1. 系统使用说明
   1. 将webserver.tar拉到一台linux服务器上然后解压

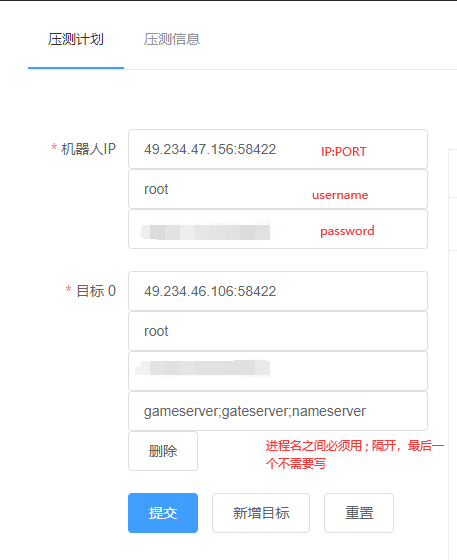
1.2把element.zip拉到本地windows机器，同时安装前端脚手架vue+elementUI+echarts

1.3在webserver所在的服务器启动go后端,之后npm start前端启动

1.4测试计划配置

我们需要指定测试机器人机器和被测服务器机器的相关信息，以及需要关注的进程名

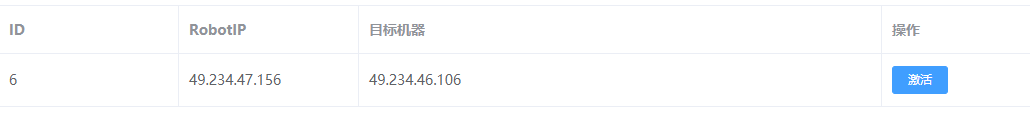
进程名（进程直接使用;隔开）最后一个不需要加上;号



点击提交后会在右边的表格中出现一行压测计划，如果填写的机器人地址和表中的某一行机器人地址相同，则这一行会被新的测试计划覆盖，页面自动刷新存在bug最好是手动刷新页面。

