ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences | 智能软件研究中心 Intelligent Software Research Center

学习V8的测试

智能软件研究中心 邱吉
qiuji@iscas.ac.cn

2021/06/13

Outline

- 为什么要学习V8的测试
- V8的测试如何运行
- V8的测试包含哪些项目
- 测试框架的运行流程
- 如何控制测试的测试集合和variant

Outline

- 为什么要学习V8的测试
- V8的测试如何运行
- V8的测试包含哪些项目
- 测试框架的运行流程
- 如何控制测试的测试集合和variant

为什么要学习V8的测试

- 测试集也是一款软件的重要部分，是学习V8的一种途径
  - 单元测试：了解软件的模块划分
  - 性能测试：软件的性能测试、分析和改进策略的评估
  - 标准测试：了解和跟踪语言特性的实现和演进
  - message测试：了解软件如何报错、如何退出
  - debugger测试：了解如何调试
- 修改V8后
  - 需要通过测试集验证是否break 了原有功能
  - 需要添加新的测试来保证新功能的正确
- Upstream过程中
  - 需要了解CI中的回归测试规则，看懂测试log

前置学习内容

- 基础：能够编译V8代码
  - https://v8.dev/docs/build
- 进阶：对V8的构建系统有一定了解
  - 从零开始分析V8的构建系统构成 part1/2/3
  - https://www.bilibili.com/video/BV1K7411a7tQ

参考内容

- 官方文档： https://v8.dev/docs/test
- 技术博客 https://medium.com/compilers/testing-the-v8-javascript-engine-cbda7d9272e6 by Peter Smith

Outline

- 为什么要学习V8的测试
- V8的测试如何运行
- V8的测试包含哪些项目
- 测试框架的运行流程
- 如何控制测试的测试集合和variant

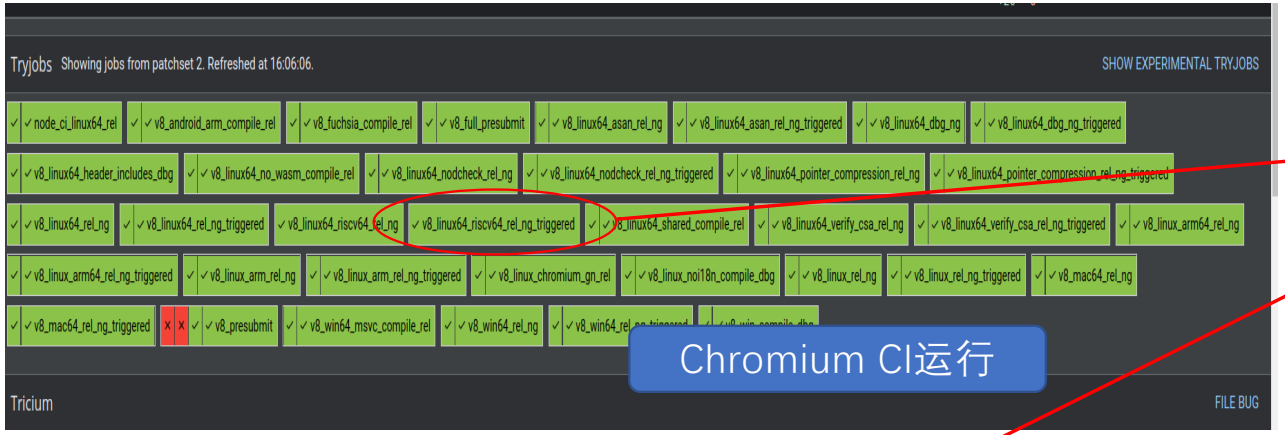
V8的测试如何运行

- 运行方法：
  - ./tools/dev/gm.py riscv64.release.check: 编译并运行default的测试集
  - ./tools/dev/gm.py riscv64.release.checkall：编译并运行所有的测试集
  - ./tools/dev/gm.py riscv64.release benchmarks：编译并运行benchmarks
  - tools/run-tests.py --progress=verbose --outdir out/build --random-seed=-976563914 bot_default --variants=more,dev --rerun-failures-count=2 --mastername tryserver.v8 --buildername v8_linux64_riscv64_rel_ng_triggered --swarming --json-test-results /b/s/w/iosm0s_iu_/output.json

