给 Node.js 插上 C++ 的翅膀

死月@蚂蚁金服

自我介绍

- 蚂蚁金服体验技术部 – Node.js 工程师
- 前大搜车无线架构组中间件团队负责人
- Node.js Core Collaborator
- 《Node.js：来一打 C++ 扩展》作者

一个故事

天造需求

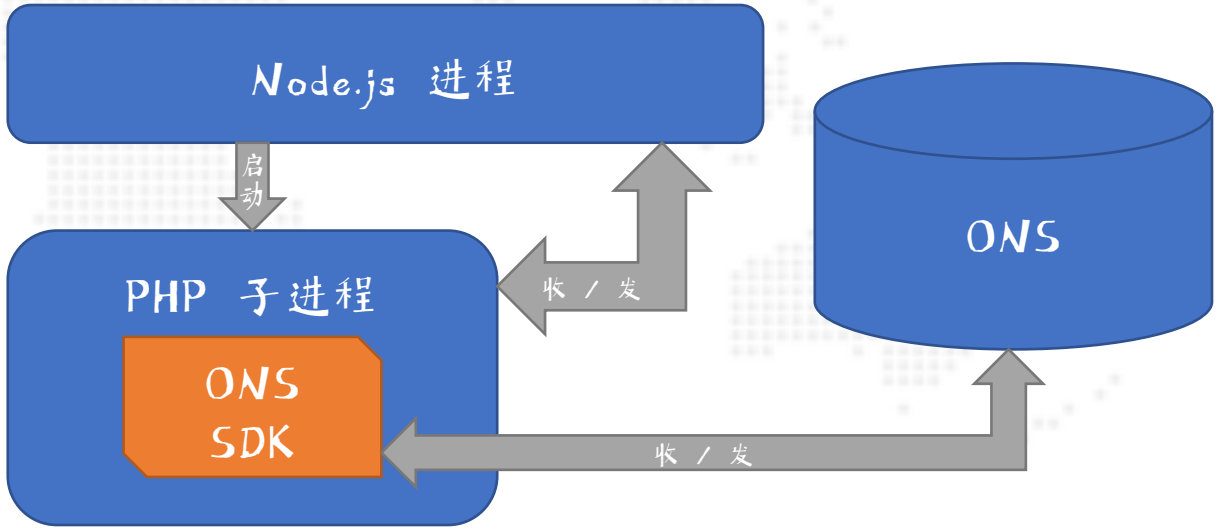
地设 Aliyun ONS

啊咧？ Node.js SDK 呢？

居然

- 有 PHP SDK!
- PHP SDK 是 C 的扩展!

脏脏的 PHP 子进程方案



有多脏?

```
project/
├── lib/
│   ├── ons/
│   │   ├── so/
│   │   │   ├── libonsclient4cpp.so
│   │   │   └── ONSClient4PHP.so # 只支持某一 PHP 版本
│   │   ├── script.php
│   │   ├── spawner.js # 启动并与子进程交互
│   │   ├── bar.js
│   │   ├── foo.js
│   │   └── ...
│   ├── app.js
│   └── ...
```

宝宝只想做一个安静的 Node.js 项目

自己动手，丰衣足食。

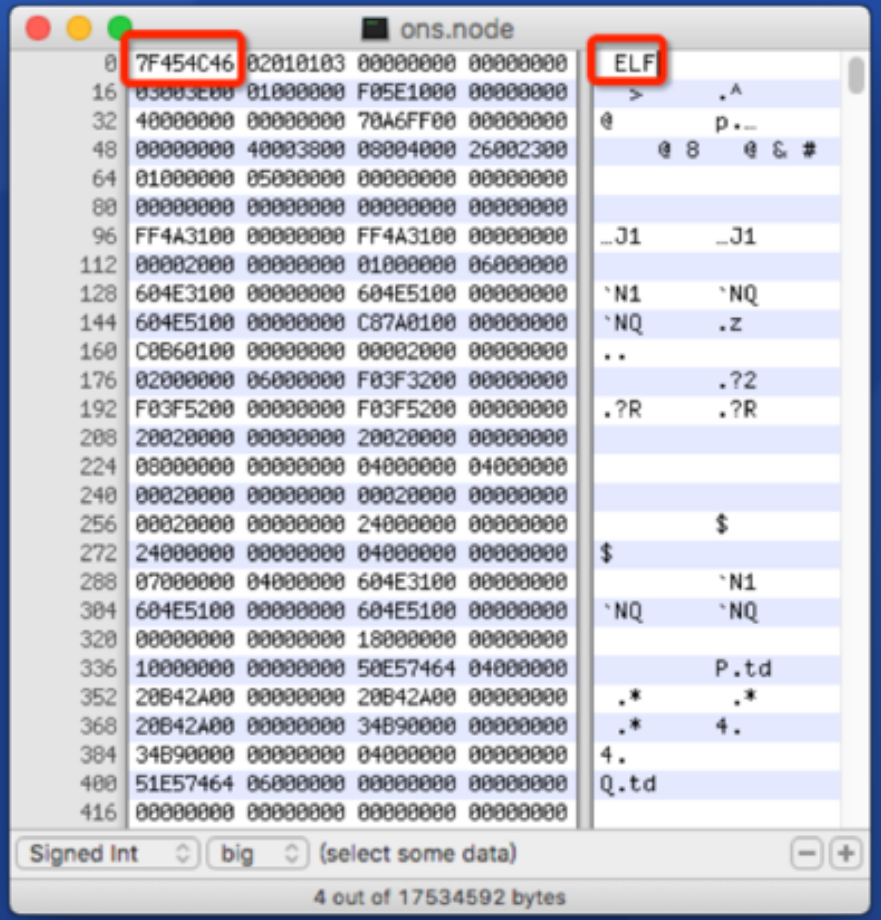
用 C++ SDK 封装一个 Node.js SDK，
从此踏上了 C++ Addons 的不归路。

