

车规级代码扫描 - 操作手册

目录

- 一、license服务的部署
- 二、扫描代码流程
- 三、接入代码扫描工具
 - 3.1 接入方式一：bcloud编译
 - 3.1.1 bcloud-标准编译
 - 1、修改对应分支中的ci.yml文件，增加如下内容
 - 2、修改代码库根目录下的BCLOUD文件
 - 3.1.2 bcloud-交叉编译
 - 3.2 接入方式二：cmake编译+标准编译
 - 1、ci.yml中增加配置
 - 2、编写parasoft_build.sh脚本
 - 3.3 接入方式三：cmake编译+交叉编译
 - 1、ci.yml增加配置
 - 2、写编译+扫描脚本
 - 3.4. 流水线配置
 - 3.5. 扫描文件过滤 规则配置
 - 1、自定义扫描指定的文件
 - 2、自定义扫描排除的文件
 - 3.6 指定扫描规则
 - 3.7 扫描报告解读
 - 1、圈复杂度 扫描报告
 - 2、查看报告中某个规则的详情
- 常见问题：
 - 1、执行生成bdf文件时 报错【XXXX.so文件file format not recognized; treating as linker script】
 - 2、扫描过程中有规则报错，导致扫描结束
 - 3、扫描任务超过2小时，自动中断。
 - 4、代码库大，扫描时间过长，并发扫描
 - 5、cmake编译的代码，如果始终无法生成.bdf文件，则可以使用json形式进行扫描

❗ 业务方接入，直接看第三部分！

i 工具说明：

当前**买的是parasoft的静态分析，没有买单测能力**（parasoft本身是支持单测的）

parasoft的静态分析主要三个功能：

- 1.编码规范，即强制代码遵从某一编码标准，例如misra c 2012
- 2.流分析，也就是找bug,可以找到100多种bug
- 3.度量指标，衡量代码各种标准，例如圈复杂度等

官方使用文档链接：

- 整体文档：<https://docs.parasoft.com/display/CPPTTESTPROEC20231>
- 静态扫描：<https://docs.parasoft.com/display/CPPTTEST20231/Running+Static+Analysis+1>

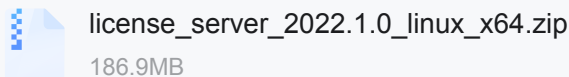
i 接parasoft工具的大概流程：

本地编译生成makefile文件 --》根据makefile生成.bdf文件 --》根据bdf文件进行代码扫描 --》将扫描报告上传至指定服务器并将报告地址写回流水线 --》从流水线上拿扫描报告

