Tizen/Artik IoT Lecture Chapter 1. IoT.js & JerryScript Overview

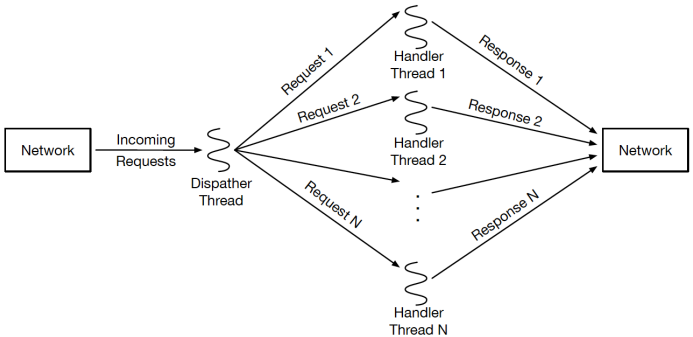
Sungkyunkwan University

Contents

- **IoT.js Overview**
  - Background
  - Design
  - Source Tree
  - How to Build IoT.js
  - How to Test IoT.js
  - How to Debug IoT.js
  - How to Contribute to IoT.js
    - Coding Style
- **JerryScript Overview**
  - Design
  - Source Tree
  - How to Build JerryScript
  - Arguments
  - How to Test JerryScript

Background (Thread-based Vs Event-based Model)

- **Thread-based Versus Event-based Model**
  - Heavy multi-threading limits system scalability
  - Thread-based model suffer from context switching overhead and memory overhead
  - Event-based model's scalability depends on the performance of single-threaded event loop



* **Figure 1: In the thread-based execution model, each incoming client request is assigned to a unique thread, which is responsible for returning a request to the client.**