1本

知其然,知其所以然。

动态链接库

```
const ons = require("./build/Release/ons.node");
```



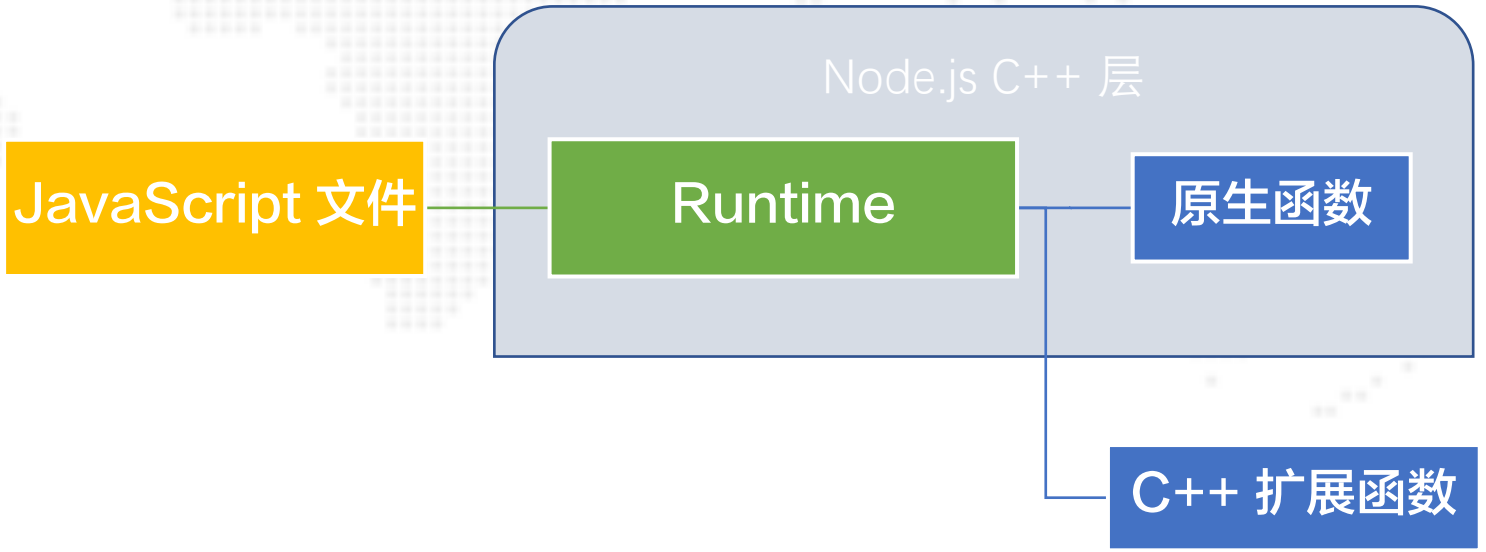
如何 require

The process.dlopen() method allows to dynamically load shared objects. It is primarily used by require() to load C++ Addons, and should not be used directly, except in special cases. In other words, require() should be preferred over process.dlopen(), unless there are specific reasons.

4 Steps to Require an Addon

```
require("./build/Release/ons.node");

↓

Module._extensions[".node"](...);

↓

process.dlopen(..., <path>);

↓

uv_dlopen(...); // 或
dlopen(...);
```

都是 C++,与 Node.js 原生函数的区别



2时

东风不与周郎便,铜雀春深锁二乔。

回到故事

同类代码复杂（翻译）

上游 SDK 限制

底层限制

rank	runtime	**secs**	mem	gz	cpu	cpu load
1	**C++ g++**	**8.24**	1,924	1763	8.23	1% 0% 100% 1%
2	**Fortran**	**8.28**	8	1524	8.28	0% 1% 100% 0%
3	**Ada**	**8.80**	2,116	2617	8.79	1% 100% 0% 0%
4	**C gcc**	**9.17**	1,228	1490	9.17	100% 0% 0% 1%
5	**Rust**	**13.31**	1,768	1805	13.31	0% 0% 1% 100%
...						
16	**Node.js 10.7.0**	**26.35**	33,228	1297	26.35	1% 0% 100% 0%

数据来源：https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/nbody.html

性能：NBody

虽然我也不是很懂，但逻辑是用简易的辛积分器模拟类木行星轨道

注意:
入不敷出

- C++ 性能高一丢丢
- 却抵不过 Node.js 打开并执行扩展时候所消耗掉的 IO
- 或抵不过 V8数据结构与你自行设计的数据结构之间进行转换所消耗的时间

3法

万法归一,一归何处。

3.1. 环境

C++ 编译环境

- macOS: xcoder
- Unix: gcc
- Windows: Visual Studio Build Tools / Visual Studio

Python 2.7

node-gyp

3.2. 瞥

hello.cpp

```cpp
#include <node.h>

using v8::FunctionCallbackInfo;
using v8::Isolate;
using v8::Local;
using v8::Object;
using v8::String;
using v8::Value;

// 函数逻辑
void Method(const FunctionCallbackInfo<Value>& args)
{
    Isolate* isolate = args.GetIsolate();
    args.GetReturnValue().Set(String::NewFromUtf8(isolate, "first build"));
}

// 相当于模块闭包
void init(Local<Object> exports)
{
    // exports.first = Method;
    NODE_SET_METHOD(exports, "first", Method);
}

// 声明模块宏
NODE_MODULE(addon, init)
```