一、license服务的部署

服务部署的机器：work@10.104.87.18

安装包目录：/home/work/license_server



服务启动命令：上述压缩包，解压后在/home/work/license_server/license-server目录下执行：
app/startLS.sh

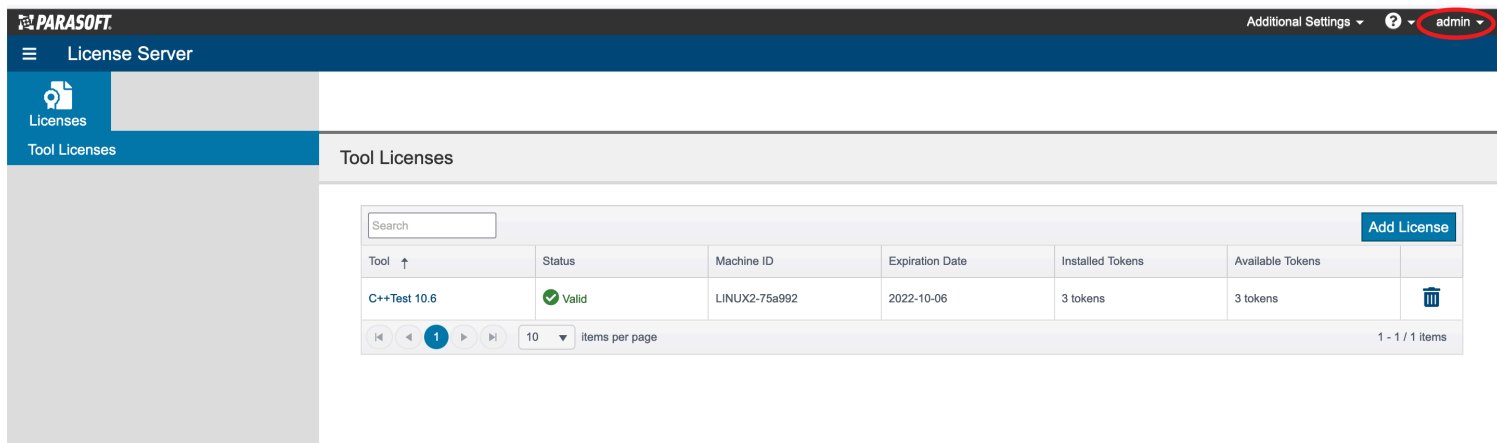
服务启动后的端口：8443

确认服务是否启动成功：

- 看进程：ps -ef|grep license-server

```
work@yq01-adu-m12xi2-043.yq01.baidu.com:~/license_server/license-server$ps -ef|grep license-server
work      3492      1   0 16:12 pts/2    00:01:51 /home/work/license_server/license-server/app/jre/bin/java -Djava.util.logging.config.file=/home/work/license_server/license-server/app/tomcat/conf/logging.properties -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Dsun.jnu.encoding=UTF-8 -Dfile.encoding=UTF-8 -Ddtp.datadir=/home/work/license_server/license-server/data -Dcom.parasoft.xtest.logging.config.file=/home/work/license_server/license-server/app/log4j.xml -Djdk.tls.ephemeralDHKeySize=2048 -Djava.protocol.handler.pkgs=org.apache.catalina.webresources -Dorg.apache.catalina.security.SecurityListener.UMASK=0027 -Dignore.endorsed.dirs= -classpath /home/work/license_server/license_server/app/tomcat/bin/bootstrap.jar:/home/work/license_server/license-server/app/tomcat/bin/tomcat-juli.jar -Dcatalina.base=/home/work/license_server/license-server/app/tomcat/temp org.apache.catalina.startup.Bootstrap start
work      39037 36073   0 19:29 pts/3    00:00:00 grep license-server
```

- 浏览器上访问：<https://10.104.87.18:8443>



二、扫描代码流程


以baidu/asd/taskmgr代码为例，扫描流程：

测试使用的机器：~~ssh work@asd-ee-01.bcc-gzhxy.baidu.com~~ （自己找个机器就行）

1、下载cpptest安装包，并配置license信息

安装目录：**/home/work/yangkang08/code_scan**

1) 下载cpptest安装包，并解压

 parasoft_cpptest_professional-2022.1.0-linux.x86_64.tar.gz
595.3MB

2) 在/home/work/yangkang08/code_scan/cpptest目录下创建settings_network.txt文件

`</> settings_network.tx`

C++

```
1 cpptest.license.network.edition=server_compliance_edition
2 cpptest.license.use_network=true
3 license.network.auth.enabled=false
4 license.network.url=https\://10.104.87.18\:8443
5 license.network.use.specified.server=true
6 bdf.import.compiler.family=gcc_7-64
7 cpptest.compiler.family=gcc_7-64
8 parasoft.eula.accepted=true
```

2、下载要扫描的代码

目录code_path：**/home/work/yangkang08/baidu/asd/taskmgr**

3、生成makefile文件

代码三级目录下，执行命令：bcloud local --target-arch="x86_64" --profile-name="x86_64" --compiler="gcc750-x86_64-ubuntu1804-gnu" --disable-cc-cxx

4、生成.bdf文件

```
/home/work/yangkang08/code_scan/cpptest/bin/cpptesttrace --cpptesttraceProjectName=XXXXXX --cpptesttraceOutputFile=XXXXXX.bdf make
```

5、扫描代码

```
cpptest_path=/home/work/yangkang08/code_scan/cpptest
```

```
code_path=/home/work/yangkang08/baidu/asd/taskmgr
```

```
module_name=taskmgr
```

执行命令：

```
${cpptest_path}/bin/cli/cpptestcli -bdf ${code_path}/${module_name}.bdf -settings
```

```
${cpptest_path}/settings_network.txt -report ${code_path}/scan_report -config "builtin://$rule" -exclude  
${cpptest_path}/exclude.lst
```

其中：-exclude 指定扫描时要排除的文件。多个需要在XX.lst文件中指定。

```
</> exclude.lst
```

Bash

```
1 **/*.pb.cc  
2 **/*.pb.h
```

上述第4、5步的自动化脚本，见代码库：baidu/adu-3rd/cpptest的run_code_scan.sh脚本

三、接入代码扫描工具

- ❗ 代码库的编译方式为bcloud的，按3.1的方法接入；
- 编译方式为cmake，且为标准编译，参考3.2的方法接入；
- 编译方式为cmake，且为交叉编译，参考3.3的方法接入；

下面以taskmgr模块为例介绍parasoft的接入方法：

3.1 接入方式一：bcloud编译

3.1.1 bcloud-标准编译

修改代码（可以完全照抄，不用改~）

1、修改对应分支中的ci.yml文件，增加如下内容

i 如果项目实际使用的bcloud local需要加额外的参数，可以修改下面的command命令中『bcloud local那部分命令』!!! 能够正常编译就不影响扫描。

</> ci.yml

Bash

```
1 - profile:
2   name : bcloud_parasoft
3   mode : AGENT
4   environment:
5     image: iregistry.baidu-int.com/idg-public/ubuntu-18_04:bcloud_dev_20220920
6   build:
7     command : export PATH=/root/.BCloud/bin:$PATH && bcloud local --target-arch="x86_64" --profile-name="x86_64" --compiler="gcc750-x86_64-ubuntu1804-gnu" --disable-cc-cxx --update && bash ../../adu-3rd/cpptest/run_code_scan.sh
8   artifacts:
9     release : true
```

