

产品简介

YHC(yashan health check) | 崖山深度健康检查工具

YHC(崖山深度健康检查工具)，是一款针对崖山数据库的深度检查工具。旨在提供更为专业的数据库检查报告及建议，为客户数据保障提供全面灵活的解决方案。

用户定位

- 崖山DBA

产品定位

- 轻量的独立工具
- 开箱即用

场景建议

- 性能监控和优化
- 故障排查和问题定位
- 安全审计和合规性检查
- 数据库升级和迁移前检查
- 数据库日常巡检
- 其他任何想要快速检查相关信息时

核心功能

服务器信息检查

- 服务器的基本信息(操作系统、硬件配置、防火墙等)
- 服务器的负载情况(网络流量、CPU占用、I/O负载、内存、磁盘容量等)
- 服务器的系统日志
- ...

数据库信息检查

- 数据库的基本信息 (版本、实例信息、主备信息等)
- 数据库的文件检查 (数据文件、控制文件、备份文件等)
- 数据库的对象检查 (表空间、表、索引、约束、序列等)
- 数据库的负载检查 (会话、事务、等待事件、锁等待、缓存池命中率等)
- 数据库的安全检查 (密码强度、用户权限、登录配置、默认表空间等)
- 数据库日志分析 (alert.log、run.log、redo日志等)
- ...

检查结果告警与建议

- 支持自定义指标告警表达式
- 灵活可配的阈值、告警级别以及告警建议

丰富可扩展的检查指标

- 提供了丰富的默认指标
- 支持自定义检查指标 (bash、sql)

灵活可配的检查策略

- 支持自定义时间周期，绝对灵活的时间周期选择
- 支持自定义路径数据检查，目录或文件均可批量选择
- ...

丰富健全的数据管理

- 支持自定义检查数据的存放路径
- 多种检查数据展示形式(html、docx)
- ...

规格说明

产品形态

一个独立的命令行工具

平台支持

- Linux(ubuntu/centos/kylin)

硬件架构

- x86
- arm

部署方式

- 无需部署，开箱即用

产品规格

1.仅支持本地信息检查

2.仅支持针对YashanDB SE的信息检查

参数规格

针对 `check` 子命令的参数有如下约束可供参考：

- r参数的优先级高于-s/-e
- r最大为30d，最小为1m，可以通过配置文件({yhc_home}/config/yhc.toml)修改
- s/-e 格式为yyyy-MM-dd:ss-mm，间隔最大为7d，最小为1m，可以通过配置文件({yhc_home}/config/yhc.toml)修改

功能规格

- 工具会根据指定的时间周期尝试检查服务器对应时段(默认24h内)的历史负载情况相关数据，需要依赖sar(System Activity Reporter)这个常用的系统性能监控工具，建议预装该工具否则历史负载数据无法检查，有效数据是基于sar工具运行起来之后才能被跟踪，且默认只保存30天内的数据。

快速开始

工具获取

1.登录数据库所在的服务器，访问地址获取工具包

```
http://192.168.19.121:9988/product/ycm/release/yhc/v0.1.4/
```

2.解压至对应目录(示例默认解压至当前目录)：

```
# 以 x86_64 架构的压缩包为例  
tar -zxvf yashan-health-check-v0.1.4-linux-x86_64.tar.gz
```

3.进入 `yashan-health-check-v0.1.4` 文件夹执行 `./yhctl -v` 成功即可

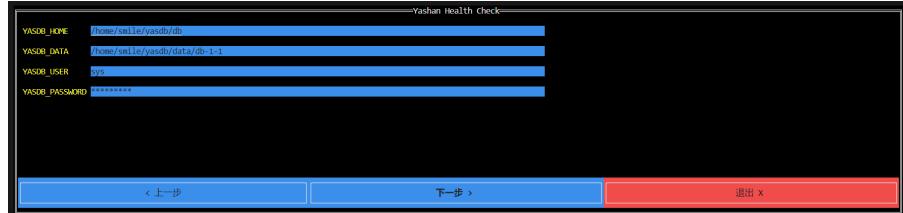
```
-bash-4.2$ ./yhctl -v  
v0.1.4
```

开始检查

强烈建议使用需要被检查的数据库进程所属用户运行该工具且给予该用户sudo权限，或使用root用户运行工具。

执行 `./yhctl check` 可以直接开始一次检查动作，所有参数不指定时，工具会默认检查24小时以内相关信息(具体检查项可移步[参考手册](#))。

1.进入前置交互页填写相关信息后，点击下一步：



- YASDB_HOME：默认填充当前环境变量YASDB_HOME的值，若无，则会尝试匹配当前环境存在的yashandb进程对应的YASDB_HOME路径填充，用户可自行修改并确认
- YASDB_DATA：默认填充当前环境变量YASDB_DATA的值，若无，则会尝试匹配当前环境存在的yashandb进程对应的YASDB_DATA路径填充，用户可自行修改并确认
- YASDB_USER：需要用户提供可访问当前数据库的用户名，至少具备 CREATE SESSION 权限，推荐使用DBA用户或者SYS用户，用户可自行修改并确认
- YASDB_PASSWORD：需要用户提供数据库YASDB_USER对应的合法密码，用户可自行修改并确认

2.工具会根据用户提供的信息检查预期检查的数据是否能被正确检查，并展示不能检查的原因，如下图：



比较常见的原因例如：

- 1.当前检查项需要root权限：是因为执行某些命令或者操作某些文件等需要root权限。
- 2.需要DBA权限：是因为对应的检查项至少需要数据库用户拥有DBA权限才能检查。
- 3.需要打开系统审计：是因为审计关闭（参数：UNIFIED_AUDITING）的状态下系统不会进行任何审计。

- 4. 用户 xxx 没有 xxx 访问权限：是因为当前用户没有操作系统文件的访问权限，若无-r-x权限则无法检查该部分信息。我们建议您使用满足权限预期的用户，运行该工具以便检查更多的信息。推荐使用：数据库的所属用户且赋予sudo权限。

3.若想继续检查则点击下一步，若想按照原因更换检查用户则可点击退出重新执行，如下图(示例为点击下一步继续执行)：



开始检查前会列出本次能够检查的各个检查项数据，包含模块、检查项、检查项告警等信息。具体检查项可移步[参考手册](#)。

默认所有检查项都勾选，如果有不想检查的项目，可以取消勾选。

4.点击下一步开始检查，观察工具检查的具体进度，等待检查完成，如下图：

```
starting yashandb health check...
OVERVIEW 100 % [=====] done
HOST_CHECK 100 % [=====] done
YASDB_CHECK 100 % [=====] done
OBJECT_CHECK 100 % [=====] done
SECURITY_CHECK 100 % [=====] done
LOG_ANALYSIS 100 % [=====] done
Yashan health check has been completed and the result was saved to /tmp/yashan-health-check/results/yhc-20231018150338.tar.gz, thanks for your use.
```

检查过程会动态更新进度，完成后会提示用户结果的存放路径，本次结果存放在 `/tmp/yashan-health-check/results/yhc-20231018150338.tar.gz`，默认检查结果会打包存放在工具执行目录的results路径下（相对路径），用户也可以使用-o参数自定义路径。

非交互模式

执行 `./yhcctl check -d` 可以开启非交互模式检查，非交互模式默认确认所有待定项目。

在非交互模式下需要通过参数指定用户以及密码，同时需要指定 `YASDB_DATA` 和 `YASDB_HOME`，如果未指定默认使用环境变量的值。

例如，在设置好 `YASDB_DATA` 和 `YASDB_HOME` 环境变量的情况下，可以执行 `./yhcctl check -d -u xxx -p xxx` 开始健康检查。

在未设置环境变量的情况下，可以执行 `./yhcctl check -d -u xxx -p xxx --yasdb-home xxx --yasdb-data xxx` 开始健康检查。

详细参数可以通过 `./yhcctl check -h` 查看。

信息查看

1.将检查到的信息包解压，会生成一个同名目录，如下：

```
# 解压文件
-bash-4.2$ tar zxvf yhc-20231018150338.tar.gz
yhc-20231018150338/
yhc-20231018150338/data/
yhc-20231018150338/data/data-20231018150338.json
yhc-20231018150338/data/report-20231018150338.json
yhc-20231018150338/data/failed-20231018150338.json
yhc-20231018150338/report-20231018150338.html
yhc-20231018150338/report-20231018150338.docx

# 查看当前目录情况
-bash-4.2$ ll
total 1196
drwxr-xr-x 3 smile smile 86 Oct 18 15:04 yhc-20231018150338
-rw-r--r-- 1 smile smile 1221824 Oct 18 15:04 yhc-20231018150338.tar.gz

# 进入yhc-20231018150338目录
-bash-4.2$ cd yhc-20231018150338/
-bash-4.2$ ll
total 3436
```

```
drwxr-xr-x 2 smile smile    106 Oct 18 15:04 data
-rw-r--r-- 1 smile smile  228794 Oct 18 15:04 report-20231018150338.docx
-rw-r--r-- 1 smile smile 3288159 Oct 18 15:04 report-20231018150338.html
```

- data：本次检查的源数据文件夹，里面包含 json 格式的原始数据，其作为生成检查报告的数据源
- report-{生成时间}.html：深度检查报告的 html 格式文件
- report-{生成时间}.docx：深度检查报告的 docx 格式文件

更多信息可参看[报告解读](#)

最佳实践

该部分为工具使用场景的最小单元拆解指导，用户可以根据实际所需场景自行整合场景单元，形成最佳实践。

- [默认检查](#)
- [指定时间范围](#)
- [指定报告存放路径](#)
- [自定义检查项](#)

参数解释

yhc目前支持一个子命令，即 `check`，以下是针对check命令的参数解释。

```
-bash-4.2$ ./yhcctl check -h
Usage: yhcctl check

The check command is used to yashan health check.

Flags:
  -h, --help           Show context-sensitive help.
  -v, --version        Show version.
  -c, --config=".~/config/yhc.toml" Configuration file.

  -r, --range=STRING      The time range of the check, such as '1M', '1d', '1h', '1m'. If <range> is given,
<start> and <end> will be discard.
  -s, --start=STRING       The start datetime of the check, such as 'yyyy-MM-dd', 'yyyy-MM-dd-hh', 'yyyy-MM-dd-hh-
mm'
  -e, --end=STRING        The end timestamp of the check, such as 'yyyy-MM-dd', 'yyyy-MM-dd-hh', 'yyyy-MM-dd-hh-
mm', default value is current datetime.
  -o, --output=STRING      The output dir of the check.
```

- `-h`: 查看check子命令的帮助信息
- `-v`: 查看yhc工具的当前版本
- `-c`: 指定工具所使用的配置文件，默认为 `{yhc_home}/config/yhc.toml`
- `-r`: 指定时间周期，默认检查从当前时间往前推24h的时间周期内数据，该配置项全局生效且优先级高于start/end参数，规格约束最大为30d，最小为1m，可以通过配置文件(`{yhc_home}/config/yhc.toml`)修改
- `-s`: 指定检查数据的开始时间点，默认为当前时间往前推24h的时间点，该配置项全局生效且优先级低于-r参数，规格约束格式 `yyyy-MM-dd-ss-mm` 且与end生效形成的时间周期最大为7d，最小为1m，可以通过配置文件(`{yhc_home}/config/yhc.toml`)修改
- `-e`: 指定检查数据的结束时间点，默认为当前时间点，该配置项全局生效且优先级低于-r参数，规格约束格式 `yyyy-MM-dd-ss-mm`，且与start生效形成的时间周期最大为30d，最小为1m，可以通过配置文件(`{yhc_home}/config/yhc.toml`)修改
- `-o`: 指定检查后的数据存放的路径，该路径至少需要具备执行工具当前用户的写权限

默认检查

直接执行 `yhcctl check` 即可

默认检查即不指定任何参数，各参数按照默认值执行，具体默认值可参看[参数解释](#)

默认检查具体执行操作指导可参看[快速开始](#)

指定时间范围

工具获取

1.登录数据库所在的服务器，访问地址获取工具包

```
http://192.168.19.121:9988/product/ycm/release/yhc/v0.1.4/
```

2.解压至对应目录(示例默认解压至当前目录)：

```
# 以 x86_64 架构的压缩包为例
tar -zxvf yashan-health-check-v0.1.4-linux-x86_64.tar.gz
```

3.进入 `yashan-health-check-v0.1.4` 文件夹执行 `./yhctl -v` 成功即可

```
-bash-4.2$ ./yhctl -v
v0.1.4
```

开始检查

以下为两种指定时间范围的方案，-r的方案优先级高于-s/-e即，若指定了-r参数，则-s/-e本次不会生效。

通过-r参数指定

- 使用示例：`-r 1d` 即一天内，`-r 1h` 即一小时内，`-r 1m` 即1分钟内
- r最大为30d，最小为1m，可以通过配置文件(`{yhc_home}/config/yhc.toml`)修改

例如，执行 `./yhctl check -r 7h` 即指定检查当前时刻往前推7个小时的数据信息

通过-s/-e参数指定

- 使用示例：`-s 2023-06-01 -e 2023-07-01` 即表示检查从6月1号至7月1号的数据信息
- s/-e 格式为`yyyy-MM-dd:ss-mm`，间隔最大为7d，最小为1m，可以通过配置文件(`{yhc_home}/config/yhc.toml`)修改
- s和-e可以单独致指定，未指定的参数按照默认值即-s为当前时间往前推24h的时刻，-e为当前时刻

例如，执行 `./yhctl check -s 2023-06-01` 即指定开始时间为23年的6月1号，而结束时间则为默认的当前时刻，但需保证时间周期长度不能超过最大规格30d，若需调整可自行修改对应配置文件(`{yhc_home}/config/yhc.toml`)

信息查看

根据工具执行检查成功之后提示的报告存放路径，找到对应的检查结果，即可查看本次检查数据的详细信息(具体报告解读可参看[报告解读](#))。

指定报告存放路径

工具获取

1. 登录数据库所在的服务器，访问地址获取工具包

```
http://192.168.19.121:9988/product/ycm/release/yhc/v0.1.4/
```

2. 解压至对应目录(示例默认解压至当前目录)：

```
# 以 x86_64 架构的压缩包为例  
tar -zxvf yashan-health-check-v0.1.4-linux-x86_64.tar.gz
```

3. 进入 `yashan-health-check-v0.1.4` 文件夹执行 `./yhctl -v` 成功即可

```
-bash-4.2$ ./yhctl -v  
v0.1.4
```

开始检查

通过-o参数指定

- o: 支持指定检查完成后的文件存放路径，若路径不存在工具会默认创建，若当前执行用户权限不足则无法继续运行

例如，执行 `./yhctl check -o /tmp/results` 即表示本次检查后的数据包将会存放在 `/tmp/results` 的路径。

使用默认路径

- 不使用-o参数指定检查完成后的文件存放路径时，将会使用配置文件中的output配置作为默认存储路径，默认保存在 `yashan-health-check` 目录下的 `results` 文件夹中

