The valid charset is A-z 0-9 and the three characters "-_".
sentinel monitor mymaster 127.0.0.1 6379 2
```

实践要点

- 1 准备主从环境
- 2 定制sentinel配置

简单实践

配置主从环境

```
# 准备基本环境
mkdir /data/server/{redis6379,redis6380,redis6381} -p

# 创建三个配置
cd /data/softs/redis-6.2.5
cp redis.conf /data/server/redis6379/
cp redis.conf /data/server/redis6380/
cp redis.conf /data/server/redis6381/

# 修改以下内容
# grep -En 'daemonize |pidfile |logfile |dir | bind |port |replicaof'
/data/server/redis6379/redis.conf | grep -v "#"
98:port 6379
```

```

257:daemonize yes
289:pidfile /var/run/redis_6379.pid
302:logfile "/data/server/redis6379/redis.log"
454:dir ./

# grep -En 'daemonize |pidfile |logfile |dir | bind |port |replicaof'
/data/server/redis6380/redis.conf | grep -v "#"
98:port 6380
257:daemonize yes
289:pidfile /var/run/redis_6380.pid
302:logfile "/data/server/redis6380/redis.log"
454:dir ./
477:replicaof 127.0.0.1 6379

# grep -En 'daemonize |pidfile |logfile |dir | bind |port |replicaof'
/data/server/redis6381/redis.conf | grep -v "#"
98:port 6381
257:daemonize yes
289:pidfile /var/run/redis_6381.pid
302:logfile "/data/server/redis6381/redis.log"
454:dir ./
477:replicaof 127.0.0.1 6379

# 启动服务
redis-server /data/server/redis6379/redis.conf
redis-server /data/server/redis6380/redis.conf
redis-server /data/server/redis6381/redis.conf

# 查看集群效果
root@python-auto:~# redis-cli -h 127.0.0.1 -p 6379 info Replication
# Replication
role:master
connected_slaves:2
slave0:ip=127.0.0.1,port=6380,state=online,offset=182,lag=1
slave1:ip=127.0.0.1,port=6381,state=online,offset=182,lag=1
master_failover_state:no-failover
master_replid:04c1ffda4f7befb833e35f598f2c04d93e42136c
master_replid2:0000000000000000000000000000000000000000000000000000000000000000
master_repl_offset:182
second_repl_offset:-1
rep1_backlog_active:1
rep1_backlog_size:1048576
rep1_backlog_first_byte_offset:1
rep1_backlog_histlen:182

```

配置sentinel

```

# 准备基本环境
mkdir /data/server/sentinel{26379,26380,26381} -p
cd /data/softs/redis-6.2.5
cp sentinel.conf /data/server/sentinel26379/
cp sentinel.conf /data/server/sentinel26380/
cp sentinel.conf /data/server/sentinel26381/

# 修改配置
root@python-auto:/data/server/sentinel26379# grep -Env '#|^$' sentinel.conf
21:port 26379

```

```

26:daemonize yes
31:pidfile "/var/run/redis-sentinel-26379.pid"
36:logfile "/data/server/sentinel26379/sentinel.log"
65:dir "/data/server/sentinel26379"
84:sentinel monitor mymaster 127.0.0.1 6379 2
148:acllog-max-len 128
295:sentinel deny-scripts-reconfig yes
332:sentinel resolve-hostnames no
338:sentinel announce-hostnames no

# grep -Env '#|^$' /data/server/sentinel26380/sentinel.conf
21:port 26380
26:daemonize yes
31:pidfile "/var/run/redis-sentinel-26380.pid"
36:logfile "/data/server/sentinel26380/sentinel.log"
65:dir "/data/server/sentinel26380"
84:sentinel monitor mymaster 127.0.0.1 6379 2
148:acllog-max-len 128
295:sentinel deny-scripts-reconfig yes
332:sentinel resolve-hostnames no
338:sentinel announce-hostnames no

# grep -Env '#|^$' /data/server/sentinel26381/sentinel.conf
21:port 26381
26:daemonize yes
31:pidfile "/var/run/redis-sentinel-26381.pid"
36:logfile "/data/server/sentinel26381/sentinel.log"
65:dir "/data/server/sentinel26381"
84:sentinel monitor mymaster 127.0.0.1 6379 2
148:acllog-max-len 128
295:sentinel deny-scripts-reconfig yes
332:sentinel resolve-hostnames no
338:sentinel announce-hostnames no

```

启动sentinel

```

/data/server/redis/bin/redis-sentinel /data/server/sentinel26379/sentinel.conf
/data/server/redis/bin/redis-sentinel /data/server/sentinel26380/sentinel.conf
/data/server/redis/bin/redis-sentinel /data/server/sentinel26381/sentinel.conf

```

注意-sentinel启动之后，会自动补全配置文件内容

```

root@python-auto:/data/server/sentinel26379# grep -Env '#|^$' /data/server/sentinel26381/sentinel.conf
21:port 26381
26:daemonize yes
31:pidfile "/var/run/redis-sentinel-26381.pid"
36:logfile "/data/server/sentinel26381/sentinel.log"
65:dir "/data/server/sentinel26381"
84:sentinel monitor mymaster 127.0.0.1 6379 2
148:acllog-max-len 128
295:sentinel deny-scripts-reconfig yes
332:sentinel resolve-hostnames no
338:sentinel announce-hostnames no
340:protected-mode no
341:user default on nopass sanitize-payload ~* &* +@all
342:sentinel myid 95438d1297a57a6f9698486dbc6c92783d3cedce
343:sentinel config-epoch mymaster 4
344:sentinel leader-epoch mymaster 4
345:sentinel current-epoch 4
346:sentinel known Replica mymaster 127.0.0.1 6380 redis的副本地址 sentinel本身的副本
347:sentinel known Replica mymaster 127.0.0.1 6381
348:sentinel known-sentinel mymaster 127.0.0.1 26380 268972b0b8504c6493d3aa6414ea5e5b031c17f0
349:sentinel known-sentinel mymaster 127.0.0.1 26379 19b6de1a4583663cc8ff496dc60e6b6c7a068220

```

查看效果

```
root@python-auto:~# tail -f /data/server/sentinel126379/sentinel.log
105560:X 29 Jul 2021 12:15:22.957 # o000o000o00o Redis is starting o000o000o000o
105560:X 29 Jul 2021 12:15:22.958 # Redis version=6.2.5, bits=64, commit=00000000, modified=0, pid=105560, just started
105560:X 29 Jul 2021 12:15:22.958 # Configuration loaded
105560:X 29 Jul 2021 12:15:22.960 * Increased maximum number of open files to 10032 (it was originally set to 1024).
105560:X 29 Jul 2021 12:15:22.960 * monotonic clock: POSIX clock_gettime
105560:X 29 Jul 2021 12:15:22.961 * Running mode=sentinel, port=26379.
105560:X 29 Jul 2021 12:15:22.962 # Sentinel ID is 19b6de1a4583663cc8ff496dc60e6b6c7a068220
105560:X 29 Jul 2021 12:15:22.962 # +monitor master mymaster 127.0.0.1 6381 quorum 2
105560:X 29 Jul 2021 12:15:33.013 * +convert-to-slave slave 127.0.0.1:6380 127.0.0.1 6380 @ mymaster 127.0.0.1 6381
105560:X 29 Jul 2021 12:15:33.013 * +convert-to-slave slave 127.0.0.1:6379 127.0.0.1 6379 @ mymaster 127.0.0.1 6381
```

连接sentinel查看效果

```
# redis-cli -h 127.0.0.1 -p 26379
127.0.0.1:26379> info sentinel
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=127.0.0.1:6379,slaves=2,sentinels=3
```

测试主从redis切换

```
# 关闭当前的主redis
redis-cli -h 127.0.0.1 -p 6379 shutdown

# 稍等几秒查看sentinel信息
127.0.0.1:26379> info sentinel
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=127.0.0.1:6381,slaves=2,sentinels=3
```

结果显示：

多个redis之间实现了 主从之间的切换效果

查看sentinel的日志切换效果

