Tizen/Artik IoT Lecture Chapter 2. IoT.js Modules

Sungkyunkwan University

Contents

- **IoT.js Modules**
- **How to Use IoT.js Modules**
  - require() vs. Global object
- **How to Load IoT.js Modules**
  - Loading Module with *require()*
  - Loading as a *Global Module*
- **How to Make IoT.js Modules**
  - JS-only Modules
  - JS-Native Binding

IoT.js Modules

- **IoT.js Modules**
  - Provide additional functions that are not specified in ECMAScript
- **How to Use IoT.js Modules**
  - Basically, user can use modules **after loading with `require()`**.
    - ex. Assert, DNS, Events, **FS**, HTTP, Net, Stream, GPIO
  - Some modules can be used **without `require()`.**
    - ex. Buffer, **Console**, Module, Process, Timers
- **How to Make IoT.js Modules**
  - Basically, IoT.js modules are written in **JavaScript**.
  - **Built-in modules** can be added to use native functions.

How to Use IoT.js Modules

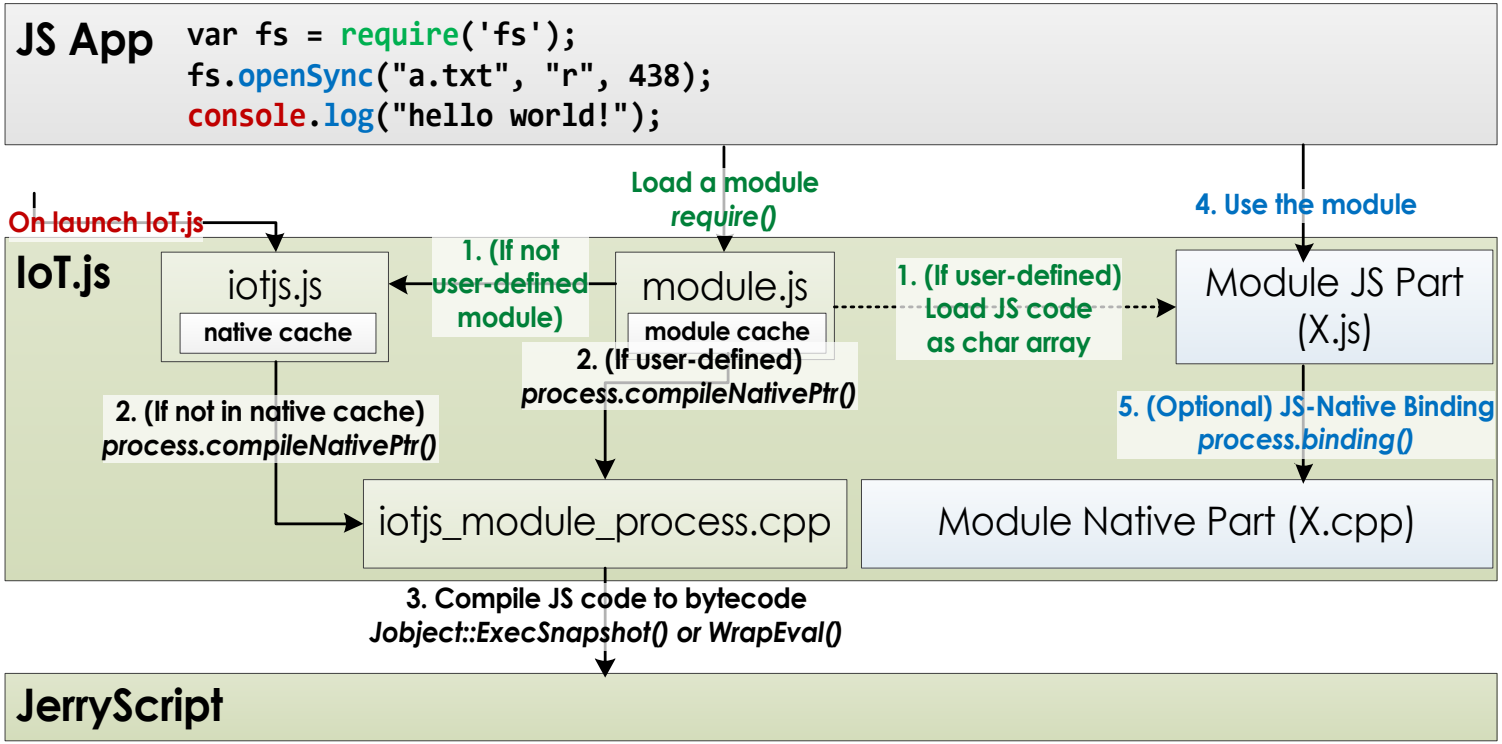

```
/* A Part of test code: test/run_pass/test_fs.js */
var fs = require('fs'); // Load 'fs' module
var assert = require('assert'); // Load 'assert' module

var filename = "../resources/greeting.txt";
var expectedContents = "Hello IoT.js!!";
var flags = "r";
var mode = 438;
try {
  var fd = fs.openSync(filename, flags, mode); // use loaded 'fs' module
  var buffer = new Buffer(64); // use global module 'buffer'
  fs.readSync(fd, buffer, 0, buffer.length, 0);
  assert.equal(buffer.toString(), expectedContents); // use global module 'assert'
  console.log(filename + " has same contents with " + expectedContents);
} catch (err) {
  throw err;
}
```