信息查看

根据工具执行检查成功之后提示的报告存放路径，找到对应的检查结果，即可查看本次检查数据的详细信息(具体报告解读可参看[报告解读](#))。

自定义检查项

工具获取

1.登录数据库所在的服务器，访问地址获取工具包

```
http://192.168.19.121:9988/product/ycm/release/yhc/v0.1.4/
```

2.解压至对应目录(示例默认解压至当前目录)：

```
# 以 x86_64 架构的压缩包为例
tar -zxvf yashan-health-check-v0.1.4-linux-x86_64.tar.gz
```

3.进入 `yashan-health-check-v0.1.4` 文件夹执行 `./yhctl -v` 成功即可

```
-bash-4.2$ ./yhctl -v
v0.1.4
```

开始配置

使用默认自定义文件

YHC提供了默认自定义指标配置文件 `{yhc_home}/config/custom_metric.toml`，可以直接在此处定义。

YHC目前支持两种类型的自定义指标：`sql` 和 `bash`。

自定义SQL指标

一个例子如下所示，定义了执行的SQL语句、列名称的别名展示以及告警配置：

```
[[metrics]]
name = "custom_check_sql"
name_alias = "自定义SQL指标"
module_name = "custom_check"
enabled = true
metric_type = "sql"
sql = "select status as instance_status, version, startup_time from v$instance;"
[metrics.column_alias]
INSTANCE_STATUS = "数据库实例状态"
STARTUP_TIME = "数据库实例启动时间"
VERSION = "数据库版本"
[metrics.item_names]
INSTANCE_STATUS = "instance_status"
[metrics.alert_rules]

[[metrics.alert_rules.critical]]
expression = "instance_status != 'OPEN'"
description = "实例状态异常"
suggestion = "建议检查实例状态"
```

如果想要了解更多关于告警表达式的用法，请移步[告警表达式](#)获取指导。

自定义Bash指标

一个例子如下所示，执行了 `uname -a` 命令，命令结果以原生字符串展示。

暂不支持解析Bash输出成表格以及Bash指标告警。

```
[[metrics]]
name = "custom_check_bash"
```

```

name_alias = "自定义BASH指标"
module_name = "custom_check"
enabled = true
command = "uname -a"
metric_type = "bash"

```

使用其他自定义文件

可以在其他文件中配置自定义指标，然后将配置文件的路径添加到 `{yhc_home}/config/yhc.toml` 中的 `metric_paths` 字段即可

例如，在 `/tmp/sql_metrics.toml` 文件中定义了新指标，可以将 `metric_paths` 字段修改为`["./config/default_metric.toml", "./config/custom_metric.toml", "/tmp/sql_metrics.toml"]`

此外，需要注意文件的访问权限，至少确保能够读取配置文件

配置报告模块

配置指标完成之后，在执行检查的时候会检查自定义指标。想要将指标展示在报告中，还需要配置好报告的模块。

打开报告模块配置文件 `{yhc_home}/config/report_module.toml`，找到自定义检查项，将新增指标的配置写入报告模块。

例如，定义了两个指标分别为 `custom_check_bash` 和 `custom_check_sql`，配置文件如下所示：

```

[[modules]]
name = "custom_check"
name_alias = "自定义检查"

[[modules.children]]
name = "custom_check_bash"
name_alias = "自定义BASH指标"
metric_names = ["bash_test"]

[[modules.children]]
name = "custom_check_sql"
name_alias = "自定义SQL指标"
metric_names = ["sql_test"]

```

验证

配置完成之后可以执行一次检查，查看自定义指标是否配置正确以及是否在报告中正确展示。

参考手册

该部分主要用来详细阐述工具内置的各项数据检查的具体实现以及检查完成后的数据包解释及工具配置项指导，以便用户更好的理解和分析检查到的数据信息及调控工具能力。

- [工具结构](#)
- [概述](#)
- [主机检查](#)
- [数据库检查](#)
- [对象检查](#)
- [安全检查](#)
- [日志分析](#)
- [报告解读](#)
- [配置文件](#)
- [告警表达式](#)

工具结构

工具获取

1.登录数据库所在的服务器，访问地址获取工具包

```
http://192.168.19.121:9988/product/ycm/release/yhc/v0.1.4/
```

2.解压至对应目录(示例默认解压至当前目录)：

```
# 以 x86_64 架构的压缩包为例
tar -zxvf yashan-health-check-v0.1.4-linux-x86_64.tar.gz
```

3.进入 `yashan-health-check-v0.1.4` 文件夹执行 `./yhctl -v` 成功即可

```
-bash-4.2$ ./yhctl -v
v0.1.4
```

工具结构

```
-bash-4.2$ cd yashan-health-check
-bash-4.2$ ll
drwxrwxr-x 2 smile smile 20 Oct 17 09:37 bin
drwxrwxr-x 2 smile smile 101 Oct 17 09:37 config
drwxrwxr-x 2 smile smile 6 Oct 17 09:37 docs
drwxrwxr-x 2 smile smile 27 Oct 17 09:37 html-template
drwxrwxr-x 2 smile smile 43 Oct 17 09:37 log
drwxrwxr-x 3 smile smile 65 Oct 18 15:08 results
drwxrwxr-x 3 smile smile 40 Oct 17 09:37 scripts
lrwxrwxrwx 1 smile smile 12 Oct 17 09:37 yhctl -> ./bin/yhctl
```

各个文件夹和文件对应的内容如下：

文件/文件夹	说明
bin	可执行二进制文件目录
config	配置文件存放目录
docs	工具文档存放目录
html-template	生成HTML报告使用的模板文件
log	工具日志存放目录
results	检查结果默认存放目录
scripts	工具脚本存放目录
yhctl	工具执行入口，软链接，指向./bin/yhctl

概述

主机概述

检查项	实现原理	备注
主机信息	调用 <code>gopsutil</code> 包的 <code>host.Info()</code>	底层调用的是各OS的基础命令
磁盘信息	调用 <code>gopsutil</code> 包的 <code>disk.Partitions()</code> 和 <code>disk.Usage()</code>	底层调用的是各OS的基础命令
磁盘块设备信息	执行 <code>lsblk -P -b</code> 命令	
内存信息	调用 <code>gopsutil</code> 包的 <code>mem.VirtualMemory()</code>	底层调用的是各OS的基础命令
网卡信息	调用 <code>gopsutil</code> 包的 <code>net.InterfacesI()</code>	底层调用的是各OS的基础命令
BIOS信息	执行 <code>dmidecode</code> 命令	需要Root权限

数据库概述

检查项	实现原理	备注
数据库信息	<p>查询数据库视图：<code>v\$instance</code>、<code>v\$database</code>、<code>v\$parameter</code>，获取数据库基本信息</p> <pre> v\$instance select status as instance_status, version, startup_time from v\$instance; v\$database select database_name, status as database_status, log_mode, open_mode, database_role, protection_mode, protection_level, create_time from v\$database; v\$parameter 监听地址 select VALUE as LISTEN_ADDR from v\$parameter where name = 'LISTEN_ADDR'; 部署形态 select count(*) as node_num from v\$parameter where value is not null and name like '%ARCHIVE_DEST%'; </pre>	如果数据库状态异常，可能无法获取部分信息
数据库文件权限	调用golang内置库 <code>filepath</code> ，实现数据库文件权限检查	底层调用的是各OS的基础命令
数据库部署形态		

主机检查

主机负载检查

检查项	实现原理	备注
CPU使用情况	<p>1.历史负载(依赖sar工具) 查看sar是否存在自定义路径配置 <code>sar_dir : /etc/sysconfig/sysstat</code> 若不存在则使用各os默认的路径执行(ubuntu: <code>/var/log/sysstat</code> ; 其他: <code>/var/log/sa</code>) 执行具体查询命令 : <code>sar -u -f {sar_dir}/{对应日期的sa文件} -s 03:41:57 -e 15:41:57</code></p> <p>2.当前负载 若sar存在，则通过sar查询当前网络负载情况，例如执行 <code>sar -u 1 10</code> 每隔1s查询一次总共查询10次 若sar不存在，则通过gopsutil的net.INCounters()查询当前网络负载情况，每隔1s查询一次总共查询10次后计算得出近似的负载情况</p>	<p>1.当sar未安装时无法获取历史负载数据，仅能算出当前负载数据 2.gopsutil是由程序自定义的算法计算得出会存在略微的偏差，仅供参考 3.间隔次数及时长均可通过配置文件调整</p>
磁盘使用情况	<p>1.历史负载(依赖sar工具) 查看sar是否存在自定义路径配置 <code>sar_dir : /etc/sysconfig/sysstat</code> 若不存在则使用各os默认的路径执行(ubuntu: <code>/var/log/sysstat</code> ; 其他: <code>/var/log/sa</code>) 执行具体查询命令 : <code>sar -d -f {sar_dir}/{对应日期的sa文件} -s 03:41:57 -e 15:41:57</code></p> <p>2.当前负载 若sar存在，则通过sar查询当前网络负载情况，例如执行 <code>sar -d 1 10</code> 每隔1s查询一次总共查询10次 若sar不存在，则通过gopsutil的net.INCounters()查询当前网络负载情况，每隔1s查询一次总共查询10次后计算得出近似的负载情况</p>	<p>1.当sar未安装时无法获取历史负载数据，仅能算出当前负载数据 2.gopsutil是由程序自定义的算法计算得出会存在略微的偏差，仅供参考 3.间隔次数及时长均可通过配置文件调整</p>
内存使用情况	<p>1.历史负载(依赖sar工具) 查看sar是否存在自定义路径配置 <code>sar_dir : /etc/sysconfig/sysstat</code> 若不存在则使用各os默认的路径执行(ubuntu: <code>/var/log/sysstat</code> ; 其他: <code>/var/log/sa</code>) 执行具体查询命令 : <code>sar -r -f {sar_dir}/{对应日期的sa文件} -s 03:41:57 -e 15:41:57</code></p> <p>2.当前负载 若sar存在，则通过sar查询当前网络负载情况，例如执行 <code>sar -r 1 10</code> 每隔1s查询一次总共查询10次 若sar不存在，则通过gopsutil的net.INCounters()查询当前网络负载情况，每隔1s查询一次总共查询10次后计算得出近似的负载情况</p>	<p>1.当sar未安装时无法获取历史负载数据，仅能算出当前负载数据 2.gopsutil是由程序自定义的算法计算得出会存在略微的偏差，仅供参考 3.间隔次数及时长均可通过配置文件调整</p>
网络使用情况	<p>1.历史负载(依赖sar工具) 查看sar是否存在自定义路径配置 <code>sar_dir : /etc/sysconfig/sysstat</code> 若不存在则使用各os默认的路径执行(ubuntu: <code>/var/log/sysstat</code> ; 其他: <code>/var/log/sa</code>) 执行具体查询命令 : <code>sar -n DEV -f {sar_dir}/{对应日期的sa文件} -s 03:41:57 -e 15:41:57</code></p> <p>2.当前负载 若sar存在，则通过sar查询当前网络负载情况，例如执行 <code>sar -n DEV 1 10</code> 每隔1s查询一次总共查询10次 若sar不存在，则通过gopsutil的net.INCounters()查询当前网络负载情况，每隔1s查询一次总共查询10次后计算得出近似的负载情况</p>	<p>1.当sar未安装时无法获取历史负载数据，仅能算出当前负载数据 2.gopsutil是由程序自定义的算法计算得出会存在略微的偏差，仅供参考 3.间隔次数及时长均可通过配置文件调整</p>
防火墙状态	<p>1.Ubuntu系统，执行执行 <code>ufw status</code> 命令 2.其他系统，执行 <code>systemctl is-active firewalld</code> 命令</p>	
端口打开情况	执行 <code>iptables -L</code> 命令	

数据库检查

主备检查

检查项	实现原理	备注
数据库主备连接状态	查询 v\$replication_status 视图： select connection, status, peer_role, peer_addr, transport_lag, apply_lag from v\$replication_status;	

数据库配置检查

检查项	实现原理	备注
数据库参数检查	查询 v\$parameter 视图： select name, value from v\$parameter where value is not null;	
OS认证用户	1.操作系统身份认证开关 查询 \${YASDB_DATA}/config/yasdb_net.ini 配置文件中的 ENABLE_LOCAL_OSAUTH 配置 2.YASDBA组中用户 解析 /etc/group 文件内容，找出目标用户组中的用户	

表空间检查

检查项	实现原理	备注
表空间	查询 dba_tablespaces 视图： SELECT TABLESPACE_NAME, CONTENTS, STATUS, ALLOCATION_TYPE , TOTAL_BYTES - USER_BYTES AS USED_BYTES, TOTAL_BYTES, (TOTAL_BYTES - USER_BYTES) / TOTAL_BYTES * 100 AS USED_RATE FROM SYS.DBA_TABLESPACES;	
数据文件	查询 dba_data_files 视图： select * from dba_data_files 调用 os.Stat() 获取数据文件权限信息	

控制文件检查

检查项	实现原理	备注
控制文件	查询 v\$controlfile 视图： select id, name, bytes/1024/1024 as MBytes from v\$controlfile;	
控制文件数量	查询 v\$controlfile 视图： select count(*) as total from v\$controlfile;	

备份检查

检查项	实现原理	备注

检查项	实现原理	备注
备份记录	<p>查询 dba_backup_set 视图 :</p> <pre>select RECID# as RECID, START_TIME, TYPE, decode(COMPLETION_TIME > sysdate, FALSE, TRUE) as SUCCESS from dba_backup_set;</pre>	
近十天完成的全量备份	<p>查询 dba_backup_set 视图 :</p> <pre>select count(*) as TOTAL from dba_backup_set where date_add(COMPLETION_TIME , INTERVAL 10 DAY) >= sysdate AND type = 'FULL';</pre>	
本地备份集路径	<p>查询 dba_backup_set 视图 :</p> <pre>"select distinct(PATH) as PATH from dba_backup_set;</pre> <p>调用 os.Stat() 获取备份文件是否存在</p>	

负载检查

检查项	实现原理	备注
会话数检查	<p>查询 v\$session 视图 :</p> <pre>select * from v\$session;</pre>	
共享池信息	<p>查询 v\$sgastat 视图 :</p> <pre>select NAME, BYTES from v\$sgastat WHERE POOL='SHARE POOL';</pre>	