```
105560:X 29 Jul 2021 12:40:30.766 # +sdown master mymaster 127.0.0.1 6379
105560:X 29 Jul 2021 12:40:30.823 # +odown master mymaster 127.0.0.1 6379 #quorum 3/2
105560:X 29 Jul 2021 12:40:30.824 # +new-epoch 5
105560:X 29 Jul 2021 12:40:30.824 # +try-failover master mymaster 127.0.0.1 6379
105560:X 29 Jul 2021 12:40:30.831 # +vote-for-leader 19b6de1a4583663cc8ff496dc60e6b6c7a068220 5
105560:X 29 Jul 2021 12:40:30.840 # 268972b0b8504c6493d3aa6414ea5e5b031c17f0 voted for 19b6de1a4583663cc8ff496dc60e6b6c7a068220 5
105560:X 29 Jul 2021 12:40:30.841 # 95438d1297a57a6f9698486dbc6c92783d3cedce voted for 19b6de1a4583663cc8ff496dc60e6b6c7a068220 5
105560:X 29 Jul 2021 12:40:30.916 # +elected-leader master mymaster 127.0.0.1 6379
105560:X 29 Jul 2021 12:40:30.916 # +failover-state-select-slave master mymaster 127.0.0.1 6379
105560:X 29 Jul 2021 12:40:30.973 # +selected-slave slave 127.0.0.1:6381 127.0.0.1 6381 @ mymaster 127.0.0.1 6379
105560:X 29 Jul 2021 12:40:30.973 # +failover-state-send-slaveof-noone slave 127.0.0.1:6381 127.0.0.1 6381 @ mymaster 127.0.0.1 6379
105560:X 29 Jul 2021 12:40:31.076 * +failover-state-wait-promotion slave 127.0.0.1:6381 127.0.0.1 6381 @ mymaster 127.0.0.1 6379
105560:X 29 Jul 2021 12:40:31.426 # +promoted-slave slave 127.0.0.1:6381 127.0.0.1 6381 @ mymaster 127.0.0.1 6379
105560:X 29 Jul 2021 12:40:31.426 # +failover-state-reconf-slaves master mymaster 127.0.0.1 6379
105560:X 29 Jul 2021 12:40:31.475 # +failover-end master mymaster 127.0.0.1 6379
105560:X 29 Jul 2021 12:40:31.475 # +switch-master mymaster 127.0.0.1 6379 127.0.0.1 6381
105560:X 29 Jul 2021 12:40:31.476 * +slave slave 127.0.0.1:6380 127.0.0.1 6380 @ mymaster 127.0.0.1 6381
105560:X 29 Jul 2021 12:40:31.476 * +slave slave 127.0.0.1:6379 127.0.0.1 6379 @ mymaster 127.0.0.1 6381
```

小结

Keepalived

基础知识

集群简介

学习目标：

这一节，我们从 基础知识、简单实践、小结 三个方面来学习。

基础知识

集群简介



类型	解析
高扩展集群	在当前业务环境集群中，所有的主机节点都处于正常的工作活动状态，它们共同承担起用户的需求带来的工作负载压力，保证用户的正常访问。
高可用集群	将核心业务使用多台(一般是2台)主机共同工作，支撑并保障核心业务的正常运行，尤其是业务的对外不间断的对外提供服务。核心特点就是"冗余"，它存在的目的就是为了解决单点故障(Single Point of Failure)问题的。
高性能集群	基于前两种技术实现的集群基础上，高效利用这些主机资源，结合某些特有的技术方案，提供的强大的计算能力，从而实现特定用户 大型任务的高复杂度数据处理功能，比如生物计算、大场景模拟计算、预测计算等。

高可用指标

简介

高可用集群是基于高扩展基础上的一个更高层次的网站稳定性解决方案。网站的稳定性体现在两个方面：网站可用性和恢复能力

网站可用性

对于网站可用性来说，它主要有这么一个指标来评判：

$$A = \text{MTBF}/(\text{MTBF}+\text{MTTR})$$

MTBF(Mean Time Between Failure)，即平均故障间隔时间，表示产品多长时间出现一次问题。

MTTR(Mean Time To Restoration)，即平均恢复时间，表示产品从故障状态到稳定状态的维修花费时间。

A值是一个介于0~1的值，我们用这个值来表示网站正常运行时间的百分比，业界用N个9来量化可用性，最常说的就是类似“4个9(也就是99.99%)”的可用性。常见的级别如下：

序号	描述	简称	可用性级别	年度停机时间
1	基本的可用性	2个9	99%	87.6小时
2	较高的可用性	3个9	99.9%	8.8小时
3	故障自动恢复的可用性	4个9	99.99%	53分钟
4	极高可用性	5个9	99.999%	5分钟

恢复能力

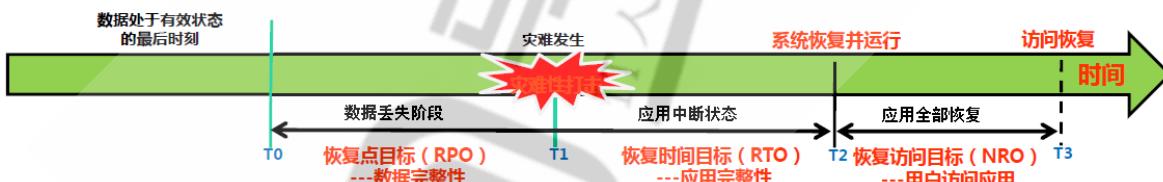
所谓的恢复能力，指的是一个网站从发生故障到故障修复这个过程的能力，而这个能力涉及到两个子内容：数据恢复能力和业务恢复能力。

数据恢复能力 RPO(Recovery Point Objective)

是指业务系统所允许的在灾难过程中的最大数据丢失量，用来衡量高可用系统的数据冗余备份能力。

业务恢复能力 RTO(Recovery Time Objective)

是指信息系统从灾难状态恢复到可运行状态所需的时间，用来衡量高可用系统的业务恢复能力。



小结

高可用方式

学习目标：

这一节，我们从基础知识、关键技术、小结三个方面来学习。

基础知识

常见术语



对于集群来说，高可用的最主要的目的就是保证网站接入口的稳定性，通过一个虚拟的接入口，将真实的请求转发到不同地址(一般是两个)的真实接入口出，进而将请求转发到后续高扩展集群主机中。这个虚拟的接入口，我们一般称之为**VIP**，这个**VIP**一般附加在后端的真实接入口，二这两个接入口一个做主一个做从，共同负担起**VIP**传输过来的信息。做主的接口主机我们一般称之为**Master**或者**Active**，做备的接口主机我们一般称之为**Slave**或者**Passive**。

集群模式

模式名称	模式解析
主备模式 (Active/Passive)	后端的高可用结点运行一致的服务实例，正常情况下，VIP配置在主节点上，只有作为主的Active结点响应用户的请求，当主节点故障，备用结点Passive自动接管一切，当主节点恢复后，通过自动或者手动方式，用户的请求重新有主节点来接管，整个过程用户感受不到任何影响。
双主模式 (Active/Active)	后端的高可用结点运行各自的服务实例，都可以正常接收用户访问请求，当A结点故障，他会将访问请求自动转移到正常运行的B结点上，从而保证不对用户产生影响。
集群模式(N+M)	在这种模式下，当主节点故障时候，后备结点有机会通过某些策略，从中选择一个结点，作为临时主节点，接管所有请求。原来的主节点恢复的时候，通过多种策略来响应。

关键技术

简介

为了实现高可用的效果，后端的高可用结点需要如下几种技术来保证：

时间同步 和 心跳检测

时间同步

作为一个集群，他们彼此间进行信息通信的一个前提是，时间状态必须一致，如果出现一个主机在2018年，一个在2020年，这种情况下，集群的通信肯定会出现问题，甚至不会传输信息。我们一般会采用时间协议，从一个专用的时间服务器上获取时间，从而保证同一个集群中的所有主机时间都是一致的，这个协议我们一般使用**ntp**协议。

心跳检测

对于高可用集群的各种模式来说，有一个关键的点就是：主节点故障了，从节点接管一切。这就涉及到了一种场景：高可用集群节点间需要知道彼此的状态，就类似于我们要知道一个人是否是活的，就看他有没有心跳。所以我们需要通过一种专用的技术来时刻了解集群节点间的状态，我们一般称这种技术为“心跳检测”，常见的软件有**VRP**。

小结

Keepalived基础

学习目标:

这一节，我们从 基础知识、软件结构、小结 三个方面来学习。

基础知识

官方介绍

Keepalived is a **routing software** written in C. The main goal of this project is to provide simple and robust[强大的] facilities[设施] for loadbalancing and high-availability to Linux system and Linux based infrastructures. Loadbalancing framework relies[依赖] on well-known and widely used Linux Virtual Server (IPVS) kernel module providing Layer4 loadbalancing. Keepalived implements[使生效] a set of checkers to dynamically and adaptively[自适应] maintain and manage loadbalanced server pool according their health.

On the other hand **high-availability is achieved[实现]** by VRRP protocol. VRRP is a fundamental brick[基石] for router failover. In addition[另外], Keepalived implements a set of hooks to the VRRP finite[有限] state machine providing low-level and high-speed protocol interactions. In order to offer fastest network failure detection[检测], Keepalived implements BFD protocol. VRRP state transition can take into account BFD hint to drive fast state transition. Keepalived frameworks can be used independently[单独] or all together to provide resilient infrastructures.

Keepalived is free software; ...

要点解析

Keepalived软件是有C语言编写的一个开源软件项目，其本质是一个路由软件。

Keepalived基于ipvs功能进行二次整合实现负载均衡功能。

Keepalived基于VRRP协议进行二次整合实现高可用性功能。

Keepalived借助于大量的功能脚本实现高质量的状态检查功能。

Keepalived框架可以单独使用，也可以和其他软件进行整合使用。

其他信息

最新版本: 2.2.2

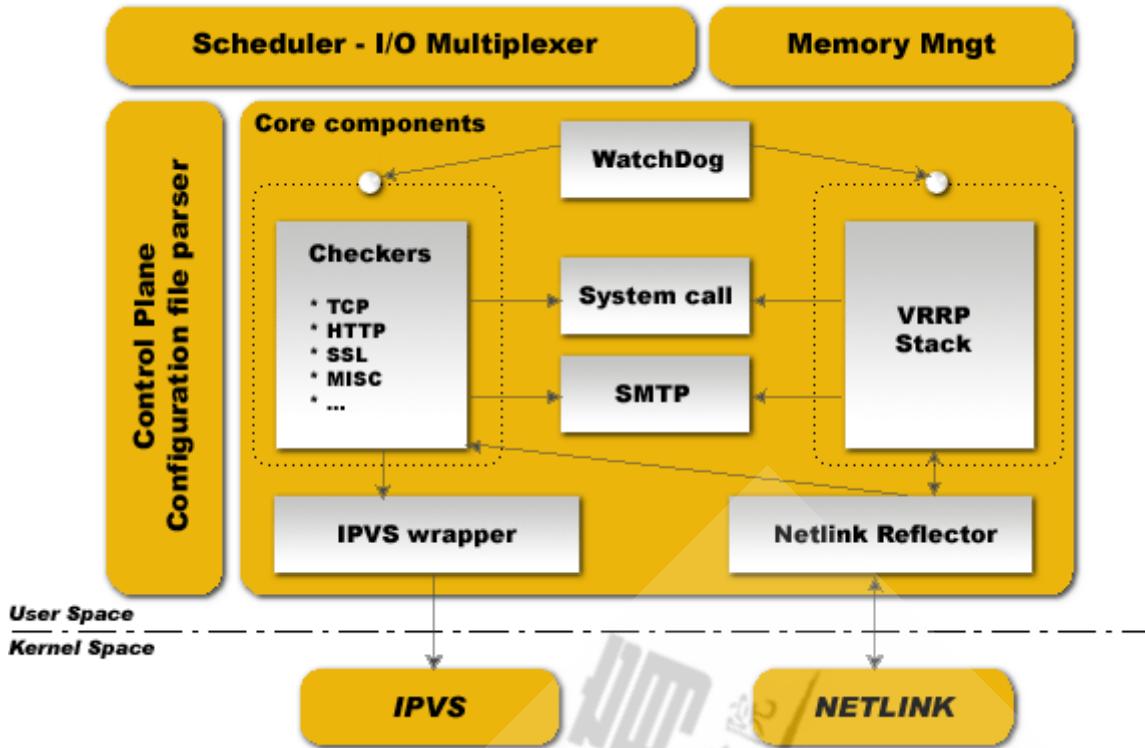
官方网站:

<http://www.keepalived.org/>

<https://www.keepalived.org/doc/index.html>

软件结构

软件结构图



图片来源: https://www.keepalived.org/doc/software_design.html

结构解析

从图中，可以看得出，Keepalived的多数核心功能模块均位于用户空间，而仅有的IPVS和NETLINK模块位于内核空间。

核心模块：

这两个内核模块却是keepalived实现负载均衡和高可用性功能的核心模块，其中NETLINK主要用于提供高可用性的路由及网络功能，IPVS主要实现虚拟主机的负载均衡功能。

功能组件：

Keepalived的大部分功能模块位于用户空间，主要有这么几个比较核心的功能组件：

WatchDog、Checkers、VRRP Stack、IPVS wrapper、Netlink Reflector

其他功能：

配置文件解析器、调度管理器、内存管理机制

工作流程

keepalived部署在多台主机上(一般是两台)，keepalived服务启动后，会加载专用的配置文件，调用内核的LVS服务以创建虚拟服务器，并根据配置启用对外的统一访问接口，并且基于相关插件脚本对服务进行监控。

为了保证服务对外的统一接口的高可用，VRRP协议将物理Router进行统一管理，并创建一个虚拟路由地址，作为外网的统一出口，物理Router内部通过VRRP机制进行内部通信，保证虚拟路由地址永远可用。

总之

通过我们对keepalived软件的结构和官方的介绍，我们可以知道，Keepalived其实就是通过整合和加强LVS和VRRP软件，从而得到的一套适用于高扩展和高可用的软件整合解决方案。

根据Keepalived软件的简单结构，我们可以了解到，我们要实现Keepalived的功能，需要在用户空间安装相应的软件才可以。

小结

前置知识

学习目标:

这一节，我们从 基础知识、软件结构、小结 三个方面来学习。

LVS

简介

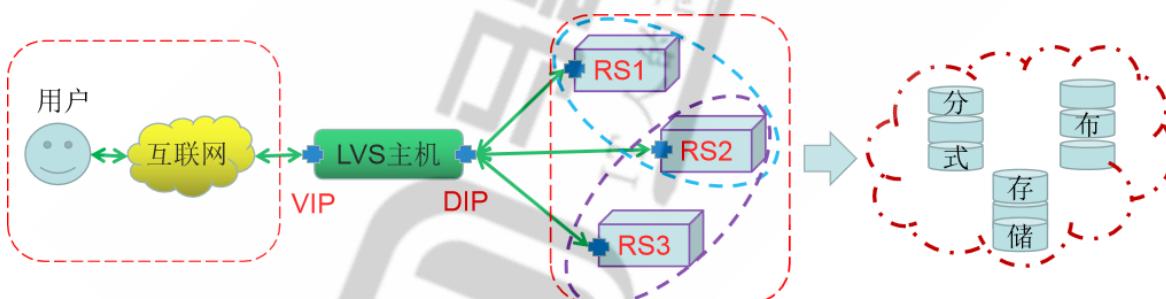
LVS是Linux Virtual Server的简称，即Linux虚拟服务器，它是由国内的章文松博士于1998年发起的开源软件项目，目前已被纳入到Linux内核中，进行统一管理。目前LVS项目已经非常成熟了，被应用于各种高扩展的服务中。

它主要有三部分组成：

最前端的负载均衡层、中间的应用服务层、后端的数据共享层。

在整个LVS负载均衡架构中，尽管整个机器内部有非常多的应用服务主机结点组成，并相应用户的请求，但是在用户看来，所有的内部应用服务都是透明的，用户只是在使用一个虚拟服务器提供的高扩展服务，这也是LVS项目核心目标所在。

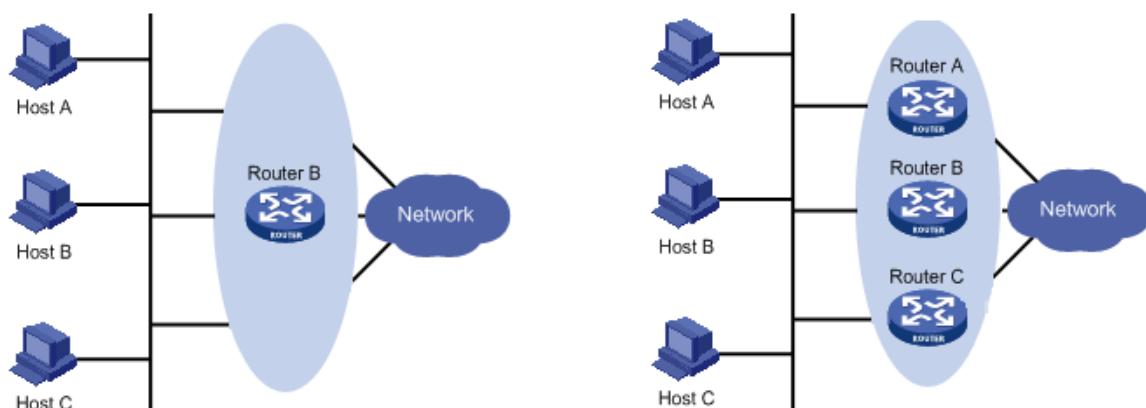
LVS结合IPVS内核模块实现为中间层的应用集群服务提供负载均衡



VRP

缺省路由

集群内部主机响应用户的请求，然后将数据报文发往到其他网段的目标地址主机，通常情况下，数据报文将通过缺省路由发往网关，再由网关进行转发到其他网段，经过层层的路由转发，从而实现主机与外部网络的通信。当网关发生故障时，本网段内所有以网关为出口的主机将无法与外部网络通信。



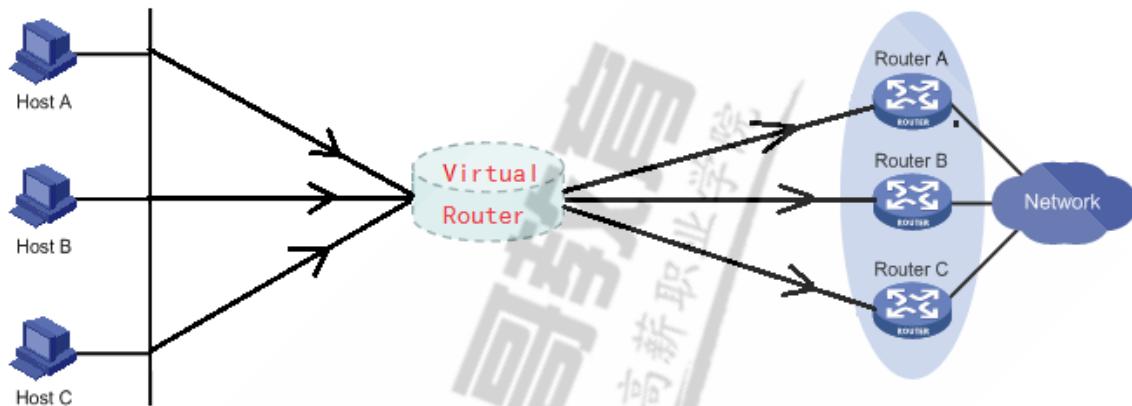
所谓的缺省路由，其实就是就是对某一网段(10.0.0.0/8)通用的网关接口，它为用户的网络配置操作提供了方便，但是由于该网关的单一性，所以其稳定性是一个很大的限制。常见的解决方法就是增加出口网关数量，即使某一个网关出现问题，我们还有备用的出口地址。而此时在多个网关出口之间或者某个网关故障时就出现了线路选择的问题。

在具有多播或广播能力的局域网中，借助**VRRP**可以完美的解决这个问题，有效避免单一链路发生故障后网络中断的现象。

vrrp

虚拟路由冗余协议(**Virtual Router Redundancy Protocol**，简称**VRRP**)是由IETF提出的解决局域网中配置静态网关出现单点故障现象的路由协议，其本质上是一种路由容错协议。

VRRP将局域网内的一组路由器虚拟为单个路由器。我们称该路由器组为**VRRP**备份组，它根据优先级选择一个**Master**主路由器，承担网关功能，其他路由器称为**Backup**从路由器。基于**VRRP**虚拟出来是路由我们称之为虚拟路由，该虚拟路由有独立的IP地址(**VIP**)。



局域网内主机不知道后端真实的路由器的ip地址，仅仅知道这个虚拟路由的ip地址，让后将VIP设定为其数据包的出网地址，通过该虚拟路由与外部的其他网络进行通信。

在整个过程中，如果**Master**路由器发生故障时，**Backup**取代**Master**继续履行网关职责，从而保证网络内的主机不间断地与外部网络进行通信。

小结

环境部署

学习目标：

这一节，我们从 软件安装、简单实践、小结 三个方面来学习。

软件安装

准备工作

序号	主机ip	主机名	安装软件	系统版本	虚拟网络模型
1	192.168.8.12	kpmaster	keepalived nginx	Ubuntu20.10	NAT
2	192.168.8.13	kpslave	keepalived nginx	Ubuntu20.10	NAT

注意：两台主机做好 时间同步 配置

软件信息