百度 云上百度 iCode 代码库 / ☆ baidu/asd/taskmgr / 文件

搜索或者跳转资源

+ 新建

△ ⓘ ≡ ⚙ ...

```
115   cluster: ADU-L3-HWP
116   build:
117     command : mkdir output && bash -c './fetch_code.sh && bash run_docker.sh --target sanxian'
118   artifacts:
119     release : true
120
121 - profile:
122   name : bcloud_x86
123   mode : BCLLOUD
124   build:
125     command : bcloud build --target-arch="x86_64" --compiler="gcc750-x86_64-ubuntu1804-gnu" --profile-name="x86_64" --ubuntu18-asd
126   artifacts:
127     release : true
128
129 - profile:
130   name : bcloud_x86_coverage
131   mode : BCLLOUD
132   build:
133     command : bcloud build --target-arch="x86_64" --compiler="gcc750-x86_64-ubuntu1804-gnu" --profile-name="x86_64_coverage" --coverage --ubuntu18-asd
134   artifacts:
135     release : true
136
137 - profile:
138   name : bcloud_aarch64
139   mode : BCLLOUD
140   build:
141     command : bcloud build --target-arch="aarch64" --compiler="gcc750-aarch64-ubuntu1804-gnu" --profile-name="aarch64" --ubuntu18-asd
142   artifacts:
143     release : true
144
145 - profile:
146   name : bcloud_parasoft
147   mode : AGENT
148   environment:
149     image: iregistry.baidu-int.com/idg-public/ubuntu-18_04:bcloud_dev_20220920
150   build:
151     command : export PATH=/root/.BCloud/bin:$PATH && bcloud local --target-arch="x86_64" --profile-name="x86_64" --compiler="gcc750-x86_64-ubuntu1804-gnu" --disable-cc-cxx --update && sh ../../adu-3rd/cpptest/run_code_scan.sh
152   artifacts:
153     release : true
```

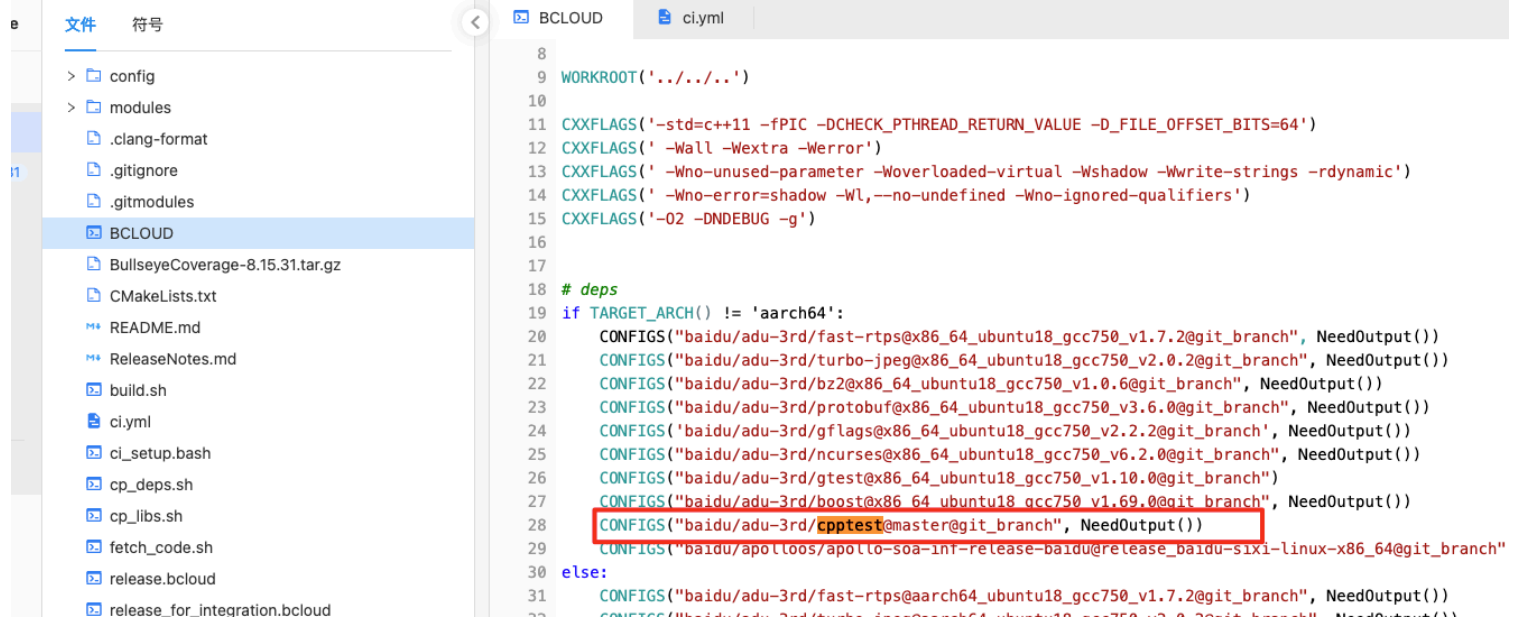
2、修改代码库根目录下的BCLLOUD文件

在x86的逻辑中增加一行：CONFIGS("baidu/adu-3rd/cpptest@master@git_branch", NeedOutput())