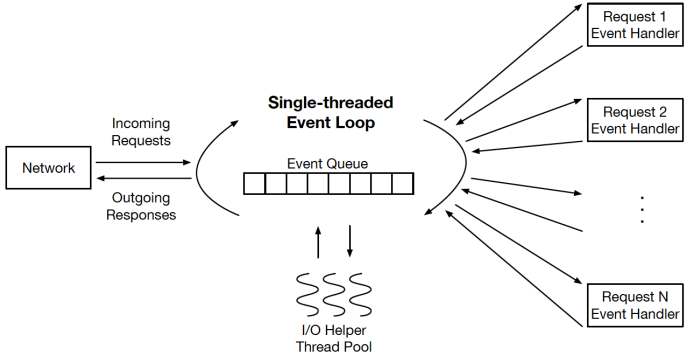


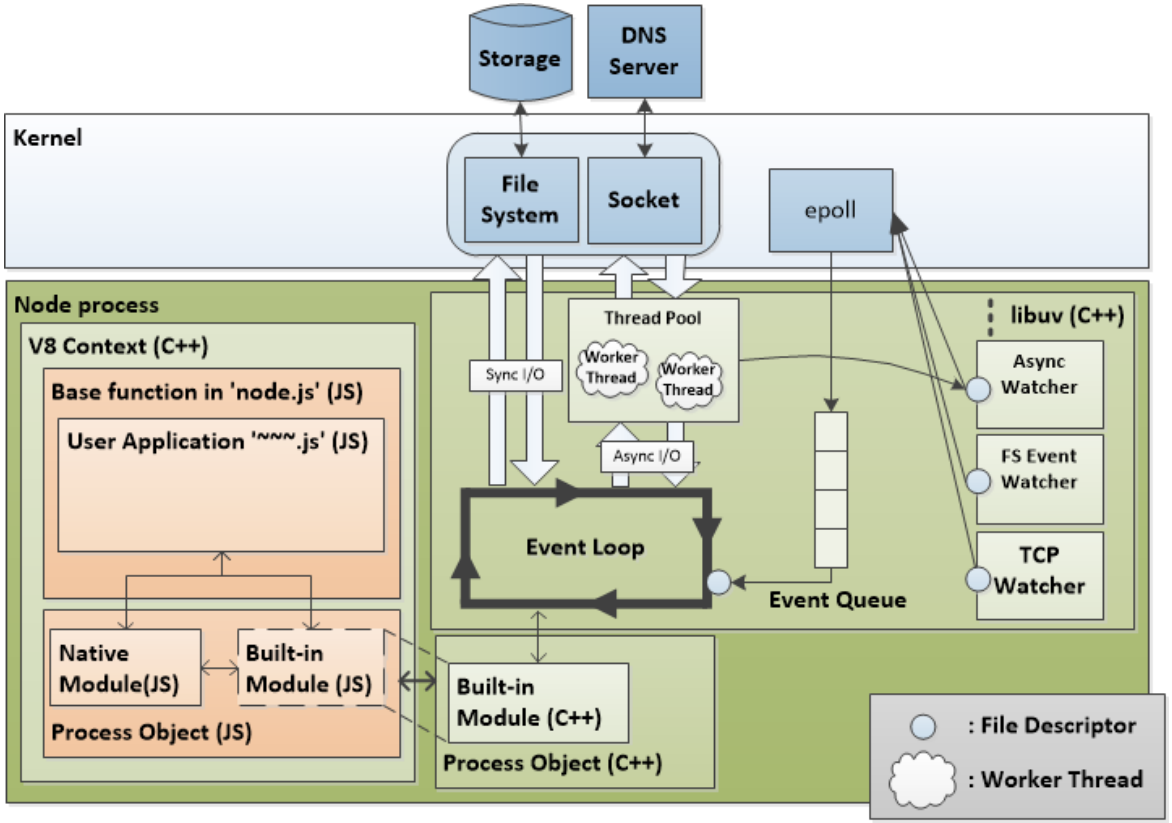
Figure 2: In the event-based execution model, each incoming client request is handled by the single-threaded event loop. I/O operations are handled asynchronously.

*Microarchitecture Implication of Event-driven Server-side Web Application (Micro'15)

Background (Event-based Model) Node.js

- **Well-known Event-based Model Platform is Node.js**
- **Node.js includes following features**
  - **V8 JavaScript engine** as an applications' runtime
  - Asynchronous I/O Framework called **libuv** (Event Loop, Helper Thread Pool, etc..)
  - Many built-in modules such as (e.g. Network, File I/O, Http, Cluster, etc..)
  - **npm** (node package module) which is dynamically attached to Node.js and supports various API set (e.g. Bluetooth API, DataBase API, etc..)

Node.js Architecture



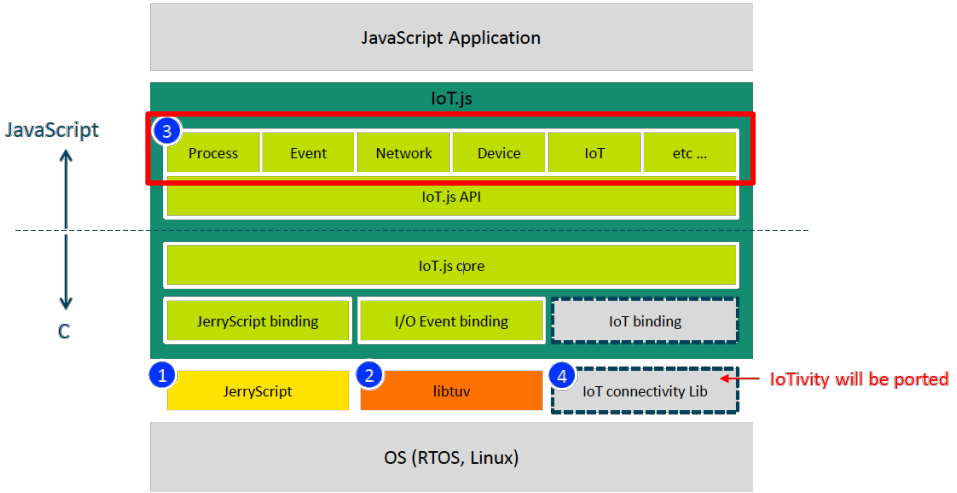
Limitation of Node.js & IoT.js

- **Memory footprint on Node.js is large**
  - V8 JavaScript Engine consumes a lot of memory at the cost of performance
  - Thus, Node.js is suitable for desktop and high-end Embedded devices
- **In order to support IoT workloads, It should be light-weight**
  - Open-source project called IoT.js suitable for Low-end IoT devices has been unveiled