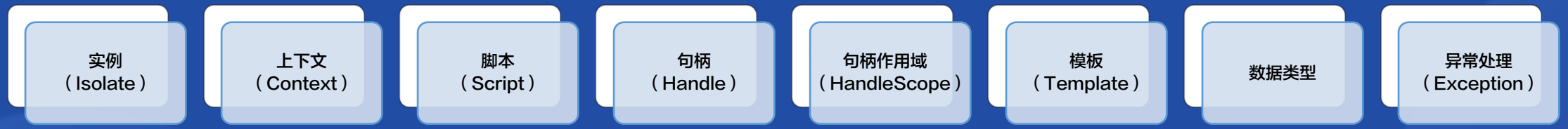
binding.gyp

```
{
  "targets": [{
    "target_name": "hello",
    "sources": [
      "hello.cpp"
    ]
  }]
}
```

bash #

```
$ node-gyp rebuild
...

$ node
> const hello = require("./build/Release/hello");
undefined
> hello.first();
'first build'
```

3.3. Chrome V8


实例
（Isolate）
上下文
（Context）
脚本
（Script）
句柄
（Handle）
句柄作用域
（HandleScope）
模板
（Template）
数据类型
异常处理
（Exception）

句柄（Handle）

本地句柄 （v8::Local）	持久句柄 （v8::Persistent）	永生句柄 （v8::Eternal）	其他句柄

对于堆内存中 JavaScript 数据对象的一个引用。

句柄作用域(HandleScope)

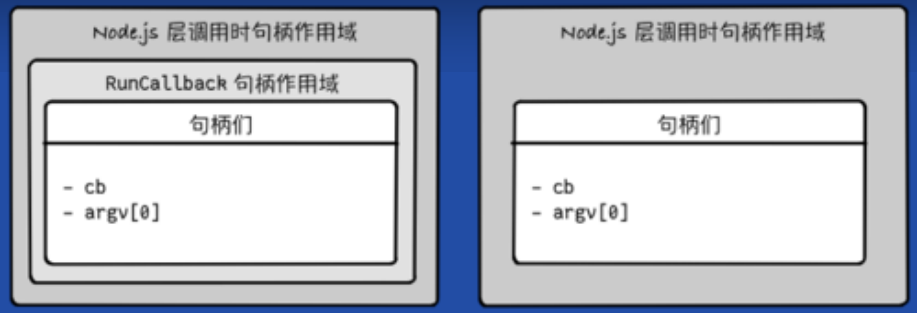
当一个句柄作用域对象的析构函数被调用时，在这个作用域中创建的所有句柄都会被从栈中抹去。于是，通常情况下这些句柄所指的对象将会失去所有引用，然后后来会被垃圾收集器统一处理。

- 一般句柄作用域
- 可逃句柄作用域

作用域是一个套一个地以栈的形式存在。在栈顶的句柄作用域处于激活状态。每次创建新的被管理的对象的时候，都会将对象交与栈顶的作用域管理，当栈顶作用域生命周期到的时候，这段时间创建的对象就会被回收。