How to Load IoT.js Module

1. **Check native cache or module cache**
2. **Load module's JavaScript code to memory**
   - ***Global* Modules**: on launching IoT.js
   - ***Loadable* Modules**: on calling *require()*
3. **Compile the JavaScript code to bytecode**
   - **JavaScript object**: If snapshot=ON, it is pre-compiled on IoT.js compile time. If not, it its compiled on calling *require()*.
   - **Built-in object (native)**: Pre-compiled on IoT.js compile time.

How to Load IoT.js Module: Execution Flow



Loading Modules with *require()* (1/4)


```
/* Example: JS app to read "a.txt" file */
var fs = require('fs');
var fd = fs.openSync("a.txt", "r", 438);
```

src/js/module.js

```
215 Module.prototype.require = function(id) {
216   return Module.load(id, this);
217 };
```

src/js/module.js

```
148 Module.load = function(id, parent, isMain) {
149   if(process.native_sources[id]){
150     return Native.require(id);
151   }
152   var module = new Module(id, parent);
153
154   var modPath = Module.resolveModPath(module.id, module.parent);
155
156   var cachedModule = Module.cache[modPath];
157   if (cachedModule) {
158     return cachedModule.exports;
159   }
160
161   if (modPath) {
162     module.filename = modPath;
163     module.SetModuleDirs(modPath);
164     module.compile();
165   }
166   else {
167     throw new Error('No module found');
168   }
169
170   Module.cache[modPath] = module;
171
172   return module.exports;
173 };
```

Loading Modules with *require()* (2/4)

Step 1. Native Cache
- Check native cache and return cached built-in module if it exists.
- **Native cache**: caches of compiled built-in modules
- **Built-in modules**: compiled on IoT.js compile time

Step 2. Module Cache
- Check module cache and return cached module if it exists.
- **Module cache**: caches of compiled modules

Step 3. Compile the JavaScript code to bytecode
- Details are described in next slide

src/js/module.js

```
148 Module.load = function(id, parent, isMain) {
149   if(process.native_sources[id]){
150     return Native.require(id);
151   }
152   var module = new Module(id, parent);
153
154   var modPath = Module.resolveModPath(module.id, module.parent);
155
156   var cachedModule = Module.cache[modPath];
157   if (cachedModule) {
158     return cachedModule.exports;
159   }
160
161   if (modPath) {
162     module.filename = modPath;
163     module.SetModuleDirs(modPath);
164     module.compile();
165   }
166   else {
167     throw new Error('No module found');
168   }
169
170   Module.cache[modPath] = module;
171
172   return module.exports;
173 };
```

Loading Modules with *require()* (3/4)

- **Step 3. Compile the JavaScript code to bytecode**

src/js/module.js

```
161   if (modPath) {
162     module.filename = modPath;
163     module.SetModuleDirs(modPath);
164     module.compile();
165   }
166   else {
167     throw new Error('No module found');
168   }
169
170   Module.cache[modPath] = module;
171
172   return module.exports;
173 };
```

src/js/module.js

```
176 Module.prototype.compile = function() {
177   var self = this;
178   var requireForThis = function(path) {
179       return self.require(path);
180   };
181
182   var source = process.readSource(self.filename);
183   var fn = process.compile(source);
184   fn.call(self, self.exports, requireForThis, self);
185 };
```