```
apt info keepalived
```

安装软件

```
apt-get install -y keepalived nginx
```

检查效果

```
keepalived --version
```

```
root@python-auto:~# keepalived --version
Keepalived v2.0.19 (10/19,2019)
Copyright(C) 2001-2019 Alexandre Cassen, <acassen@gmail.com>
Built with kernel headers for Linux 5.4.18
Running on Linux 5.8.0-38-generic #43-20.04.1-Ubuntu SMP Tue Jan 12 16:39:47 UTC 2021

configure options: --build=x86_64-linux-gnu --prefix=/usr --includedir=${prefix}/include --mandir=${prefix}/share/man --infodir=${prefix}/share/info --sysconfdir=/etc --localstatedir=/var --disable-silent-rules --libdir=${prefix}/lib/x86_64-linux-gnu --runstatedir=/run --disable-maintainer-mode --disable-dependency-tracking --with-kernel-dir=debian/ --enable-snmp --enable-sha1 --enable-snmp-rcv2 --enable-snmp-rcv3 --enable-dbus --enable-json --enable-bfd --enable-regex build_alias=x86_64-linux-gnu CFLAGS=-g -O2 -fdebg-prefix-map=/build/keepalived-sJ1e 4/keepalived-2.0.19_ --fstack-protector-strong -Wformat -Werror=format-security LDFLAGS=-Wl,-Bsymbolic-functions -Wl,-z,relro CPPFLAGS=-Wdate-time -D_FORTIFY_SOURCE=2

Config options: NFTABLES LVS REGEX VRPP VRRP_VRRP_AUTH JSON BFD OLD_CHKSUM_COMPAT FIB_ROUTING SNMP_V3_FOR_V2 SNMP_VRRP SNMP_CHECKER SNMP_P_RFCV2 SNMP_RFCV3 DBUS