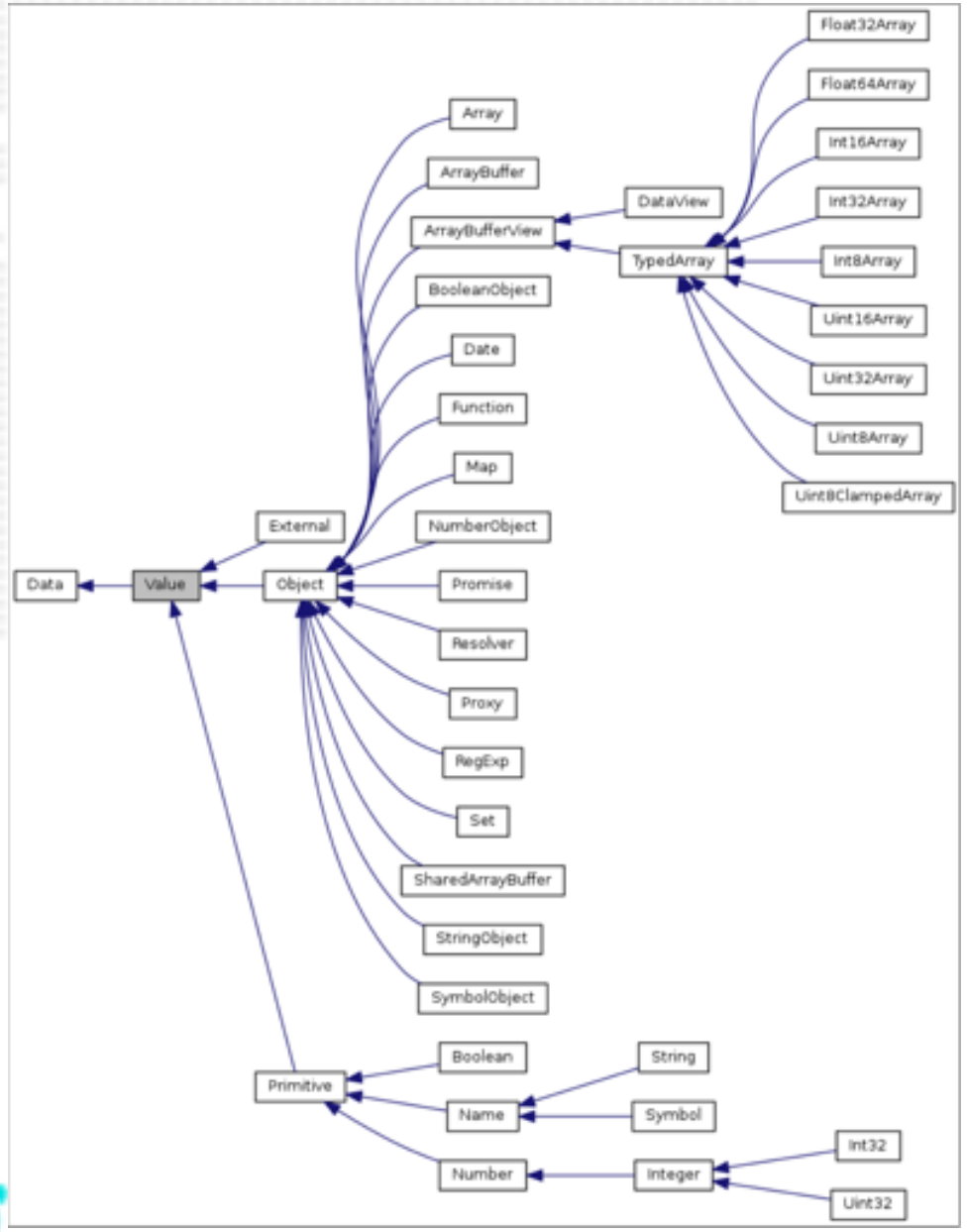
demo.cpp

```cpp
void RunCallback(const FunctionCallbackInfo<Value>& args)
{
    Isolate* isolate = Isolate::GetCurrent();
    HandleScope scope(isolate); // 就是这句 🤣

    Local<Function> cb = Local<Function>::Cast(args[0]);
    const unsigned argc = 1;
    Local<Value> argv[argc] = { String::NewFromUtf8(isolate, "hello world") };
    cb->Call(isolate->GetCurrentContext()->Global(), argc, argv);
}
```



Node.js 在调起 C++ 扩展之前就已经为我们在上层 C++ 函数中创建了一个句柄作用域了。

常用数据类型


Float32Array
Float64Array
Array
Int16Array
ArrayBuffer
Int32Array
DataView
ArrayBufferView
TypedArray
Int8Array
BooleanObject
Uint16Array
Date
Uint32Array
Function
Uint8Array
Map
Uint8ClampedArray
External
NumberObject
Data
Value
Object
Promise
Resolver
Proxy
RegExp
Set
SharedArrayBuffer
StringObject
SymbolObject
Boolean
String
Primitive
Name
Symbol
Int32
Number
Integer
Uint32

3.4. NAN

A header file filled with macro and utility goodness for making add-on development for Node.js easier across versions 0.8, 0.10, 0.12, 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10.

binding.gyp

```
{
  "targets": [{
    "target_name": "echo",
    "sources": [ "echo.cc" ],
    "include_dirs": [
      "<!(node -e \"require('nan')\")"
    ]
  }]
}
```

echo.cc

```
#include <nan.h>

using v8::String;
using v8::FunctionTemplate;

NAN_METHOD(Echo)
{
    if(info.Length() < 1)
    {
        Nan::ThrowError("Wrong number of arguments.");
        return;
    }

    info.GetReturnValue().Set(info[0]);
}

NAN_MODULE_INIT(Init)
{
    Nan::Set(target, Nan::New<String>("echo").ToLocalChecked(),
            Nan::GetFunction(Nan::New<FunctionTemplate>(Echo)).ToLocalChecked());
}

NODE_MODULE(echo, Init)
```

CityHash

city.js

```
var city = require("./build/Release/city");
var Long = require("long");

exports.city64 = function(str) {
    var _int64 = city.cityHash64(str);
    _int64 = new Long(_int64.readInt32LE(0), _int64.readInt32LE(4), true);
    return _int64;
};
```

