本周工作（6.24-6.29）

1.阅读 SVN 下载的 DCS 现有文档以及白皮书中相关的 DCS 文档，了解 DCS 架构以及其每一部分的架构，包括主要有六部分组成

（1） DCManageService：管理服务后台，提供创建、删除业务集群，增加节点扩容，删除节点等远程服务，由 gPassAdmin 管理台远程调用。

（2） DCAgent：通过实时轮询 Redis 节点，监控节点状态并上报给 ETCD，并依据 ETCD 中节点的状态变动执行相关的操作，如关闭或启动 Redis 节点。

（3） DCSDK：它是 DCS 对外提供的 API，业务程序通过 DCSDK 服务端访问 Redis 集群，实现 set 或 get 数据等功能；DCSDK 初始化时，会从 ETCD 中查询带集群对应的 Redis 节点相信后，后面不再与 ETCD 打交道，而是直连 redis 操作数据。

（4） ETCD：统一配置存储，存储节点和业务集群的配置信息，作为 DCSManage、DCSAgent、DCSDK 沟通的桥梁。

（5） DCConsumer：双云同步消费者，用于双云场景，消费另一个机房部署的 DCS 发过来的信息，并在当前机房还原数据。

（6） Redis：分布式缓存，用于程序数据的存储。

2.学习评估一个 redis 集群的只要指标，参考云眼实时监测的数据核对云平台分布式缓存的服务容量度量指标日报。根据已给的公式理解日报，其中

（1）TPS 水位线值=（节点 TPS 基线/max（1，昨天平均输出大小（B）/1024）/冗余倍数）*主节点数；

（2）内存容量水位线=昨天集群分配最大内存*节点内存阈值/冗余倍数；

（3）TPS 水位线比例=昨天 TPS 峰值/昨天 TPS 水位线；

（4）内存水位线比例=昨天内存使用率峰值/昨天内存水位线的取值

（5）TPS 扩容节点数=（(天数*近一周 TPS 峰值平均每日增量+昨日 TPS 峰值-TPS 水位线）/TPS 水位线）*节点个数；

（6）内存扩容节点数=（(天数*近一周集群内存峰值每日增量+昨日使用内存峰值-内存水位线值）/内存水位线值）*节点个数

3.参考 DCS 开发指南及已有代码实现一个简单的 DEMO，通过连接廊坊服务器实现向集群内 set 值和 get 值的功能。

4.通过学习 Redis 开发与运维，安装一个 Redis 集群。

（1）安装 Ruby 环境

（2）安装 rubygem redis 依赖

（3）验证 redis-trib.rb 命令确认环境是否正确

```
[root@localhost config]# redis-trib.rb
Usage: redis-trib <command> <options> <arguments ...>

  create          host1:port1 ... hostN:portN
                  --replicas <arg>
  check           host:port
  info            host:port
  fix             host:port
                  --timeout <arg>
  reshard         host:port
                  --from <arg>
                  --to <arg>
                  --slots <arg>
                  --yes
                  --timeout <arg>
                  --pipeline <arg>
  rebalance       host:port
                  --weight <arg>
                  --auto-weights
                  --use-empty-masters
                  --timeout <arg>
                  --simulate
                  --pipeline <arg>
                  --threshold <arg>
  add-node        new_host:new_port existing_host:existing_port
                  --slave
                  --master-id <arg>
  del-node        host:port node_id
  set-timeout     host:port milliseconds
  call            host:port command arg arg .. arg
  import          host:port
                  --from <arg>
                  --copy
                  --replace
```