System options: PIPE2 SIGNALFD INOTIFY_INIT1 VSYSLOG EPOLL CREATE1 IPV4 DEVCNF IPV6 ADVANCED API LIBNL3 RTA ENCAP RTA EXPIRES RTA_NEWDST RTA_PREF FRA_SUPPRESS_IFGROUP FRA_TUN_ID RTAX_CC_ALGO RTAX_QUICKACK RTEXT_FILTER SKIP_STATS FRA_L3MD EV_FRA_UID_RANGE RTAX_FASTOPEN_NO_COOKIE RTA_VIA FRA_OIFNAME FRA_PROTOCOL FRA_IP_PROTO FRA_SPORT RANGE FRA_DPORT RANGE RTA_TTL TTL_PROP AGATE IFA_FLAGS IP_MULTICAST_ALL LWTUNNEL_ENCAP_MPLS LWTUNNEL_ENCAP_IHLA_NET_LINUX_IF_H_COLLISION LIBIPVS_NETLINK IPVVS_DEST_ATTR ADD_R_FAMILY IPVFS_SYNCD ATTRIBUTES IPVFS_6BIT_STATS IPVFS_TUN_CSUM IPVFS_TUN_GRE IPVFS_TUN_VMAC IPVFS_TUN_VRRP IPVLAN_IFLA_LINK_NETNSID CN PROC SOCK NONBLOCK SOCK_CLOEXEC O_PATH_GLOB_BRACE INET6_ADDR_GEN_MODE_VRF SO_MARK_SCHED_RT_SCHED_RESET_ON_FORK
```

配置结构目录

```
root@python-auto:~# dpkg -L keepalived
/.
/etc
/etc/dbus-1
/etc/dbus-1/system.d
/etc/dbus-1/system.d/org.keepalived.Vrrp1.conf
/etc/default
/etc/default/keepalived
/etc/init.d
/etc/init.d/keepalived
/etc/keepalived # 默认工作目录
/lib
/lib/systemd
/lib/systemd/system
/lib/systemd/system/keepalived.service # 服务配置文件
/usr
/usr/bin
/usr/bin/genhash
/usr/sbin
/usr/sbin/keepalived # 软件命令
...
/usr/share/doc/keepalived/samples # 模板文件目录，这里面包含了大量的配置文件
```

```
root@python-auto:~# cat /lib/systemd/system/keepalived.service
[Unit]
Description=Keepalive Daemon (LVS and VRRP)
After=network-online.target
Wants=network-online.target
# Only start if there is a configuration file
ConditionFileNotEmpty=/etc/keepalived/keepalived.conf      # 核心配置文件， 默认不存在
...
...
```

根据我们对 /usr/share/doc/keepalived/samples 目录下的模板文件查看 , keepalived的配置文件结构如下

```
root@python-auto:/usr/share/doc/keepalived/samples# cat keepalived.conf.sample
! Configuration File for keepalived
# 全局配置段
global_defs {
    notification_email {
        acassen
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}
# VRRP配置段
vrrp_instance VI_1 {
    interface ens33
    virtual_router_id 50
    nopreempt
    priority 100
    advert_int 1
    virtual_ipaddress {
        192.168.200.11
        192.168.200.12
        192.168.200.13
    }
}
# 虚拟主机配置段
virtual_server 10.10.10.2 1358 {
    delay_loop 6
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
    sorry_server 192.168.200.200 1358
    real_server 192.168.200.2 1358 {
        weight 1
        HTTP_GET {
            url {
                path /testurl3/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334d
            }
            connect_timeout 3
            retry 3
        }
    }
}
```

```
    delay_before_retry 3
}
}
}
```

结果显示

我们的keepalived默认配置文件主要有三部分组成，`global_defs`、`vrrp_instance`、`virtual_server`

其中最重要的配置内容是`vrrp_instance`，在这个配置段中，设置了我们keepalived对外提供的统一入口

配置简介

作为快速入门阶段来说，此时它的配置信息没有那么的复杂，我们这里只需要修改两台keepalived主机的`global_defs`和`vrrp_instance`部分的属性内容，就可以让keepalived正常启动了。

在这里我们来简单的看一下这里面的基础重要的属性配置：

`global_defs`

在这部分的配置段中，我们暂时只需要关心`router_id`即可

`router_id` 设定当前keepalived提供的路由标识，它在keepalived集群中必须唯一

`vrrp_instance`

在这部分的配置段中，我们优先关心以下几处的配置信息：

`state` 描述keepalived主机间的角色定位的，一般只有两个值MASTER、

BACKUP

`interface` 指定在哪个网卡上绑定VIP

`virtual_router_id` 指定VIP的唯一标识，在keepalived集群中，此配置必须一致。

`priority` 被VRRP协议来判断那个`router_id`作为主路由，值越大，优先级越高

`authentication` 多个路由之间通信的认证

`virtual_ipaddress` 指定VIP的地址，可以是多个。

简单实践

nginx环境定制

```
# 安装nginx软件
apt install nginx -y

# 定制nginx主页信息
8.12 主机定制nginx主页信息
echo 'keepalived master' > /var/www/html/index.nginx-debian.html
8.13 主机定制nginx主页信息
echo 'keepalived slave' > /var/www/html/index.nginx-debian.html
```

确认效果

```
root@python-auto:~# curl 192.168.8.12
keepalived master
root@python-auto:~# curl 192.168.8.13
keepalived slave
```

定制keepalived的配置

```
# 准备配置文件
cd /usr/share/doc/keepalived/samples
cp keepalived.conf.sample /etc/keepalived/keepalived.conf

# 设定主keepalived配置文件
# cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
    router_id kpmaster
}

vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 50
    priority 100
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.8.100
    }
}

# 设定从keepalived配置文件
# cat /etc/keepalived/keepalived.conf
global_defs {
    router_id kpslave
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 50
    priority 99
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.8.100
    }
}

# 重启keepalived服务
systemctl restart keepalived.service
```

查看8.12主机网卡效果

```
root@python-auto:/usr/share/doc/keepalived/samples# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:50:56:2e:61:91 brd ff:ff:ff:ff:ff:ff
        altname enp2s1
        inet 192.168.8.12/24 brd 192.168.8.255 scope global noprefixroute ens33
            valid_lft forever preferred_lft forever
        inet 192.168.8.100/32 scope global ens33
            valid_lft forever preferred_lft forever
        inet6 fe80::250:56ff:fe2e:6191/64 scope link
            valid_lft forever preferred_lft forever
```

查看8.13主机网卡效果

```
root@python-auto:/usr/share/doc/keepalived/samples# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:50:56:21:d2:7a brd ff:ff:ff:ff:ff:ff
        altname enp2s1
        inet 192.168.8.13/24 brd 192.168.8.255 scope global noprefixroute ens33
            valid_lft forever preferred_lft forever
        inet6 fe80::250:56ff:fe21:d27a/64 scope link
            valid_lft forever preferred_lft forever
```

结果显示：

在kpmaster主机的ens33上多出来了一个虚拟ip
只能使用ip addr命令来查看，ifconfig 命令检查不出来效果

查看启动信息

我们可以通过 systemctl status keepalived 方式查看不同主机的软件运行状态，
更详细的信息，可以通过 tail -f /var/log/syslog 方式来进行查询

```
root@python-auto:/usr/share/doc/Keepalived/samples# systemctl status keepalived.service
● keepalived.service - Keepalive Daemon (LVS and VRRP)
   Loaded: loaded (/lib/systemd/system/keepalived.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2021-07-29 16:14:31 CST; 3min 25s ago
     Main PID: 11612 (keepalived)
        Tasks: 2 (limit: 4619)
       Memory: 3.4M
      CGroup: /system.slice/keepalived.service
              └─11612 /usr/sbin/keepalived --dont-fork
                  ├─11613 /usr/sbin/keepalived --dont-fork
8.12 主机查看状态
7月 29 16:14:32 python-auto Keepalived_vrrp[11613]: Registering Kernel netlink reflector
7月 29 16:14:32 python-auto Keepalived_vrrp[11613]: Registering Kernel netlink command channel
7月 29 16:14:32 python-auto Keepalived_vrrp[11613]: Opening file '/etc/keepalived/keepalived.conf'.
7月 29 16:14:32 python-auto Keepalived_vrrp[11613]: Registering gratuitous ARP shared channel
7月 29 16:14:32 python-auto Keepalived_vrrp[11613]: (VI_1) Entering BACKUP STATE (init)
7月 29 16:14:32 python-auto Keepalived_vrrp[11613]: (VI_1) received lower priority (99) advert from 192.168.8.13 - discarding
7月 29 16:14:33 python-auto Keepalived_vrrp[11613]: (VI_1) received lower priority (99) advert from 192.168.8.13 - discarding
7月 29 16:14:34 python-auto Keepalived_vrrp[11613]: (VI_1) received lower priority (99) advert from 192.168.8.13 - discarding
7月 29 16:14:35 python-auto Keepalived_vrrp[11613]: (VI_1) received lower priority (99) advert from 192.168.8.13 - discarding
7月 29 16:14:35 python-auto Keepalived_vrrp[11613]: (VI_1) Entering MASTER STATE
7月 29 16:14:26 python-auto Keepalived[10344]: Opening file '/etc/keepalived/keepalived.conf'.
7月 29 16:14:26 python-auto Keepalived[10344]: Starting VRRP child process, pid=10345
8.13 主机查看状态
7月 29 16:14:26 python-auto Keepalived_vrrp[10345]: Registering Kernel netlink reflector
7月 29 16:14:26 python-auto Keepalived_vrrp[10345]: Registering Kernel netlink command channel
7月 29 16:14:26 python-auto Keepalived_vrrp[10345]: Opening file '/etc/keepalived/keepalived.conf'.
7月 29 16:14:26 python-auto Keepalived_vrrp[10345]: Registering gratuitous ARP shared channel
7月 29 16:14:26 python-auto Keepalived_vrrp[10345]: (VI_1) Entering BACKUP STATE (init)
7月 29 16:14:30 python-auto Keepalived_vrrp[10345]: (VI_1) Entering MASTER STATE
7月 29 16:14:35 python-auto Keepalived_vrrp[10345]: (VI_1) Master received advert from 192.168.8.12 with higher priority 100, ours 99
7月 29 16:14:35 python-auto Keepalived_vrrp[10345]: (VI_1) Entering BACKUP STATE
```

nginx访问效果

```
# 访问虚拟ip
~]# for i in {1..10};do curl 192.168.8.100;done
keepalived master
...
keepalived master
```

结果显示：
VIP效果完全正常，可以替代原来的web地址访问效果

小结

故障演练

学习目标

这一节，我们从 主机故障、服务故障、小结 三个方面来学习。

主机故障

模拟master主机故障

```
关停kpmaster主机上的keepalived服务
systemctl stop keepalived.service
```

kpmaster主机日志效果

```
Mar 27 15:51:02 [localhost] Keepalived[14369]: Stopping
Mar 27 15:51:02 [localhost] systemd: Stopping LVS and VRRP High Availability
Monitor...
Mar 27 15:51:02 [localhost] Keepalived_healthcheckers[14370]: Stopped
Mar 27 15:51:02 [localhost] Keepalived_vrrp[14371]: VRRP_Instance(VI_1) sent 0
priority
Mar 27 15:51:02 [localhost] Keepalived_vrrp[14371]: VRRP_Instance(VI_1) removing
protocol VIPs.
Mar 27 15:51:03 [localhost] Keepalived_vrrp[14371]: Stopped
Mar 27 15:51:03 [localhost] Keepalived[14369]: Stopped Keepalived v1.3.5
(03/19/2017), git commit v1.3.5-6-g6fa32f2
Mar 27 15:51:03 [localhost] systemd: Stopped LVS and VRRP High Availability
Monitor.
```

可以看到：

该节点移除了VIP内容

kpslave结点检查效果