归档日志检查

检查项	实现原理	备注
数据库归档情况	<p>查询 v\$archived_log, V\$PARAMETER, V\$SYSTEM_PARAMETER 视图 :</p> <pre>WITH value_stats AS (SELECT SUM(BLOCK_SIZE * BLOCKS) AS total_blocks FROM V\$ARCHIVED_LOG), arch_clean_upper_threshold AS (SELECT CASE WHEN VALUE LIKE '%T' THEN TRIM(TRAILING 'T' FROM VALUE) * 1024 * 1024 * 1024 * 1024 WHEN VALUE LIKE '%G' THEN TRIM(TRAILING 'G' FROM VALUE) * 1024 * 1024 * 1024 WHEN VALUE LIKE '%M' THEN TRIM(TRAILING 'M' FROM VALUE) * 1024 * 1024 WHEN VALUE LIKE '%K' THEN TRIM(TRAILING 'K' FROM VALUE) * 1024 WHEN REGEXP_LIKE(VALUE, '^[-]+[0-9]*\$') = TRUE THEN TO_NUMBER(VALUE) END AS value FROM V\$PARAMETER WHERE name = 'ARCH_CLEAN_UPPER_THRESHOLD'), usable_pct AS (SELECT TO_CHAR(100 * (b.total_blocks / a.value), '99.99') AS USABLE_PCT FROM arch_clean_upper_threshold a, value_stats b), space_limit AS (SELECT round(value/1024/1024/1024,2) AS SPACE_LIMIT FROM arch_clean_upper_threshold), number_of_files AS (SELECT COUNT(1) AS NUMBER_OF_FILES FROM V\$ARCHIVED_LOG), space_used AS (SELECT ROUND(SUM(BLOCK_SIZE * BLOCKS)/1024/1024/1024,2) AS SPACE_USED FROM V\$ARCHIVED_LOG), space_reclaimable AS (SELECT ROUND(SUM(b.size)/1024/1024/1024,2) AS SPACE_RECLAIMABLE FROM (SELECT REGEXP_SUBSTR(RCY_POINT, '[-]+', 1, 2, 'i') AS value FROM v\$database) a,(SELECT SEQUENCE# AS value, BLOCK_SIZE * BLOCKS AS size FROM V\$ARCHIVED_LOG) b WHERE a.value > b.value), archive_dest AS (select value AS ARCHIVE_DEST from V\$SYSTEM_PARAMETER where name='ARCHIVE_LOCAL_DEST') select USABLE_PCT,SPACE_LIMIT,NUMBER_OF_FILES,SPACE_USED,SPACE_RECLAIMABLE,ARCHIVE_DEST FROM usable_pct,space_limit,number_of_files,space_used,space_reclaimable,archive_dest;</pre>	
数据库归档日志	<p>查询 v\$archived_log 视图 :</p> <pre>select NAME, SEQUENCE# AS SEQUENCE, to_char(FIRST_TIME, 'yyyy-mm-dd hh24:mi:ss') as FIRST_TIME, to_char(NEXT_TIME, 'yyyy-mm-dd hh24:mi:ss') as NEXT_TIME, to_char(COMPLETION_TIME, 'yyyy-mm-dd hh24:mi:ss') as COMPLETION_TIME, BLOCKS, BLOCK_SIZE, COMPRESSED, FAL from v\$archived_log;</pre>	

性能分析

检查项	实现原理	备注
VM 转换率	<p>查询 v\$vm 、 v\$sysstat 视图 :</p> <pre>SELECT t1.SWAPPED_OUT_BLOCKS / t2.value AS RATE FROM (SELECT SWAPPED_OUT_BLOCKS FROM v\$vm) t1, (SELECT value FROM V\$SYSSTAT WHERE NAME = 'VM ALLOC') t2;</pre>	
TOP SQL	<p>1.平均耗时 TOP10 SQL</p> <pre>SELECT round(CPU_TIME / 1000, 2) AS CPU_TIME, EXECUTIONS , round(ELAPSED_TIME / 1000, 2) AS ALL_ELAPSED_TIME , round(ELAPSED_TIME / 1000 / EXECUTIONS, 2) AS AVG_TIME , to_char(LAST_ACTIVE_TIME, 'YYYY-MM-DD HH24:MI:SS') AS LAST_TIME, SQL_ID , SQL_TEXT FROM v\$sqlarea WHERE EXECUTIONS > 0 ORDER BY round(ELAPSED_TIME / 1000 / EXECUTIONS, 2) DESC LIMIT 10;</pre> <p>2.从缓存区获取Buffer次数 TOP10 SQL</p> <pre>SELECT BUFFER_GETS, EXECUTIONS , round(BUFFER_GETS / EXECUTIONS, 2) AS GETS_PER_EXEC , round(ELAPSED_TIME / 1000, 2) AS ALL_ELAPSED_TIME , to_char(LAST_ACTIVE_TIME, 'YYYY-MM-DD HH24:MI:SS') AS LAST_TIME, SQL_ID , SQL_TEXT FROM v\$sqlarea WHERE EXECUTIONS > 0 ORDER BY BUFFER_GETS DESC LIMIT 10;</pre> <p>3.磁盘读取次数 TOP10 SQL</p> <pre>SELECT DISK_READS, EXECUTIONS , round(DISK_READS / EXECUTIONS, 2) AS READS_PER_EXEC , round(ELAPSED_TIME / 1000, 2) AS ALL_ELAPSED_TIME , to_char(LAST_ACTIVE_TIME, 'YYYY-MM-DD HH24:MI:SS') AS LAST_TIME, SQL_ID , SQL_TEXT FROM v\$sqlarea WHERE EXECUTIONS > 0 ORDER BY DISK_READS DESC LIMIT 10;</pre> <p>4.磁盘解析次数 TOP10 SQL</p> <pre>SELECT PARSE_CALLS, EXECUTIONS , round(PARSE_CALLS / EXECUTIONS, 2) AS CALLS_PER_EXEC , round(ELAPSED_TIME / 1000, 2) AS ALL_ELAPSED_TIME , to_char(LAST_ACTIVE_TIME, 'YYYY-MM-DD HH24:MI:SS') AS LAST_TIME, SQL_ID , SQL_TEXT FROM v\$sqlarea WHERE EXECUTIONS > 0 ORDER BY round(ELAPSED_TIME / 1000 / EXECUTIONS, 2) DESC LIMIT 10;</pre>	
高频率 SQL	<p>查询 v\$sql1 视图 :</p> <pre>select SQL_ID, SQL_TEXT, PLSQL_EXEC_TIME, EXECUTIONS from v\$sql where EXECUTIONS >= 10000;</pre>	
数据库历史负载	<p>查询 sys.wrm\$_.database_instance 、 sys.wrh\$_.sysstat 、 sys.wrm\$_.snapshot 视图 :</p> <pre>WITH dbinfo AS (SELECT DISTINCT dbid FROM sys.wrm\$_.database_instance LIMIT 1), t1 AS (SELECT snap_id, value FROM SYS.wrh\$_.sysstat, dbinfo WHERE SYS.wrh\$_.sysstat.dbid = dbinfo.dbid AND stat_id = 604), t2 AS (SELECT snap_id, begin_interval_time + (end_interval_time - begin_interval_time) / 2 AS snap_time FROM SYS.wrm\$_.snapshot, dbinfo WHERE SYS.wrm\$_.snapshot.dbid = dbinfo.dbid) SELECT t1.value AS db_times, to_char(t2.snap_time, 'YYYY-MM-DD HH24:MI:SS') AS snap_time FROM t1, t2 WHERE t1.snap_id = t2.snap_id AND t2.snap_time >= TIMESTAMP('%s') AND t2.snap_time <= TIMESTAMP('%s');</pre> <p>先通过上述SQL查询出快照事件以及DBTime递增值，在计算两个时间的中间值和DBTime之前的差值得到最终结果</p>	
内存池命中率	<p>1.当前内存命中率</p> <p>查询 v\$sysstat 视图 :</p> <pre>select (sum(decode(NAME, 'BUFFER GETS', VALUE, 0)) + sum(decode(NAME, 'BUFFER CR GETS', VALUE, 0))) - sum(decode(NAME, 'DISK READS', VALUE, 0))) / (sum(decode(NAME, 'BUFFER GETS', VALUE, 0)) + sum(decode(NAME, 'BUFFER CR GETS', VALUE, 0))) * 100 AS HIT_RATE FROM v\$sysstat;</pre> <p>2.历史内存命中率</p> <p>查询 sys.wrm\$_.database_instance 、 sys.wrh\$_.sysstat 、 sys.wrm\$_.snapshot 视图 :</p> <pre>WITH dbinfo AS (SELECT DISTINCT dbid FROM SYS.wRM\$_.database_instance LIMIT 1), dbstat AS (SELECT snap_id, value, stat_id FROM SYS.wrh\$_.sysstat, dbinfo WHERE SYS.wrh\$_.sysstat.dbid = dbinfo.dbid), t1 AS (SELECT snap_id, value AS b_cr_get FROM dbstat WHERE stat_id = 120), t2 AS (SELECT snap_id, value AS b_buf_get FROM dbstat WHERE stat_id = 121), t3 AS (SELECT snap_id, value AS e_phys_read FROM dbstat WHERE stat_id = 131), t4 AS (SELECT t1.snap_id , (t1.b_cr_get + t2.b_buf_get) / (t1.b_cr_get + t2.b_buf_get + t3.e_phys_read) * 100 AS hit_rate FROM t1 JOIN t2 ON t1.snap_id = t2.snap_id JOIN t3 ON t1.snap_id = t3.snap_id), t5 AS (SELECT snap_id, begin_interval_time + (end_interval_time - begin_interval_time) / 2 AS snap_time FROM SYS.wrm\$_.snapshot, dbinfo WHERE SYS.wrm\$_.snapshot.dbid = dbinfo.dbid) SELECT to_char(t5.snap_time, 'YYYY-MM-DD HH24:MI:SS') AS snap_time, t4.hit_rate FROM t4 JOIN t5 ON t4.snap_id = t5.snap_id WHERE t5.snap_time >= TIMESTAMP('%s') AND t5.snap_time <= TIMESTAMP('%s') ORDER BY t5.snap_time;</pre>	
性能配置检查	<p>1.标准大页</p> <p>执行命令 : grep -oP '(?<=[\[]).+(?=])' /sys/kernel/mm/transparent_hugepage/enabled</p> <p>2.Swap内存</p> <p>调用 gopstui 包的 mem.SwapMemory()</p>	
锁等待	<p>1.行锁等待</p> <pre>select count(*) as ROW_LOCK_WAIT_COUNT from v\$lock where REQUEST in ('ROW');</pre> <p>2.表锁等待</p> <pre>select count(*) as TABLE_LOCK_WAIT_COUNT from v\$lock where REQUEST in ('TS', 'TX');</pre>	

检查项	实现原理	备注
长事务	查询 v\$transaction 、 v\$session 视图： <pre>select t.XID, to_char(t.START_DATE, 'yyyy-mm-dd hh24:mi:ss') as START_DATE, t.STATUS , t.RESIDUAL, s.USERNAME, t.SID, t.USED_UBLK from v\$transaction t, v\$session s where t.START_DATE < sysdate - 3 / 24 and t.SID = s.SID;</pre>	

对象检查

对象数量统计

检查项	实现原理	备注
对象总数	1.对象总数 <pre>select count(*) as total_count from dba_objects;</pre> 2.各owner对象统计 <pre>SELECT owner, object_type, COUNT(*) AS owner_object_count FROM dba_objects WHERE owner NOT IN ('SYS', 'SYSTEM') AND object_type NOT LIKE 'BIN\$%' GROUP BY owner, object_type ORDER BY owner,object_type;</pre> 3.各tablespace对象统计 <pre>SELECT tablespace_name, COUNT(*) AS tablespace_object_count FROM dba_segments WHERE segment_type IN ('TABLE', 'INDEX', 'VIEW', 'SEQUENCE') GROUP BY tablespace_name ORDER BY tablespace_name;</pre>	

对象状态检查

检查项	实现原理	备注
失效对象列表	查询 dba_objects 视图： <pre>select OBJECT_ID, OWNER, OBJECT_NAME, OBJECT_TYPE, STATUS from dba_objects where STATUS = 'INVALID';</pre>	
不可见索引列表	查询 dba_indexes 视图： <pre>select OWNER, INDEX_NAME, VISIBILITY from dba_indexes where VISIBILITY !='VISIBLE';</pre>	
不可用约束列表	查询 dba_constraints 视图： <pre>select OWNER, CONSTRAINT_NAME, CONSTRAINT_TYPE, STATUS from dba_constraints where STATUS ='DISABLED';</pre>	

表

检查项	实现原理	备注
字段过多的表	查询 dba_tab_cols 视图： <pre>select OWNER, TABLE_NAME, count(*) as COLUMN_COUNT from dba_tab_cols group by OWNER, TABLE_NAME having count(*)>80;</pre>	
索引过多的表	查询 dba_indexes 视图： <pre>select TABLE_OWNER, TABLE_NAME, count(*) as INDEX_COUNT from dba_indexes group by TABLE_OWNER, TABLE_NAME having count(*) >8;</pre>	

检查项

实现原理

备注

没有分区索引的分区表

查询 dba_indexes 、 dba_part_key_columns 、 dba_part_tables 视图：

```

SELECT b.OWNER ,b.name,a.PARTITIONING_TYPE ,b.tab_cols from (SELECT owner,TABLE_NAME,PARTITIONING_TYPE FROM
DBA_PART_TABLES) a, (SELECT OWNER ,NAME ,LISTAGG(COLUMN_NAME,',') WITHIN group(ORDER BY COLUMN_POSITION) AS
tab_cols FROM DBA_PART_KEY_COLUMNS WHERE OBJECT_TYPE ='TABLE' GROUP BY OWNER ,NAME ) b WHERE a.OWNER = b.owner AND
a.TABLE_NAME =b.name AND a.owner<>'SYS' minus (WITH t1 AS ( SELECT b.OWNER ,b.name,a.PARTITIONING_TYPE ,b.tab_cols
from (SELECT owner,TABLE_NAME,PARTITIONING_TYPE FROM DBA_PART_TABLES) a, (SELECT OWNER ,NAME ,
LISTAGG(COLUMN_NAME,',') WITHIN group(ORDER BY COLUMN_POSITION) AS tab_cols FROM DBA_PART_KEY_COLUMNS WHERE
OBJECT_TYPE ='TABLE' GROUP BY OWNER ,NAME ) b WHERE a.OWNER = b.owner AND a.TABLE_NAME =b.name ), t2 AS ( SELECT
t2.INDEX_OWNER ,t2.INDEX_NAME,t2.TABLE_OWNER ,t2.TABLE_NAME,t2.ind_cols FROM (SELECT OWNER ,INDEX_NAME ,TABLE_OWNER
,TABLE_NAME FROM DBA_INDEXES WHERE PARTITIONED ='Y') t1, (SELECT INDEX_OWNER ,INDEX_NAME ,TABLE_OWNER
,TABLE_NAME ,LISTAGG(COLUMN_NAME,',') WITHIN group(ORDER BY COLUMN_POSITION) AS ind_cols FROM DBA_IND_COLUMNS WHERE
INDEX_OWNER <>'SYS1' GROUP BY INDEX_OWNER ,INDEX_NAME ,TABLE_OWNER ,TABLE_NAME) t2 WHERE t1.OWNER=t2.INDEX_OWNER AND
t1.INDEX_NAME=t2.INDEX_NAME AND t1.TABLE_OWNER=t2.TABLE_OWNER AND t1.TABLE_NAME=t2.TABLE_NAME ) SELECT t1.OWNER
,t1.name,t1.PARTITIONING_TYPE ,t1.tab_cols FROM t1,t2 WHERE t1.OWNER=t2.TABLE_OWNER AND t1.name=t2.TABLE_NAME AND
t1.tab_cols=t2.ind_cols);

