N-API

Next Generation

Node.js Native Module API

@legendecas

Why we want them

- Vast module ecosystem makes Node.js Powerful
- Node.js growth depends on the stability of this ecosystem
- All possible future uses!

Error: the module '/Users/user1/demo-without-napi/node_modules/leveldown/build/Release/leveldown.node' was compiled against a different Node.js version

What we want

- Upgrading Node.js with less work
- Easily swap back and forth between Node versions
- Binaries were compatible
- Leverage innovation in difference Node flavors

Native add-on use cases

- Access native APIs on the platform unavailable via JavaScript
- Derive better performance for compute bound code using C/C++
- Use of legacy code/logic available in C/C++ via JavaScript

Native Add-ons

node-sass

bcrypt

sqlite3

leveldown

websocket

icov

~30%
estimated
ecosystem
dependecy

First attempt NAN

- Available since 0.8
- Only provide so much isolation
- Does not address deployment issue

New Approach Needed

What is N-API

- A Stable Node API layer for native modules
- Provides ABI Stability guarantees across different Node versions & flavors
- Enables native modules just work across different Node versions & flavors without recompilation

JavaScript

NAN | Native Module

Node.js

JS Engine

Module Maintainers

- Have to update modules to support new Node.js versions

Module Consumers

- Have to recompile modules
- Have to wait for updated modules

JavaScript

Native Module

N-API

Node.js

JS Engine

Module Maintainers

- Don’t have to update modules to support new Node.js versions

Module Consumers

- Don’t have to recompile modules
- Don’t have to wait for updated modules

API Shape

```
napi_status napi_create_array(napi_env env, napi_value* result);
napi_status napi_get_and_clear_last_exception(napi_env env, napi_value* result);
napi_status napi_is_exception_pending(napi_env env, bool* result);
napi_status napi_throw(napi_env env, napi_value error);
```

Plain API Usage

```
#include <node_api.h>;

napi_value RunCallback(napi_env env, napi_cb_info info) {
  ...
}

napi_value Init(napi_env env, napi_value exports) {
  napi_status status;
  napi_value func;

  status = napi_create_function(env, NULL, 0, RunCallback, NULL, &func);
  assert(status == napi_ok);

  return func;
}

NAPI_MODULE(demo, Init);
```

```
napi_value RunCallback(napi_env env, napi_callback_info info) {
  napi_status status;

  size_t argc = 1;
  napi_value args[1];
  status = napi_get_cb_info(env, info, &argc, args, NULL, NULL);

  napi_value cb = argv[0];

  napi_value argv[1];
  status = napi_create_string_utf8(env, "Hello World", NAPI_AUTO_LENGTH, argv);

  napi_value global;
  status = napi_get_global(env, &global);

  napi_value result;
  status = napi_call_function(env, global, cb, 1, argv, &result);

  return NULL;
}
```

Conversion to N-API

https://github.com/nodejs/node-addon-api

Intuitive C++ Wrapper

```cpp
#include <napi.h>;

void RunCallback(const Napi::CallbackInfo& info) {
  Napi::Env env = info.Env();
  Napi::Function cb = info[0].As<Napi::Function>();
  cb.Call(env.Global(), {
    Napi::String::New(env, "Hello World")
  });
}

Napi::Object Init(Napi::Env env, Napi::Object exports) {
  return Napi::Function::New(env, RunCallback);
}

NODE_API_MODULE(demo, Init)
```

the callback problem

Callback never make ease

- Non thread safe JS Engine API
- Not guaranteed ABI Stability on libuv
- Manually reference count on the callback function value
- Self managed context & finalization

N-API Thread Safe Functions

- Can be called safely from threads other than JS Engine main thread
- ABI Stability guaranteed
- N-API managed garbage collection
- Easy to use context & finalization
- Async Hooks integrated

```
napi_status
napi_create_threadsafe_function(napi_env env,
                                napi_value func,
                                napi_value async_resource,
                                napi_value async_resource_name,
                                size_t max_queue_size,
                                size_t initial_thread_count,
                                void* thread_finalize_data,
                                napi_finalize thread_finalize_cb,
                                void* context,
                                napi_threadsafe_function_call_js call_js_cb,
                                napi_threadsafe_function* result);
napi_status
napi_call_threadsafe_function(napi_threadsafe_function func,
                              void* data,
                              napi_threadsafe_function_call_mode is_blocking);
```

```
napi_status
napi_acquire_threadsafe_function(napi_threadsafe_function func);
napi_status
napi_release_threadsafe_function(napi_threadsafe_function func,
                                 napi_threadsafe_function_release_mode mode);
```

Questions?

Thanks

@legendecas