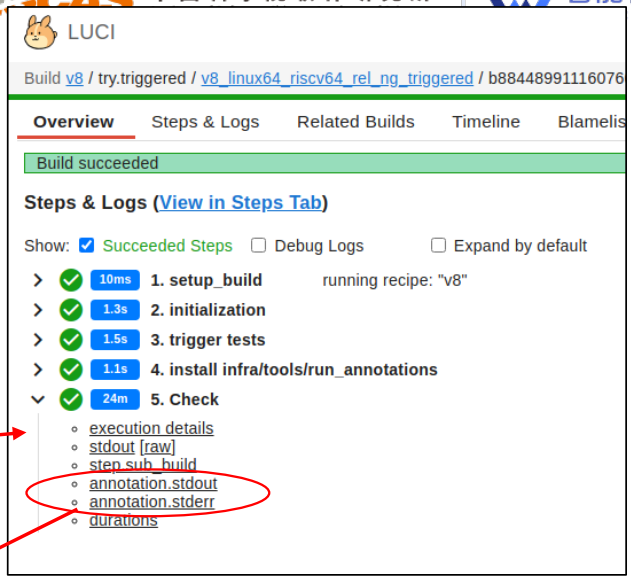
成功运行的结果

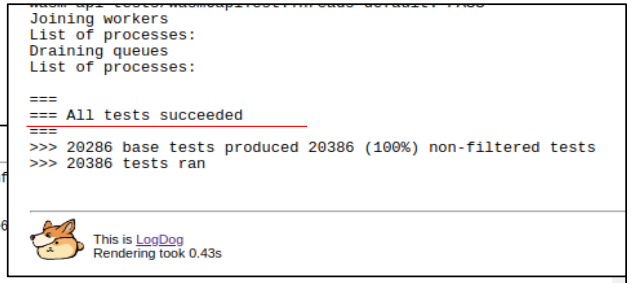
```
v8@plct-build-7:./tools/dev/gm.py riscv64.release benchmarks
# autoninja -C out/riscv64.release d8
ninja: Entering directory `out/riscv64.release'
ninja: no work to do.
# "/usr/bin/python2" tools/run-tests.py --outdir=out/riscv64.release benchmarks
Build found: /home/qjivy/work/v8/v8/out/riscv64.release
>>> Autodetected:
webassembly
>>> Running tests for riscv64.release
>>> Running with test processors
[02:43|% 100|+  55|-   0]: Done
>>> 55 base tests produced 55 (100%) non-filtered tests
>>> 55 tests ran
Done! - V8 compilation finished successfully.
```

本地运行



Chromium CI运行





https://chromium-review.googlesource.com/c/v8/v8/+/2945175

失败运行的结果


```
Command: /b/s/w/ir/out/build/d8 --test /b/s/w/ir/test/mjsunit/mjsunit.js /b/s/w/ir/test/mjsunit

===
=== 22 tests failed
===
>>> 20303 base tests produced 20402 (100%) non-filtered tests
>>> 20402 tests ran
Force exit code 0 after failures. Json test results file generated with failure information.
```

shorturl.at/uwQU4

Outline

- 为什么要学习V8的测试
- V8的测试如何运行
- V8的测试包含哪些项目
- 测试框架的运行流程
- 如何控制测试的测试集合和variant

v8的测试包含哪些项目： v8/test目录

语言标准测试	test262	ECMAScript的语言标准测试
	wasm-spec-tests	Wasm兼容性的测试
性能测试	benchmarks	性能测试，包括SunSpider/Kraken/Octane
	js-perf-test	用于性能测试的微benchmark，通过tools/run_perf.py来运行
语言特性测试	bigint	ECMAScript大整数特性测试
	intl	测试国际化特性
单元测试	mjsunit	用JS代码写的单元测试
	unittests	C++实现的单元测试
	cctest	C++实现的单元测试
	common	C++测试的公共代码
	memory	给cctest提供一个用于测试追踪内存信息工具的json文件
	torque	给cctest提供一个测试用的tq文件，用于测试内置语言的Torque编译器
debugger测试	debugger	验证内置debugger的命令
	debugging	Wasm的gdb调试测试用例
模糊测试	fuzzer	fuzzer非法的JS输入，验证V8是否输出正确的错误提示
	fuzzilli	V8内置的一款JS语言的开源fuzzer code产生代码
观察器测试	inspector	验证Inspector Protocol，该协议用于从外部的调试器来观察和控制V8引擎
输出信息的测试	message	验证非法JS代码产生的错误信息是否能正常输出
minidump调试工具测试	mkgrokdump	用于测试grokdump工具（grokdump用于对Windows系统下的minidump文件进行解析
Wasm的API调用测试	wasm-api-tests	通过C++ API接口实现的Wasm的测试
	wasm-js	通过JS API接口实现的Wasm的测试
来自其他JS引擎的测试	webkit	从Webkit的JavaScriptCore引擎借鉴的测试用例
	mozilla	从Mozilla的SpiderMonkey引擎借鉴的测试用例