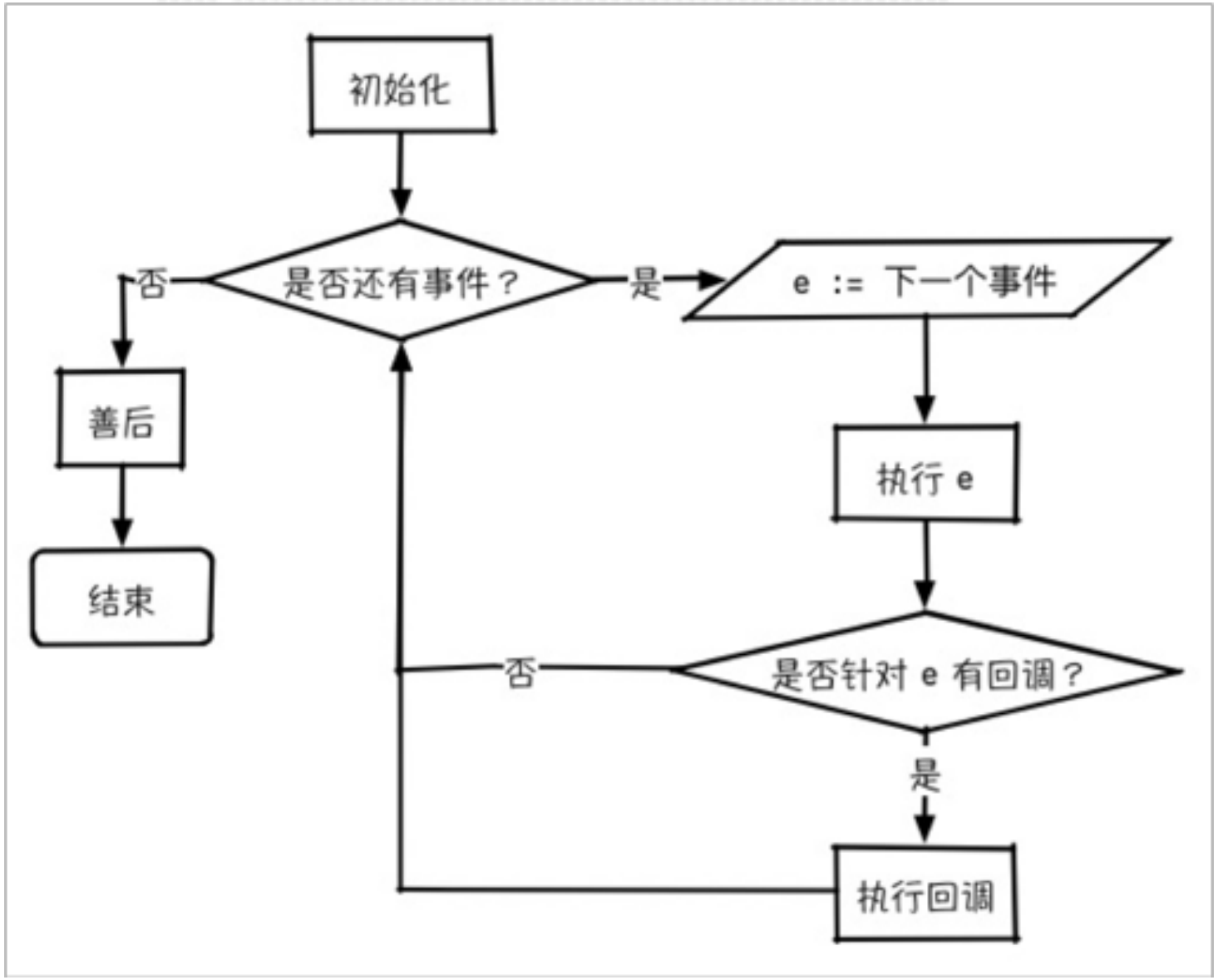
city.cpp

```
#include "cityhash.h"
...

void __CityHash64FreeCallback(char* data, void* hint)
{
    uint64* hash = (uint64*)data;
    delete hash;
}

...

NAN_METHOD(_CityHash64)
{
    if(info.Length() < 1)
    {
        return Nan::ThrowError("invalid argument count");
    }

    String::Utf8Value v8_source_string(info[0]->ToString());
    std::string source_string = *v8_source_string;
    unsigned int len = source_string.size();

    uint64* hash = new uint64(CityHash64(source_string.c_str(), len));
    info.GetReturnValue().Set(
            Nan::NewBuffer(
                (char*)hash,
                sizeof(uint64),
                __CityHash64FreeCallback,
                NULL).ToLocalChecked());
}

NAN_MODULE_INIT(Init)
{
    ...

    Set(target, New<String>("cityHash64").ToLocalChecked(),
            GetFunction(New<FunctionTemplate>(_CityHash64)).ToLocalChecked());
}

NODE_MODULE(city, Init)
```

3.5. libuv——事件循环与异步

“事件”一词：

- 文件已经准备好，处于可写状态；
- 一个 Socket 已有数据；
- 一个计时器超时了；
- ……。



句柄(Handle)与请求(Request)

句柄

uv_loop_t：事件循环句柄

uv_handle_t：所有句柄类型的基础类型

uv_tcp_t：TCP 句柄

uv_udp_t：UDP 句柄

uv_timer_t：定时任务句柄

uv_async_t：异步句柄，允许用户从另一个非 libuv 维护的线程中唤醒 libuv 事件循环

...

请求

uv_req_t：所有请求类型的基础类型

uv_getaddrinfo_t：用于 uv_getaddrinfo()

uv_connect_t：用于 uv_listen()

...

没什么卵用的睡眠排序

sleep.cc

```cpp
#include <nan.h>
#include <vector>

...

void _Sleep(int sleep_ms)
{
    usleep(sleep_ms * 1000);
}

struct ThreadArg {
    std::vector<uint32_t>* vec;
    uint32_t num;
};

void Sleep(void* _)
{
    ThreadArg* arg = (ThreadArg*)_;

    // 睡上 num * 100 毫秒
    _Sleep(arg->num * 100);
    arg->vec->push_back(arg->num);

    delete arg;
}

NAN_METHOD(Sort)
{
    if(info.Length() < 1 || !info[0]->IsArray())
    {
        return Nan::ThrowError("Wrong argument");
    }

    Local<Array> array = info[0].As<Array>();
    if(!array->Length())
    {
        return info.GetReturnValue().Set(Nan::New<Array>());
    }
```

demo.js

```js
const sleep = require("./build/Release/sleep");
sleep.sort([ 2, 4, 6, 9, 1 ]);

// 900 毫秒后
// < [ 1, 2, 4, 6, 9 ]

sleep.sort([ 2, 4, 6, 9, 1, 1 ]);

// 一定几率崩溃
```