如果不是x86的，根据业务需要 在对应位置添加即可。

百度云 | iCode | 代码库 | ☆ baidu/asd/taskmgr / 文件

搜索或者



```
8
9 WORKROOT(' ../../..')
10
11 CXXFLAGS('-std=c++11 -fPIC -DCHECK_PTHREAD_RETURN_VALUE -D_FILE_OFFSET_BITS=64')
12 CXXFLAGS('-Wall -Wextra -Werror')
13 CXXFLAGS('-Wno-unused-parameter -Woverloaded-virtual -Wshadow -Wwrite-strings -rdynamic')
14 CXXFLAGS('-Wno-error=shadow -Wl,--no-undefined -Wno-ignored-qualifiers')
15 CXXFLAGS('-O2 -DDEBUG -g')
16
17
18 # deps
19 if TARGET_ARCH() != 'aarch64':
20     CONFIGS("baidu/adu-3rd/fast-rtps@x86_64_ubuntu18_gcc750_v1.7.2@git_branch", NeedOutput())
21     CONFIGS("baidu/adu-3rd/turbo-jpeg@x86_64_ubuntu18_gcc750_v2.0.2@git_branch", NeedOutput())
22     CONFIGS("baidu/adu-3rd/bz2@x86_64_ubuntu18_gcc750_v1.0.6@git_branch", NeedOutput())
23     CONFIGS("baidu/adu-3rd/protobuf@x86_64_ubuntu18_gcc750_v3.6.0@git_branch", NeedOutput())
24     CONFIGS("baidu/adu-3rd/gflags@x86_64_ubuntu18_gcc750_v2.2.2@git_branch", NeedOutput())
25     CONFIGS("baidu/adu-3rd/ncurses@x86_64_ubuntu18_gcc750_v6.2.0@git_branch", NeedOutput())
26     CONFIGS("baidu/adu-3rd/gtest@x86_64_ubuntu18_gcc750_v1.10.0@git_branch")
27     CONFIGS("baidu/adu-3rd/boost@x86_64_ubuntu18_gcc750_v1.69.0@git_branch", NeedOutput())
28     CONFIGS("baidu/adu-3rd/cpptest@master@git_branch", NeedOutput())
29     CONFIGS("baidu/apolloos/apollo-soa-int-release-baidu@release-baidu-sixi-linux-x86_64@git_branch"
30 else:
31     CONFIGS("baidu/adu-3rd/fast-rtps@aarch64_ubuntu18_gcc750_v1.7.2@git_branch", NeedOutput())
32     CONFIGS("baidu/adu-3rd/turbo-jpeg@aarch64_ubuntu18_gcc750_v2.0.2@git_branch", NeedOutput())
33     CONFIGS("baidu/adu-3rd/bz2@aarch64_ubuntu18_gcc750_v1.0.6@git_branch", NeedOutput())
34     CONFIGS("baidu/adu-3rd/protobuf@aarch64_ubuntu18_gcc750_v3.6.0@git_branch", NeedOutput())
35     CONFIGS("baidu/adu-3rd/gflags@aarch64_ubuntu18_gcc750_v2.2.2@git_branch", NeedOutput())
36     CONFIGS("baidu/adu-3rd/ncurses@aarch64_ubuntu18_gcc750_v6.2.0@git_branch", NeedOutput())
37     CONFIGS("baidu/adu-3rd/gtest@aarch64_ubuntu18_gcc750_v1.10.0@git_branch")
38     CONFIGS("baidu/adu-3rd/boost@aarch64_ubuntu18_gcc750_v1.69.0@git_branch", NeedOutput())
39     CONFIGS("baidu/adu-3rd/cpptest@master@git_branch", NeedOutput())
40     CONFIGS("baidu/apolloos/apollo-soa-int-release-baidu@release-baidu-sixi-linux-aarch64@git_branch"
```

3.1.2 bcloud-交叉编译

提供以下两个内容，找yangkang08接入：

1. 编译器的名称
2. 本地编译的命令

3.2 接入方式二：cmake编译+标准编译

目前使用的模块：baidu/acu/gaia_sixi (develop_sixi分支)、baidu/acu/autosar_tda4 (MCU_master_Platform)

1、ci.yml中增加配置

以下两个参数，需要根据自己的模块进行修改。

- image：下面的镜像是公共的，里面装的软件较少。
- command：本地编译产出makefile文件+扫描逻辑。

</> ci.yml (以baidu/acu/gaia_sixi模块为例)