V8的测试项目和对应的实体程序

```
TESTSUITES_TARGETS = {"benchmarks": "d8",
                      "bigint": "bigint_shell",
                      "cctest": "cctest",
                      "debugger": "d8",
                      "fuzzer": "v8_fuzzers",
                      "inspector": "inspector-test",
                      "intl": "d8",
                      "message": "d8",
                      "mjsunit": "d8",
                      "mozilla": "d8",
                      "test262": "d8",
                      "unittests": "unittests",
                      "wasm-api-tests": "wasm_api_tests",
                      "wasm-js": "d8",
                      "wasm-spec-tests": "d8",
                      "webkit": "d8"}
"tools/dev/gm.py" line 51 of 485 --10%-- col 1
```

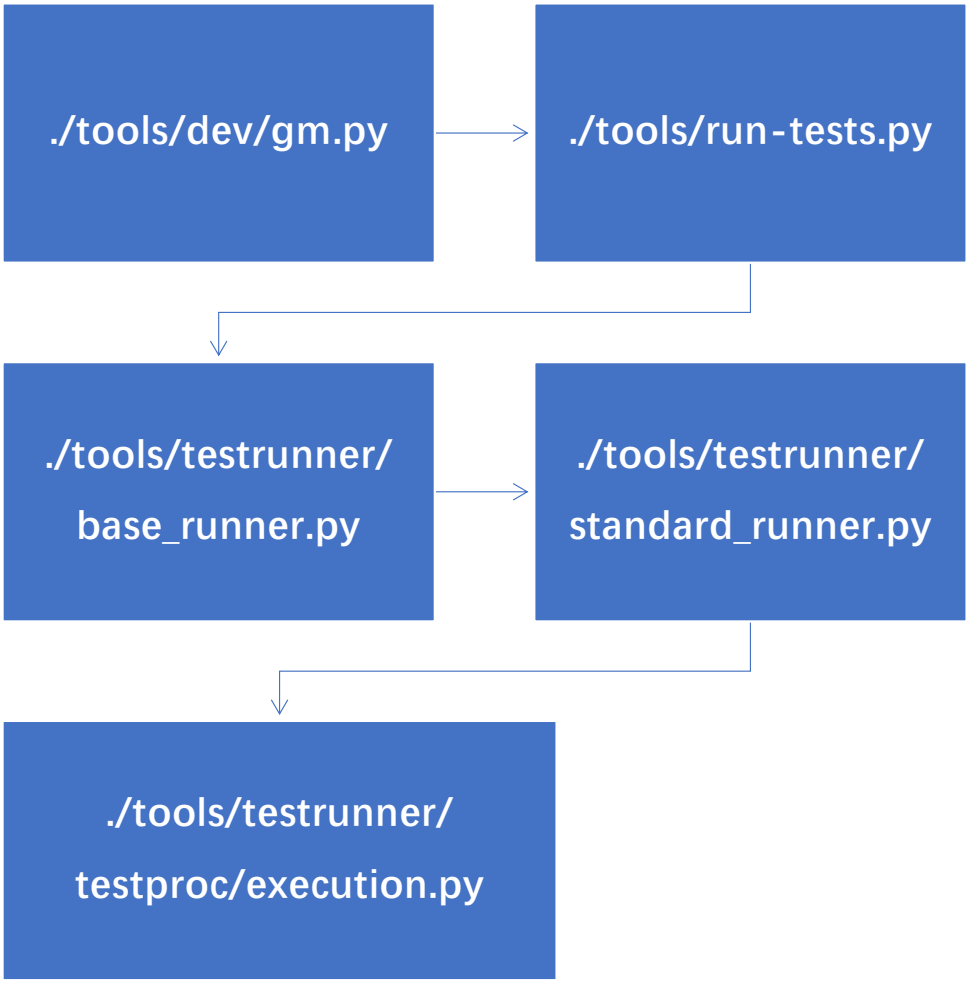
测试项目名称 : 测试实体程序名称

测试运行实体：
d8
bigint_shell
cctest
v8_fuzzers
inspector-test
unittests
wasm_api_tests