IoT.js Design

- **JavaScript runtime for IoT devices based on JerryScript and libuv**
  - **JerryScript**: Lightweight JavaScript engine
  - **libuv**: asynchronous event handling
- **Supported Platforms**
  - Linux, NuttX
  - x86_64, ARM(RPi2, STM32F4)
- **Modules**
  - JavaScript functions to extend functionality

▪ ① JerryScript engine + ② Async. I/O event library + ③ App Framework + ④ IoT Connectivity

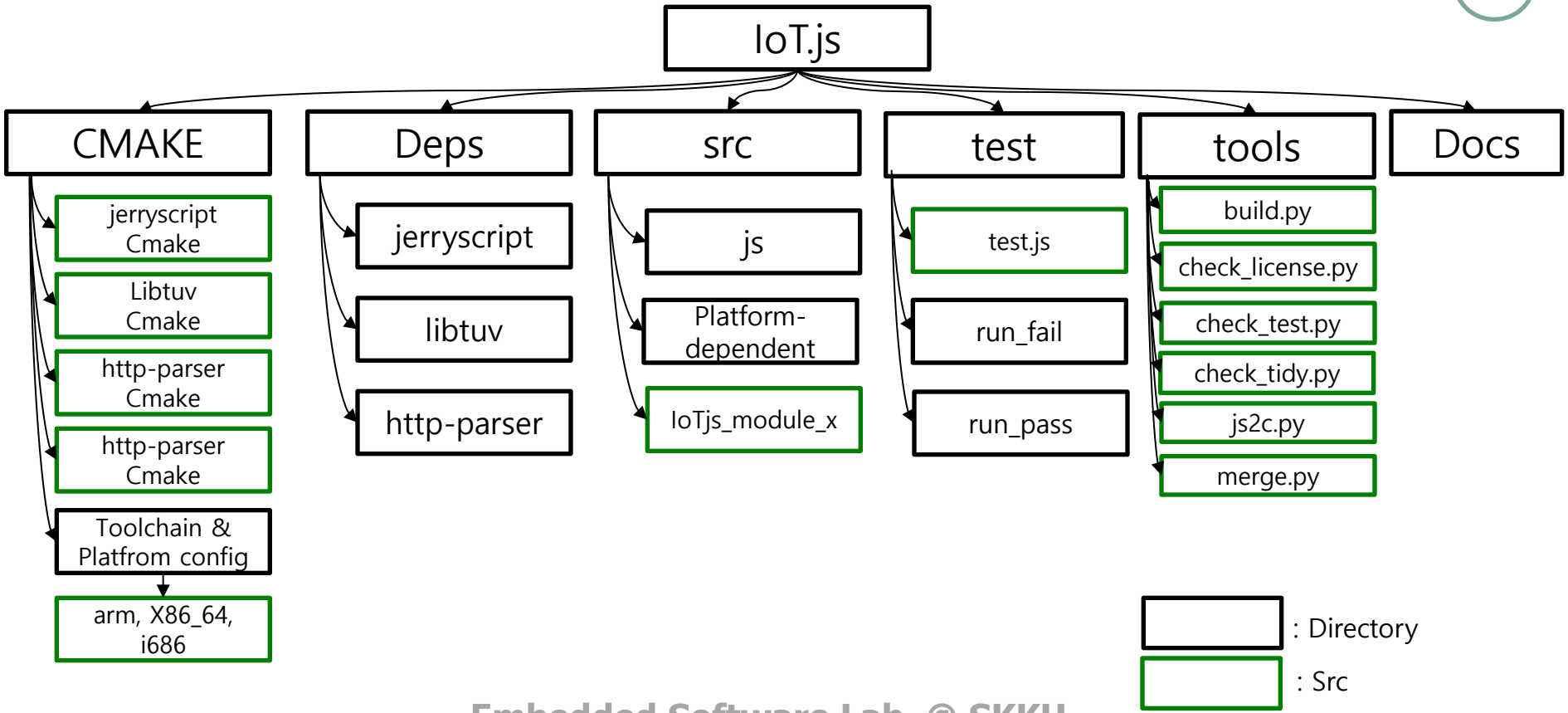


Life cycle of IoT.js

- Initialization Phase
- Running Phase
- Termination Phase

1. **Initialize JerryScript Engine**
2. **Execute empty script**
   - Create initial JavaScript Context
3. **Initialize Builtin modules**
4. **Evaluate Iotjs.js**
   - Generate Entry Function
5. **Run the entry function passing 'e.g. process'**
   1. Initialize 'process' module
   2. Load user application script
   3. Run user application
6. **Run Event loop until there are no more events to be handled**
7. **Clean up**