Bash

```
1 - profile:
2   name: bcloud_parasoft
3   mode: AGENT
4   environment:
```

```

5      image: iregistry.baidu-int.com/idg-public/ubuntu-
      18_04:bcloud_dev_20221011_with_cmake
6      resourceType: SMALL
7      build:
8        command: bash parasoft_build.sh all linux-x86_64 wm
9      artifacts:
10     release: true

```

2、编写parasoft_build.sh脚本

脚本所做的事情：对模块进行本地编译生成MakeFile文件，然后调parasoft的扫描流程。扫描完成后会将报告地址写到流水线上，从流水线上获取报告。

- parasoft_build.sh脚本：代码位置：~~baidu/acu/gaia_sixi~~代码库develop_sixi分支
- parasoft_build.sh → ~~build_board_parasoft.sh~~ 在board/wm目录下。
- ~~build_board_parasoft.sh~~ → acuAutosarTda4_scan.sh脚本，进行的代码扫描
- acuAutosarTda4_scan.sh脚本在：~~baidu/adu-3rd/cpptest~~ 代码库master分支的根目录下

步骤：1、下载cpptest代码库； 2、执行代码扫描脚本。以下接入代码 仅供参考

</>

Bash

```

1  #!/bin/bash
2
3  function download_parasoft() {
4      # 下载parasoft工具
5      echo "开始下载parasoft工具"
6      cd $current_pwd
7      if [ ! -d cpptest ]; then
8          #git clone ssh://yangkang08@icode.baidu.com:8235/baidu/adu-3rd/cpptest
          cpptest
9          git clone
          ssh://${AGILE_PIPELINE_TRIGGER_USER}@icode.baidu.com:8235/baidu/adu-3rd/cpptest
          cpptest && curl -s http://icode.baidu.com/tools/hooks/commit-msg >
          cpptest/.git/hooks/commit-msg && chmod u+x cpptest/.git/hooks/commit-msg && git
          config -f cpptest/.git/config user.name ${AGILE_PIPELINE_TRIGGER_USER} && git
          config -f cpptest/.git/config user.email ${AGILE_PIPELINE_TRIGGER_USER}@baidu.com
10     else
11         git pull
12     fi
13     echo "下载结束。。。。"
14 }
15 # =====main=====
16 # 使用: ci.yml中: mkdir -p output && bash build.sh -ncbi && bash parasoft_build.sh
17 # 1. 拉cpptest
18 download_parasoft

```

```
19
20 # 2. 执行扫描逻辑
21 rule_name=${rule_name} # 从流水线传
22 # 参数说明：以根目录为坐标，参数1：cpptest的目录 参数2： 是json还是bdf 参数3： json文件路径
    或Makefile路径 参数4：要扫的规则名称
23 # json格式的传参demo
24 bash ./cpptest/run_code_scan_cmake.sh "cpptest" "json"
    "src/build/compile_commands.json" ${rule_name}
25 # bdf格式的传参demo,如果生成的Makefile在代码根目录下，则参数3传空字符串。 如果不在根目录下，则
    传makefile所在的目录
26 bash ./cpptest/run_code_scan_cmake.sh "cpptest" "bdf" "src/build" ${rule_name}
```

3.3 接入方式三：cmake编译+交叉编译

使用的模块：baidu/apolloos/ota (byd-anp分支)

1、ci.yml增加配置

</> ci.yml


Bash

```
1 - profile:
2     name: bcloud_parasoft
3     env: ADU-L3-CENTOS7
4     command: mkdir -p output && bash build.sh -d && bash build.sh -pcm && bash
    ./cpptest/run_code_scan_apolloos.sh ./cpptest
5     release: true
```

上述command中：

- mkdir -p output && bash build.sh -d && bash build.sh -pcm 命令负责生成makefile文件，并拉cpptest代码库的代码。
 - bash ./cpptest/run_code_scan_apolloos.sh ./cpptest 进行代码扫描并将报告地址回传到流水线上
- run_code_scan_apolloos.sh扫描脚本，在baidu/adu-3rd/cpptest代码库中。

2、写编译+扫描脚本

 交叉编译，通用的扫描命令要改下：

- 1、生成bdf文件的命令，增加参数指定编译器名字：--cpptesttraceTraceCommand="aarch64-none-linux-gnu-g++" 编译器名称根据业务需要修改
- 2、扫描命令，增加参数：-compiler gcc_9-64_custom0 （参数的值 由parasoft的同学给的）

</> 拉cpptest的代码 (parasoft的工具)