表单提交成功后webserver会给目标机器安装monitor程序和脚本到/root/tools目录下，请确保该目录可用

填写完成后可以点击激活按钮开始上传被测进程数据，在点击激活前被测目标机器上面必须运行被测的进程，否则将无法激活



激活后需要手动去机器人机器上运行机器人启动脚本

1.5测试开始

当机器人启动后会生成一份机器人机器对应的测试报告，该行显示正在测试，测试时详细信息可以查看，但报告不可用



1.6测试结束

结束后会显示测试结束，此时系统会关闭刚才激活在目标机器的监控程序

下次启动需要重新开启

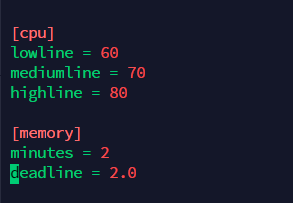
此时报告和信息都可以查询。

报告生成依据webserver程序目录下的report.ini文件

此文件定义了cpu占用率预警线和内存增长幅度预警线

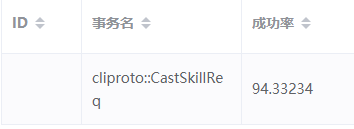
memory中的minutes指测量时间段，是结束时间往前算多少分钟

deadline是计算这段时间的线性回归曲线的斜率值超过这个值预警

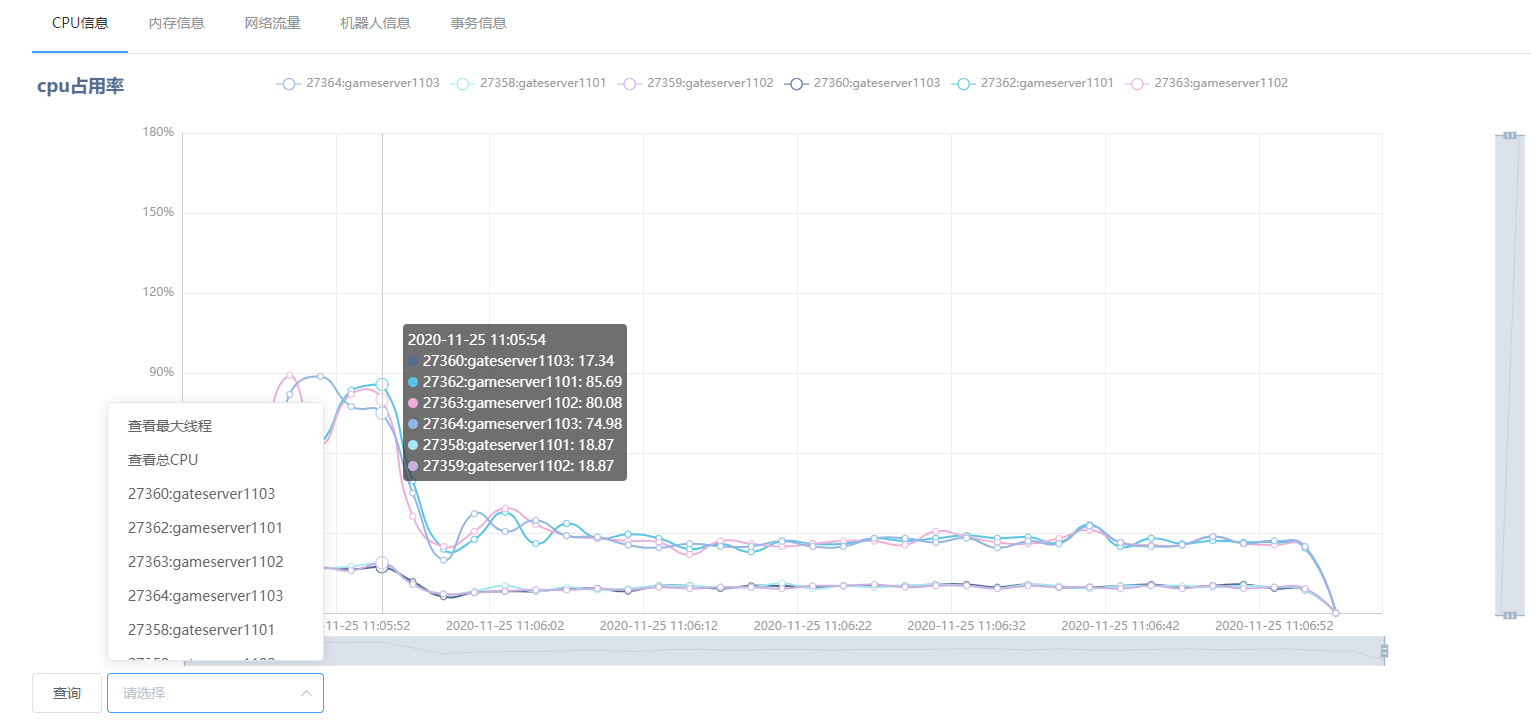


报告中会显示到达预警线的进程名以及一小段曲线，事务成功率固定98需要在前端修改





cpu信息默认显示总cpu占用，可选择查看最大线程占用和某一进场的各个线程占用



关于事务信息，由于数据量比较大需要生成加速表，测试完成后最好等待一分钟。

1. 代码说明

代码分为三部分——monior程序，webserver，前端vue。

monitor编译好后的程序，配置文件和脚本——monitor information.ini run\_monitor.sh需要和webserver在同一目录下。 前端打包好的dist文件夹，报告配置文件report.ini 也必须和webserver在同一目录下。

2. 1 monitor程序

monitor分为三个主要部分，监控类——XXXMonitor继承自IMonitor 完成主要的功能实现。Log类——完成POST请求。其他类——完成配置读取，timer线程创建等功能。

XXXMonitor继承后按照需求重载4个函数。

virtual void Calculate() = 0; 这个函数主要用于计算，每次timer循环都会被调用

virtual void WriteLogs() = 0; 主要是记录数据到log的缓存区

这两个函数没有本质上的区别，各自都能包含对方的功能，只是为了功能划分写成了两个而已。

virtual void SetLogFiles() = 0; 这个函数在线程创建时（timer循环前）调用一次用来创建log文件和log的缓冲区

virtual void FileOpenReady(); 这个函数一样在线程创建时（timer循环前）调用一次 用来打开文件句柄。如果不需要可以不重载。

如果重载了FileOpenReady函数必须先调用一次基类的 FileOpenReady 并且要在析构函数中关闭你打开的文件句柄

Log 类 是全局单例要使用时请指定缓冲区名，也可以调用SetFilename创建缓冲区和对应文件。每次写入缓存时会拿到锁 \_swap\_lock

如果某一缓存大于 MESSAGE\_MAX\_NUM 定义的数量 会临时创建一个线程，并且将整个缓存拷贝个线程里面。

Timer调用一次setInterval 会开启一个线程每隔interval调用一次function

ini.h只要include之后就能使用，mINI::INIFile file(path); mINI::INIStructure ini; file.read(ini); 读取ini文件，存储到ini中，只要按照域查找就能找到字段值。

Ex: value = ini["section"]["key"];

在使用前保证ini文件格式正确

2.2 webserver

webserver 使用了gin框架，api开头的方法是与前端交互使用的。

在全局维护了一个pool。用来检测机器人是否存活。机器人每次上传数据都会将这个pool更新。webserver启动后会开启一个协程ActiveCheck 每隔1s遍历一次pool里面的连接，如果长时间(10s)没有更新，就判断为超时。

当机器人发送stop或者超时时会启动两个协程updateTransResult

和updateTransRecord 这两个是用来创建新的数据库表，来加速事务信息查找。

同时也会关闭机器人对应目标机器激活的monitor程序。

/api/truncate 这个接口会清空所有数据库。慎用

2.3 前端vue

本次前端使用了vue+elementUI+echarts来完成。

App.vue是最上面菜单栏。只有压测计划和压测信息。

压测计划 —— TestPlan.vue

压测信息 —— TestInfo.vue

点击压测信息后会拿到测试的表格。点击查看详细信息会跳转到Index.vue

这个页面有5个子路由来分别查询5个测试信息。

echarts 需要先创建好容器在进行渲染。所以如果要在表格中进行渲染，并且这个表格是从后端请求的话，需要使用this.$nextTick(() =>{}) 来延迟到下一个周期渲染。

2.4 数据库表

三个基本表cpu\_infomation memory\_infomation net\_flow 来记录monitor采集的信息。webserver拿到数据时并不是使用JSON方式，而是自定义数据结构。

robot\_info trans\_info 是用来记录机器人传输的信息。这两个表在webserver上是使用JSON方式。target\_ip 是记录sid和目标机器的表，sid和ip是一对多的关系。

plan\_info 是记录robot机器是和sid的表，一一对应。如果robot重复会覆盖原来的。

server\_info存储服务器账号密码等。

test\_info 记录instanceID sid 开始结束时间以及目标等信息的压测信息表。这个表会在机器人发送start控制信息时进行增加。

trans\_result 和 trans\_record\_result 是用来加速 trans\_info查找的 会在机器人调用stop或超时后进行创建。由于trans\_info数据量巨大，这个过程比较慢(大概需要1分钟)。