IoT.js Source Tree



How to Build IoT.js

1. Cross-compilation (Host : X86_64-linux, Target : Raspberry-pi2)

- $ sudo apt-get install gcc-arm-linux-gnueabihf g++-arm-linux-gnueabihf
  - Toolchain Installation
- $ ./tools/build.py --buildtype=release --no-init-submodule --target-arch=arm --target-os=linux --target-board=rpi2

2. Native-Compilation (Raspberry-pi2)

- $ git clone https://github.com/Samsung/iotjs.git
- $ ./tools/build.py --target-board=rpi2

Debugging IoT.js

- **Debugging mode Compilation**
  - $ ./tools/build.py **--buildtype=debug** --no-init-submodule --target-arch=arm --target-board=rpi2
- **Using GDB**
  - $ gdb {pathToIoTJS}/build/arm-linux/debug/iotjs
  - (gdb) run {param1}, {param2}, …
  - (gdb) bt # backtrace Print

Segment-fault Error Tracking

Error Occurrence Point(Libtuv)

```
(gdb) bt
#0  0x75fd031c in internal_getent (result=0x76ffbaa0,
    result@entry=0x76ffb634, buffer=0x75fd69ac "/etc/hosts",
    buffer@entry=0x76ffb680 "", buflen=buflen@entry=1056,
    errnop=0x76ffb680, errnop@entry=0x76ffbb44, herrnop=0x76ffbb54,
    herrnop@entry=0xd5b21 <gaih_inet+2040>, af=af@entry=0,
    flags=flags@entry=0) at nss_files/files-XXX.c:268
#1  0x75fd1490 in _nss_files_gethostbyname4_r (
    name=0x21d510 "localhost", pat=0x76ffbb44, buffer=0x76ffb680 "",
    buflen=1056, errnop=0x76ffbb48, herrnop=0x76ffbb54, ttlp=0x0)
    at nss_files/files-hosts.c:402
#2  0x000d5b20 in gaih_inet ()
#3  0x000d6f54 in getaddrinfo ()
#4  0x0009046a in uv__getaddrinfo_work (w=0x21d498)
    at /home/mini/iotjs/deps/libtuv/source/unix/uv_unix_getaddrinfo.c:121
#5  0x0008bb16 in worker (arg=0x0)
    at /home/mini/iotjs/deps/libtuv/source/unix/uv_unix_threadpool.c:97
#6  0x0008c648 in uv__thread_start (arg=0x21cc78)
    at /home/mini/iotjs/deps/libtuv/source/unix/uv_unix_thread.c:61
#7  0x000b6512 in start_thread (arg=0x0) at pthread_create.c:335
#8  0x000d8bec in ?? ()
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
```

<gdb backtrace>

Coding Style

- **Formatting**
  - Maximum 80 characters in a line
  - 2 space indent at a time. Do not use a tab for indentation
- **Naming**
  - Type names
    - Use UpperCamelCase for class, structs, typedef, enums
    - Use lower cases and underscore for variable names
    - Constant names: use a 'K' followed by UpperCamel case
    - Function names : Use UpperCamelCase for regular function names

```
Class MyClassName {
    ...
}

typedef MyTypeName {
    ...
}
```