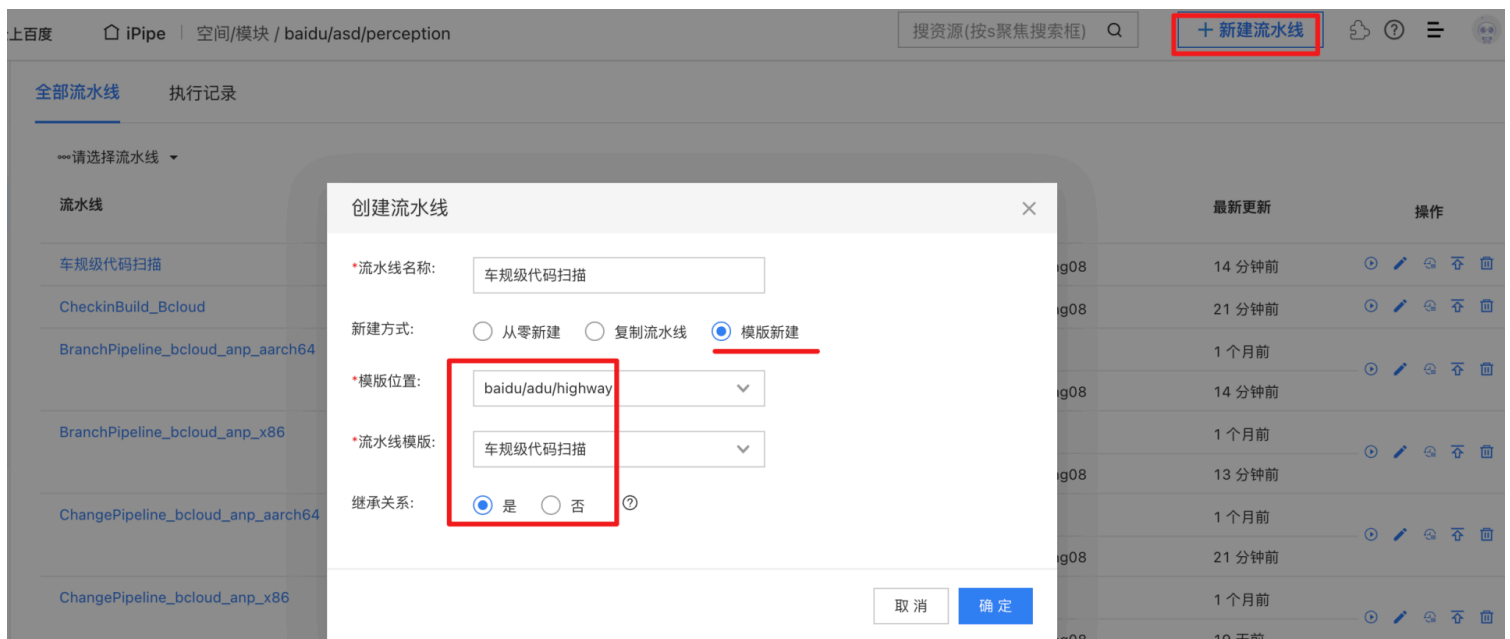
Bash

```
1 function download_parasoft() {
2     # download parasoft tool
3     echo "start download parasoft tool"
4     cd $current_pwd
5     if [ ! -d cpptest ]; then
6         #git clone
7         ssh://${AGILE_PIPELINE_TRIGGER_USER}@icode.baidu.com:8235/baidu/adu-3rd/cpptest
8         cpptest
9         git clone
10        ssh://${AGILE_PIPELINE_TRIGGER_USER}@icode.baidu.com:8235/baidu/adu-3rd/cpptest
11        cpptest && curl -s http://icode.baidu.com/tools/hooks/commit-msg >
12        cpptest/.git/hooks/commit-msg && chmod u+x cpptest/.git/hooks/commit-msg && git
13        config -f cpptest/.git/config user.name ${AGILE_PIPELINE_TRIGGER_USER} && git
14        config -f cpptest/.git/config user.email ${AGILE_PIPELINE_TRIGGER_USER}@baidu.com
15    else
16        git pull
17    fi
18    echo "download complete"
19 }
```

3.4. 流水线配置

按模板的方式创建流水线，操作如下：

点击确定后，直接保存不用做任何修改！



3.5. 扫描文件过滤 规则配置



提了代码找杨康或陈潜过

1、自定义扫描指定的文件

下载[baidu/adu-3rd/cpptest](#)的master分支的代码，修改src/cpptest/includes下面对应的文件。文件名是以 **include_模块名.lst**命名，找到对应的文件，增加要扫描的正则表达式即可（一个表达式一行）

以**baidu/asd/em_avp**模块为例进行说明：

模块名：代码库路径的最后一段内容：em_avp

文件名：include_em_avp.lst

文件内容：

```
</> include_em_avp.lst
1 **/em_avp/AVP/**
2 **/em_avp/ANP/**
```

Bash

2、自定义扫描排除的文件

下载[baidu/adu-3rd/cpptest](#)的master分支的代码，修改src/cpptest/excludes下面对应的文件。文件名是以 **exclude_模块名.lst**命名，找到对应的文件（没有就新建一个），增加要扫描的正则表达式即可（一个表达式一行）

说明：如果没有查到模块对应的排除文件，则默认使用公共的排除文件：**exclude_common.lst**

```
</> exclude_common.lst内容（以代码库中的内容为准）
1 **/**/*.pb.cc
2 **/**/*.pb.h
3 **/bc_out/**
```