（4）配置节点并启动

```
port 7000
daemonize yes
dir "/tmp/redis-4.0.10/data"
logfile "7000.log"
dbfilename "dump-7000.rdb"
cluster-enabled yes
cluster-config-file nodes-7000.conf
cluster-require-full-coverage no
~
```

```
[root@localhost config]# sed 's/7000/7001/g' redis-7000.cofg>redis-7001.cofg
[root@localhost config]# sed 's/7000/7002/g' redis-7000.cofg>redis-7002.cofg
```

```
[root@localhost config]# sed 's/7000/7003/g' redis-7000.cofg>redis-7003.cofg
[root@localhost config]# sed 's/7000/7004/g' redis-7000.cofg>redis-7004.cofg
[root@localhost config]# sed 's/7000/7005/g' redis-7000.cofg>redis-7005.cofg
```

```
[root@localhost config]# ps -ef | grep redis
root      22773  14090  0 07:08 pts/2    00:00:02 ./redis-server *:6379
root      22955      1  0 07:44 ?        00:00:00 redis-server *:7000 [cluster]
root      22970      1  0 07:44 ?        00:00:00 redis-server *:7001 [cluster]
root      22977      1  0 07:44 ?        00:00:00 redis-server *:7002 [cluster]
root      22984      1  0 07:45 ?        00:00:00 redis-server *:7003 [cluster]
root      22991      1  0 07:45 ?        00:00:00 redis-server *:7004 [cluster]
root      22998      1  0 07:45 ?        00:00:00 redis-server *:7005 [cluster]
root      23003  22785  0 07:45 pts/0    00:00:00 grep --color=auto redis
```

（5）创建集群

```
[root@localhost config]# redis-trib.rb create --replicas 1 127.0.0.1:7000 127.0.
0.1:7001 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005
>>> Creating cluster
>>> Performing hash slots allocation on 6 nodes...
Using 3 masters:
127.0.0.1:7000
127.0.0.1:7001
127.0.0.1:7002
Adding replica 127.0.0.1:7004 to 127.0.0.1:7000
Adding replica 127.0.0.1:7005 to 127.0.0.1:7001
Adding replica 127.0.0.1:7003 to 127.0.0.1:7002
>>> Trying to optimize slaves allocation for anti-affinity
[WARNING] Some slaves are in the same host as their master
M: c3ae0a2e67e55bc392b3bef87a2017d9ad42cc87 127.0.0.1:7000
   slots:0-5460 (5461 slots) master
M: dff4f14fd8b2d48af689901c506b89fa41c4b300 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
M: 0b5c90031a61f1894d4c6daa39d39198c1779ad6 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
S: 3b2572e086ec1358635fa4c1c399f6ed83b3a3da 127.0.0.1:7003
   replicates c3ae0a2e67e55bc392b3bef87a2017d9ad42cc87
S: 9e9fab23b10c3c98842713301cc8bcaaee8ba434 127.0.0.1:7004
   replicates dff4f14fd8b2d48af689901c506b89fa41c4b300
S: a2db10e4cf1da8f1569feb8cff0a07cb9a66a9be 127.0.0.1:7005
   replicates 0b5c90031a61f1894d4c6daa39d39198c1779ad6
Can I set the above configuration? (type 'yes' to accept): yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
```

```
Waiting for the cluster to join.....
>>> Performing Cluster Check (using node 127.0.0.1:7000)
M: c3ae0a2e67e55bc392b3bef87a2017d9ad42cc87 127.0.0.1:7000
   slots:0-5460 (5461 slots) master
   1 additional replica(s)
M: 0b5c90031a61f1894d4c6daa39d39198c1779ad6 127.0.0.1:7002
   slots:10923-16383 (5461 slots) master
   1 additional replica(s)
M: dff4f14fd8b2d48af689901c506b89fa41c4b300 127.0.0.1:7001
   slots:5461-10922 (5462 slots) master
   1 additional replica(s)
S: 3b2572e086ec1358635fa4c1c399f6ed83b3a3da 127.0.0.1:7003
   slots: (0 slots) slave
   replicates c3ae0a2e67e55bc392b3bef87a2017d9ad42cc87
S: a2db10e4cf1da8f1569feb8cff0a07cb9a66a9be 127.0.0.1:7005
   slots: (0 slots) slave
   replicates 0b5c90031a61f1894d4c6daa39d39198c1779ad6
S: 9e9fab23b10c3c98842713301cc8bcaaee8ba434 127.0.0.1:7004
   slots: (0 slots) slave
   replicates dff4f14fd8b2d48af689901c506b89fa41c4b300
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```