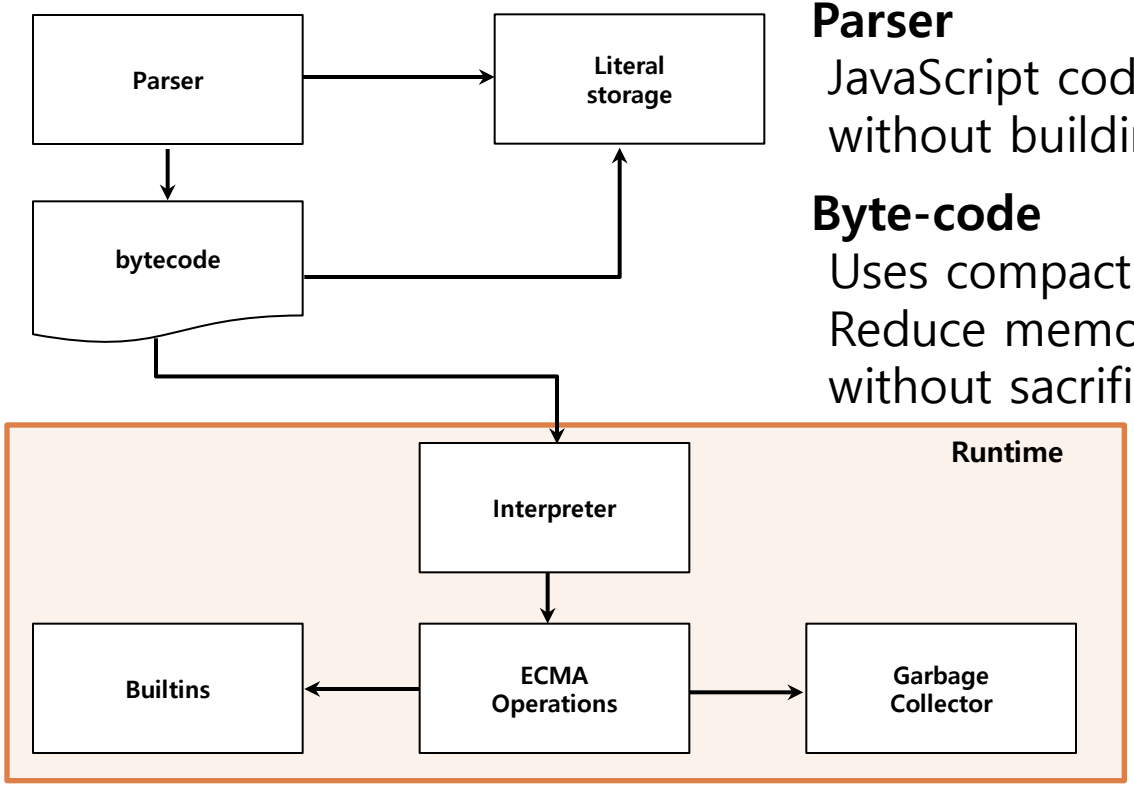
```
int lower_case_variable;
```

```
#define kConstantUse
```

JerryScript

- **JerryScript is a lightweight ECMAScript 5.1 engine, which is optimized for low-end systems (Open Source)**
- **ECMAScript**
  - ECMAScript: **Script-language specification** standardized by ECMA International in ECMA-262
  - ECMAScript Naming History: Mocha, LiveScript, and **JavaScript** now
  - File extensions: .es for ECMAScript, .js for JavaScript
    - Generally **JavaScript** aims to **be compatible** with ECMAScript, and also provides **additional features** not described in the ECMA specifications
- **Low-end**
  - Embedded systems with 32 bit CPU and 64K or less RAM
- **Small binary size**
  - Only 173Kbyte on ARM

JerryScript Design



Parser

JavaScript code → bytecode
without building an AST

Byte-code

Uses compact byte-code
Reduce memory consumption
without sacrificing considerable perf