Bash

3.6 指定扫描规则

目前仅支持自定义：MISRA C 2012、MISRA C++ 2008、**AUTOSAR C++14 Coding Guidelines**、MISRA C 2012 & MISRA C++ 2008 和 **HIS Source Code Metrics**（这个是扫圈复杂的）、**Flow Analysis Standard**（扫描代码缺陷的）6种规则集

指定方法：**默认使用MISRA C 2012 & MISRA C++ 2008规则集**

需要修改流水线配置：增加一个rule_name参数，如下图所示。

百度智能云 | 度厂版

iPipe | 空间/模块 / baidu/asd/dpc / 车规级代码扫描

流水线iPipe

空间/模块

流水线

模板

场景

发布记录

回收站

数据统计

配置

基本信息

阶段任务设置

通用设置

源与触发事件

代码库源

baidu/asd/dpc

代码库触发

baidu/asd/dpc

dev_pangu

1 车规级代码扫描

车规级代码扫描

+ 新建并行任务

编辑阶段

*阶段名称: 车规级代码扫描

触发方式: ☐ 自动触发 ☒ 手动触发 ☐ 定时触发

失败策略: ☐ 快速失败 ☒ 自然失败

构建参数: + 新增参数

排序	参数名	参数类型	执行时隐藏	默认值	说明	操作
三	report_url	单行文本	否		扫描报告url	✎
三	rule_name	单行文本	否	HIS Source Code Metrics		✎

这个名称一定不要改，照抄

按文档写对应的规则名称即可

如果没有办法加新参数，可以先跟模板解除后，再重新编辑试下。解除方法如下：

百度智能云 | 度厂版

iPipe | 空间/模块 / baidu/asd/dpc_avp / 车规级代码扫描

🔍 K 搜资源(按s聚焦搜索)

流水线iPipe

空间/模块

流水线

模板

场景

发布记录

回收站

数据统计

配置

效能数据

矩阵

基本信息

阶段任务设置

通用设置

*流水线名称: 车规级代码扫描

流水线备注: 请输入流水线备注 0 / 200

解除关联: ☒ 解除模版与流水线的关联

勾上 点保存后 再加rule_name参数就行了

保存

关闭

3.7 扫描报告解读

基本的报告信息 比较好理解，不在此赘述。 简单介绍以下几点：

1、圈复杂度 扫描报告

圈复杂度扫描时用的parasoft内置的规则集： builtin://HIS Source Code Metrics，除了圈复杂度，还包括其他规则。

圈复杂度，阈值是10，如果代码的圈复杂度超过10，在报告中才会显示！

如只看圈复杂度，则在报告中找METRICS-18规则对应的扫描内容，如果没有找到对应规则的记录，就说明

当前代码的圈复杂度值都在10以内。

All Findings by Category

[14] Coding Conventions (CODSTA)	
[14] A function shall have at most one exit point (CODSTA-91-3)	
[1] Metrics (METRICS)	
[2] The number of statements within function should be in range 1 - 50 (METRICS-38-3)	圈复杂度，找对应规则对应的记录，前面的数字代表 不符合的有几个
[2] The value of VOCF metric for a function should not be higher than 4 (METRICS-39-3)	
[8] The number of blocks of comments before and inside function to the number of statements in function should be > 0.2 (METRICS-41-3)	

Findings by File

Expand All Collapse All Back to Top

+ 17 (0) Total (Suppressed)		
+ 17 (0) VehicleStateMgr		
+ 3 (0) base_mock		
+ 3 (0) mock_function		
+ 3 (0) mock.c		
18: Number of blocks of comments per statement in the function 'GetMcuLocalTime': 0.0	work	METRICS-41-3
27: Number of blocks of comments per statement in the function 'umb_pub': 0.0	work	METRICS-41-3
53: Number of blocks of comments per statement in the function 'check_current_automode': 0.0	work	METRICS-41-3
0 (0) mock_misrc.c		
+ 13 (0) VehicleStateMgr.c		
40: Number of blocks of comments per statement in the function 'get_state_m_result': 0.0	work	METRICS-41-3
45: Number of blocks of comments per statement in the function 'set_avp_buildings': 0.0	work	METRICS-41-3
50: Number of blocks of comments per statement in the function 'umbstate_machine_pub': 0.0	work	METRICS-41-3
73: Function 'VehicleStateMgr_20ms_Runnable' has high Cyclomatic Complexity value: 23	work	METRICS-18-3
73: Function 'VehicleStateMgr_20ms_Runnable' contains over 50 statements: 66	work	METRICS-38-3
73: The value of VOCF metric for function 'VehicleStateMgr_20ms_Runnable' should not be higher than 4: VOCF = 7.17	work	METRICS-39-3
157: Avoid statements nested deeper than 4 levels	work	METRICS-40-3
160: Avoid statements nested deeper than 4 levels	work	METRICS-40-3
168: Avoid statements nested deeper than 4 levels	work	METRICS-40-3