Outline

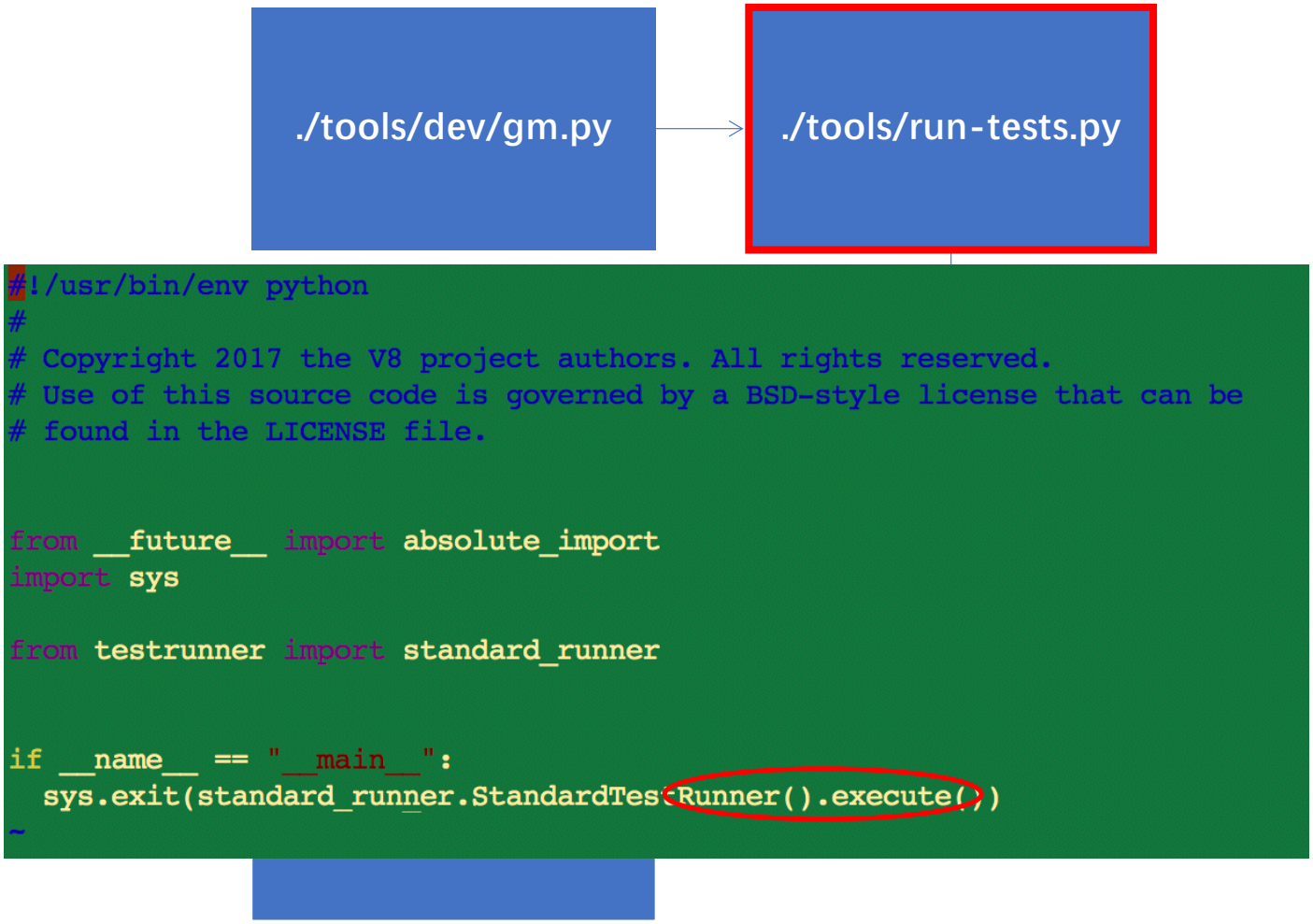
- 为什么要学习V8的测试
- V8的测试如何运行
- V8的测试包含哪些项目
- 测试框架的运行流程
- 如何控制测试的测试集合和variant