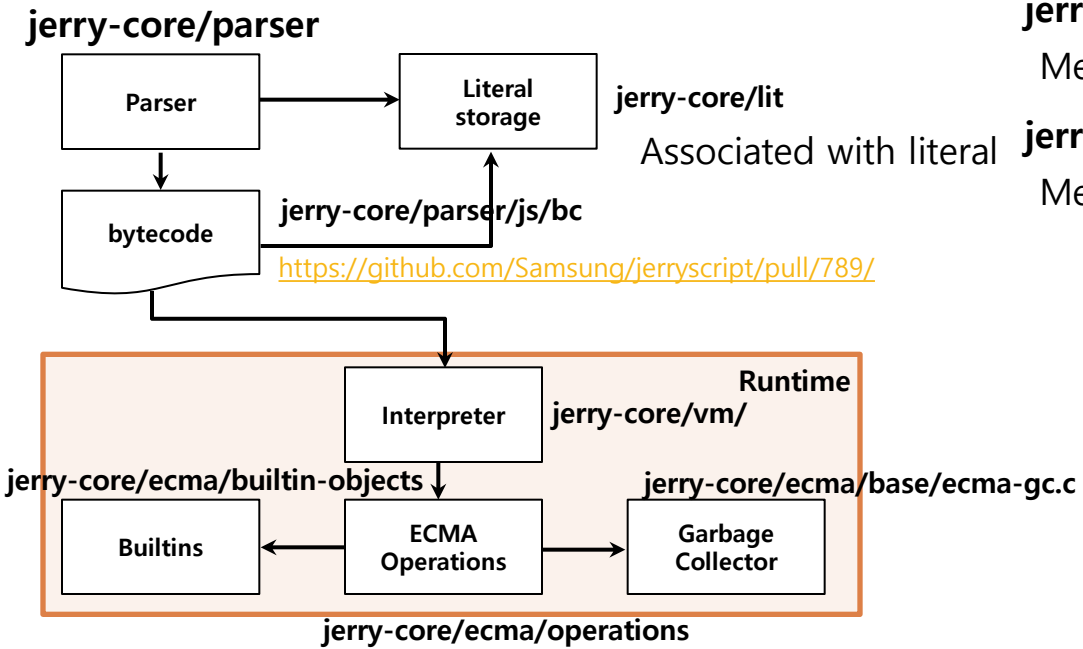
Literal Storage

JerryScript stores literals into
Literal store during parsing

JerryScript Source Tree

- **Build files**
  - build/configs/[toochain_ARCH].cmake
  - Makefile
- **Dependent Libraries**
  - jerry-libc (memcpy, srand, printf ...)
  - jerry-libm (math libraries)
- **tests**
  - *.js (Exclude SunSpider → Use it from webkit)
- **tools**
  - Script files for build, test, code check, and git
- **Jerry-core**
  - jerry-api.h
  - jerry.c

JerryScript Source Tree: jerry-core



jerry-core/jcontext

Memory context

jerry-core/jmem

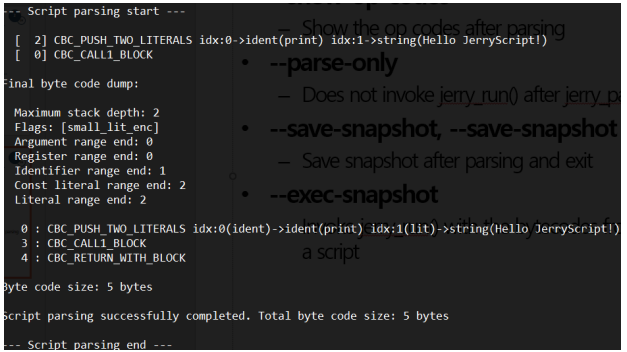
Memory allocation

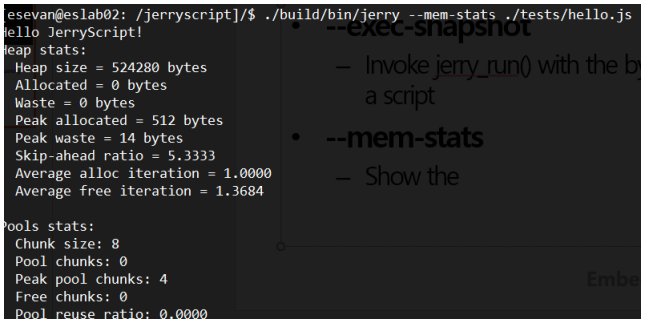
How to Build JerryScript

1. **Install Required Packages**
   - `$ sudo apt-get install gcc g++ gcc-arm-none-eabi cmake cppcheck vera++`
2. **Build**
   - `$ make debug.linux -j4`
   - `$ make release.linux -j4`

