FREE eBook

LEARNING v8

Free unaffiliated eBook created from
Stack Overflow contributors.

#v8

Table of Contents

About 1

Chapter 1: Getting started with v8 2

Remarks 2

Versions 2

v8 Releases 2

Examples 2

Installation or Setup 2

Chapter 2: Getting started with v8 4

Examples 4

Running v8 on a file 4

Useful built-in functions and objects in d8 4

Chapter 3: Weak Callbacks 7

Remarks 7

Examples 7

Running user-specified code when an Object is garbage collected. 7

Credits 9

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: v8

It is an unofficial and free v8 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official v8.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with v8

Remarks

This section provides an overview of what v8 is, and why a developer might want to use it.

It should also mention any large subjects within v8, and link out to the related topics. Since the Documentation for v8 is new, you may need to create initial versions of those related topics.

Versions

v8 Releases

Version	Release Date
0.1.0	2008-07-03
1.0.0	2009-02-09
2.0.0	2009-11-18
3.0.0	2010-12-07
3.10.0	2012-10-30
3.20.0	2013-06-28
4.2.1	2015-01-14
4.10.1	2016-01-16

Examples

Installation or Setup

`v8` uses the Google `depot_tools` for getting the source and building the library. To install `depot_tools`, follow the instructions here:

https://www.chromium.org/developers/how-tos/install-depot-tools

Get the `v8` source by running

```
/path/to/depot_tools/fetch v8
```

After running this command, to pull new versions in the future, run

```
/path/to/depot_tools/gclient sync
```

Build instructions are provided here: https://github.com/v8/v8/wiki/Building%20with%20GN

Read Getting started with v8 online: https://riptutorial.com/v8/topic/7639/getting-started-with-v8

Chapter 2: Getting started with v8

Examples

Running v8 on a file

Use the d8 shell to run the v8 engine. Use the following pattern to run on a file:

```
/path/to/d8 [flags] [file].js
```

For example:

```
./d8 --log-gc script.js
```

will run d8 on script.js with garbage collection logging enabled

Useful built-in functions and objects in d8

In addition to libraries defined in EcmaScript language specification and EcmaScript internationalization API specification, d8 also implements the following functions and objects.

- `print(args...): function`. Print to `stdout`.
- `printErr(args...): function`. Print to `stderr`.
- `write(args...): function`. Same as `print` but no newline at the end.
- `read(filename): function`. Read text from file and returned as `String`.
- `readbuffer(filename): function`. Read binary from file and returned as `ArrayBuffer`.
- `readline(): function`. Read line from `stdin`. Use `'\'` for multi-lines input.
- `load(filename): function`. Load and execute JavaScript file.
- `quit([exitCode]): function`. Quit with optional exit code.
- `version(): function`. Return version code as `String`.
- `os: object`. OS-related utilities, only available for POSIX.
    - `os.system(command): function`: Execute system command.
    - `os.chdir(path): function`: Change current directory.
    - `os.setenv(name, value): function`: Set environment variable.
    - `os.unsetenv(name): function`: Unset environment variable.
    - `os.umask(alue) function`: Calls the umask system call and returns the old umask.
    - `os.mkdirp(path[, mask]): function`: Creates a directory. The mask (if present) is anded with the current umask. Intermediate directories are created if necessary.
    - `os.rmdir(path): function`: Remove directory.
- `performance: object`: Use for performance analysis.
    - `performance.now(): function`. Returns a time stamp as double, measured in milliseconds.
- `Worker: object`: Modified from HTML5 Web Worker (see example below for detail)
- `Realm: object`: Create and manage isolated environment (realm).
    - `Realm.create(): function`: Create a new realm with distinct security token and return its

index.
- `Realm.createAllowCrossRealmAccess(): function`: Create a new realm with same security token as the current realm and return its index.
- `Realm.current(): function`: Returns the index of the currently active realm. Index of global realm is 0.
- `Realm.global(i): function`: Returns the global object of realm i.
- `Realm.owner(globalObj): function`: Returns the index of the realm that created globalObj.
- `Realm.eval(i, s): function`: Evaluates s in realm i and returns the result.
- `Realm.switch(i): function`: Switches to the realm i for consecutive interactive inputs.
- `Realm.dispose(i): function`: Disposes the reference to the realm i.
- `Realm.shared: object`: An accessor for a single shared value across realms.

Detailed implementations and comments can be found in d8.h, d8.cc, d8-posix.cc and d8-windows.cc.