```

行大小超过块大小的表

查询 dba_tab_columns 视图：

```

SELECT a.OWNER ,a.TABLE_NAME FROM ( SELECT OWNER , TABLE_NAME , SUM(DATA_LENGTH) AS MAX_DL FROM DBA_TAB_COLUMNS
WHERE OWNER <> 'SYS' AND DATA_TYPE NOT LIKE '%LOB' GROUP BY OWNER , TABLE_NAME ) a, ( SELECT to_number(decode(value,
'8K', '8192', '16K', '16384', '32K', '32768', value)) AS VALUE FROM v$parameter WHERE NAME = 'DB_BLOCK_SIZE' ) b
WHERE a.max_dl > b.value;

```

哈希分区数量不是2的幂的分区表

查询 dba_part_tables 视图：

```

select OWNER,TABLE_NAME,PARTITIONING_TYPE,PARTITION_COUNT from dba_part_tables where PARTITIONING_TYPE ='HASH' and
abs(floor(log(2, PARTITION_COUNT)))!=log(2, PARTITION_COUNT) or log(2, PARTITION_COUNT)='Nan';

```

约束

检查项

实现原理

备注

检查项

实现原理

备注

无索引的子表外键

查询 dba_constraints、all_cons_columns、dba_ind_columns、dba_indexes 视图：

```
WITH t1 AS ( SELECT owner, CONSTRAINT_NAME, TABLE_NAME , LISTAGG(COLUMN_NAME, ',') WITHIN GROUP ( ORDER BY posi) AS col_lst FROM ( SELECT b.OWNER, b.CONSTRAINT_NAME, b.TABLE_NAME, b.COLUMN_NAME, b.posi FROM ( SELECT OWNER, CONSTRAINT_NAME, TABLE_NAME FROM DBA_CONSTRAINTS WHERE CONSTRAINT_TYPE = 'R' ) a, ( SELECT b.OWNER, b.CONSTRAINT_NAME, b.TABLE_NAME, b.COLUMN_NAME, b."POSITION" AS posi FROM ALL_CONS_COLUMNS b ) b WHERE a.owner = b.OWNER AND a.CONSTRAINT_NAME = b.CONSTRAINT_NAME AND a.table_name = b.TABLE_NAME ) GROUP BY owner, CONSTRAINT_NAME, TABLE_NAME ), t2 AS ( SELECT INDEX_OWNER, INDEX_NAME, TABLE_OWNER, TABLE_NAME , LISTAGG(COLUMN_NAME, ',') WITHIN GROUP ( ORDER BY COLUMN_POSITION) AS ind_lst FROM DBA_IND_COLUMNS GROUP BY INDEX_OWNER, INDEX_NAME, TABLE_OWNER, TABLE_NAME ) SELECT DISTINCT t1.owner, t1.CONSTRAINT_NAME, t1.TABLE_NAME, t1.col_lst FROM t1, t2 WHERE t1.owner <> 'SYS' AND t1.OWNER = t2.TABLE_OWNER AND t1.TABLE_NAME = t2.TABLE_NAME AND t1.col_lst <> t2.ind_lst;
```

数据类型隐式转换的外键

查询 dba_constraints、db_cons_columns、dba_tab_columns 视图：

```
WITH t1 AS ( SELECT b.OWNER, b.CONSTRAINT_NAME, b.TABLE_NAME, b.COLUMN_NAME AS CHD_COL, b.posi , c.DATA_TYPE AS CHD_TYP FROM ( SELECT OWNER, CONSTRAINT_NAME, TABLE_NAME FROM DBA_CONSTRAINTS WHERE CONSTRAINT_TYPE = 'R' ) a, ( SELECT b.OWNER, b.CONSTRAINT_NAME, b.TABLE_NAME, b.COLUMN_NAME, b."POSITION" AS posi FROM DBA_CONS_COLUMNS b ) b, ( SELECT OWNER, TABLE_NAME, COLUMN_NAME, DATA_TYPE FROM DBA_TAB_COLUMNS ) c WHERE a.owner = b.OWNER AND a.CONSTRAINT_NAME = b.CONSTRAINT_NAME AND a.table_name = b.TABLE_NAME AND b.OWNER = c.OWNER AND b.TABLE_NAME = c.TABLE_NAME AND b.COLUMN_NAME = c.COLUMN_NAME ), t2 AS ( SELECT DISTINCT A.FK_OWNER, A.FK_CON_NAME, A.CHD_TAB, B.PRT_OWNER, B.PRT_CON_NAME , B.PRT_TAB, B.COLUMN_NAME AS PRT_COL, C.DATA_TYPE AS PRT_TYP, B.posi FROM ( SELECT OWNER AS FK_OWNER, CONSTRAINT_NAME AS FK_CON_NAME, TABLE_NAME AS CHD_TAB, R_OWNER, R_CONSTRAINT_NAME FROM DBA_CONSTRAINTS WHERE CONSTRAINT_TYPE = 'R' ) a, ( SELECT owner AS PRT_OWNER, CONSTRAINT_NAME AS PRT_CON_NAME, TABLE_NAME AS PRT_TAB, COLUMN_NAME, 'POSITION' AS posi FROM DBA_CONS_COLUMNS ) b, ( SELECT OWNER, TABLE_NAME, COLUMN_NAME, DATA_TYPE FROM DBA_TAB_COLUMNS ) c WHERE a.R_OWNER = b.PRT_OWNER AND a.R_CONSTRAINT_NAME = b.PRT_CON_NAME AND B.PRT_OWNER = c.OWNER AND B.PRT_TAB = C.TABLE_NAME AND B.COLUMN_NAME = C.COLUMN_NAME ) SELECT t2.FK_OWNER, t2.FK_CON_NAME, t2.CHD_TAB, t1.CHD_COL, t1.CHD_TYP , t2.PRT_OWNER, t2.PRT_CON_NAME, t2.PRT_TAB, t2.PRT_COL, T2.PRT_TYP FROM t1, t2 WHERE t1.owner = t2.FK_OWNER AND t1.CONSTRAINT_NAME = t2.FK_CON_NAME AND t1.TABLE_NAME = t2.CHD_TAB AND t1.posi = t2.posi AND t1.CHD_TYP <> t2.PRT_TYP;
```

索引

超过三层的索引

查询 dba_indexes 视图：

```
select OWNER, INDEX_NAME, BLEVEL from dba_indexes where BLEVEL>3;
```

字段过多的索引

查询 dba_ind_columns 视图：

```
select INDEX_OWNER, INDEX_NAME, count(*) as column_count from dba_ind_columns group by INDEX_OWNER,INDEX_NAME having count(*) > 10;
```

不可见索引

查询 dba_indexes 视图：

```
select OWNER, INDEX_NAME, TABLE_OWNER, TABLE_NAME FROM dba_indexes where owner<> 'SYS' and VISIBILITY <> 'VISIBLE';
```

过大的索引

查询 dba_segments 视图：

```
SELECT ind.OWNER AS ind_owner,ind SEGMENT_NAME AS ind_name,ind SEGMENT_TYPE as IND_SEGMENT_TYPE ,tab.SEGMENT_TYPE as TAB_SEGMENT_TYPE,tab.OWNER AS tab_owner ,tab.SEGMENT_NAME AS tab_name,ind.BYTES AS ind_bytes,tab.BYTES AS tab_bytes FROM DBA_SEGMENTS ind,DBA_SEGMENTS tab,DBA_INDEXES di WHERE IND.SEGMENT_TYPE IN ('INDEX','INDEX PARTITION') AND tab.SEGMENT_TYPE IN ('TABLE','TABLE PARTITION') AND ind.OWNER =di.OWNER AND ind.SEGMENT_NAME =di.INDEX_NAME AND tab.OWNER =di.TABLE_OWNER AND tab.SEGMENT_NAME =di.TABLE_NAME AND ind.BYTES > tab.BYTES;
```

表和索引不在一个 schema 下

查询 dba_indexes 视图：

```
SELECT OWNER,INDEX_NAME ,TABLE_OWNER ,TABLE_NAME FROM dba_indexes WHERE OWNER <> TABLE_OWNER;
```

序列

检查项	实现原理	备注
无可用值的序列	<p>查询 dba_sequences 视图 :</p> <pre>SELECT SEQUENCE_OWNER ,SEQUENCE_NAME ,LAST_NUMBER / MAX_VALUE * 100 as USED_RATE FROM DBA_SEQUENCES ds WHERE LAST_NUMBER / MAX_VALUE > 7/10;</pre>	

任务

检查项	实现原理	备注
运行中的任务	<p>查询 dba_scheduler_jobs 视图 :</p> <pre>select OWNER ,JOB_NAME ,JOB_STYLE ,JOB_CREATOR ,JOB_ACTION from DBA_SCHEDULER_JOBS where STATE='RUNNING';</pre>	

包

检查项	实现原理	备注
没有package的 package body	<p>查询 dba_source 视图 :</p> <pre>SELECT OWNER ,NAME, JOIN_STR FROM (SELECT OWNER ,NAME, LISTAGG(TYPE, '-') AS JOIN_STR FROM DBA_SOURCE GROUP by OWNER ,NAME) WHERE JOIN_STR<>'PACKAGE-PACKAGE BODY';</pre>	

安全检查

登录检查

检查项	实现原理	备注
密码强度控制	查询 x\$parameter 视图： SELECT VALUE FROM x\$parameter WHERE name = '_CHECK_PASSWORD_COMPLEXITY';	
未限制登录次数的Profile	查询 dba_profiles 视图： select PROFILE,RESOURCE_NAME ,RESOURCE_TYPE, LIMIT from DBA_PROFILES where PROFILE<>'DEFAULT' and RESOURCE_NAME='FAILED_LOGIN_ATTEMPTS' and LIMIT='UNLIMITED';	

用户与权限检查

检查项	实现原理	备注
非OPEN状态的用户	查询 dba_users 视图： select username,ACCOUNT_STATUS from dba_users where ACCOUNT_STATUS!= 'OPEN';	
拥有系统表权限的用户	查询 dba_users 、 dba_tab_privs 视图： select GRANTEE , TABLE_NAME from DBA_TAB_PRIVS where OWNER='SYS' and TYPE='TABLE' and GRANTEE in (select username from dba_users);	
所有DBA角色的用户	查询 dba_role_privs 视图： select GRANTEE from dba_role_privs where GRANTED_ROLE='DBA';	
拥有ALL PRIVILEGE SYSTEM的用户	查询 dba_users 、 dba_sys_privs 视图： select GRANTEE from dba_sys_privs where PRIVILEGE='ALL PRIVILEGES' AND GRANTEE IN (SELECT USERNAME FROM DBA_USERS);	
以SYSTEM表空间为默认 表空间的用户	查询 dba_users 视图： select username,default_tablespace from dba_users where default_tablespace = 'SYSTEM';	

审计定时清理任务详情

检查项	实现原理	备注
审计 定 时 清 理 任 务 详 情	查询SQL： select AUDIT_JOB.COUNT AS AUDIT_JOB_CLEAN_JOB, AUDIT_CLEAN_TIME.COUNT AS AUDIT_CLEAN_TIME_COUNT , AUDIT_RECORD.COUNT AS AUDIT_RECORD_COUNT from (SELECT COUNT(*) AS COUNT FROM DBA_AUDIT_MGMT_CLEANUP_JOBS) AS AUDIT_JOB, (select count(*) AS COUNT from DBA_AUDIT_MGMT_LAST_ARCH_TS) AS AUDIT_CLEAN_TIME , (select count(*) AS COUNT from UNIFIED_AUDIT_TRAIL) AS AUDIT_RECORD;	

审计文件大小

检查项	实现原理	备注
审计定时清理任务详情	查询 dba_segments 视图： <pre>select segment_name ,bytes/1024/1024/1024 as size_gb from dba_segments where segment_name like '%AUD\$%' and SEGMENT_TYPE = 'TABLE';</pre>	

日志分析

错误日志分析

检查项	实现原理	备注
	1.run.log错误分析 查询 <code>run.log</code> 文件，过滤出包含 <code>errno</code> 的日志	
	2.alert.log错误分析 查询 <code>alert.log</code> 文件，过滤出告警信息	
数据	3.内核错误分析 执行 <code>dmesg</code> 命令，将输出结果中包含"error", "warning", "warn", "failed", "invalid", "fault", "faulty", "timeout", "unable", "cannot", "corrupt", "corruption"等关键字的信息	
库主		
备连		
接状态		
态	4.操作系统错误日志分析 查询 <code>/var/log/messages</code> 或者 <code>/var/log/syslog</code> 文件，过滤出包含："BIOS Error", "Error", "FAILED Result", "Hardware Error", "I/O error", "iptables denied", "refused connect", "Possible SYN flooding", "drop open request", "connection reset", "error", "Out of memory", "Killed process"等关键字的日志	

数据库变更日志

检查项	实现原理	备注
数据库变更日志	查询 <code>run.log</code> 文件，过滤出 <code>INFO</code> 级别的日志	<p>INFO级别记录数据库正常运行中发生的关键事件，主要包括：</p> <p>数据库状态变更：例如启动、关闭、升主降备。</p> <p>数据库关键资源的变更：例如表空间、用户增加删除等。</p> <p>数据库资源变更：例如线程的启动和停止等。</p> <p>数据库关键活动：例如重启恢复、残留事务等。</p> <p>数据库参数变化：修改系统配置、隐藏参数等。</p>

慢日志分析

检查项	实现原理	备注
慢日志参数	查询 <code>v\$parameter</code> 视图	
慢日志系统表	查询 <code>sys.slow_log\$</code> 视图	
慢日志文件	查询 <code>slow.log</code> 文件	

REDO日志分析

检查项	实现原理	备注
REDO 日志 分析	查询 <code>v\$logfile</code> 视图： <code>select ID, NAME, STATUS, BLOCK_SIZE, BLOCK_COUNT, USED_BLOCKS, SEQUENCE# AS SEQUENCE from v\$logfile;</code>	

检查项	实现原理	备注
REDO 日志 数量 分析	查询 v\$logfile 视图： <pre>select count(*) as total_count, SUM(CASE WHEN STATUS = 'CURRENT' THEN 1 ELSE 0 END) AS current_count,SUM(CASE WHEN STATUS = 'ACTIVE' THEN 1 ELSE 0 END) AS active_count, SUM(CASE WHEN STATUS = 'INACTIVE' THEN 1 ELSE 0 END) AS inactive_count from v\$logfile;</pre>	

UNDO日志分析

检查项	实现原理	备注
正在使用的 UNDO空间大小	查询 v\$transaction 、 v\$parameter 视图： <pre>SELECT round(a.USED_UBLK * b.value /1024/1024,3) AS SIZE_MB, XID from V\$TRANSACTION as a ,(SELECT VALUE FROM V\$PARAMETER WHERE NAME='DB_BLOCK_SIZE') AS B ;</pre>	
使用过多UNDO 块	查询 v\$transaction 视图： <pre>SELECT SUM(USED_UBLK) as TOTAL_BLOCK from V\$TRANSACTION ;</pre>	
当前正在运行的 事务及UNDO使 用情况	查询 v\$transaction 视图： <pre>SELECT XID, SID,XRMID,XEXT, XNODE,XSN,STATUS,RESIDUAL, USED_UBLK, FIRST_UBAFIL,FIRST_UBABLK,FIRST_UBAVER , FIRST_UBAREC,LAST_UBAFIL,LAST_UBABLK, PTX_XID, START_DATE,ISOLATION_LEVEL from V\$TRANSACTION ;</pre>	