JerryScript Arguments

- **--show-op-codes**
  - Show the op codes after parsing and run
- **--parse-only**
  - Does not invoke jerry_run() after jerry_parse()
- **--save-snapshot, --save-snapshot**
  - Save snapshot after parsing and exit
- **--exec-snapshot**
  - Invoke jerry_run() with the bytecodes from the snapshot instead of parsing a script
- **--mem-stats**
  - Show the memory analysis in execution





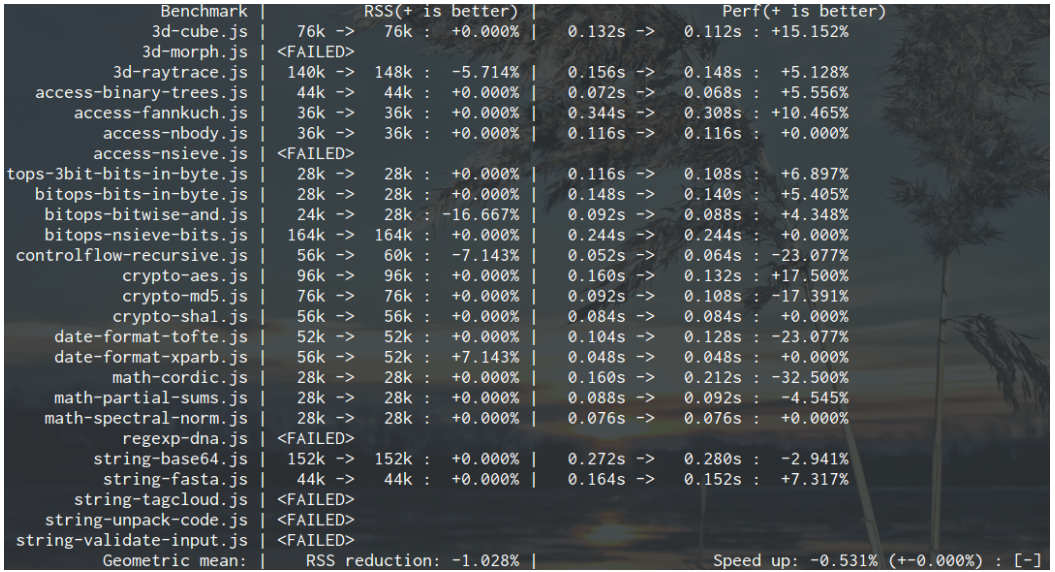
How to Test JerryScript

- **JerryScript provides the profiling shell scripts under the tools directory**
  - **`run-mem-stat-tests.sh`** `[Engine without mem-stat build] [Engine with mem-stat build] [Script file]`
    - Memory usage profiling script
  - **`run-perf-test.sh`** `[Engine 1] [Engine 2] [Repeats] [timeout] [Scripts directory] [-m output file name]`
    - Performance tools
    - Usually using ubench & sunspider

Benchmark	RSS (+ is better)	Perf (+ is better)
3d-cube.js	136 -> 136 (0)	3.728 -> 3.71 (0.4828)
3d-raytrace.js	304 -> 264 (13.1579)	5.918 -> 5.554 (6.1507)
access-binary-trees.js	88 -> 92 (-4.545)	2.72 -> 2.72 (0)
access-fannkuch.js	52 -> 52 (0)	10.03 -> 9.94 (0.8973)
access-nbody.js	68 -> 68 (0)	4.65 -> 4.63 (0.4301)
bitops-3bit-bits-in-byte.js	40 -> 40 (0)	3.18 -> 3.13 (1.5723)
bitops-bits-in-byte.js	40 -> 40 (0)	4.35 -> 4.36 (-0.23)
bitops-bitwise-and.js	36 -> 36 (0)	4.26 -> 4.26 (0)
controlflow-recursive.js	224 -> 224 (0)	3.23 -> 3.17 (1.8576)
crypto-aes.js	156 -> 152 (2.5641)	5.96 -> 5.71 (4.1946)
crypto-md5.js	216 -> 212 (1.8519)	33.57 -> 10.53 (68.6327)
crypto-sha1.js	156 -> 152 (2.5641)	15.1 -> 5.71 (62.1854)
date-format-xparb.js	104 -> 96 (7.6923)	2.03 -> 1.82 (10.3448)
math-cordic.js	48 -> 48 (0)	4.25 -> 4.23 (0.4706)
math-partial-sums.js	40 -> 40 (0)	2.55 -> 2.5 (1.9608)
math-spectral-norm.js	52 -> 52 (0)	3.22 -> 3.18 (1.2422)
string-base64.js	192 -> 164 (14.5833)	235.22 -> 38.6 (83.5898)
string-fasta.js	60 -> 60 (0)	5.54 -> 4.72 (14.8014)
Geometric mean:	RSS reduction: 2.2246%	Speed up: 21.6495%