1. Read module js file
   (cf. Default modules are pre-loaded)
2. Compile the module file
   (cf. If snapshot=ON, default modules are pre-compiled)
3. Run the module's initialization function

src/js/iotjs.js

```
212   Native.prototype.compile = function() {
213     // process.native_sources has a list of pointers to
214     // the source strings defined in 'iotjs_js.h', not
215     // source strings.
216
217     var fn = process.compileNativePtr(this.id);
218     fn(this.exports, Native.require, this);
219   };
```

JS-Native Binding

src/iotjs_module_process.cpp

```
159 JHANDLER_FUNCTION(CompileNativePtr){
174   if (natives[i].name != NULL) {
175 #ifdef ENABLE_SNAPSHOT
176     JResult jres = JObject::ExecSnapshot(natives[i].code,
177                                          natives[i].length);
178 #else
179     JResult jres = WrapEval((const char*)natives[i].code, natives[i].length);
180 #endif
181
```

Loading Modules with *require()* (4/4)

- **Default modules are pre-loaded in code segment.**
  - Pre-loaded as a char[] array in iotjs_js.cpp
  - On compiling IoT.js, `*.js` files of default modules are transformed to `src/iotjs_js.cpp`. (by using `tools/js2c.py`)

tools/build.py

```
614 def build_iotjs(option):
615     # Run js2c
616     os.chdir(SCRIPT_PATH)
617     check_run_cmd('python', ['js2c.py', option.buildtype,
```

tools/js2c.py

```
 88 JS_PATH = SRC_PATH + 'js/'
 99 fout_h = open(SRC_PATH + 'iotjs_js.h', 'w')
100 fout_cpp = open(SRC_PATH + 'iotjs_js.cpp', 'w')
106 files = glob.glob(JS_PATH + '*.js')
107 for path in files:
108     name = extractName(path)
109     fout_cpp.write('const char ' + name + '_n [] = "' + name + '";\n')
110     fout_h.write('extern const char ' + name + '_n [];\n')
111     fout_h.write('extern const int ' + name + '_l;\n')
```

src/iotjs_js.cpp

```
703 const char console_n [] = "console";
704 const unsigned char console_s [] = {
705   0x04, 0x00, 0x00, 0x00, 0x18, 0x01, 0x00, 0x00,
706   0x08, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
707   0x03, 0x00, 0x01, 0x00, 0x01, 0x00, 0x01, 0x00,
708   0x00, 0x00, 0x00, 0x01, 0x05, 0x00, 0x28, 0x00,
709   0x08, 0x45, 0x0a, 0x00, 0x01, 0x20, 0x00, 0x00,
710   0x10, 0x00, 0x01, 0x00, 0x01, 0x00, 0x07, 0x03,
711   0x03, 0x0a, 0x14, 0x17, 0x00, 0x00, 0x00, 0x00,
712   0x00, 0x00, 0xd5, 0x00, 0xef, 0x00, 0x92, 0x00,
713   0x83, 0x00, 0x75, 0x00, 0x39, 0x00, 0x66, 0x00,
714   0xd5, 0x00, 0x57, 0x00, 0x48, 0x00, 0x31, 0x00,
715   0x29, 0x00, 0x1d, 0x00, 0x11, 0x00, 0x04, 0x00,
```

Loading Module as *a Global Object* (1/2)

- ***Global* objects are loaded on running `iotjs.js`**
  - `src/js/iotjs.js`: First JavaScript file running on IoT.js

src/js/iotjs.js

```
23  function startIoTjs() {
24    initGlobal();
25    initTimers();
26
27    initProcess();
28
29    var module = Native.require('module');
30
31    module.runMain();
32  };
```

```
35  function initGlobal() {
36    global.process = process;
37    global.global = global;
38    global.GLOBAL = global;
39    global.root = global;
40    global.console = Native.require('console');
41    global.Buffer = Native.require('buffer');
42  };
```

Declare console, buffer modules as global objects.

src/js/iotjs.js

```
194  Native.require = function(id) {
195    if (id == 'native') {
196      return Native;
197    }
198
199    if (Native.cache[id]) {
200      return Native.cache[id].exports;
201    }
202
203    var nativeMod = new Native(id);
204
205    Native.cache[id] = nativeMod;
206    nativeMod.compile();
207
208    return nativeMod.exports;
209  };
```

Check Native cache

src/js/iotjs.js

```
184  function Native(id) {
185    this.id = id;
186    this.filename = id + '.js';
187    this.exports = {};
188  };
```