```
Mar 27 15:51:03 [localhost] Keepalived_vrrp[1762]: VRRP_Instance(VI_1)
Transition to MASTER STATE
Mar 27 15:51:04 [localhost] Keepalived_vrrp[1762]: VRRP_Instance(VI_1) Entering
MASTER STATE
Mar 27 15:51:04 [localhost] Keepalived_vrrp[1762]: VRRP_Instance(VI_1) setting
protocol VIPs.
Mar 27 15:51:04 [localhost] Keepalived_vrrp[1762]: Sending gratuitous ARP on
ens33 for 192.168.8.100
```

结果显示：

该节点接管了VIP内容

```
kpslave结点查看ip信息
# ip addr
...
2: ens33: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 00:0c:29:b0:56:23 brd ff:ff:ff:ff:ff:ff
        inet 192.168.8.15/16 brd 192.168.255.255 scope global ens33
            valid_lft forever preferred_lft forever
        inet 192.168.8.100/32 scope global ens33
            valid_lft forever preferred_lft forever
结果显示:
    keepalived的VIP自动被kpslave主机接管了。
```

```
检查nginx效果
~]# for i in {1..10};do curl 192.168.8.100;done
keepalived slave
...
keepalived slave
结果显示
后端服务访问正常
```

模拟master主机恢复

```
开启kpmaster主机上的keepalived服务
systemctl start keepalived.service
```

```
kpslave主机日志效果
Mar 27 15:59:20 [localhost] Keepalived_vrrp[1762]: VRRP_Instance(VI_1) Received
advert with higher priority 100, ours 99
Mar 27 15:59:20 [localhost] Keepalived_vrrp[1762]: VRRP_Instance(VI_1) Entering
BACKUP STATE
Mar 27 15:59:20 [localhost] Keepalived_vrrp[1762]: VRRP_Instance(VI_1) removing
protocol VIPs.
可以看到:
```

该节点被移除了VIP内容

```
kpmaster结点检查效果
Mar 27 15:59:20 [localhost] Keepalived_vrrp[14466]: VRRP_Instance(VI_1)
Transition to MASTER STATE
Mar 27 15:59:21 [localhost] Keepalived_vrrp[14466]: VRRP_Instance(VI_1) Entering
MASTER STATE
Mar 27 15:59:21 [localhost] Keepalived_vrrp[14466]: VRRP_Instance(VI_1) setting
protocol VIPs.
Mar 27 15:59:21 [localhost] Keepalived_vrrp[14466]: Sending gratuitous ARP on
ens33 for 192.168.8.100
结果显示:
```

该节点接管了VIP内容，恢复到了MASTER角色

```
kpmaster结点查看ip信息
# ip addr
...
2: ens33: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 00:0c:29:9f:ea:4e brd ff:ff:ff:ff:ff:ff
        inet 192.168.8.14/24 brd 192.168.8.255 scope global ens33
            valid_lft forever preferred_lft forever
        inet 192.168.8.100/32 scope global ens33
```

```
valid_lft forever preferred_lft forever  
结果显示:  
keepalived的VIP自动被kpmaster主机抢占回来了。
```

```
检查nginx效果  
~]# for i in {1..10};do curl 192.168.8.100;done  
keepalived master  
...  
keepalived master  
结果显示  
后端服务访问正常
```

服务故障

模拟服务故障

```
关闭master主机上的nginx服务  
systemctl stop nginx  
  
访问vip效果  
~]# for i in {1..10};do curl 192.168.8.100;done  
curl: (7) Failed connect to 192.168.8.100:80; 拒绝连接  
...  
curl: (7) Failed connect to 192.168.8.100:80; 拒绝连接  
结果显示:  
因为VIP附加在kpmaster主机上，所以只能访问kpmaster主机上的nginx  
因为该主机上的nginx服务被终止了，所以会提示拒绝连接
```

模拟服务恢复

```
开启master主机上的nginx服务  
systemctl start nginx  
  
访问vip效果  
~]# for i in {1..10};do curl 192.168.8.100;done  
keepalived master  
...  
keepalived master  
结果显示:  
nginx访问成功了
```

小结

梳理

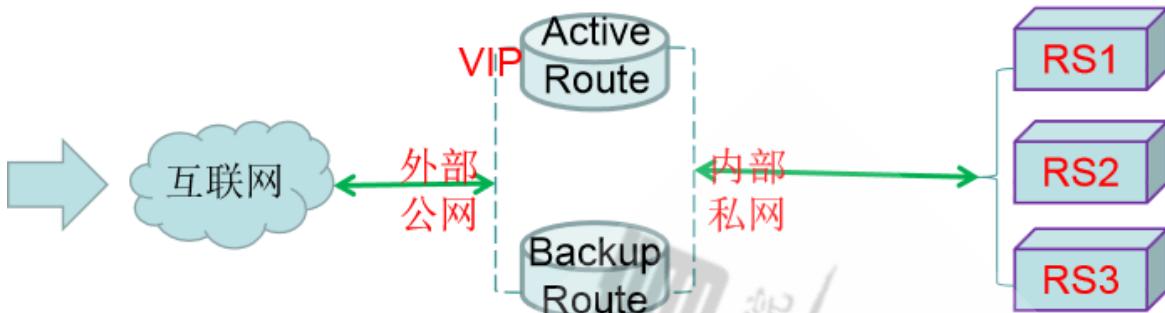
根据我们的两种不同情况的验证效果，我们知道：

VIP就是网站服务的入口，只要keepalived集群主机没有问题，那么后端的服务访问就不受影响。

后端的服务效果由真实主机的nginx服务来决定的，而web的访问入口是主机ip，和VIP关系不大。

由此可得出高可用集群的特点：

Keepalived的设计其实是具有二层拓扑的负载均衡架构，该架构分为两层。前一层为负载均衡层，由多个keepalive主机提供的Routers组成，每个Routers都有两个接口，一个接入Internet网络(--VIP)，一个接入内部私有网络(--192.168.8.0/24)，Active主机的Router在这两个接口间进行数据转发，将Internet网络中的数据转发到私网的物理服务器结点中。第二层的物理服务器运行相应的应用服务，并响应转发过来请求的服务。



负载均衡

学习目标

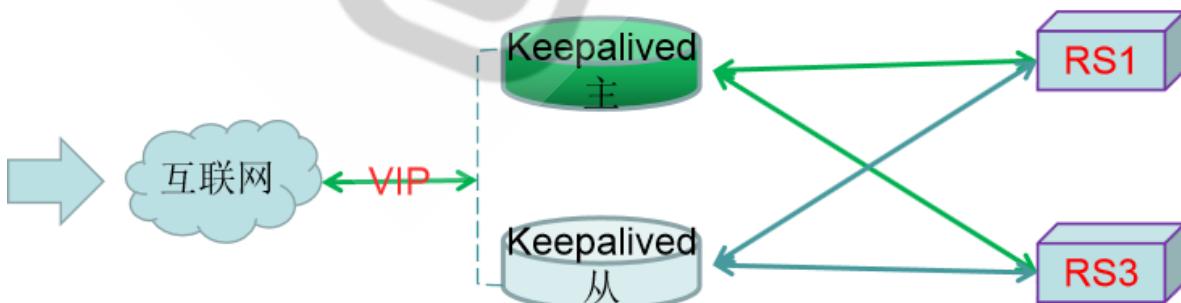
这一节，我们从基础知识、配置详解、操作实践、实践小结六个方面来学习

基础知识

简介

在上一节的故障演练中，尤其是后端服务的中断后，显示“拒绝连接”报错信息，这是由于后端的真实服务并没有实现负载均衡的效果，而我们知道keepalived其实由这个功能，它是基于内核中的lvs模块来实现的Linux虚拟机负载均衡效果。

接下来，我们就借助于keepalived的虚拟服务功能来实现负载均衡两个Nginx服务。



主机资源

序号	主机角色	ip信息	ip角色	网卡设备	安装软件
1	kpmaster	192.168.8.12	网卡ip	eth0-nat模型	keepalived
		192.168.8.100	VIP	eth0别名	
2	kpslave	192.168.8.13	网卡ip	eth0-nat模型	keepalived
		192.168.8.100	VIP	eth0别名	
3	lvs-rs1	192.168.8.14	网卡ip	eth0-nat模型	nginx
		192.168.8.100	VIP	lo回环网卡即可	
4	lvs-rs2	192.168.8.15	网卡ip	eth0-nat模型	nginx
		192.168.8.100	VIP	lo回环网卡即可	

注意：

所有主机都是单网卡，网络适配器采用NAT模式。

需求分析

为了满足上面的场景需求，我们需要通过以下几个步骤来完成整个动作

- 0 网络环境配置
- 1 nginx服务正常
- 2 keepalived配置虚拟服务
- 3 效果检查

关键点分析

- 0 网络环境配置
- 1 lvs配置nginx服务正常
 - 1.1 保证nginx服务正常访问即可
 - 1.2 配置DR模型的lvs
- 2 keepalived配置虚拟服务
 - 2.1 配置keepalived的虚拟服务
- 3 效果检查

配置详解

属性配置

这里简单的罗列一下常见的虚拟服务配置属性，我们可以直接根据默认的配置内容进行学习。配置样例如下：

```
virtual_server 10.10.10.2 1358 {
    delay_loop 6
    lb_algo rr
    lb_kind NAT
    protocol TCP
    real_server 192.168.200.2 1358 {
        ...
    }

    real_server 192.168.200.3 1358 {
        ...
    }
}
```

属性简介

虚拟主机配置格式：virtual_server <VIP> <port> { ... }

```
delay_loop 用于服务轮询的延迟计时器  
lb_algo 设定访问后端服务的调度策略  
lb_kind 设定数据转发的模型  
protocol 设定数据通信的协议  
real_server 设定后端主机的信息
```

注意：

VIP的完整设置格式是：

```
<IP地址>/<掩码> brd <广播地址> dev <网卡名称> scope <作用范围> label <网卡  
标签>
```

默认格式：<IPADDR>/32 dev ens33 scope global

标签样式：192.168.200.18/24 dev eth2 label eth2:1

注意：

没有特殊要求的话，推荐直接使用ip地址即可。

后端主机配置格式：real_server <IP> <port> { ... }

注意：

以上5项是必须设置的。

real_server 后面的"{}"中没有属性，不能写成"{}"，必须使用Enter换行格式隔开，否则的话会报错