在此处找：METRICS-18-3对应的记录，就是圈复杂度不满足10的代码

2、查看报告中某个规则的详情

Findings by File

Expand All Coll

+ 17 (0) Total (Suppressed)		
+ 17 (0) VehicleStateMgr		
+ 3 (0) base_mock		
+ 3 (0) mock_function		
+ 3 (0) mock.c		
18: Number of blocks of comments per statement in the function 'GetMcuLocalTime': 0.0	work	METRICS-41-3
27: Number of blocks of comments per statement in the function 'umb_pub': 0.0	work	METRICS-41-3
53: Number of blocks of comments per statement in the function 'check_current_automode': 0.0	work	METRICS-41-3

从对应的记录中拿到对应的规则名称，然后在下面的规则文档中搜索。如记录中显示**METRICS-41-3**，则就用**METRICS-41**搜

-  cpptest_rules中文版.pdf
16.4MB
-  cpptest_rules英文版.pdf
15.0MB

常见问题：

1、执行生成bdf文件时 报错【XXXX.so文件file format not recognized; treating as linker script】

可找到so文件执行fileXXX.so文件，如果格式变成了ASCII text，则安装上git lfs，删掉os文件对应的源码和bc_out内容，重新生成makefile和bdf文件

2、扫描过程中有规则报错，导致扫描结束

调试：在settings_network.txt中增加以下3行，然后本地进行扫描，把【techsupport.archive.location】目录下的文件发给parasoft的同学进行排查

```
techsupport.enabled=true
```

```
techsupport.create.on.exit=true
```

```
techsupport.archive.location=/home/work/yangkang08/baidu/adu-3rd/cpptest/test
```

3、扫描任务超过2小时，自动中断。

解决：在ci.yml中设置agent的超时时间

[http://buildcloud.baidu.com/bcloud/5-best-practice#11.-](http://buildcloud.baidu.com/bcloud/5-best-practice#11.-%E8%87%AA%E5%AE%9A%E4%B9%89%E6%9E%84%E5%BB%BA%E8%B6%85%E6%97%B6%E6%97%B6%E9%97%B4)

<http://buildcloud.baidu.com/bcloud/5-best-practice#11.-%E8%87%AA%E5%AE%9A%E4%B9%89%E6%9E%84%E5%BB%BA%E8%B6%85%E6%97%B6%E6%97%B6%E9%97%B4>

4、代码库大，扫描时间过长，并发扫描

1) 按3.5.1 配置include_XX文件，按文件内容并发扫描。（按行并发扫）

2) ci.yml文件 扫描的脚本改为：bash ../../adu-3rd/cpptest/run_code_scan_multi.sh 1

5、cmake编译的代码，如果始终无法生成.bdf文件，则可以使用json形式进行扫描

在编译的cmake ..命令中 增加 -DCMAKE_EXPORT_COMPILE_COMMANDS=1

编译完成后会生成一个XXX.json文件。如： 评审：ADUQA-COMPONENT-300 接入parasoft

因此接入parasoft时可以跳过生成bdf的部署，直接进行扫描，扫描命令中-bdf改为-input，后面的bdf文件改

为json文件路径。

如：

库 / ☆ baidu/adu-3rd/cpptest / 文件

⌘ K

搜索或者跳转资源

Q

+ 新建

⚠

?

≡

🎁

🔍

▼

...

run_code_scan.sh

scan_code.sh

```
4 code_path=$2
5 module_name=$3
6 rule=$4
7 include_cmd=$5
8 exclude_cmd=$6
9 compiler_name=$7
10 check_type=$8
11 file_path=$9
12
13 if [ "${compiler_name}" = "aarch64-unknown-nto-qnx7.1.0-gcc" ];then
14     compiler_cmd="-compiler gcc_8-64_custom0"
15 elif [ "${compiler_name}" = "aarch64-none-linux-gnu-gcc" ];then
16     compiler_cmd="-compiler gcc_9-64_custom0"
17 else
18     compiler_cmd=""
19 fi
20
21 echo "check_type=${check_type}"
22 if [ "$check_type" == "json" ];then
23     scan_cmd="${cpptest_path}/bin/cli/cpptestcli -compiler gcc_9-64_custom0 -input ${file_path} -settings ${cpptest_path}/settings_network.txt -report ${code_path}/scan_report -config \"${rule}\" ${include_cmd} ${exclude_cmd}"
24 else
25     scan_cmd="${cpptest_path}/bin/cli/cpptestcli -compiler gcc_9-64_custom0 -bdf ${code_path}/${module_name}.bdf -settings ${cpptest_path}/settings_network.txt -report ${code_path}/scan_report -config \"${rule}\" ${include_cmd} ${exclude_cmd}"
26 fi
27 echo "scan_cmd=$scan_cmd"
28 eval $scan_cmd
```

进入IDE阅读