How to Test JerryScript: SunSpider


```
$ ./tools/run-perf-test.sh \
    ./build/bin/release.linux/jerry ./build/bin/release.linux/
jerry 1 30 ./sunspider1.0.2/sunspider-1.0.2/ -m result.m
```



Benchmark	RSS(+ is better)	Perf(+ is better)
3d-cube.js	76k -> 76k : +0.000%	0.132s -> 0.112s : +15.152%
3d-morph.js	<FAILED>	
3d-raytrace.js	140k -> 148k : -5.714%	0.156s -> 0.148s : +5.128%
access-binary-trees.js	44k -> 44k : +0.000%	0.072s -> 0.068s : +5.556%
access-fannkuch.js	36k -> 36k : +0.000%	0.344s -> 0.308s : +10.465%
access-nbody.js	36k -> 36k : +0.000%	0.116s -> 0.116s : +0.000%
access-nsieve.js	<FAILED>	
tops-3bit-bits-in-byte.js	28k -> 28k : +0.000%	0.116s -> 0.108s : +6.897%
bitops-bits-in-byte.js	28k -> 28k : +0.000%	0.148s -> 0.140s : +5.405%
bitops-bitwise-and.js	24k -> 28k : -16.667%	0.092s -> 0.088s : +4.348%
bitops-nsieve-bits.js	164k -> 164k : +0.000%	0.244s -> 0.244s : +0.000%
controlflow-recursive.js	56k -> 60k : -7.143%	0.052s -> 0.064s : -23.077%
crypto-aes.js	96k -> 96k : +0.000%	0.160s -> 0.132s : +17.500%
crypto-md5.js	76k -> 76k : +0.000%	0.092s -> 0.108s : -17.391%
crypto-sha1.js	56k -> 56k : +0.000%	0.084s -> 0.084s : +0.000%
date-format-tofte.js	52k -> 52k : +0.000%	0.104s -> 0.128s : -23.077%
date-format-xparb.js	56k -> 52k : +7.143%	0.048s -> 0.048s : +0.000%
math-cordic.js	28k -> 28k : +0.000%	0.160s -> 0.212s : -32.500%
math-partial-sums.js	28k -> 28k : +0.000%	0.088s -> 0.092s : -4.545%
math-spectral-norm.js	28k -> 28k : +0.000%	0.076s -> 0.076s : +0.000%
regexp-dna.js	<FAILED>	
string-base64.js	152k -> 152k : +0.000%	0.272s -> 0.280s : -2.941%
string-fasta.js	44k -> 44k : +0.000%	0.164s -> 0.152s : +7.317%
string-tagcloud.js	<FAILED>	
string-unpack-code.js	<FAILED>	
string-validate-input.js	<FAILED>	
Geometric mean:	RSS reduction: -1.028%	Speed up: -0.531% (+-0.000%) : [-]

How to Test JerryScript: Memory Status

- **--mem-stat option**

1. `$ make release.linux-mem_stats`
2. `$ ./build/bin/release.linux-mem_stats/jerry \`
   `./sunspider1.0.2/sunspider-1.0.2/3d-cube.js --mem-stats`

```
Heap stats:
  Heap size = 524280 bytes
  Allocated = 0 bytes
  Waste = 0 bytes
  Peak allocated = 16376 bytes
  Peak waste = 642 bytes
  Skip-ahead ratio = 8.4250
  Average alloc iteration = 1.0565
  Average free iteration = 6.6216
```

```
Pools stats:
  Chunk size: 8
  Pool chunks: 0
  Peak pool chunks: 746
  Free chunks: 0
  Pool reuse ratio: 11.2659
```