报告解读

健康巡检报告主要包括健康检查的概览信息和7从不同维度触发的用来衡量系统健康程度的模块及模块下的健康巡检结果。

健康检查概览信息

概览信息主要包括本次巡检的一些基础信息、得分详情、告警信息以及每个模块的检查项信息及该检查项对应的告警数量。

YashanDB 深度巡检报告	
模块查看概况	
概述	
主机信息	
数据源检测	
对象扫描	
安全检测	
日志分析	
自定义检查	

健康检查概况信息	
健康检查开始时间	2024-01-10 19:01:55
健康检查花费时间	30 秒
检查项共计	92 个
存在问题的检查项	5 个
YashanDB Home 目录	/home/mile/yashan/db
YashanDB Data 目录	/home/mile/yashan/db/data/db-1-1
YashanDB 日志	99%

健康检查得分详情	
健康检查总分	100.00
本次健康的香港分	99.19
本次检测健康状况	优秀
本次检测告警统计	严重级别的0个，警告级别的2个，建议确认并处理相关问题
得分详细说明	优秀(95.00->分数<100.00) 良好(90.00->分数<95.00) 商用(70.00->分数<90.00) 较差(60.00->分数<70.00) 危急(0.00->分数<60.00)
告警权重	单机级告警权重: 10.00, 相同指标范围内的综合告警: false, 单机告警权重: 严重(1.00) 警告(2.00) 常规(1.00)

告警详情							
告警名称	模块	告警级别	告警描述	表达式	值	告警建议	告警标记
数据库参数检查	数据库参数检查	数据	数据库参数已关闭	parameter.recycle_bin.enabled == 0	OFF	数据库参数已关闭，建议开启，关注日志	敏感功能，关注日志

左侧菜单栏的三种颜色数字标识当前菜单下检查项中存在的告警数量

左侧菜单支持一键收起和展开，便于快速找到存在告警的检查项

概述模块

包含主机概述和数据库概述两部分：

- 主机概述
 - 主机信息
 - BIOS信息
- 数据库概述

主机概述

主机信息

主机信息包含主机的操作系统信息、CPU信息、网卡信息、磁盘信息、磁盘块设备信息和内存信息，以图表的形式展示出来。

YashanDB 深度巡检报告	
模块查看概况	
概述	
主机概述	
BIOS 概述	
数据源概述	
对象扫描概述	
安全检测	
日志分析	
自定义检查	

主机信息	
主机名称	mg_4
操作系统的名称	linux
内核架构	x86_64
内核版本号	3.10.0-1160.71.1.el7.x86_64
操作系统的识别	centos
操作系统的名称	rhel
操作系统的版本号	7.9.2009
开机时间	2023-09-04 18:54:11
运行时间	1149h22m0s
进程数	214
CPU 型号	Intel Xeon Processor (Skylake, IBRS)
CPU 厂商	GenuineIntel
CPU 速度	2.00GHz

BIOS 信息

BIOS 信息展示当前主机的 BIOS 信息，是在 linux 系统上执行 dmidecode 命令获取的信息，将其中与 BIOS 相关的信息截取展示出来。

YashanDB 深度巡检报告

模块: 硬件信息概览

硬件信息概览

YashanDB 22.2.0.0-3444

BIOS Information

Vendor: Seaboard
Version: 1.3
Release Date: 01/01/2021
Address: 00E000
Base Address: 00E000
ROM Size: 64 MB
Characteristics:
 - ROM
 - Non-volatile
 - Targeted content distribution is supported
 - ROM Revision: 1.0

模块: 硬件评估

模块: 安全评估

模块: 性能分析

模块: 日志分析

模块: 自定义检查

数据库概述

数据库信息

数据库信息记录了数据库及实例的基本信息，包括数据库名称，数据库版本，数据库创建时间，主备角色，部署形态，实例状态、实例启动时间等信息。

YashanDB 深度巡检报告

模块: 数据库信息

数据库创建时间	2023-09-13 02:03:49.457636 +0800 CST
数据库名称	test
数据库主备角色	PRIMARY
数据库状态	NORMAL
数据库实例状态	OPEN
数据库IP及端口	192.168.4.117:1400
数据库归档模式	ARCHIVELOG
部署形态	主备
数据库打开模式	READ_WRITE
保护级别	MAXIMUM PERFORMANCE
保护模式	MAXIMUM PERFORMANCE
数据库实例启动时间	2023-09-13 02:03:48.670527 +0800 CST
数据库版本	Release 22.2.0-x64_64

模块: 数据库文件权限

文件路径	所有者	所有组	文件权限
/opt/yasom/yashandb/test/data/db-1/1/archive/arch_0_10.ARC	golang	golang	-rw-r-----

数据库文件权限展示了当前节点的数据库文件的详细权限，包括文件所有者和所有组，以及文件权限。如果数据库的数据文件权限过高（其他用户拥有写权限），将会产生告警。

YashanDB 深度巡检报告

模块: 数据库文件权限

文件路径	所有者	所有组	文件权限
/opt/yasom/yashandb/test/data/db-1/1/archive/arch_0_10.ARC	golang	golang	-rw-r-----
/opt/yasom/yashandb/test/data/db-1/1/archive/arch_0_50.ARC	golang	golang	-rw-r-----
/opt/yasom/yashandb/test/data/db-1/1/archive/arch_0_40.ARC	golang	golang	-rw-r-----
/opt/yasom/yashandb/test/data/db-1/1/archive/arch_0_30.ARC	golang	golang	-rw-r-----
/opt/yasom/yashandb/test/data/db-1/1/dml/double_write	golang	golang	-rw-r-----
/opt/yasom/yashandb/test/data/db-1/1/archive/arch_0_30.ARC	golang	golang	-rw-r-----
/opt/yasom/yashandb/test/data/db-1/1/archive/arch_0_40.ARC	golang	golang	-rw-r-----
/opt/yasom/yashandb/test/data/db-1/1/archive/arch_0_50.ARC	golang	golang	-rw-r-----
/opt/yasom/yashandb/test/data/db-1/1/archive/arch_0_60.ARC	golang	golang	-rw-r-----

模块: 数据库文件及目录权限要求过低

表达式: file_permissions == '/tmp/*', 值: -rwxrwxrwx, 警告建议: 建议调整的属性文件和目录权限, 如果文件读写权限过高, 存在文件被其他用户读写的风险, 切忌: [文件路径: /opt/yasom/yashandb/test/data/db-1/1/test0]

主机检查

主要包括主机的负载模块

- 主机的负载模块
 - CPU使用情况
 - 磁盘使用情况
 - 内存使用情况

- 网络使用情况
- 防火墙状态
- 端口打开情况等

主机负载检查

CPU使用情况

CPU使用情况以曲线图的形式展示历史CPU负载和当前CPU使用负载。

历史CPU负载展示指定的时间范围内的CPU使用情况，如空闲时间百分比，等待I/O时间百分比等信息。当前CPU负载展示检查过程中CPU的使用情况。



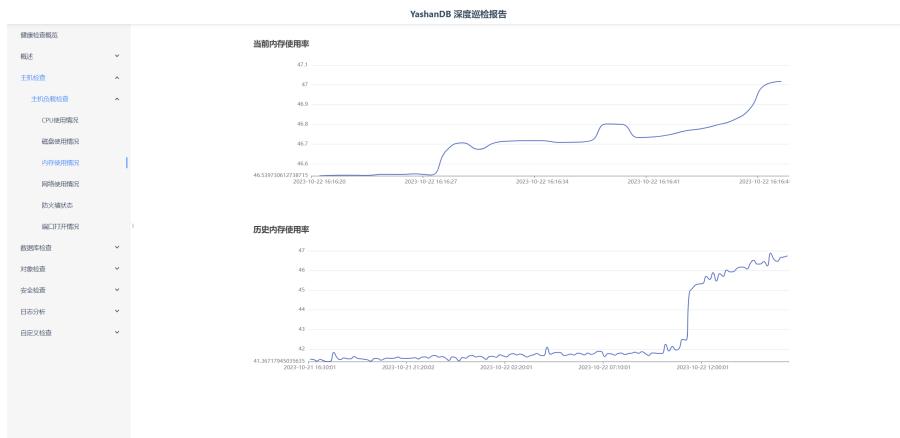
磁盘使用情况

磁盘使用情况以曲线图的形式展示磁盘的历史、当前的每秒读写数据量，tps等信息。



内存使用情况

内存使用情况以曲线图的形式展示当前和历史的内存使用率。



网络使用情况

网络使用情况以曲线图的形式展示当前和历史的网络每秒接收和发送的数据量。



防火墙状态

防火墙状态展示当前机器是否开启了防火墙。为true表示开启了防火墙，为false表示未开启防火墙

端口打开情况

端口打开情况展示当前机器的防火墙规则，是由linux命令iptables -L获取的。

数据库检查

主要包括以下几个模块：

- 主备检查
 - 主备连接状态
- 数据库配置检查
 - 数据库参数检查
 - OS认证用户
- 表空间检查
 - 表空间
 - 数据文件
- 控制文件检查
 - 控制文件
- 备份检查
 - 备份记录
 - 近10天完成的全量备份
 - 本地备份集路径
- 负载检查
 - 会话数检查
 - 共享池检查
- 归档日志检查
 - 数据库归档情况
 - 数据库归档日志
- 性能分析
 - VM转换率
 - 等待事件
 - TOP10 SQL
 - 高频SQL
 - 数据库历史负载
 - 内存池命中率
 - 性能配置检查
 - 锁等待
 - 长事务

主备检查

主备连接状态

主备检查检查数据库主备的连接状态，数据来源于数据库v\$replication_status视图。

只有备节点的视图才有该数据，因此如果对主节点进行检查，将无法获取到该数据

数据库配置检查

数据库参数

数据库参数检查以表格的形式展示数据库v\$parameter视图中不为空的参数。

YashanDB 深度巡检报告		
数据库参数检查		
模块	参数名	参数值
主备检查	AC_MAX_SOURCE_SLICE_COUNT	20
数据库参数	AC_SLAVE_THRESHOLD_SIZE	64M
主备检查	ARCHIVE_LOCAL_DEST	/archive
数据库参数设置	ARCH_CLEAN_JONCH_MODE	NONE
数据库参数设置	ARCH_CLEAN_LOWER_THRESHOLD	12G
OS认证用户	ARCH_CLEAN_UPPER_THRESHOLD	16G
表空间检查	AUDIT_FLUSH_INTERVAL	100
控制文件检查	AUDIT_QUEUE_SIZE	16M
备份检查	AUDIT_QUEUE_WRITE	TRUE
负载检查	BLOCK_REPAIR_ENABLED	TRUE
性能分析	BLOCK_REPAIR_TIMEOUT	60
对象检查	BLOOM_FILTER_FACTOR	3
安全检查	BROADCAST_GTS_TIME	5
日志分析	BUCKET_RESERVED_SPACE	1G
自定义检查	CHARACTER_SET	UTF8
	CHECKPOINT_INTERVAL	100000

OS身份认证

OS身份认证则检查当前数据库是否开启了操作系统身份认证，如果开启了操作系统身份认证，还将展示YASDBA组中所有用户信息。

The screenshot shows the 'YashanDB Depth Inspection Report' interface. On the left is a sidebar with a tree view of inspection modules: General Configuration, System Configuration, Database Configuration, Performance Monitoring, Diagnostic Tools, and Audit Log. The 'OS Identity Authentication' section is expanded, showing the 'System Identity Authentication Status' as 'on'. Below it, the 'YASDBA Group User' section lists 'yashan.mongodbsysadmin,yashan,golang'.

表空间检查

表空间

表空间页面以表格的形式展示数据库表空间相关信息，包括表空间名称、类别、状态、总字节数、使用率等信息。还会对开启了自动拓展的表空间进行告警。

The screenshot shows the 'Table Space' section of the report. It displays a table with columns: 表空间名称 (Table Space Name), 表空间类型 (Table Space Type), 表空间状态 (Table Space Status), EXTEND分步方式 (Extend Step Method), 已使用字节数 (Used Bytes), 总字节数 (Total Bytes), and 使用率(%) (Usage %). The table includes rows for SYSTEM, SYSAUX, UNDO, TEMP, SWAP, and USERS. Below the table, three yellow warning boxes highlight '表空间开启自动扩展' (Table space automatic extension enabled) for SYSTEM, SYSAUX, and USERS, with detailed explanations about the allocation type being AUTO and the potential risks of auto-expanding UNDO tablespaces.

数据文件

数据文件页面以表格的形式展示数据库数据文件相关信息，包括数据文件名称、ID、表空间名称、状态、是否自动拓展等信息。

The screenshot shows the 'Data File' section of the report. It displays a table with columns: 数据文件名称 (Data File Name), ID, 表空间名称 (Table Space Name), 状态 (Status), 自动拓展 (Automatic Extension), and 文件权限 (File Permissions). The table lists multiple data files (system, sysaux, undo, temp, swap, users) across different tablespaces (SYSTEM, SYSAUX, UNDO, TEMP, SWAP, USERS) with their respective IDs and statuses.