sleep.cc 续

```cpp
    std::vector<uint32_t> orig;
    for(uint32_t i = 0; i < array->Length(); i++)
    {
        if(!Nan::Get(array, i).ToLocalChecked()->IsUint32())
        {
            return Nan::ThrowTypeError("Elements should be unsigned int.");
        }

        orig.push_back(Nan::To<uint32_t>(Nan::Get(array, i).ToLocalChecked()).FromJust());
    }

    std::vector<uint32_t> vec;
    std::vector<uv_thread_t> handles(array->Length());
    for(uint32_t i = 0; i < array->Length(); i++)
    {
        ThreadArg* arg = new ThreadArg();
        arg->vec = &vec;
        arg->num = orig[i];
        uv_thread_create(&handles[i], Sleep, arg);
    }

    for(uint32_t i = 0; i < handles.size(); i++)
    {
        uv_thread_join(&handles[i]);
    }

    Local<Array> ret = Nan::New<Array>();
    for(uint32_t i = 0; i < array->Length(); i++)
    {
        Local<Value> v = Nan::New(vec[i]);
        Nan::Set(ret, i, v);
    }

    info.GetReturnValue().Set(ret);
}

NAN_MODULE_INIT(Init)
{
    Nan::Export(target, "sort", Sort);
}

NODE_MODULE(sleep, Init)
```

同步原语

互斥锁（Mutex Lock）

读 / 写锁（Write-Read Lock）

信号量（Semaphore）

条件变量（Condition Variable）

屏障（Barrier）

睡眠排序 + 同步原语之互斥锁

scope_lock.cc

```cpp
struct ScopeLock {
    uv_mutex_t* handle;

    ScopeLock(uv_mutex_t* handle) : handle(handle)
    {
        uv_mutex_lock(handle);
    }

    ~ScopeLock()
    {
        uv_mutex_unlock(handle);
    }
};
```

sleep.cc

```cpp
struct ThreadArg {
    ...
    uv_mutex_t* mutex_handle;
};

void Sleep(void* _)
{
    ThreadArg* arg = (ThreadArg*)_;

    // 睡上 num * 100 毫秒
    _Sleep(arg->num * 100);

    ScopeLock lock(arg->mutex_handle);
    arg->vec->push_back(arg->num);

    delete arg;
}

NAN_METHOD(Sort)
{

    ...

    std::vector<uint32_t> vec;
    std::vector<uv_thread_t> handles(array->Length());
    uv_mutex_t handle;
    uv_mutex_init(&handle);
    for(uint32_t i = 0; i < array->Length(); i++)
    {
        ThreadArg* arg = new ThreadArg();
        ...
        arg->mutex_handle = &handle;
        uv_thread_create(&handles[i], Sleep, arg);
    }

    ...
    uv_mutex_destroy(&handle);
}
```

ONS 中 Consume 线程与事件循环线程交互

消费线程

```cpp
COMMON_ACTION ONSConsumerBaseListener::Consume(Message& message)
{
    const char* msg_id = message.getMsgID();

    ONSConsumerACKInner* ack_inner = new ONSConsumerACKInner(msg_id);
    MessageHandlerParam* param = new MessageHandlerParam();
    param->message = &message;
    param->ons = parent;
    param->ack_inner = ack_inner;

    // GetAsync() 函数中会返回一个初始化好的 async 句柄
    uv_async_t* async = this->GetAsync();
    ...

    async->data = (void*)param;

    // 发送给事件循环线程处理
    // 在 GetAsync() 中定义了返回的 async 由 HandleMessage 处理
    uv_async_send(async);

    // 等待 Ack
    COMMON_ACTION result = ack_inner->WaitResult();

    async->data = NULL;
    param->ons = NULL;
    param->ack_inner = NULL;
    param->message = NULL;

    delete ack_inner;
    delete param;

    // 释放 async 句柄
    RestoreAsync(async);

    return result;
}
```

事件循环线程

```cpp
void ONSConsumerV8::HandleMessage(uv_async_t* handle)
{
    Nan::HandleScope scope;

    AliyunONS::MessageHandlerParam* param = (AliyunONS::MessageHandlerParam*)handle->data;
    Message* message = param->message;

    ONSConsumerV8* ons = param->ons;

    ONSConsumerACKInner* ack_inner = param->ack_inner;

    // 将 ack_inner 包装成一个 JavaScript 可调用的对象
    // 一旦对象被调用 .ack()，ack_inner.WaitResult() 就会完成
    v8::Local<v8::Function> cons = Nan::New<v8::Function>(ONSConsumerACKV8::GetConstructor());
    v8::Local<v8::Object> ack_obj = cons->NewInstance(0, {});
    ONSConsumerACKV8* ack = ObjectWrap::Unwrap<ONSConsumerACKV8>(ack_obj);
    ack->SetInner(ack_inner);

    // 将消息对象中的字段一一赋值
    v8::Local<v8::Object> result = Nan::New<v8::Object>();
    result->Set(
            Nan::New<v8::String>("topic").ToLocalChecked(),
            Nan::New<v8::String>(message->getTopic()).ToLocalChecked());
    ...

    // 三个参数
    //   + err: undefined
    //   + message: result
    //   + ack: ack_obj
    v8::Local<v8::Value> argv[3] = { Nan::Undefined(), result, ack_obj };

    // 将三个参数传给监听函数
    Nan::Callback* callback = ons->GetListenerFunc();
    callback->Call(3, argv);
}
```