Example:

```
print("Hello World!");
write("Hello ");
write("again!\n");
printErr("Nothing went wrong.");

write("Your name: ");
var name = readline();
print("Hello, ", name, "!");

load("external.js");

var string = read("text.txt");
var buffer = readbuffer("binary.bin");

print("Version: ", version());

quit(0); // bye
```

Worker example:

main.js

```
var workerScript = read("worker.js");
var worker = new Worker(workerScript);
worker.postMessage(12);
```

worker.js

```
onmessage = ev => {
  print(ev); // 12
};
```

Realm example:

```
print(Realm.current()); // 0

var rIndex = Realm.create();
print(rIndex); // 1

Realm.eval(rIndex, "var x = 100");
Realm.eval(rIndex, "print(x)"); // 100
var result = Realm.eval(rIndex, "x * 2");
print(result); // 200

Realm.eval(rIndex, "var rIndex2 = Realm.create()");
Realm.eval(rIndex, "print(rIndex2)"); // 2
Realm.eval(rIndex, "print(Realm.owner(this))"); // 1

try {
  var childGlobal = Realm.global(rIndex);
  print(childGlobal); // error
} catch (e) {
  print("Global object cannot be read/written cross-realm.");
}

var rIndex3 = Realm.createAllowCrossRealmAccess();
Realm.eval(rIndex, "var x = 50");
var childGlobal = Realm.global(rIndex3);
childGlobal.x++;
Realm.eval(rIndex3, "print(x)"); // 51

try {
  Realm.dispose(rIndex3);
  Realm.eval(rIndex3, "print(x)"); // error
} catch (e) {
  print("The realm is dereferenced");
}

Realm.shared = "Hello from another world";
Realm.eval(rIndex, "print(Realm.shared)");
```

Read Getting started with v8 online: https://riptutorial.com/v8/topic/7745/getting-started-with-v8

Chapter 3: Weak Callbacks

Remarks

Weak callbacks are primarily used for cleaning up C++ objects embedded in the `InternalField` of a `v8::Object` created from a `v8::ObjectTemplate`. When the JavaScript object is garbage collected, often times the C++ object must be deleted as well. By setting a weak callback, you can get notification that a javascript object has been garbage collected and take appropriate action.

It is *VERY* important to remember that garbage collection is *NOT* deterministic. Your program may exit with objects with weak reference callbacks registered that are never called. These callbacks are important for a properly behaving long-running program, but should not be relied on for releasing critical-path resources in a consistent or prompt fashion.

In order for the garbage collector to know when it should run, you have to tell it about the amount of space your C++ objects are using via the `v8::Isolate::AdjustAmountOfExternalAllocatedMemory` call. The parameter to this call is the *change* in bytes, so when you allocate it, you'd often send in `sizeof(T)` and when you clean up in your weak reference callback, you'd send in `-sizeof(T)`.

Examples

Running user-specified code when an Object is garbage collected.

```
/**
 * Runs user-specified code when the given javascript object is garbage collected
 */
template<class CALLBACK_FUNCTION>
void global_set_weak(v8::Isolate * isolate, const v8::Local<v8::Object> & javascript_object,
CALLBACK_FUNCTION function)
{
    struct SetWeakCallbackData{
        SetWeakCallbackData(CALLBACK_FUNCTION function, v8::Isolate * isolate, const
v8::Local<v8::Object> & javascript_object) :
            function(function) {
                this->global.Reset(isolate, javascript_object);
        }
        // function to call for cleanup
        CALLBACK_FUNCTION function;

        // this is the weak reference
        v8::Global<v8::Object> global;
    };

    // This must be dynamically allocated so it sticks around until the object
    //   is garbage collected.   It cleans itself up in the callback.
    auto callback_data = new SetWeakCallbackData(function, isolate, javascript_object);

    // set the callback on the javascript_object to be called when it's garbage collected
    callback_data->global.template SetWeak<SetWeakCallbackData>(callback_data,
        [](const v8::WeakCallbackInfo<SetWeakCallbackData> & data) {
            SetWeakCallbackData * callback_data = data.GetParameter();
```

```
            callback_data->function(); // run user-specified code
            callback_data->global.Reset(); // free the V8 reference
            delete callback_data; // delete the heap variable so it isn't leaked
        }, v8::WeakCallbackType::kParameter);
}
```

Read Weak Callbacks online: https://riptutorial.com/v8/topic/7655/weak-callbacks

Credits

S. No	Chapters	Contributors
1	Getting started with v8	Community, Paul Sweatte, RamenChef, xaxxon
2	Weak Callbacks	RamenChef, tbodt, xaxxon