控制文件检查

控制文件

检查数据库的控制文件信息，如控制文件名称和大小等，还会对控制文件数量进行告警。如果控制文件过少，可能会出现单点故障而引起数据库不可用。

YashanDB 深度巡检报告

控制文件			
名称	ID	大小(MB)	
/opt/json/yashandb/test/data/db-1-1/bfile/ctrl1	0	27.3964375	
/opt/json/yashandb/test/data/db-1-1/bfile/ctrl2	1	27.3964375	
< 1 >			

控制文件数量	
2	

备份检查

数据库备份记录

备份记录展示数据库中的备份集记录。

YashanDB 深度巡检报告

备长集编号	开始时间	备长类型	备长是否成功
1	2023-10-20 01:11:46.534075 +0000 CST	FULL	true
< 1 >			

近十天的完成的全量备份

近十天的完成的全量备份统计最近10天内完成的全量备份文件的数量。

YashanDB 深度巡检报告

备份数量
1

本地备份集路径

本地备份集路径则检查完成的备份是否保存在本地对应的路径，如果不存在，将会产生告警。

YashanDB 深度巡检报告

本地备份集路径

备份集文件: /opt/yashandb/test/data/db-1/backup/bak_bak_test_instance1_20231019171144
是否配置: FALSE

告警信息

● 备份集文件不存在或者检查用户无权限访问
表达式: yashd_backup_set_path_exist == TRUE, 值: FALSE, 告警建议: 建议设置备份文件, 必要时重新备份, 标题: 备份集文件: /opt/yashandb/test/data/db-1/backup/bak_bak_test_instance1_20231019171144

负载检查检查

会话使用情况

检查数据库当前的系统会话数、用户会话数、最大会话数、总会话数、会话使用率等信息。

YashanDB 深度巡检报告

系统会话数	21
用户会话数	5
最大会话数	1024
会话使用率(%)	254
空会话数	26

共享池信息检查

共享池信息检查数据库共享池使用情况，包括共享池总大小、空闲大小、已使用大小和使用率。

YashanDB 深度巡检报告

空闲大小	0B
总大小	255.73M
共享池使用率	100.00%
已使用大小	255.73M

归档日志检查

数据库归档情况

数据库归档情况展示数据库的归档路径、归档文件的空间使用情况等信息。

YashanDB 深度巡检报告	
模块检查概览	
概述	已监控对象数 20
主机检查	已检测大空间(GB) 16
数据完整性检查	已检测已用空间(GB) 2.37
主备检查	已检测可用空间(GB) 2.37
数据完整性检查	可用空间(GB) 14.84
表空间检查	
控制文件检查	
备份检查	
负数检查	
性能分析	
对像检查	
安全检查	
日志分析	
白名单检查	
数据库归档日志	
数据库归档日志情况	
数据库归档日志	
性能分析	
对像检查	
安全检查	
日志分析	
白名单检查	
展开所有项	

数据库归档日志

数据库归档日志展示数据库各归档日志相关信息。

YashanDB 深度巡检报告	
模块检查概览	
概述	/opt/yassom/yashandb/test/data/db-1/1/archive/arch_1,1,ARC
主机检查	/opt/yassom/yashandb/test/data/db-1/1/archive/arch_2,2,ARC
数据完整性检查	/opt/yassom/yashandb/test/data/db-1/1/archive/arch_3,3,ARC
主备检查	/opt/yassom/yashandb/test/data/db-1/202004/020004
数据完整性检查	2023-12-09 02:00:07
表空间检查	2023-12-10 02:00:08
控制文件检查	2023-12-10 02:00:09
备份检查	2023-12-11 02:00:07
负数检查	2023-12-12 02:00:06
性能分析	2023-12-13 02:00:05
对像检查	2023-12-13 02:00:04
安全检查	2023-12-14 02:00:03
日志分析	2023-12-15 02:00:05
白名单检查	2023-12-16 02:00:08
数据库归档日志	
数据库归档日志情况	
数据库归档日志	
性能分析	
对像检查	
安全检查	
日志分析	
白名单检查	
展开所有项	

< 1 2 > 跳至 页

性能分析

VM转换率

VM转换率展示了数据库当前的VM转换率信息。

YashanDB 深度巡检报告	
数据完整性检查	
表空间检查	
控制文件检查	
备份检查	
负数检查	
性能分析	
VM转换率	
等待事件	
TOP10 SQL	
高耗SQL	
数据完整性失败	
内存使用率	
性能配置设置	
故障转	
长事务	
对像检查	
安全检查	
日志分析	
白名单检查	
展开所有项	

等待事件

等待事件以表格的形式展示了检查时间范围内数据库的TOP10等待事件，包含事件名称、事件类别等信息。

YashanDB 深度逆检报告						
数据源连接池	等待事件名称	等待事件类别	总等待事件(s)	平均等待时间(ms)	DB Time	等待次数
表空间锁	SQL-Net message from client	idle	107473.82	191	147302.34	875517
控制文件检查						
备机检测	DB CPU		113.65		99.96	
负载均衡	SQL-Net message to client	Network	12.37	0	10.88	875494
性能分析	db_file sequential read	User I/O	.11	0	.1	3978
VN线程读取	db_file scattered read	User I/O	.01	0	.01	882
等待事件	free buffer wait	Configuration	0	0	0	16791
TOP5 SQL	log file sync	Commit	0	1	0	2
高级SQL	log file parallel write	System I/O	0	1	0	2
数据表反向加载	xslist busy wait	Configuration	0	0	0	0
内存命中率统计	exclusive lock wait	Concurrency	0	0	0	0
性能数据检测						
故障转移						
长事务						
对表检测						
安全配置						
日志分析						
日志文件检查						

TOP10 SQL

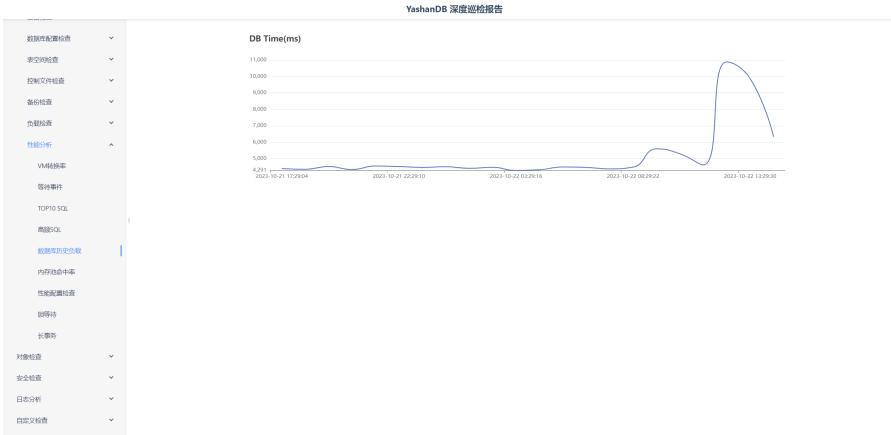
TOP10 SQL展示了平均耗时最长的10条SQL的信息，包括SQL ID，SQL文本的前1000个字符等。

高频SQL

高频SQL展示了执行次数最长的10条SQL的信息。

数据库历史负载

数据库历史负载以曲线图的形式展示指定时间范围内数据库的平均DB Time。



内存池命中率

内存池命中率展示了数据库历史和当前的内存池命中率。



性能配置检查

性能配置检查检查数据库性能相关的标准大页和Swap内存配置是否开启。



锁等待

锁等待界面展示数据库的行锁和表锁等待数量。

YashanDB 深度巡检报告

长事务

长事务检查当前数据库执行时间过长的事务信息，包括事务ID、事务状态等。

YashanDB 深度巡检报告

对象检查

对象数量统计

对象总数

统计数据库的对象总数、各owner拥有的对象数以及各tablespace拥有的对象数。

YashanDB 深度巡检报告

对象状态检查

对象检查主要对数据库中表、索引、序列、任务、包等对象进行检查，主要包含以下几个模块：

检查失效对象

失效对象列表展示了失效状态的对象的详细信息，包括对象名称、类型、所属用户等。

对象所属用户	对象名称	对象类型	对象状态	对象ID
SYS	VA_PROC	PROCEDURE	INVALID	2232

失效对象 ID
表达式: invalid_object_id != 0, 值: 2232, 备註建议: 当前存在失效对象, 可能影响业务运行, 列名: 对象所属用户: SYS; 对象名称: VA_PROC

不可见索引

不可见索引列表展示了不可见状态的索引的详细信息，包括索引所属用户、索引名称等。

索引所属用户	索引名称	索引可见性
TEST	TEST_INDEX_INVISIBLE	INVISIBLE
TEST	TEST_INDEX_INVISIBLE1	INVISIBLE

不可见索引名称
表达式: invisible_index_name != "", 值: TEST_INDEX_INVISIBLE, 备註建议: 当前存在不可见索引, 需检查并恢复需要该对象, 列名: (索引的列名: TEST; 索引名称: TEST_INDEX_INVISIBLE)

不可见索引名称
表达式: invisible_index_name != "", 值: TEST_INDEX_INVISIBLE1, 备註建议: 当前存在不可见索引, 需检查并恢复需要该对象, 列名: (索引的列名: TEST; 索引名称: TEST_INDEX_INVISIBLE1)

不可用约束

不可用约束列表展示了不可用状态的约束的详细信息，包括约束所属用户、约束名称、约束类型等。

约束所属用户	约束名称	约束状态	约束类型
SYS	CONS_T10	DISABLED	C

不可用的约束名称
表达式: disabled_constraint_name != "", 值: CONS_T10, 备註建议: 当前存在不可用约束, 请检查, 列名: (约束的列名: SYS; 约束名称: CONS_T10)

表

字段过多的表

字段过多的表页面详细展示了字段超过80的表的名称、所属用户以及字段数量。

YashanDB 深度巡检报告

对象位置	表所属用户	表名称	索引数量
TEST	TEST_COLUMN_LARGE	81	

索引过多的表

索引过多的表页面详细展示了索引数量超过8个的表的名称、所属用户以及索引数量。

对象位置	表所属用户	表名称	索引数量
TEST	TEST_COLUMN_LARGE	10	

没有分区索引的分区表

没有分区索引的分区表页面详细展示了此类表的名称、所属用户、分区类型和分区键的列信息。

YashanDB 深度巡检报告

对象位置	表所属用户	表名称	分区类型	分区键 (Key)
YD_C5_BILL	T_SRSJ_GMVCFQK22_23	RANGE	F_1_NY	

行大小超过块大小的表

行大小超过块大小的表页面展示了此类表的名称和所属用户信息。

YashanDB 深度巡检报告

表名	表名	表名
TEST	T14	

哈希分区数量不是2的幂的分区

哈希分区数量不是2的幂的分区页面详细展示了此类表的名称、所属用户、分区类型和分区个数。

YashanDB 深度巡检报告

表名	表名	分区类型	分区个数
SYS	SALES_INFO_HASH	HASH	3

约束

无索引子表外键

无索引子表外键界面详细展示了此类表的父表、子表、约束名称和相关列的信息。

YashanDB 深度巡检报告

父表	子表	约束名称	相关列
AAAORDERS	MEMBERS	FK_MEM_ID	MEM_ID

数据类型隐式转换的外键

由于yashandb暂不支持数据类型隐式转换，数据类型隐式转换的外键页面通常为空页面。

索引

超过三层的索引

主要展示超过三层的索引信息。

YashanDB 深度巡检报告

索引名称	索引用户	字节数量
IND_18	SYS	8

左侧菜单栏包括：主备检查、数据完整性检查、表空间检查、控制文件检查、备份检查、负载均衡、性能分析、对象位置、对象数量统计、对象总数、对象状态检查、表、约束、索引（显示了二级索引的使用情况）、序列。

底部地址栏显示：<http://127.0.0.1:8080/yashandb/20230221161616/report/20230221161616.html>

字段过多的索引

主要展示字段过多的索引。

YashanDB 深度巡检报告

索引名称	索引用户	字段数量
TEST_INDEX_COLS	SYS	11

左侧菜单栏包括：主备检查、数据完整性检查、表空间检查、控制文件检查、备份检查、负载均衡、性能分析、对象位置、对象数量统计、对象总数、对象状态检查、表、约束、索引（显示了二级索引的使用情况）、序列。

不可见索引

主要展示不可见索引信息。

YashanDB 深度巡检报告

索引名称	索引用户	表名	表用户
TEST_INDEX_INVISIBLE	TEST	TEST_INDEX	TEST

左侧菜单栏包括：主备检查、数据完整性检查、表空间检查、控制文件检查、备份检查、负载均衡、性能分析、对象位置、对象数量统计、对象总数、对象状态检查、表、约束、索引（显示了二级索引的使用情况）、序列。

过大的索引

主要展示过大的索引信息。

YashanDB 深度巡检报告

左侧导航栏包括：主备份、数据完整性检查、表空间检查、控制文件检查、备机检查、负载均衡、性能分析、对像位置、对象数量统计、对象函数、对象状态检查、表、约束、索引（显示了三个级别的索引：超过三层的索引、字段过多的索引、过大索引）、过大索引、表和索引不在一个schema下、序列。

右侧显示“过大的索引”结果：

索引名称	索引用户	索引类型	索引大小(字节)	表名称	表用户	表类型	表大小(字节)
LHSTGMM1	SYS	INDEX	8388008	HISTGMS	SYS	TABLE	4194304
WRH_COSTAT_FK	SYS	INDEX PARTITION	131072	WRH_COSTAT	SYS	TABLE PARTITION	65536
WRH_MEM_USED_COMP_PK	SYS	INDEX PARTITION	131072	WRH_MEM_USED_COMP	SYS	TABLE PARTITION	65536

表和索引不在同一个schema下

主要展示表和索引不在同一个schema的索引信息和表信息。

YashanDB 深度巡检报告

左侧导航栏包括：主备份、数据完整性检查、表空间检查、控制文件检查、备机检查、负载均衡、性能分析、对像位置、对象数量统计、对象函数、对象状态检查、表、约束、索引（显示了三个级别的索引：超过三层的索引、字段过多的索引、过大索引）、过大索引、表和索引不在一个schema下、序列。

右侧显示“表和索引不在一个schema下”结果：

表名	索引名	索引用户	表名称	表用户
TEST_SCHEMA	VASOM	TEST_SCHEMA	TEST	

序列

无可用值的序列

主要展示序列使用率超过70%的序列信息。

YashanDB 深度巡检报告

左侧导航栏包括：备份检查、负载均衡、性能分析、对像位置、对象数量统计、对象函数、对象状态检查、表、约束、索引（显示了三个级别的索引：超过三层的索引、字段过多的索引、过大索引）、过大索引、表和索引不在一个schema下、序列、任务、角色、权限、安全检查。

右侧显示“无可用值的序列”结果：

序列名称	序列用户	自由值(%)
TEST_SEQ	TEST	80
TEST_SEQ1	TEST	90

任务

运行中的任务

主要展示当前正在运行的数据库JOB信息。

YashanDB 深度巡检报告							
常规检查							
负载检查		JOB名 JOB JOB类型 调度 JOBID 用户名 名					
性能分析							
对称检测							
对像数量统计		BATCH INS SYS REGULAR SYS					
对像对象数							
对像状态检查							
表	▼						
视图	▼						
索引	▼						
索引	▼						
超过三倍的索引							
字段过多的索引							
不可见索引							
过大的索引							
表和索引在一个schema下							
序列	▼						
无可重用的序列							
任务	▼						
运行中的任务							
包	▼						

包

没有package的package body

主要展示有没有没有package的package body。

YashanDB 深度巡检报告

没有package的package body			
类别	对象名称	对象所属的包名	对象类型及连接字符串
	VA_FROC	SYS	PROCEDURE
● 存在没有 package 的 package body			
表达式: yashan_package.package_type_json != 'PACKAGE PACKAGE BODY', 或 PROCEDURE, 告警建议: 建议使用 package 以充分利用其封装、抽象和优化代码的特性。, 标签: 对象名称: VA_FROC			
表和索引不在一个 schema 下			
存储过程			
无可用的游标			
任务			
运行中的任务			
包			
使用 packaged 的 package body			
安全检测			
日志分析			
错误日志分析			
nevlog错误分析			
alertlog错误分析			
内核错误分析			
操作数错误日志分析			
剩余空间			

安全检查

安全检查主要包含针对用户登录，用户权限以及审计进行检查，主要包含以下几个模块及检查项：

- 登录检查
 - 密码强度
 - 未限制登录次数的profile
 - 用户与权限检查
 - 非OPEN状态的用户
 - 拥有系统表权限的用户
 - 拥有DBA角色的用户
 - 拥有ALL PRIVILEGES|SYSTEM的的用户
 - 以SYSTEM为默认表空间的用户
 - 审计定时任务详情
 - 审计文件大小

登录检查

密码强度

YashanDB 深度巡检报告

健康检查概况

概述

主机连接

数据完整性

对象检查

安全检查

审计检查

密码强度

密码强度控制

表达式: yndb_password_strength_complexity := FALSE; 值: FALSE, 警告建议: 建议打开此参数, 防止普通用户使用弱密码

FALSE

未开启密码强度控制

其他功能

未限制登录次数的Profile

用户与权限检查

由OPEN状态的用户

拥有系统表权限的用户

所有DBA角色的用户

拥有ALL PRIVILEGES...
USSTEM权限的用户...