```
Missing '{' at beginning of configuration block  
Unknown keyword 'real_server'
```

配置样例

```
virtual_server 192.168.8.100 80 {  
    delay_loop 2  
    lb_algo rr  
    lb_kind NAT  
    protocol TCP  
    real_server 192.168.8.14 80 {  
    }  
    real_server 192.168.8.15 80 {  
    }  
}
```

操作实践

网络环境配置



配置后端nginx服务

安装软件

```
yum install nginx -y
```

配置首页

```
# lvs-RS1主机
```

```
echo "nginx-RS1" > /var/www/html/index.nginx-debian.html
# lvs-RS2主机
echo "nginx-RS2" > /var/www/html/index.nginx-debian.html
```

启动两台主机nginx服务
systemctl restart nginx

检查服务效果
]# curl 192.168.8.14
nginx-RS1
]# curl 192.168.8.15
nginx-RS2

配置LVS的DR模式

后端主机配置arp抑制
echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce

启动后端主机VIP
ifconfig lo:0 192.168.8.100 netmask 255.255.255.255 broadcast 192.168.8.100

lvs主机配置虚拟ip(-以kpmaster主机为例)
ip addr add 192.168.8.100 dev ens33:0
注意：

这里的 网卡别名仅仅是为了 ipvs 演示用的，后面的keepalived用不到

配置ipvs规则(-以kpmaster主机为例)
apt-get install ipvsadm -y
ipvsadm -A -t 192.168.8.100:80 -s rr
ipvsadm -a -t 192.168.8.100:80 -r 192.168.8.14
ipvsadm -a -t 192.168.8.100:80 -r 192.168.8.15
ipvsadm -Ln

检查效果
curl 192.168.8.100
结果显示：

LVS的DR模型配置成功。

注意(特别注意)：

只能在lvs主机和RS主机之外的客户端主机上测试，我们这里以kpslave主机为例
因为keepalived的master和两个RS上都有 8.100，无法测试效果

注意：

为例不影响后续keepalived的操作，需要ipvsadm清空规则(-以kpmaster主机为例)
ipvsadm -C
ip addr del 192.168.8.100 dev ens33:0

配置keepalived的虚拟服务

```
# 修改kpmaster配置文件
global_defs {
```

```
router_id kpmaster
}

vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 50
    priority 100
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.8.100 dev ens33 label ens33:0
    }
}

virtual_server 192.168.8.100 80 {
    delay_loop 2
    lb_algo rr
    lb_kind DR
    protocol TCP
    real_server 192.168.8.14 80 {
    }
    real_server 192.168.8.15 80 {
    }
}

# 修改kpslave配置文件
global_defs {
    router_id kpslave
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 50
    priority 99
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.8.100 dev ens33 label ens33:0
    }
}

virtual_server 192.168.8.100 80 {
    delay_loop 2
    lb_algo rr
    lb_kind DR
    protocol TCP
    real_server 192.168.8.14 80 {
    }
    real_server 192.168.8.15 80 {
    }
}
```

```
# 启动所有keepalived服务  
systemctl restart keepalived.service
```

检查效果

```
检查ip地址  
ip addr  
结果显示:  
    只有kpmaster主机存在VIP  
  
访问vip  
[root@kpslave ~]# curl 192.168.8.100  
nginx-RS2  
[root@kpslave ~]# curl 192.168.8.100  
nginx-RS1  
注意:  
    只能去没有192.168.8.100的主机上进行测试。否则
```

故障模拟

```
关闭RS1主机nginx服务  
[root@lvs-rs1 ~]# systemctl stop nginx  
检查效果  
[root@kpslave ~]# curl 192.168.8.100  
curl: (7) Failed connect to 192.168.8.100:80; 拒绝连接  
[root@kpslave ~]# curl 192.168.8.100  
nginx-RS2  
[root@kpslave ~]# curl 192.168.8.100  
curl: (7) Failed connect to 192.168.8.100:80; 拒绝连接  
[root@kpslave ~]# curl 192.168.8.100  
nginx-RS2
```

```
开启RS1主机nginx服务  
[root@lvs-rs1 ~]# systemctl start nginx  
检查效果  
[root@kpslave ~]# curl 192.168.8.100  
nginx-RS2  
[root@kpslave ~]# curl 192.168.8.100  
nginx-RS1  
[root@kpslave ~]# curl 192.168.8.100  
nginx-RS2  
[root@kpslave ~]# curl 192.168.8.100  
nginx-RS1
```

```
主机故障模拟  
[root@kpmaster ~]# systemctl stop keepalived.service  
检查效果  
[root@kpmaster ~]# curl 192.168.8.100  
nginx-RS1  
[root@kpmaster ~]# curl 192.168.8.100  
nginx-RS2  
[root@kpmaster ~]# curl 192.168.8.100  
nginx-RS1  
[root@kpmaster ~]# curl 192.168.8.100  
nginx-RS2
```

结果显示：

我们的keepalived实现了后端主机的负载均衡效果

实践小结

对于负载均衡场景，我们可以结合keepalived的virtual_server属性来完成，完成步骤如下

- 1 规划服务的网络拓扑
- 2 定制负载均衡的效果
- 3 keepalived使用配置负载均衡
- 4 效果测试

注意

keepalived在使用负载均衡配置的时候，一定要注意：

选择合适的负载均衡策略

virtual_ipaddress 中使用的名称，一定要与 virtual_server 中指定的名称一致

virtual_server中必备的属性一定不要少，而且注意real_server的属性格式

进阶实践

状态检测

学习目标

这一节，我们从 基础知识、配置详解、操作实践、小结 四个方面来学习。

基础知识

需求

根据我们对keepalived的了解，我们知道它基于VRRP协议不但实现了高可用性的功能，而且还实现了基于脚本文件实现了状态监测的功能，而这个功能使用的场景非常多。

拓展开来，只要我们掌握这个状态监测的方法，我们就可以对keepalived管理的所有内容统统进行状态监测，而且还可以基于本身的邮件通知功能实现更为精细化的服务管理。

我们的需求就是：keepalived采用脚本的方式动态方式来检查keepalived服务的状态，然后通过手工操作模拟keepalived故障，检查keepalived的服务检查状态。

分析

为了满足上面的场景需求，我们需要通过以下几个步骤来完成整个动作

- 1 编写检查脚本文件
- 2 keepalived配置文件使用脚本
- 3 效果检查

关键点分析

- 1 编写检查脚本文件
 - 1.1 编写检测服务脚本
- 2 keepalived配置文件使用脚本
 - 2.1 配置检测脚本
- 3 效果检查

配置详解

为了更高效的使用keepalived的脚本检测功能，我们先来学习一下，如何配置脚本的检查流程。

如果要实现状态监测效果，我们需要按照以下步骤来做：

- 1 配置 VRRP script(s) 的状态检查命令
- 2 在VRRP实例中使用指定的检查命令

VRRP script配置

根据 `man keepalived.conf` 方式来查看VRRP script的配置信息显示：vrrp_script配置是专门用于设定监控VRRP实例用的配置，格式效果如下：

```
vrrp_script <SCRIPT_NAME> {
    script <STRING>|<QUOTED-STRING>          # 设定执行脚本的路径，也可以是命令行的检测命令
    interval <INTEGER>                         # 设定脚本执行的间隔时间，默认1s
    timeout <INTEGER>                          # 设定脚本失败的超时时间
    weight <INTEGER:-254..254>                 # 设定脚本的权重，默认是0
    rise <INTEGER>                            # 执行多少次，表示成功OK
    fall <INTEGER>                            # 执行多少次，表示失败KO
    user USERNAME [GROUPNAME]                  # 设定脚本的用户/组属性信息
    init_fail                                  # 假设脚本最初处于失败状态
}
```

注意：

我们一般使用前面的两条属性

检测命令的<SCRIPT_NAME>是VRRP专用的，一定要保证在VRRP场景下是唯一的

`script` 属性要求尽量使用脚本格式，不包含特殊字符和表达式的简单命令也可以，如果命令里包含特殊字符 就不能执行，比如"[[]]"符号，报错效果如下

```
Mar 27 23:09:51 [localhost] Keepalived_vrrp[1422]: Unable to access script
`[[`
```

配置示例

配置示例：

```
vrrp_script chk_keepalived {
    script "/bin/bash /path/to/script.file"
    interval 1
    weight -10
}
```

注意：

此处的`weight`很重要，它与`vrrp_instance`的`priority`属性共同决定了集群中MASTER和BACKUP角色切换动作，这个值一般有正数和负数之分，最终功能都是一样的，我们一般用负数。

`weight`为负(-n)：

脚本执行失败，主节点的"priority-weight值"，若差值小于BACKUP节点的`priority`，则切换备为主，否 则不切换角色状态

使用检测

我们可以在vrrp_instance配置段中，基于track_script属性来执行已设定好的vrrp_script。

```
track_script {  
    <SCRIPT_NAME>  
    <SCRIPT_NAME> weight <-254..254>  
}
```

注意：

我们一般使用第一条配置样式即可。

必须保证<SCRIPT_NAME>在上面的vrrp_script中定义好了。

配置示例

```
track_script {  
    chk_keepalived  
}
```

操作实践

编写检测服务脚本

创建专用脚本目录

```
mkdir /data/scripts/ -p
```

创建脚本

```
]# vim /data/scripts/keepalived_check.sh  
#!/bin/bash  
if [ -f /tmp/keepalived.fail ];then  
    weight -2  
fi
```

注意：

脚本中的 "weight -2" 表示，当我们存在keepalived的fail文件的时候，表明该软件已经故障，需要降低本主机的优先级，便于BACKUP主机提升为主角色
-2 的值，需要根据MASTER和BACKUP的优先级进行规划，保证 "MASTER优先级-2 < BACKUP优先级"

keepalived配置文件使用脚本

修改后的keepalived配置文件内容如下

```
global_defs {  
    router_id kpmaster  
}  
vrrp_script chk_keepalived {  
    script "/bin/bash /data/scripts/keepalived_check.sh"  
    interval 1  
}  
vrrp_instance VI_1 {  
    state MASTER  
    interface ens33  
    virtual_router_id 51  
    priority 100  
    virtual_ipaddress {  
        192.168.8.100  
    }  
}
```

```
track_script {
    chk_keepalived
}
```

注意：

`track_script` 中使用的名称，一定要在 `vrrp_script` 中定义
`vrrp_script` 中 `script` 属性的脚本使用，应该使用标准的“`/bin/bash /path/to/file`”格式

避免因为权限问题导致脚本无法执行，从而影响 `keepalived` 的正常使用
两台 `keepalived` 主机必须做同样的配置修改

重启服务

启动服务