ONS 中的 Ack 类

...::Ack

```
1 void Ack(COMMON_ACTION result = COMMON_ACTION::SUCCESS)
2 {
3     uv_mutex_lock(&mutex);
4     bool _acked = this->acked;
5
6     if(_acked)
7     {
8         uv_mutex_unlock(&mutex);
9         return;
10    }
11
12    ack_result = result;
13
14    acked = true;
15
16    // 发送条件变量并解锁
17    uv_cond_signal(&cond);
18    uv_mutex_unlock(&mutex);
19 }
```

...::WaitResult

```
1 COMMON_ACTION WaitResult()
2 {
3     uv_mutex_lock(&mutex);
4
5     if(acked)
6     {
7         COMMON_ACTION result = ack_result;
8         uv_mutex_unlock(&mutex);
9         return result;
10    }
11
12    // 等待条件变量收到
13    uv_cond_wait(&cond, &mutex);
14
15    COMMON_ACTION result = ack_result;
16    uv_mutex_unlock(&mutex);
17
18    return result;
19 }
```

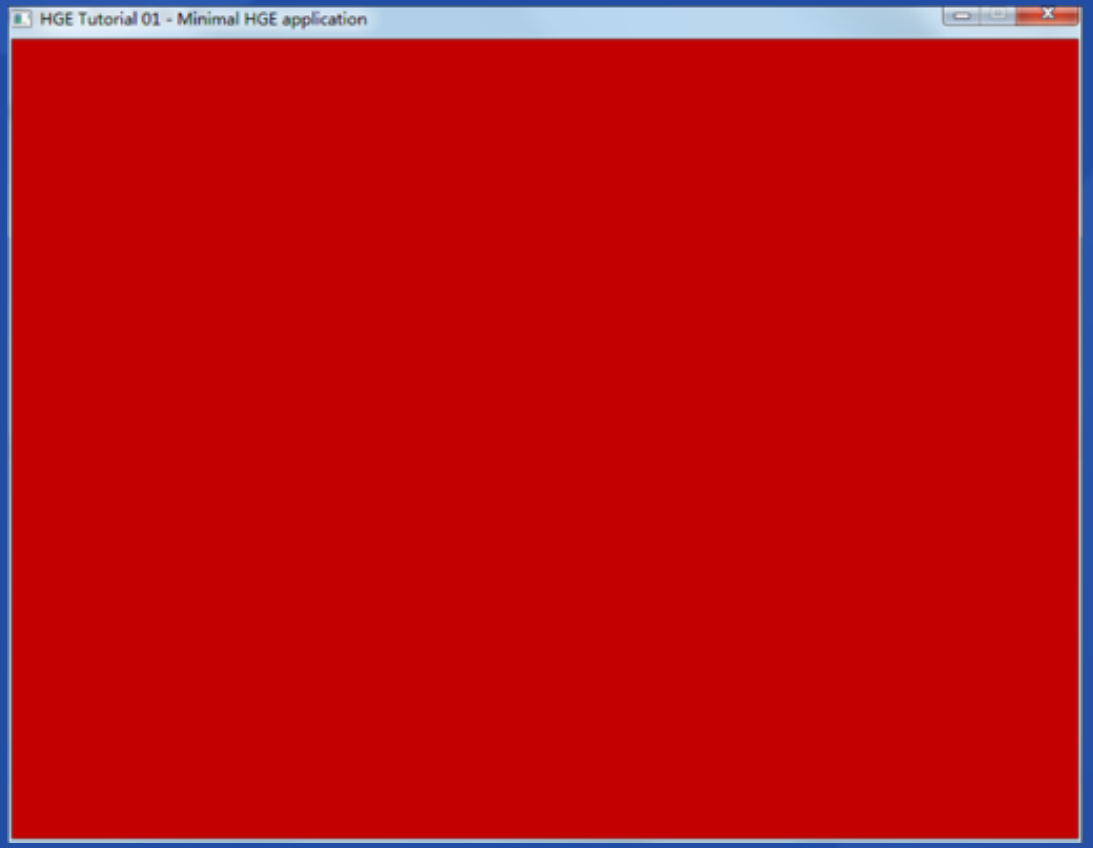
4 跳

旋转、跳跃，我不停歇。

HGE——一个老掉牙的 2D 游戏引擎

game.cc

```cpp
#include <hge.h>

HGE* hge = nullptr;

bool RenderFunc()
{
    hge->Gfx_BeginScene();
    hge->Gfx_Clear(0xffff0000);
    hge->Gfx_EndScene();
    return false;
}

bool FrameFunc()
{
    if (hge->Input_GetKeyState(HGEK_ESCAPE)) return true;
    return false;
}

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    hge = hgeCreate(HGE_VERSION);

    hge->System_SetState(HGE_FRAMEFUNC, frame_func);
    hge->System_SetState(HGE_TITLE, "Minimal HGE application");
    hge->System_SetState(HGE_WINDOWED, true);
    hge->System_SetState(HGE_USESOUND, false);

    if (hge->System_Initiate())
    {
        hge->System_Start();
    }
    else
    {
        MessageBox(nullptr, hge->System_GetErrorMessage(), "Error",
                   MB_OK | MB_ICONERROR | MB_APPLMODAL);
    }

    hge->System_Shutdown();
    hge->Release();

    return 0;
}
```