审计定时器任务详细

审计文件大小

日志分析

自定义检查

未限制登录次数的Profile

YashanDB 深度巡检报告

健康检查概况

概述

主机连接

数据完整性

对象检查

安全检查

审计检查

密码强度

未限制登录次数的Profile

限制 PROFILE名称 资源名称 密码强度控制

UNLIMITED PRO2 FAILED_LOGIN_ATTEMPTS PASSWORD

未限制用户登录尝试次数

表达式: yndb_maximum_login_limit := UNLIMITED; 值: UNLIMITED, 警告建议: 建议限制用户尝试登录次数, 防止暴力破解密码 (启用系统账户模块设置尝试登录次数), 标签: [PROFILE名称: PRO2]

FAILED_LOGIN_ATTEMPTS

其他功能

未限制登录次数的Profile

用户与权限检查

由OPEN状态的用户

拥有系统表权限的用户

所有DBA角色的用户

拥有ALL PRIVILEGES...
USSTEM权限的用户...

审计定时器任务详细

审计文件大小

日志分析

自定义检查

用户与权限检查

非OPEN状态的用户

YashanDB 深度巡检报告

健康检查概况

概述

主机连接

数据完整性

对象检查

安全检查

审计检查

密码强度

非OPEN状态的用户

账户状态 用户名称

LOCKED AAA

存在非OPEN状态的用户

表达式: yndb_user_account_status != 'OPEN'; 值: LOCKED, 警告建议: 如果用户被锁定, 应及时解锁。

其他功能

未限制登录次数的Profile

用户与权限检查

由OPEN状态的用户

拥有系统表权限的用户

所有DBA角色的用户

拥有ALL PRIVILEGES...
USSTEM权限的用户...

审计定时器任务详细

审计文件大小

日志分析

自定义检查

拥有系统表权限的用户

YashanDB 深度巡检报告

拥有系统表权限的用户

被授权者名	用户名
AAA	USERAUTHS

非SYS用户拥有系统表权限
表达式: `yashan_user_with_system_table != 'SYS'`, 值: AAA, 告警建议: 建议立即取消该用户的权限, 这会增加该用户破坏整个数据库的风险, 标签: (用户名: USERAUTHS)

拥有DBA角色的用户

YashanDB 深度巡检报告

所有DBA角色的用户

拥有DBA角色的用户
YD_CS_BILL
TEST
YASOM

非SYS用户拥有DBA角色
表达式: `yashan_user_with_dba_role != 'SYS'`, 值: YD_CS_BILL, 告警建议: 建议立即取消该用户的DBA角色, 标签: (拥有DBA角色的用户: YD_CS_BILL)

非SYS用户拥有DBA角色
表达式: `yashan_user_with_dba_role != 'SYS'`, 值: TEST, 告警建议: 建议立即取消该用户的DBA角色, 标签: (拥有DBA角色的用户: TEST)

非SYS用户拥有DBA角色
表达式: `yashan_user_with_dba_role != 'SYS'`, 值: YASOM, 告警建议: 建议立即取消该用户的DBA角色, 标签: (拥有DBA角色的用户: YASOM)

拥有ALL PRIVILEGES|SYSTEM的的用户

!1697966315349

以SYSTEM为默认表空间的用户

YashanDB 深度巡检报告

默认表空间

用户名
SYSTEM

用户名
SYS

非SYS用户拥有DBA角色
表达式: `yashan_user_with_dba_role != 'SYS'`, 值: SYSTEM, 告警建议: 建议立即取消该用户的DBA角色, 标签: (用户名: SYSTEM)

审计定时清理任务详情

主要展示审计清理任务的数量、清理审计日志时间点数量、审计日志数量。



审计文件大小

主要展示与审计相关的segment信息。



日志分析

日志分析模块主要对数据库日志、操作系统日志、REDO、UNDO日志进行过滤与查询，主要包含如下：

- 错误日志分析（主要查询满足时间范围的符合特定条件的日志）
 - run.log错误分析
 - alert.log错误分析
 - 内核错误分析
 - 操作系统错误日志分析
- 数据库变更日志（run.log中的数据库变更日志）
- REDO日志分析
- UNDO日志分析

错误日志分析

run.log错误分析

主要收集满足时间范围且日志主体中包含***errno***的日志。

YashanDB 深度巡检报告

健康状态概览	
概述	2023-10-22 14:13:39,031 14964 [DEBUG]lerrno=0x016: need to specify the database
主从配置	2023-10-22 14:14:48,035,2014 [DEBUG]lerrno=0x000: type convert error , not a valid number
数据完整性	2023-10-22 14:14:48,036,2014 [DEBUG]lerrno=0x000: type convert error , not a valid number
对象检查	2023-10-22 14:14:48,037,2014 [DEBUG]lerrno=0x000: type convert error , not a valid number
安全检查	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
容灾检查	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
用户权限检查	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
审计控制台日志环境	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
审计文件大小	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
日志分析	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
错误日志分析	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
run.log错误分析	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
alert.log错误分析	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
内核错误分析	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
操作系统错误日志分析	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
数据库变更	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
REDO日志分析	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
UNDO日志分析	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number
白名单检查	2023-10-22 14:14:51,599,2004 [DEBUG]lerrno=0x000: type convert error , not a valid number

alert.log错误分析

主要收集满足时间范围且告警action为0即告警日志状态为***上报***的告警日志。

YashanDB 深度巡检报告	
健康状态概览	2023-10-18 00:00:00,15,209 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
概述	2023-10-18 00:00:15,210 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
主从配置	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
数据完整性	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
对象检查	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
安全检查	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
日志分析	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
错误日志分析	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
run.log错误分析	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
alert.log错误分析	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
内核错误分析	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
操作系统错误日志分析	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
数据库变更	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
REDO日志分析	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
UNDO日志分析	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
白名单检查	2023-10-18 03:00:43,176 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
健康状态概览	2023-10-18 04:07:02,15,960 [Database]lerrno=0x000:the incident was controlled by flood, problem id=1, last incident id=15
概述	2023-10-18 04:07:02,15,960 [Database]lerrno=0x000:the incident was controlled by flood, problem id=1, last incident id=15
主从配置	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
数据完整性	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
对象检查	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
安全检查	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
日志分析	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
错误日志分析	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
run.log错误分析	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
alert.log错误分析	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
内核错误分析	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
操作系统错误日志分析	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
数据库变更	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
REDO日志分析	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
UNDO日志分析	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs
白名单检查	2023-10-18 04:07:04,13,448 [Database]lerrno=0x000:database is set to read-only, the reason: there is not enough disk space to archive the logs

内核错误分析

主要收集dmesg日志中包含如下关键字的日志

- error
- warning
- warn
- failed
- invalid
- fault
- faulty
- timeout
- unable
- cannot
- corrupt
- corruption



操作系统错误日志分析

操作系统中 /var/log/messages 或者 /var/log/syslog 中的错误日志。



数据库变更日志



run.log 中的数据库变更日志。

慢日志分析

包括慢日志参数，慢日志系统表和慢日志文件。

慢查询日志配置						
慢查询配置项	配置值	描述	单位	默认值	修改说明	操作
慢查询时长	1000	慢查询的时长，单位毫秒	毫秒	1000	设置为1000毫秒，表示大于1000毫秒的查询将被记录到慢查询日志中。	修改
慢查询条数限制	1000	慢查询日志条数限制	条数	1000	设置为1000条，表示慢查询日志最多存储1000条记录。	修改
慢查询输出位置	/var/log/mysql/mysqld.log	慢查询日志的输出位置	路径	/var/log/mysql/mysqld.log	设置为 /var/log/mysql/mysqld.log，表示将慢查询日志输出到该文件。	修改
慢查询日志开关	ON	慢查询日志开关	开/关	ON	设置为 ON，表示开启慢查询日志功能。	修改
慢查询日志文件大小	100M	慢查询日志文件大小	大小	100M	设置为 100M，表示慢查询日志文件的最大容量为 100MB。	修改
慢查询日志文件数量	10	慢查询日志文件数量	数量	10	设置为 10，表示慢查询日志文件的数量为 10 个。	修改
慢查询日志文件保留天数	30	慢查询日志文件保留天数	天数	30	设置为 30，表示慢查询日志文件的保留时间为 30 天。	修改
慢查询日志文件名前缀	slow_query_log	慢查询日志文件名前缀	前缀	slow_query_log	设置为 slow_query_log，表示慢查询日志文件的前缀名称。	修改
慢查询日志文件后缀	.log	慢查询日志文件后缀	后缀	.log	设置为 .log，表示慢查询日志文件的后缀名为 .log。	修改
慢查询日志文件模式	追加	慢查询日志文件模式	模式	追加	设置为 追加，表示慢查询日志文件采用追加模式写入。	修改
慢查询日志文件权限	644	慢查询日志文件权限	权限	644	设置为 644，表示慢查询日志文件的权限为 644。	修改
慢查询日志文件权限位	0644	慢查询日志文件权限位	权限位	0644	设置为 0644，表示慢查询日志文件的权限位为 0644。	修改
慢查询日志分析						
慢查询分析项	分析结果	分析时间	分析次数	用户名	用户主机	SQL ID
所有慢查询分析	分析完成	2024-01-10 17:22:49	118	sys	192.168.8.236	gmvw3h3tzhewg5
慢查询分析	分析完成	2024-01-10 17:22:49	118	sys	192.168.8.236	gmvw3h3tzhewg5
REDO日志分析	分析完成	2024-01-10 16:42:21	112	sys	192.168.8.236	gmvw3h3tzhewg5
UNDO日志分析	分析完成	2024-01-10 16:42:21	112	sys	192.168.8.236	gmvw3h3tzhewg5
自定义检测	分析完成	2024-01-10 16:42:21	112	sys	192.168.8.236	gmvw3h3tzhewg5

REDO日志分析

包括，REDO日志文件的信息、REDO日志数量分析。

YashanDB 深度巡检报告

REDO日志分析							
	ID	日志文件名	状态	序列号	块数量	块大小	已使用块数量
主日志数	0	/opt/yason/yashandb/test/data/db-1-1/dfiles/redo1	INACTIVE	64	32768	4096	32768
数据性检查	1	/opt/yason/yashandb/test/data/db-1-1/dfiles/redo2	INACTIVE	65	32768	4096	32768
对撞检查	2	/opt/yason/yashandb/test/data/db-1-1/dfiles/redo3	CURRENT	66	32768	4096	7093
安全校验							
日志分析							
错误日志分析							
curllog数据源分析							
alert/log配置分析							
内部错误源分析							
操作日志错误源日志分析							
数据性变差							
REDO日志分析							
UNDO日志分析							
日志文件名							

REDO日志数量分析							
	REDO日志文件总数	active状态日志文件数	inactive状态日志文件数	current状态日志文件数			
	3	0	2	1			

⚠ REDO日志数量检查

表达式: redo_total_count < 4, 阈值: 3, 告警建议: REDO后进数量较少, 建议增加REDO日志组

UNDO日志分析

主要包括正在使用的UNDO空间大小、使用过多UNDO块、当前正在运行的事务及UNDO使用情况。

配置文件

工具配置文件 : {yhc_home}/config/yhc.toml

```
log_level = "DEBUG"
max_duration = "30d"
min_duration = "1m"
range = "24h"
output = "./results"
sql_timeout = 10
scrape_interval = 1
scrape_times = 30
metric_paths = ["./config/default_metric.toml", "./config/custom_metric.toml"]
default_module_path = "./config/report_module.toml"
evaluate_model_path = "./config/evaluate_model.toml"
network_io_discard = "^lo$,^veth.*|^virbr.*|^br.*|^tap.*|^tun.*|^docker.*|^flannel.*"
```

该配置文件是YHC工具的配置文件入口，其中各个字段的含义如下：

- log_level: 工具检查信息的时间范围最大值即指定的-r或-s/-e生效的功能，默认30天
- max_duration: 工具检查信息的时间范围最大值即指定的-r或-s/-e生效的功能，默认30天
- min_duration: 工具检查信息的时间范围最小值即指定的-r或-s/-e生效的功能，默认1分钟
- range: 工具检查的时间范围，默认为从当前时间往前24h
- output: 检查到的数据最后存放的主路径，默认在工具运行当前目录的results目录下
- sql_timeout: AWR生成的最大超时时间，默认10秒
- scrape_interval: 网络负载相关数据检查时的时间间隔，默认为1s
- scrape_times: 网络负载相关数据检查时基于时间间隔重复的总次数，默认为30次
- metric_paths: 检查指标配置文件，用于配置检查指标，默认包含默认指标配置文件和自定义指标配置文件
- default_module_path: 报告模块配置文件，用于配置报告的展示内容
- evaluate_model_path: 得分评估模型配置文件，用于配置健康检查的得分评估模型
- network_io_discard: 查询服务器网络负载情况时，默认丢弃的数据

报告模块配置文件 : {yhc_home}/config/report_module.toml

```
[[modules]]
name = "overview"
name_alias = "概述"

[[modules.children]]
name = "overview_host"
name_alias = "主机概述"
metric_names = ["host_info", "host_cpu_info", "host_disk_info", "host_disk_block_info", "host_bios_info",
"host_memory_info", "host_network_info"]

[[modules.children]]
name = "overview_yasdb"
name_alias = "数据库概述"
metric_names = ["yasdb_instance", "yasdb_database", "yasdb_file_permission", "yasdb_listen_address"]

[[modules]]
name = "host_check"
name_alias = "主机检查"

[[modules.children]]
name = "host_workload_check"
name_alias = "主机负载检查"
metric_names = ["host_history_cpu_usage", "host_current_cpu_usage"]

# .....
# 更多文件内容省略，不重复赘述，详见配置文件
```

该配置文件是YHC工具的报告模块配置文件，负责组织报告展示的内容，其中各个字段的含义如下：

- modules: 定义每个模块的报告内容
 - name : 定义模块的名称（推荐英文）
 - name_alias : 定义模块的别名（用于在报告上展示，一般是中文），如果不定义别名则使用模块名称展示
 - modules.children : 定义模块的子模块
 - name : 定义子模块的名称（推荐英文）
 - name_alias : 定义子模块的别名（用于在报告上展示，一般是中文），如果不定义别名则使用模子模块的名称展示
 - metric_names : 定义模块下的指标名称，即模块下展示哪些检查指标信息

默认指标配置文件： {yhc_home}/config/default_metric.toml

```
[[metrics]]
name = "yasdb_instance"
name_alias = "实例信息"
module_name = "overview"
metric_type = "sql"
default = true
enabled = true
sql = "select status as instance_status, version, startup_time from v$instance;"
```