```
systemctl start keepalived
```

注意：

两台主机都执行服务启动的命令

`kpmaster` 查看日志效果

```
Mar 27 23:43:44 [localhost] Keepalived_vrrp[5069]: VRRP_Script(chk_keepalived)
succeeded
Mar 27 23:43:44 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1)
Transition to MASTER STATE
Mar 27 23:43:46 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1) Entering
MASTER STATE
Mar 27 23:43:46 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1) setting
protocol VIPs.
Mar 27 23:43:46 [localhost] Keepalived_vrrp[5069]: Sending gratuitous ARP on
ens33 for 192.168.8.100
```

检查 IP 地址

```
[root@kpmaster ~]# ip addr
...
2: ens33: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 00:0c:29:9f:ea:4e brd ff:ff:ff:ff:ff:ff
    inet 192.168.8.12/24 brd 192.168.8.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet 192.168.8.100/32 scope global ens33
```

结果显示：

主从两个节点状态都正常

模拟故障测试

`kpmaster` 主机创建 `fail` 文件

```
touch /tmp/keepalived.fail
```

`kpmaster` 主机查看日志

```
Mar 27 23:50:49 [localhost] Keepalived_vrrp[5069]: /bin/bash
/data/scripts/keepalived_check.sh exited with status 127
Mar 27 23:50:49 [localhost] Keepalived_vrrp[5069]: VRRP_Script(chk_keepalived)
failed
Mar 27 23:50:50 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1) Entering
FAULT STATE
```

```
Mar 27 23:50:50 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1) removing protocol VIPs.
Mar 27 23:50:50 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1) Now in FAULT state
Mar 27 23:50:50 [localhost] Keepalived_vrrp[5069]: /bin/bash
/data/scripts/keepalived_check.sh exited with status 127
...

```

kpslave主机查看日志

```
Mar 27 23:50:51 [localhost] Keepalived_vrrp[1756]: VRRP_Instance(VI_1) Transition to MASTER STATE
Mar 27 23:50:52 [localhost] Keepalived_vrrp[1756]: VRRP_Instance(VI_1) Entering MASTER STATE
Mar 27 23:50:52 [localhost] Keepalived_vrrp[1756]: VRRP_Instance(VI_1) setting protocol VIPs.
Mar 27 23:50:52 [localhost] Keepalived_vrrp[1756]: Sending gratuitous ARP on ens33 for 192.168.8.100
```

kpmaster主机检查ip

```
[root@kpmaster ~]# ip addr
...
2: ens33: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 00:0c:29:9f:ea:4e brd ff:ff:ff:ff:ff:ff
    inet 192.168.8.12/24 brd 192.168.8.255 scope global ens33
        valid_lft forever preferred_lft forever
```

kpslave主机检查ip

```
[root@kpslave ~]# ip addr
...
2: ens33: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 00:0c:29:b0:56:23 brd ff:ff:ff:ff:ff:ff
    inet 192.168.8.13/16 brd 192.168.255.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet 192.168.8.100/32 scope global ens33
```

模拟故障修复

```
kpmaster主机移除fail文件
rm -f /tmp/keepalived.fail
```

kpmaster主机查看日志

```
Mar 27 23:52:23 [localhost] Keepalived_vrrp[5069]: VRRP_Script(chk_keepalived) succeeded
Mar 27 23:52:24 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1) Entering BACKUP STATE
Mar 27 23:52:25 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1) forcing a new MASTER election
Mar 27 23:52:26 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1) Transition to MASTER STATE
Mar 27 23:52:27 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1) Entering MASTER STATE
Mar 27 23:52:27 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1) setting protocol VIPs.
Mar 27 23:52:27 [localhost] Keepalived_vrrp[5069]: Sending gratuitous ARP on ens33 for 192.168.8.100
```

```
Mar 27 23:52:27 [localhost] Keepalived_vrrp[5069]: VRRP_Instance(VI_1)
Sending/queueing gratuitous ARPs on ens33 for 192.168.8.100

kpslave主机查看日志
Mar 27 23:52:25 [localhost] Keepalived_vrrp[1756]: VRRP_Instance(VI_1) Received
advert with higher priority 100, ours 99
Mar 27 23:52:25 [localhost] Keepalived_vrrp[1756]: VRRP_Instance(VI_1) Entering
BACKUP STATE
Mar 27 23:52:25 [localhost] Keepalived_vrrp[1756]: VRRP_Instance(VI_1) removing
protocol VIPs.
```

```
kpmaster主机检查ip
[root@kpmaster ~]# ip addr
...
2: ens33: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 00:0c:29:9f:ea:4e brd ff:ff:ff:ff:ff:ff
    inet 192.168.8.12/24 brd 192.168.8.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet 192.168.8.100/32 scope global ens33
```

```
kpslave主机检查ip
[root@kpslave ~]# ip addr
...
2: ens33: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP group default qlen 1000
    link/ether 00:0c:29:b0:56:23 brd ff:ff:ff:ff:ff:ff
    inet 192.168.8.13/16 brd 192.168.255.255 scope global ens33
        valid_lft forever preferred_lft forever
结果显示：
keepalived可以结合内置的vrrp_script和track_script两个属性来实现自动服务检测的作用。
```

小结

对于任何场景下的服务检测功能，我们可以结合keepalived的内置属性来完成，完成步骤如下：

- 1 规划服务的检测内容
- 2 定制服务检测脚本
- 3 keepalived使用脚本检测配置
- 4 效果测试

注意

keepalived在使用脚本检测配置的时候，一定要注意：
track_script 中使用的名称，一定要在 vrrp_script 中定义
vrrp_script中script属性的尽量脚本方式，而且使用标准的"/bin/bash /path/to/file"格式

健康检测

学习目标：

这一节，我们从基础知识、简单实践、小结三个方面来学习。

基础知识

默认配置

```
virtual_server 10.10.10.2 1358 {
    delay_loop 6
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP

    sorry_server 192.168.200.200 1358

    real_server 192.168.200.2 1358 {
        weight 1
        HTTP_GET {
            url {
                path /testurl/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334d
            }
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
        }
    }
}

...
```

基本属性

delay_loop 用于服务轮询的延迟计时器

lb_algo 设定访问后端服务的调度策略

在当前的keepalived版本下，它支持7种调度算法 **rr|wrr|lc|wl|c|lbc|sh|dh**

lb_kind 设定数据转发的模型

因为是基于IPVS进行负载均衡的，所以数据转发的模型与LVS一致 **NAT|DR|TUN**

persistence_timeout 设定数据持久性的超时时间，默认为6分钟

protocol 设定数据通信的协议，默认支持三种 **TCP|UDP|SCTP**

这一项是必备的，因为我们使用**ipvsadm** 创建集群的时候，必须指定通信协议

sorry_server 备用真实主机，当所有RS失效后，开始启用该后备主机。

real_server 设定后端主机的信息

真实主机属性

weight 设定真实主机的权重，默认是1

HTTP_GET 以HTTP方式来检查后端主机

TCP_CHECK 以TCP方式来检查后端主机

SMTP_CHECK 以SMTP方式来检查后端主机

DNS_CHECK 以DNS方式来检查后端主机

MISC_CHECK 以MISC方式来检查后端主机

http检测属性

```

url {
    path <path/to/url> 指定要检查的URL的路径
    digest <STRING> 校验码验证, 校验计算: genhash -s <IP> -p <port> -u
    /path/to/url
    status_code <INT> 状态码方式验证, 推荐使用这种方法
}
注意:
由于digest方式太繁琐, 而且文件一旦变动, 老的digest就不能用了, 所以不推荐
connect_timeout 连接超时时间, 默认5s
nb_get_retry get方式获取检测url页面的重试次数
delay_before_retry 重试的间隔时间, 默认1s

```

tcp检测属性

connect_ip	测试目标的ip, 默认是RS的ip, 可省略
connect_port	测试目标的端口, 默认是RS的port, 可省略
connect_timeout	连接超时时间
retry	重试次数, 刻省略
delay_before_retry	重试间隔时间, 默认1s

简单实践

需求

在上一节的故障演练中, 尤其是后端服务的中断后, 显示"拒绝连接"报错信息, 按照我们对负载均衡的理解, 这种情况下, 当某一个后端主机出现故障, 我们就不应该给他传递数据请求了。
接下来我们的实验需求就是, 当我们检测到后端主机服务故障的时候, 数据信息就不再给他传递了。

思路

我们使用 http 或者 tcp 都是可以的

定制keepalived的属性(两台主机一致)

```

virtual_server 192.168.8.100 80 {
    delay_loop 2
    lb_algo rr
    lb_kind DR
    protocol TCP
    sorry_server 127.0.0.1 80
    real_server 192.168.8.14 80 {
        HTTP_GET {
            url {
                path /
                status_code 200
            }
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
        }
    }
    real_server 192.168.8.15 80 {
        HTTP_GET {
    }
}
```

```
        url {
            path /
            status_code 200
        }
        connect_timeout 3
        nb_get_retry 3
        delay_before_retry 3
    }
}
```

注意：

只需要 在每个real_server中添加 HTTP_GET 属性即可

重启服务

两台keepalived重启服务
systemctl restart keepalived

正常效果

查看当前的web效果
~]# for i in {1..10}; do curl 192.168.8.100;done
nginx-RS2
nginx-RS1
...
nginx-RS2
nginx-RS1

查看当前的ipvsadm规则
]# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port Forward Weight ActiveConn InActConn
TCP 192.168.8.100:80 rr
-> 192.168.8.16:80 Route 1 0 1
-> 192.168.8.17:80 Route 1 0 2

模拟后端故障

停止后端RS1的nginx服务
[root@lvs-rs1 ~]# systemctl stop nginx

查看当前的web效果
~]# for i in {1..10}; do curl 192.168.8.100;done
nginx-RS2
nginx-RS2
...
nginx-RS2

kpmaster上检查日志效果

```
Mar 28 11:29:08 [localhost] Keepalived_healthcheckers[5313]: Error connecting server [192.168.8.16]:80.
Mar 28 11:29:11 [localhost] Keepalived_healthcheckers[5313]: Error connecting server [192.168.8.16]:80.
Mar 28 11:29:14 [localhost] Keepalived_healthcheckers[5313]: Error connecting server [192.168.8.16]:80.
Mar 28 11:29:17 [localhost] Keepalived_healthcheckers[5313]: Error connecting server [192.168.8.16]:80.
Mar 28 11:29:17 [localhost] Keepalived_healthcheckers[5313]: Check on service [192.168.8.16]:80 failed after 3 retry.
Mar 28 11:29:17 [localhost] Keepalived_healthcheckers[5313]: Removing service [192.168.8.16]:80 from VS [192.168.8.100]:80
```

kpslave上检查日志效果