V8测试框架的运行流程



V8测试框架的运行流程

V8测试框架的运行流程



V8测试框架的运行流程

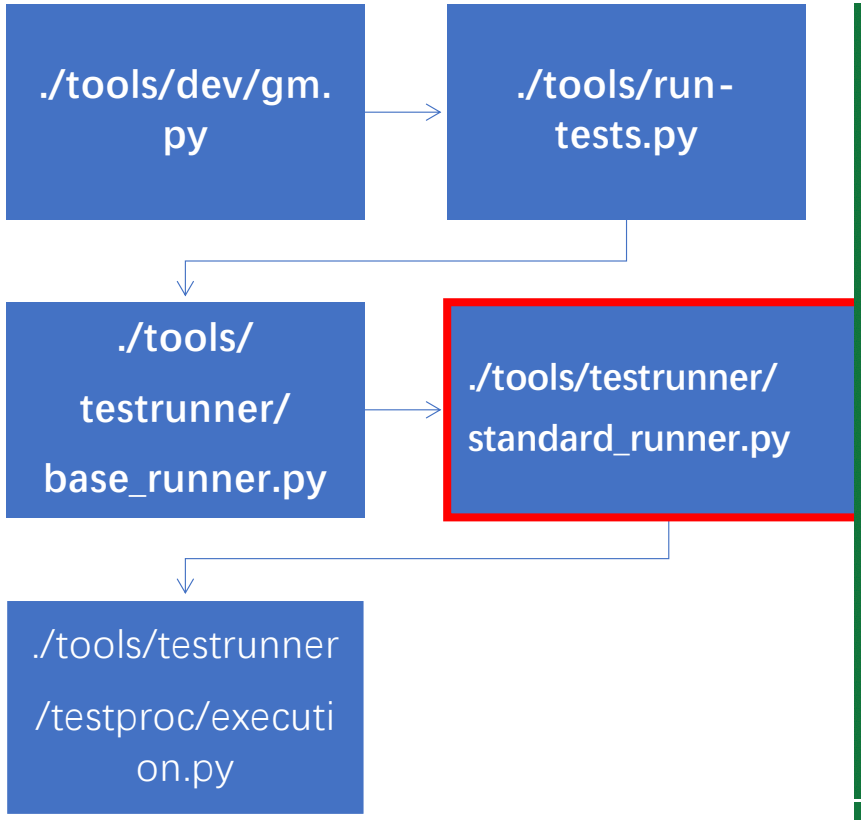
./tools/dev/gm.py

./tools/testrunner/base_runner.py

./tools/testrunner/testproc/execution.py

```
def execute(self, sys_args=None):
  if sys_args is None:  # pragma: no cover
    sys_args = sys.argv[1:]
  try:
    parser = self._create_parser()
    options, args = self._parse_args(parser, sys_args)
    if options.swarming:
      # Swarming doesn't print how isolated commands are called. Lets make
      # this less cryptic by printing it ourselves.
      print(' '.join(sys.argv))

      # Kill stray processes from previous tasks on swarming.
      util.kill_processes_linux()

    self._load_build_config(options)
    command.setup(self.target_os, options.device)

    try:
      self._process_default_options(options)
      self._process_options(options)
    except TestRunnerError:
      parser.print_help()
      raise

    args = self._parse_test_args(args)
    tests = self._load_testsuite_generators(args, options)
    self._setup_env()
    print(">>> Running tests for %s.%s %s %s" % (self.build_config.arch,
                                                 self.mode_options.label,args,tests))
    exit_code = self._do_execute(tests, args, options)
```

V8测试框架的运行流程




```
def _do_execute(self, tests, args, options):
  jobs = options.j

  print('>>> Running with test processors')
  loader = LoadProc(tests)
  results = self._create_result_tracker(options)
  indicators = self._create_progress_indicators(
      tests.test_count_estimate, options)

  outproc_factory = None
  if self.build_config.predictable:
    outproc_factory = predictable.get_outproc
  execproc = ExecutionProc(jobs, outproc_factory)
  sigproc = self._create_signal_proc()

  procs = [
    loader,
    NameFilterProc(args) if args else None,
    StatusFileFilterProc(options.slow_tests, options.pass_fail_tests),
    VariantProc(self._variants),
    StatusFileFilterProc(options.slow_tests, options.pass_fail_tests),
    self._create_predictable_filter(),
    self._create_shard_proc(options),
    self._create_seed_proc(options),
    SequenceProc(options.max_heavy_tests),
    sigproc,
  ] + indicators + [
    results,
```

```
  # This starts up worker processes and blocks until all tests are
  # processed.
  execproc.run()
```