Set Module's JS part file and JS object

```
212  Native.prototype.compile = function() {
213    // process.native_sources has a list of pointers to
214    // the source strings defined in 'iotjs_js.h', not
215    // source strings.
216
217    var fn = process.compileNativePtr(this.id);
218    fn(this.exports, Native.require, this);
219  };
```

Compile module JS part to bytecode

Loading Module as *a Global Object* (2/2)

src/js/iotjs.js

```
212  Native.prototype.compile = function() {
213    // process.native_sources has a list of pointers to
214    // the source strings defined in 'iotjs_js.h', not
215    // source strings.
216
217    var fn = process.compileNativePtr(this.id);
218    fn(this.exports, Native.require, this);
219  };
```

src/iotjs_module_process.cpp

```
159 JHANDLER_FUNCTION(CompileNativePtr){
160   JHANDLER_CHECK(handler.GetArgLength() == 1);
161   JHANDLER_CHECK(handler.GetArg(0)->IsString());
162
163   String id = handler.GetArg(0)->GetString();
164
165   int i=0;
166   while (natives[i].name != NULL) {
167     if (!strcmp(natives[i].name, id.data())) {
168       break;
169     }
170
171     i++;
172   }
173
174   if (natives[i].name != NULL) {
175 #ifdef ENABLE_SNAPSHOT
176     JResult jres = JObject::ExecSnapshot(natives[i].code,
177                                          natives[i].length);
178 #else
179     JResult jres = WrapEval((const char*)natives[i].code, natives[i].length);
180 #endif
181
182     if (jres.IsOk()) {
183       handler.Return(jres.value());
184     } else {
185       handler.Throw(jres.value());
186     }
187   } else {
188     JObject jerror = JObject::Error ("Unknown native module");
189     handler.Throw(jerror);
190   }
191
192   return !handler.HasThrown();
193 }
```

Functions to compile JS to Bytecode
JObject::Eval()
JObject::ExecSnapshot()

How to Make IoT.js Modules: JS-only Modules

- **Basically, modules are written in JavaScript.**

```
/* hello.js */
var hello = require('./hello.js'); // Load 'hello.js' module
hello.log("Hello world");
```

```
/* hello.js: A module written in only JavaScript */
var hello = exports;
hello.log = function(str) {
  console.log(str);
};
```

How to Make IoT.js Modules: JS-Native Binding

14

```
/* Example: JS app */
console.log("Hello World!");
```

src/js/console.js

```
17 var util = require('util');
18 var consoleBuiltin = process.binding(process.binding.console);
19
20
21 function Console() {
22 }
23
24
25 Console.prototype.log =
26 Console.prototype.info = function() {
27   consoleBuiltin.stdout(util.format.apply(this, arguments) + '\n');
28 };
29
30
31 Console.prototype.warn =
32 Console.prototype.error = function() {
33   consoleBuiltin.stderr(util.format.apply(this, arguments) + '\n');
34 };
35
36
37 module.exports = new Console();
38 module.exports.Console = Console;
```

JS-Native Binding

src/iotjs_module_process.cpp

```
103 JHANDLER_FUNCTION(Binding) {
104   JHANDLER_CHECK(handler.GetArgLength() == 1);
105   JHANDLER_CHECK(handler.GetArg(0)->IsNumber());
106
107   int module_kind = handler.GetArg(0)->GetInt32();
108
109   Module* module = GetBuiltinModule(static_cast<ModuleKind>(module_kind));
110   IOTJS_ASSERT(module != NULL);
111
112   if (module->module == NULL) {
113     IOTJS_ASSERT(module->fn_register != NULL);
114     module->module = module->fn_register();
115     IOTJS_ASSERT(module->module);
116   }
117
118   handler.Return(*module->module);
119
120   return true;
121 }
```

src/iotjs_module_console.cpp

```
36 JHANDLER_FUNCTION(Stdout) {
37   return Print(handler, stdout);
38 }
```

```
46 JObject* InitConsole() {
47   Module* module = GetBuiltinModule(MODULE_CONSOLE);
48   JObject* console = module->module;
49
50   if (console == NULL) {
51     console = new JObject();
52     console->SetMethod("stdout", Stdout);
53     console->SetMethod("stderr", Stderr);
54
55     module->module = console;
56   }
57
58   return console;
59 }
```