[metrics.column_alias]

INSTANCE_STATUS = "数据库实例状态"
STARTUP_TIME = "数据库实例启动时间"
VERSION = "数据库版本"

[metrics.item_names]

INSTANCE_STATUS = "instance_status"

[metrics.alert_rules]

[[metrics.alert_rules.critical]]

expression = "instance_status != 'OPEN'"
description = "实例状态异常"
suggestion = "建议检查实例状态"

[[metrics]]
name = "yasdb_tablespace"
name_alias = "表空间"
module_name = "yasdb_check"
metric_type = "sql"
default = true
enabled = true
column_order = ["TABLESPACE_NAME", "CONTENTS", "STATUS", "ALLOCATION_TYPE", "USED_BYTES", "TOTAL_BYTES", "USED_RATE",
"DATA_PERCENTAGE"]
number_columns = ["USED_RATE", "USED_BYTES", "TOTAL_BYTES"]
labels = ["TABLESPACE_NAME"]
sql = "SELECT TABLESPACE_NAME, CONTENTS, STATUS, ALLOCATION_TYPE, TOTAL_BYTES - USER_BYTES AS USED_BYTES, TOTAL_BYTES,
(TOTAL_BYTES - USER_BYTES) / TOTAL_BYTES * 100 AS USED_RATE FROM SYS.DBA_TABLESPACES;"

[metrics.column_alias]

TABLESPACE_NAME = "表空间名称"
CONTENTS = "表空间类型"
STATUS = "表空间状态"
ALLOCATION_TYPE = "EXTEND分配方式"
USED_BYTES = "已使用字节数"
TOTAL_BYTES = "总字节数"
USED_RATE = "使用率(%)"
DATA_PERCENTAGE = "数据段使用占比"

[metrics.item_names]

STATUS = "tablespace_status"
ALLOCATION_TYPE = "allocation_type"
USED_RATE = "tablespace_used_rate"

[metrics.alert_rules]

[[metrics.alert_rules.warning]]

expression = "tablespace_status != 'ONLINE'"
description = "表空间状态检查"
suggestion = "表空间状态不正常，建议检查表空间状态，确认异常原因"

[[metrics.alert_rules.warning]]

expression = "allocation_type == 'AUTO'"

```

description = "表空间开启自动扩展"
suggestion = "建议关闭此功能，避免有事务占用过多的UNDO导致磁盘空间不可用"

[[metrics.alert_rules.warning]]
expression = "tablespace_used_rate >= 80 && tablespace_used_rate < 90"
description = "表空间使用率"
suggestion = "表空间使用率过高，建议检查表空间使用情况并通过清理空间或者增加数据文件的方法降低表空间使用率，表空间完全满之后可能导致业务挂起或者数据库挂起"

[[metrics.alert_rules.critical]]
expression = "tablespace_used_rate >= 90"
description = "表空间使用率"
suggestion = "表空间使用率过高，建议检查表空间使用情况并通过清理空间或者增加数据文件的方法降低表空间使用率，表空间完全满之后可能导致业务挂起或者数据库挂起"

# .....
# 更多文件内容省略，不重复赘述，详见配置文件

```

该配置文件是YHC工具的默认指标配置文件，负责定义默认的检查指标，其中各个字段的含义如下：

- metrics: 定义每个检查指标的内容
 - name : 定义检查指标的名称（推荐英文）
 - name_alias : 定义检查指标的别名（用于在报告上展示，一般是中文），如果不定义别名则使用检查指标的名称展示
 - module_name : 指标所属的模块名称
 - default : 是否为默认指标
 - enabled : 默认是否启用指标，如果为 `false` 则在展示可选指标时，不会展示该检查指标
 - metric_type : 指标类型，支持 `sql` 和 `bash` 两种，默认指标可以不指定指标类型，主要用于自定义指标
 - column_order : 如果检查结果是表格的形式，对于每一列的排列顺序可以通过该字段指定
 - metrics.column_alias : 用于配置检查结果的列名称别名，在报告中会优先展示别名，如果没有别名则展示原始列名称
 - metrics.item_names : 用于配置检查结果的列名对应的告警指标名称，告警指标是组成告警表达式的基本元素
 - number_columns : 用于将相应列对应的告警指标结果转换成数字，数字类型的告警指标可以进行数学运算，默认告警指标的结果为字符串
 - labels : 用于给告警指标添加标签，当检查结果有多列时，为了区分每一列，需要指定某些字段作为每一列的唯一标识。例如 `dba tablespaces` 视图中，每一列的唯一标识字段为：`TABLESPACE_NAME`
 - sql : 用于配置检查相应指标使用的SQL查询语句，`sql` 类型的指标需要
 - metrics.alert_rules : 用于定义检查项的告警和建议，如果需要的话
 - metrics.alert_rules.info : 用于定义提示级别的告警
 - expression : 用于定义告警表达式
 - description : 用于定义告警描述
 - suggestion : 用于定义告警建议
 - metrics.alert_rules.warning : 用于定义警告级别的告警
 - 子字段可参考 `metrics.alert_rules.info`
 - metrics.alert_rules.critical : 用于定义严重级别的告警
 - 子字段可参考 `metrics.alert_rules.info`

健康检查得分评估模型配置文件： `{yhc_home}/config/evaluate_model.toml`

```

# 用于评估检查结果
total_score = 100 # 总分
default_metric_weight = 5 # 未在metrics_weight字段显式指定权重的指标，其默认权重
max_alert_total_weight = 10 # 单个指标产生告警的总权重
ignore_same_alert = false # 单个指标同一级别告警是否只扣一次分，例如如果某一指标产生了十个严重的告警，将只扣一次严重告警的分数
ignore_failed_metric = true # 是否忽略检查失败的指标项

[metrics_weight] # 显式指定某一指标的权重
# 一类指标
yasdb_database = 30
host_disk_info = 30
yasdb_replication_status = 30
yasdb_controlfile_count = 30

# 二类指标
yasdb_tablespace = 20
yasdb_datafile = 20

```

```

yasdb_session = 20
yasdb_undo_size = 20

# 三类指标

# 四类指标
host_huge_page = 7
host_swap_memory = 7
yasdb_security_user_use_system_tablespace = 7
yasdb_redo_log_count = 7

# 五类指标
yasdb_file_permission = 5
yasdb_parameter = 5
yasdb_os_auth = 5
yasdb_controlfile = 5
yasdb_security_password_strength = 5
yasdb_security_maximum_login_attempts = 5
yasdb_security_audit_cleanup_task = 5

[alerts_weight] # 指定告警扣分的权重，某项告警扣分公式为：单个指标对应分数*告警扣分权重/单个指标产生告警的总权重
critical = 3
warning = 2
info = 1

[health_model] # 健康模型，分数对应的健康状态
[health_model.excellent]
min = 95
max = 100
[health_model.good]
min = 80
max = 95
[health_model.fair]
min = 70
max = 80
[health_model.poor]
min = 60
max = 70
[health_model.critical]
min = 0
max = 60

[health_status_alias]
critical = "危急"
excellent = "优秀"
fair = "尚可"
good = "良好"
poor = "较差"

```

该配置文件是YHC工具的健康检查得分评估模型配置文件，负责定义得分评估模型，其中各个字段的含义如下：

- total_score : 健康检查的总分
- default_metric_weight : 未显式指定某一指标权重时，该指标的默认权重
- max_alert_total_weight : 单项指标的所产生的告警的最大的总权重
- ignore_same_alert : 单项指标的同一级别的告警是否只扣一次分，例如当该字段为true时，当某个指标产生10个严重级别的告警，将会只扣一次严重级别的告警的分数
- ignore_failed_metric : 是否忽略检查失败的指标，例如当该字段为true时，检查失败的指标将不参与得分计算
- metrics_weight : 用于显式指定某一具体指标的权重
- modele_weight : 用于批量指定某一模块下所有指标的权重
- alerts_weight : 指定某一级别告警的权重，用于计算该指标对应级别告警应该扣除的分数
- health_model : 用于指定健康模型，即不同分数对应的数据库健康状态
 - health_model.excellent : excellent状态的对应的分数范围，即 [min, max]的闭区间
 - min : 分数最小值
 - max : 分数最大值
 -

- `health_status_alias` : 健康状态的中文别名映射

YHC工具的健康检查得分评估模型采取扣分制，最终得分为各指标实际得分之和。

各指标实际得分 = 指标总分 - 指标产生的告警扣分

指标总分 = 指标权重 / 参与检查所有指标总权重 * 总分

指标产生的告警扣分 = 指标各告警的权重之和 / 指标总权重 * 指标总分

(其中，指标产生的告警扣分最多为单项指标总分，即各指标实际得分最低为0分，不会出现负分情况)

自定义指标配置文件：`{yhc_home}/config/custom_metric.toml`

该配置文件是YHC工具的自定义指标配置文件，如有需要自定义检查指标时可以在此新增，可以移步[自定义检查项](#)获取指导。

以上所有配置项均可按规范自行调整，再次执行时生效

告警表达式

告警表达式简介

告警表达式用于定义告警条件。指标和运算是告警表达式的基本元素，指标+运算组成告警表达式。

指标 (Metric)

一个指标可以由三个部分组成，指标名称、指标值和指标标签，其中指标标签是可选项。

- 指标名称：以字母开始，后续使用的字符可以是：字母、下划线或者数字。例如：cpu_usage (CPU 使用率)。
- 指标值：指标值的数据类型为基本类型，目前支持数字和字符串类型，其中字符串使用英文单引号包裹。例如：cpu_usage == 0.1 (CPU 使用率等于 0.1)，hostname == 'localhost' (主机名称为 localhost)。
- 指标标签：当一个指标名称有多项数据时，可以使用标签标识指标项的额外属性，标签为字符串类型。例如：tablespace_usage{name == 'UNDO'} (名称为 UNDO 的表空间的使用率)。

运算 (Operator)

目前支持以下三种运算：

- 逻辑运算：[&&, ||]。
- 比较运算：[>, >=, <, <=, ==, !=, ~=, in]。
- 算数运算：[+, -, *, /, %]。

常见的运算符使用方法和原本的含义类似，主要说明下以下两个运算符：

- ~=: 用于正则匹配字符串，例如：firewall_status =~ '.*active'，tablespace_usage{name =~ '^U.+'} > 0.5。
- in : 用于判断字符串是否在列表中，例如：firewall_status in ['active', 'inactive']，tablespace_usage{name in ['UNDO', 'USER']} > 0.5。

定义告警表达式

直接说明或许有些抽象，可以通过案例来快速入门。

单一指标告警

不带标签的单一指标告警

一、CPU 使用率告警，假设指标名称为：cpu_usage。指标值的类型为数字，以下是一些告警案例：

告警表达式	告警条件
cpu_usage > 0.75	CPU 使用率 大于 75%
cpu_usage > 0.75 && cpu_usage <= 0.9	CPU 使用率 大于 75% 并且 小于等于 90%
cpu_usage > 0.75 cpu_usage <= 0.001	CPU 使用率 大于 75% 或者 小于等于 0.1%

二、防火墙状态告警，假设指标名称为：firewall_status。指标值的类型为字符串，以下是一些告警案例：

告警表达式	告警条件
firewall_status != 'active'	防火墙状态 不是 活跃状态
firewall_status == 'inactive'	防火墙状态 是 不活跃状态
firewall_status == 'inactive' firewall_status == 'dead'	防火墙状态 是 不活跃状态 或者 关闭状态
firewall_status in ['inactive', 'dead']	防火墙状态 是 不活跃状态 或者 关闭状态
firewall_status =~ '.*active'	防火墙状态 匹配 任意字符后面接 active

额外说明：

- `in` 运算用于判断字符串值是否在字符串列表内，其中字符串列表使用中括号包裹，字符串之间使用英文逗号分割。
- `~=` 运算用于正则匹配字符串，能够匹配则为满足条件。

带标签的单一指标告警

一、表空间使用率告警，假设指标的名称为：`tablespace_usage`。指标值的类型为数字，指标的标签有一个：表空间名称（`name`），以下是一些告警案例：

告警表达式	告警条件
<code>tablespace_usage > 0.8</code>	所有表空间使用率 大于 80%
<code>tablespace_usage{name == 'UNDO'} > 0.9 && tablespace_usage{name == 'UNDO'} < 0.95</code>	UNDO 表空间使用率 大于 90% 并且 小于等于 95%
<code>tablespace_usage{name != 'UNDO'} > 0.7</code>	非 UNDO 表空间的其他表空间使用率 大于 70%
<code>tablespace_usage{name == 'UNDO'} > 0.7 tablespace_usage{name != 'UNDO'} > 0.8</code>	UNDO 表空间的使用率 大于 70% 或者 其他表空间的使用率 大于 80%
<code>tablespace_usage{name != 'UNDO' && name != 'DATA'} > 0.7</code>	非 UNDO 表空间 并且 非 DATA 表空间的其他表空间使用率 大于 70%
<code>tablespace_usage{name != 'UNDO' && name != 'DATA'} > 0.7</code>	非 UNDO 表空间 并且 非 DATA 表空间的其他表空间使用率 大于 70%
<code>tablespace_usage{name =~ '^U.*'} > 0.7</code>	名字以字母 U 开始的表空间使用率 大于 70%
<code>tablespace_usage{name == 'UNDO' name == 'DATA'} > 0.7</code>	名字为 UNDO 或者 DATA 的表空间使用率 大于 70%
<code>tablespace_usage{name in ['UNDO', 'DATA']} > 0.7</code>	名字为 UNDO 或者 DATA 的表空间使用率 大于 70%

多个标签的告警类似，例如：

告警表达式	告警条件
<code>metric_name{label1 == 'value1' label2 == 'value2'} > 1</code>	标签1 等于 value1 或者 标签2 等于 value2 的指标值大于1
<code>metric_name{label1 == 'value1' && label2 == 'value2'} > 1</code>	标签1 等于 value1 并且 标签2 等于 value2 的指标值大于1

多指标组合告警

不带标签的多指标告警

一、CPU 使用率，内存使用率联合告警。假设指标名称分别为：`cpu_usage`，`memory_usage`，以下是一些告警案例：

告警表达式	告警条件
<code>cpu_usage > 0.75 memory_usage > 0.7</code>	CPU 使用率 大于 75% 或者 内存使用率大于 70%
<code>cpu_usage > 0.75 && memory_usage > 0.7</code>	CPU 使用率 大于 75% 并且 内存使用率大于 70%
<code>cpu_usage/memory_usage > 1</code>	CPU 使用率 除以 内存使用率 大于 1 (只是举例，无特殊含义)

带标签的多指标告警

一、当前会话占用率告警，使用当前会话数和最大会话数指标联合告警。假设指标名称分别为：`current_session`，`max_session`，包含一个标签：数据库类型（`database_type`），以下是一些告警案例：

告警表达式	告警条件
<code>current_session/max_session > 0.7</code>	会话占用率大于 70%

告警表达式	告警条件
current_session{database_type == 'se'}/max_session{database_type == 'se'} > 0.7	SE 类型的数据库会话占用率大于 70%

Q&A

TODO 常见问题解决方案指引

术语解释

参考资料