V8测试框架的运行流程

./tools/dev/gm.py

./tools/testrunner/base_runner.py

./tools/testrunner/testproc/execution.py

```
class ExecutionProc(base.TestProc):
  """Last processor in the chain. Instead of passing tests further it creates
  commands and output processors, executes them in multiple worker processes and
  sends results to the previous processor.
  """

  def __init__(self, jobs, outproc_factory=None):
    super(ExecutionProc, self).__init__()
    self._pool = pool.Pool(jobs, notify_fun=self.notify_previous)
    self._outproc_factory = outproc_factory or (lambda t: t.output_proc)
    self._tests = {}

  def connect_to(self, next_proc):
    assert False, 'ExecutionProc cannot be connected to anything'

  def run(self):
    it = self._pool.imap_unordered(
        fn=run_job,
        gen=[],
        process_context_fn=create_process_context,
        process_context_args=[self._prev_requirement],
    )
    for pool_result in it:
      self._unpack_result(pool_result)
```

V8测试框架的运行流程

./tools/dev/gm.py

./tools/testrunner/base_runner.py

./tools/testrunner/testproc/execution.py

```
class ExecutionProc(base.TestProc):
  """Last processor in the chain. Instead of passing tests further it creates
  commands and output processors, executes them in multiple worker processes and
  sends results to the previous processor.
  """

  def __init__(self, jobs, outproc_factory=None):
    super(ExecutionProc, self).__init__()
    self._pool = pool.Pool(jobs, notify_fun=self.notify_previous)
    self._outproc_factory = outproc_factory or (lambda t: t.output_proc)
    self._tests = {}

  def connect_to(self, next_proc):
    assert False, 'ExecutionProc cannot be connected to anything'

  def run(self):
    it = self._pool.imap_
        fn=run_job,
        gen=[],
        process_context_fn=create_process_context,
        process_context_args=[self._prev_requirement],
    )
    for pool_result in it:
      self._unpack_result(pool_result)
```

```
def run_job(job, process_context):
  return job.run(process_context)
```

V8测试框架的运行流程

./tools/dev/gm.py

./tools/testrunner/base_runner.py

./tools/testrunner/testproc/execution.py

```
class ExecutionProc(base.TestProc):
  """Last processor in the chain. Instead of passing tests further it creates
  commands and output processors, executes them in multiple worker processes and
  sends results to the previous processor.
  """

  def __init__(self, jobs, outproc_factory=None):
    super(ExecutionProc, self).__init__()
    self._pool = pool.Pool(jobs, notify_fun=self.notify_previous)
    ctory or (lambda t: t.output_proc)
```

```
def run_job(job, process_context):
  return job.run(process_context)
```

```
class Job(object):
  def __init__(self, test_id, cmd, outproc, keep_output):
    self.test_id = test_id
    self.cmd = cmd
    self.outproc = outproc
    self.keep_output = keep_output

  def run(self, process_ctx):
    output = self.cmd.execute()
    reduction = process_ctx.result_reduction if not self.keep_output else None
    result = self.outproc.process(output, reduction)
    return JobResult(self.test_id, result)
```

Outline

- 为什么要学习V8的测试
- V8的测试如何运行
- V8的测试包含哪些项目
- 测试框架的运行流程
- 如何控制测试的测试集合和variant

如何控制测试项目

- 通过gm.py invoke的方式：
  - ./tools/dev/gm.py riscv64.release[.option1] [option2]/*
  - option1: check/checkall
  - option2: 单个的测试集名字(某些项目需要加上/*)
    - e.g.: ./tools/dev/gm.py riscv64.release cctest/*
- 通过run-tests.py invoke 的方式：
  - ./tools/run-tests.py --outdir=out/riscv64.release [option]
  - option: 单个的测试集名称 or 约定的测试集合MAP

如何控制测试项目

- 在gm.py 后添加参数
  - ./tools/dev/gm.py riscv64.rele
-

约定的测试集合MAP：
参考 tools/testrunner/base_runner.py

bot_default
default
d8_default
optimize_for_size
unittests

```
# Map of test name synonyms to lists of test suites. Should be ordered by
# expected runtimes (suites with slow test cases first). These groups are
# invoked in separate steps on the bots.
# The mapping from names used here to GN targets (which must stay in sync)
# is defined in infra/mb/gn_isolate_map.pyl.
TEST_MAP = {
  # This needs to stay in sync with group("v8_bot_default") in test/BUILD.gn.
  "bot_default": [
    "debugger",
    "mjsunit",
    "cctest",
    "wasm-spec-tests",
    "inspector",
    "webkit",
    "mkgrokdump",
    "wasm-js",
    "fuzzer",
    "message",
    "intl",
    "unittests",
    "wasm-api-tests",
  ],
  # This needs to stay in sync with group("v8_default") in test/BUILD.gn.
  "default": [
    "debugger",
    "mjsunit",
    "cctest",
    "wasm-spec-tests",
    "inspector",
    "mkgrokdump",
    "wasm-js",
    "fuzzer",
    "message",
    "intl",
    "unittests",
    "wasm-api-tests",
"tools/testrunner/base_runner.py" [readonly] line 70 of 830 --8%-- col 3
```