```
Mar 28 11:28:38 [localhost] systemd: Started LVS and VRRP High Availability Monitor.
Mar 28 11:28:38 [localhost] Keepalived_healthcheckers[5073]: Opening file '/etc/keepalived/keepalived.conf'.
Mar 28 11:28:38 [localhost] Keepalived_healthcheckers[5073]: Activating healthchecker for service [192.168.8.100]:80
Mar 28 11:28:38 [localhost] Keepalived_healthcheckers[5073]: Activating healthchecker for service [192.168.8.100]:80
Mar 28 11:28:38 [localhost] Keepalived_vrrp[5074]: VRRP_Instance(VI_1) Entering BACKUP STATE
Mar 28 11:28:38 [localhost] Keepalived_vrrp[5074]: VRRP_sockpool: [ifindex(2), proto(112), unicast(0), fd(10,11)]
Mar 28 11:29:07 [localhost] Keepalived_healthcheckers[5073]: Error connecting server [192.168.8.16]:80.
Mar 28 11:29:10 [localhost] Keepalived_healthcheckers[5073]: Error connecting server [192.168.8.16]:80.
Mar 28 11:29:13 [localhost] Keepalived_healthcheckers[5073]: Error connecting server [192.168.8.16]:80.
Mar 28 11:29:16 [localhost] Keepalived_healthcheckers[5073]: Error connecting server [192.168.8.16]:80.
Mar 28 11:29:16 [localhost] Keepalived_healthcheckers[5073]: Check on service [192.168.8.16]:80 failed after 3 retry.
Mar 28 11:29:16 [localhost] Keepalived_healthcheckers[5073]: Removing service [192.168.8.16]:80 from VS [192.168.8.100]:80
```

模拟后端恢复

恢复后端RS1的nginx服务

```
[root@lvs-rs1 ~]# systemctl start nginx
```

查看当前的web效果

```
~]# for i in {1..10}; do curl 192.168.8.100;done
```

nginx-RS2

nginx-RS1

...

nginx-RS2

nginx-RS1

结果显示：

后端服务可以自由的监控到，然后又恢复正常了

kpmaster上检查日志效果

```
Mar 28 11:37:17 [localhost] Keepalived_healthcheckers[5313]: HTTP status code success to [192.168.8.16]:80 url(1).
Mar 28 11:37:17 [localhost] Keepalived_healthcheckers[5313]: Remote Web server [192.168.8.16]:80 succeed on service.
Mar 28 11:37:17 [localhost] Keepalived_healthcheckers[5313]: Adding service [192.168.8.16]:80 to VS [192.168.8.100]:80
```

kpslave上检查日志效果

```
Mar 28 11:37:17 [localhost] Keepalived_healthcheckers[5073]: HTTP status code success to [192.168.8.16]:80 url(1).
Mar 28 11:37:17 [localhost] Keepalived_healthcheckers[5073]: Remote Web server [192.168.8.16]:80 succeed on service.
Mar 28 11:37:17 [localhost] Keepalived_healthcheckers[5073]: Adding service [192.168.8.16]:80 to VS [192.168.8.100]:80
```

小结

对于http场景下的服务检测功能，我们可以结合keepalived的HTTP检测功能来完成，完成步骤如下：

- 1 规划服务的检测内容
- 2 keepalived使用HTTP检测配置
- 3 效果测试

注意

keepalived在使用HTTP检测配置的时候，一定要注意：

url 中的检测文件路径必须准确

关于状态检测，我们推荐使用status_code的方式，不推荐使用digest方式

双主实践

学习目标：

这一节，我们从基础知识、操作实践、小结三个方面来学习。

基础知识

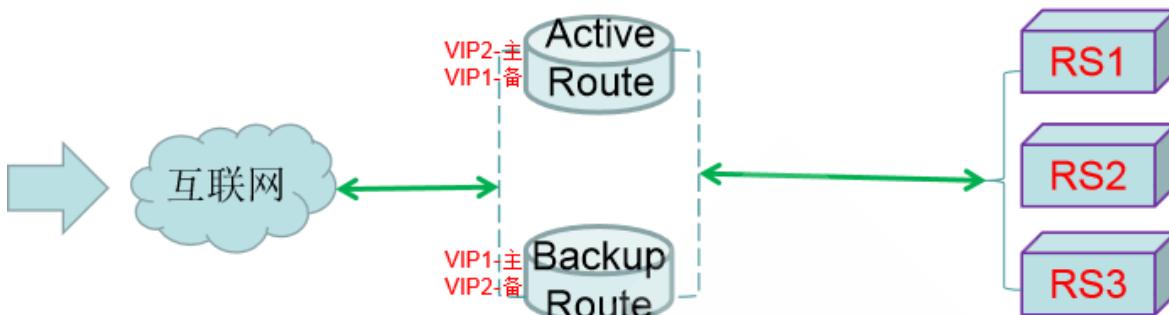
需求

在实际的工作中，我们的网站服务一般都是以域名的方式对外提供服务，对于这种情况下一般有这么两种现象：

一个域名对应一个ip地址，万一域名解析的ip地址故障，就出现单点故障现象

一个域名可以解析不同的后端服务，我们可以基于同域名解析多个不同服务的ip地址，更精确的响应用户

也就是说，我们在实现keepalived的高可用时候，需要在同一组高可用集群中，设置多个对外的VIP即多组VRRP实例，任意一组失效，都不影响用户的访问体验。



主机资源

序号	主机角色	ip信息	ip角色	网卡设备	安装软件
1	kpmaster	192.168.8.12	网卡ip	ens33-nat模型	keepalived
		192.168.8.100	VIP1-主	ens33别名	
		192.168.8.200	VIP2-备	ens33别名	
2	kpslave	192.168.8.13	网卡ip	ens33-nat模型	keepalived
		192.168.8.100	VIP1-备	ens33别名	
		192.168.8.200	VIP2-主	ens33别名	

操作实践

定制基本环境配置

后端主机配置多vip

```
echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
ifconfig lo:0 192.168.8.100 netmask 255.255.255.255 broadcast 192.168.8.100
ifconfig lo:1 192.168.8.200 netmask 255.255.255.255 broadcast 192.168.8.200
```

检查效果

```
[~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
        inet 192.168.8.100/32 brd 192.168.8.100 scope global lo:0
            valid_lft forever preferred_lft forever
        inet 192.168.8.200/32 brd 192.168.8.200 scope global lo:1
            valid_lft forever preferred_lft forever
```

结果显示：

可以看到，真实主机上有两个VIP地址，都绑定在lo网卡上。

定制keepalived配置

```
配置kpmaster 多个vrrp示例
global_defs {
    router_id kpmaster
}

vrrp_script chk_keepalived {
    script "/bin/bash /data/scripts/keepalived_check.sh"
    interval 1
}

vrrp_instance VI_1 {
    state MASTER
    interface ens33
    virtual_router_id 50
    priority 100
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.8.100 dev ens33 label ens33:0
    }
    track_script {
        chk_keepalived
    }
}

vrrp_instance VI_2 {
    state BACKUP
    interface ens33
    virtual_router_id 51
    priority 99
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.8.200 dev ens33 label ens33:1
    }
    track_script {
        chk_keepalived
    }
}
virtual_server 192.168.8.100 80 {
    ...
}
virtual_server 192.168.8.200 80 {
    ...
}

# 配置kpslave 多个vrrp实例
global_defs {
    router_id kpslave
}
```

```
vrrp_script chk_keepalived {
    script "/bin/bash /data/scripts/keepalived_check.sh"
    interval 1
}

vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 50
    priority 99
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.8.100 dev ens33 label ens33:0
    }
    track_script {
        chk_keepalived
    }
}

vrrp_instance VI_2 {
    state MASTER
    interface ens33
    virtual_router_id 51
    priority 100
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.8.200 dev ens33 label ens33:1
    }
    track_script {
        chk_keepalived
    }
}
virtual_server 192.168.8.100 80 {
    ...
}
virtual_server 192.168.8.200 80 {
    ...
}
```

保证所有依赖服务都处于开启状态

```
[root@kpmaster ~]# systemctl start httpd
[root@kpslave ~]# systemctl start httpd
[root@lvs-rs1 ~]# systemctl start nginx
[root@lvs-rs2 ~]# systemctl start nginx
```

依次启动两台keepalived主机服务
systemctl start keepalived.service

查看日志效果

结果显示：

在kpslave结点keepalived服务未启动时候，两个vrrp实例都在kpmaster主机上

当kpslave结点keepalive服务启动时候，VI_2的实例优先级较kpmaster主机高，所以将8.200的VIP枪过来了。

检查效果

检查两台主机的ip

`ip addr`

检查ipvs规则

```
[root@kpmaster ~]# ipvsadm -Ln
```

Prot	LocalAddress:Port	Scheduler	Flags	Forward	weight	ActiveConn	InActConn
->	RemoteAddress:Port						
TCP	192.168.8.100:80	rr					
->	192.168.8.16:80			Route	1	0	4
->	192.168.8.17:80			Route	1	1	4
TCP	192.168.8.200:80	rr					
->	192.168.8.16:80			Route	1	0	0
->	192.168.8.17:80			Route	1	0	0

可以看到：

这里存在两个Jvs集群，管理的内容是一致的。

访问效果

单独找一台客户端来验证效果

```
[root@client ~]# for i in {1..8}; do curl 192.168.8.100:done
```

nginx-RS1

nginx-RS2

nginx-RS1

nginx-RS2

nginx-RS1

nginx-RS2

nginx-RS1

nginx-RS2

[root@cli

nginx-RS2

nginx-RS1

nginx-RS2

nginx-RS1

nginx-RS2

nginx-RS1

nginx-RS2

```
[root@lvs-rs1 ~]# systemctl stop nginx
[root@lvs-rs2 ~]# systemctl stop nginx
查看keepalived的ipvs规则
[root@kpmaster ~]# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port           Forward weight ActiveConn InActConn
TCP  192.168.8.100:80  rr
-> 127.0.0.1:80                 Route   1      0      0
TCP  192.168.8.200:80  rr
-> 127.0.0.1:80                 Route   1      0      0
```

小结

对于多应用场景或者请求入口高可用的场景，我们可以基于vrrp多实例的样式来完成，完成步骤如下

- 1 规划服务的检测内容
- 2 后端服务的正常运行
- 3 keepalived使用配置多vrrp实例
- 4 效果测试

注意

keepalived在配置多vrrp实例的时候，一定要注意：

同一实例间的优先级和state必须写清楚

不同实例的名称和VIP一定要合理的规划