gm.py check所包含的测试（6项）

语言标准测试	test262	ECMAScript的语言标准测试
	wasm-spec-tests	Wasm兼容性的测试
性能测试	benchmarks	性能测试，包括SunSpider/Kraken/Octane
	js-perf-test	用于性能测试的微benchmark，通过tools/run_perf.py来运行
语言特性测试	bigint	ECMAScript大整数特性测试
	intl	测试国际化特性
单元测试	mjsunit	用JS代码写的单元测试
	unittests	
	cctest	
	common	
	memory	
	torque	
debugger测试	debugger	验证内置debugger的命令
	debugging	Wasm的gdb调试测试用例
模糊测试	fuzzer	fuzzer非法的JS输入，验证V8是否输出正确的错误提示
	fuzzilli	V8内置的一款JS语言的开源fuzzer code产生代码
观察器调试	inspector	验证Inspector Protocol，该协议用于从外部的调试器来观察和控制V8引擎
输出信息的测试	message	验证非法JS代码产生的错误信息是否能正常输出
minidump调试工具测试	mkgrokdump	用于测试grokdump工具（grokdump用于对Windows系统下的minidump文件进行解析
Wasm的API调用测试	wasm-api-tests	通过C++ API接口实现的Wasm的测试
	wasm-js	通过JS API接口实现的Wasm的测试
来自其他JS引擎的测试	webkit	从Webkit的JavaScriptCore引擎借鉴的测试用例
	mozilla	从Mozilla的SpiderMonkey引擎借鉴的测试用例

```
# Tests that run-tests.py would run by default that can be run with
# BUILD_TARGETS_TESTS.
DEFAULT_TESTS = ["cctest", "debugger", "intl", "message", "mjsunit",
                 "unittests"]
"tools/dev/gm.py" line 17 of 485 --3%-- col 1
```

gm.py checkall所包含的测试（12项）

语言标准测试	test262
	wasm-spec-tests
性能测试	benchmarks
	js-perf-test
语言特性测试	bigint
	intl
单元测试	mjsunit
	unittests
	cctest
	common
	memory
	torque
debugger测试	debugger
	debugging
模糊测试	fuzzer
	fuzzilli
观察器调试	inspector
输出信息的测试	message
minidump调试工具测试	mkgrokdump
Wasm的API调用测试	wasm-api-tests
	wasm-js
来自其他JS引擎的测试	webkit
	mozilla

```
# Map of test name synonyms to lists of test suites. Should be ordered by
# expected runtimes (suites with slow test cases first). These groups are
# invoked in separate steps on the bots.
# The mapping from names used here to GN targets (which must stay in sync)
# is defined in infra/mb/gn_isolate_map.pyl.
TEST_MAP = {
  # This needs to stay in sync with group("v8_bot_default") in test/BUILD.gn.
  "bot_default": [
    "debugger",
    "mjsunit",
    "cctest",
    "wasm-spec-tests",
    "inspector",
    "webkit",
    "mkgrokdump",
    "wasm-js",
    "fuzzer",
    "message",
    "intl",
    "unittests",
    "wasm-api-tests",
  ],
  # This needs to stay in sync with group("v8_default") in test/BUILD.gn.
  "default": [
    "debugger",
    "mjsunit",
    "cctest",
    "wasm-spec-tests",
    "inspector",
    "mkgrokdump",
    "wasm-js",
    "fuzzer",
    "message",
    "intl",
    "unittests",
    "wasm-api-tests",
"tools/testrunner/base_runner.py" [readonly] line 70 of 830 --8%-- col 3
```

如何控制测试选项-了解variants参数

- variants参数用于把不同的选项传递给测试实体（如d8），从而开启v8引擎的不同功能或者模式（例如jitless模式）
- run-test.py脚本接受variants参数
  - 默认是default模式
  - Google的CI是：“more, dev”模式
- 在tools/testrunner/standard_runner.py中定义了VARIANT的别名

如何控制测试选项-了解variants参数

- variants参数用于把不同的选项传递给测试实体（如d8），从而开启v8引擎的不同功能
- run-test.py脚本指定CI是：“more, d
- 在tools/testrunn

```
VARIANTS = ['default']

MORE_VARIANTS = [
  'jitless',
  'stress',
  'stress_js_bg_compile_wasm_code_gc',
  'stress_incremental_marking',
]

VARIANT_ALIASES = {
  # The default for developer workstations.
  'dev': VARIANTS,
  # Additional variants, run on all bots.
  'more': MORE_VARIANTS,
  # Shortcut for the two above ('more' first - it has the longer running tests)
  'exhaustive': MORE_VARIANTS + VARIANTS,
  # Additional variants, run on a subset of bots.
  'extra': ['nooptimization', 'future', 'no_wasm_traps', 'turboprop',
            'instruction_scheduling', 'always_sparkplug'],
}
```

如何控制测试选项-了解variants参数

- variants参数用于把不同的选项传递给测试实体（如d8），从而开启v8引擎的不同功能或者模式（例如jitless模式，不同的GC选项）
- run-test.py脚本接受variants参数，默认是default模式，Google的CI是：“more, dev”模式
- 在tools/testrunner/standard_runner.py中定义了VARIANT的别名
- 在tools/testrunner/local/variant.py中定义了每一个VARIANT代表的运行参数

如何控制测试选项-了解variants参数

- variants参数用于把不同的选项
  v8引擎的不同功能或者模式（例
- run-test.py脚本接受variants参
  CI是：“more, dev”模式
- 在tools/testrunner/standard_
- 在tools/testrunner/local/varia
  的运行参数

```
# Use this to run several variants of the tests.
ALL_VARIANT_FLAGS = {
  "assert_types": [["--assert-types"]],
  "code_serializer": [["--cache=code"]],
  "default": [[]],
  "future": [["--future"]],
  "gc_stats": [["--gc-stats=1"]],
  # Alias of exhaustive variants, but triggering new test framework features.
  "infra_staging": [[]],
  "interpreted_regexp": [["--regexp-interpret-all"]],
  "experimental_regexp":  [["--default-to-experimental-regexp-engine"]],
  "concurrent_inlining":  [["--concurrent-inlining"]],
  "jitless": [["--jitless"]],
  "sparkplug": [["--sparkplug"]],
  "always_sparkplug": [[ "--always-sparkplug", "--sparkplug"]],
  "minor_mc": [["--minor-mc"]],
  "no_lfa": [["--no-lazy-feedback-allocation"]],
  # No optimization means disable all optimizations. OptimizeFunctionOnNextCall
  # would not force optimization too. It turns into a Nop. Please see
  # https://chromium-review.googlesource.com/c/452620/ for more discussion.
  # For WebAssembly, we test "Liftoff-only" in the nooptimization variant and
  # "TurboFan-only" in the stress variant. The WebAssembly configuration is
  # independent of JS optimizations, so we can combine those configs.
  "nooptimization": [["--no-opt", "--liftoff", "--no-wasm-tier-up"]],
  "slow_path": [["--force-slow-path"]],
  "stress": [["--stress-opt", "--no-liftoff", "--stress-lazy-source-positions",
              "--no-wasm-generic-wrapper"]],
  "stress_concurrent_allocation": [["--stress-concurrent-allocation"]],
  "stress_concurrent_inlining": [["--stress-concurrent-inlining",
                                  "--concurrent-inlining"]],
  "stress_js_bg_compile_wasm_code_gc": [["--stress-background-compile",
                                         "--stress-wasm-code-gc"]],
  "stress_incremental_marking": [["--stress-incremental-marking"]],
  "stress_snapshot": [["--stress-snapshot"]],
  # Trigger stress sampling allocation profiler with sample interval = 2^14
  "stress_sampling": [["--stress-sampling-allocation-profiler=16384"]],
  "trusted": [["--no-untrusted-code-mitigations"]],
  "no_wasm_traps": [["--no-wasm-trap-handler"]],
  "turboprop": [["--turboprop"]],
  "turboprop_as_toptier": [["--turboprop-as-toptier", "--turboprop"]],
  "instruction_scheduling": [["--turbo-instruction-scheduling"]],
  "stress_instruction_scheduling": [["--turbo-stress-instruction-scheduling"]],
  "top_level_await": [["--harmony-top-level-await"]],
  "wasm_write_protect_code": [["--wasm-write-protect-code-memory"]],
}
```

Thanks

2020/06/13