Node.js HGE Demo

addon.cc

```cpp
HGE* hge = 0;
Nan::Callback* node_frame_callback;
Nan::Callback* node_render_callback;
Nan::Callback* hge_done_callback;
uv_cond_t cond;
uv_cond_t cond_render;
uv_mutex_t mutex;
uv_mutex_t mutex_render;

void JSFrame(uv_async_t* handle)
{
    Nan::HandleScope scope;
    float* delta_time = (float*)handle->data;
    v8::Local<v8::Number> dt = Nan::New<v8::Number>(*delta_time);
    delete delta_time;
    v8::Local<v8::Value> args[] = { dt };
    node_frame_callback->Call(1, args);
    uv_mutex_lock(&mutex);
    uv_cond_signal(&cond);
    uv_mutex_unlock(&mutex);
}

void JSRender(uv_async_t* handle)
{
    Nan::HandleScope scope;
    node_render_callback->Call(0, NULL);
    uv_mutex_lock(&mutex_render);
    uv_cond_signal(&cond_render);
    uv_mutex_unlock(&mutex_render);
}
```

addon.cc

```cpp
void OnHGEDone(uv_async_t* handle)
{
    Nan::HandleScope scope;
    hge_done_callback->Call(0, NULL);
    delete handle;
}

bool RenderFunc()
{
    uv_async_t async;
    uv_async_init(uv_default_loop(), &async, JSRender);
    hge->Gfx_BeginScene();
    uv_mutex_lock(&mutex_render);
    uv_async_send(&async);

    uv_cond_wait(&cond_render, &mutex_render);
    uv_mutex_unlock(&mutex_render);
    hge->Gfx_EndScene();
    return false;
}

bool FrameFunc()
{
    uv_async_t async;
    uv_async_init(uv_default_loop(), &async, JSFrame);
    float* delta_time = new float(hge->Timer_GetDelta());
    async.data = (void*)delta_time;
    uv_mutex_lock(&mutex);
    uv_async_send(&async);
    uv_cond_wait(&cond, &mutex);
    uv_mutex_unlock(&mutex);
    return false;
}
```

addon.cc

```cpp
void GameThreadFunc(void*)
{
    hge->System_Initiate();
    hge->System_Start();
    hge->System_Shutdown();
    hge->Release();
    uv_async_t* async = new uv_async_t();
    uv_async_init(uv_default_loop(), async, OnHGEDone);
    uv_async_send(async);
}

NAN_METHOD(SetFrameCallback)
{
    node_frame_callback = new Nan::Callback(info[0].As<v8::Function>());
}

NAN_METHOD(SetRenderCallback)
{
    node_render_callback = new Nan::Callback(info[0].As<v8::Function>());
}

NAN_METHOD(SetHGEDoneCallback)
{
    hge_done_callback = new Nan::Callback(info[0].As<v8::Function>());
}

NAN_METHOD(Start)
{
    hge = hgeCreate(HGE_VERSION);
    hge->System_SetState(HGE_FRAMEFUNC, FrameFunc);
    hge->System_SetState(HGE_RENDERFUNC, RenderFunc);
    ...
    uv_thread_t handle;
    uv_thread_create(&handle, GameThreadFunc, NULL);
}

NAN_METHOD(GfxClear)
{
    uint32_t color = Nan::To<uint32_t>(info[0]).FromJust();
    hge->Gfx_Clear(color);
}
```

Demo 的样例

hge.js

```js
const hge = require("../build/Release/node_hge");

process.stdin.resume();

let time = 0;

hge.setFrameCallback(function(dt) {
    time += dt;
    if(time > 5) time -= 5;
    if(time < 0) time = -time;
    console.log("FPS:", 1 / dt);
});

hge.setRenderCallback(function() {
    const ret = `ff${parseInt((255 * ((5 - time) / 5))).toString(16)}0000`;
    hge.gfxClear(parseInt(ret, 16));
});

hge.setHGEDoneCallback(function() {
    console.log("Bye!");
    process.exit();
});

hge.start();
```

5 猫

白猫黑猫,抓老鼠就是好猫。

N-API

ABI 兼容性：编译产物与 Node.js 版本无关

C API

C++ Wrapper：node-addon-api

N-API

	1	2	3
v4.x			
v6.x			v6.14.2*
v8.x	v8.0.0*	v8.10.0*	
v9.x	v9.0.0*	v9.3.0*	v9.11.0*
v10.x			v10.0.0

* 代表实验性版本

WebAssembly

编译成 *.wasm

编译成 *.js

CRC64-ECMA182

代码片段

```
1 void crc64(uint64_t* crc, void *buf, size_t len)
2 {
3     uint64_t
4     *crc = *(
5
6 }
7
8 void str_to_u
9 {
10    sscanf(st
11 }
12
13 void uint64_t
14 {
15    sprintf(str, "%llu", *ret);
16 }
```

compile

```
1 emcc --bind -o dist/crc.js webassembly/crc64_ecma_182.cc -s \
2     EXPORTED_FUNCTIONS="[\
3         '_crc64_init',\
4         '_crc64',\
5         '_str_to_uint64',\
6         '_uint64_to_str'\
7     ]"
```

CRC64-ECMA182

代码片段

```
const binding = require('./dist/crc');

const raw = {
    crc64: binding.cwrap('crc64', 'null', [ 'number', 'number', 'number' ]),
    crc64Init: binding.cwrap('crc64_init', 'null', []),
    strToUint64Ptr: binding.cwrap('str_to_uint64', 'null', [ 'number', 'number' ]),
    uint64PtrToStr: binding.cwrap('uint64_to_str', 'null', [ 'number', 'number' ])
};

raw.crc64Init();

function strToUint64Ptr(str) {
    const strPtr = binding._malloc(str.length + 1);
    binding.stringToUTF8(str, strPtr, str.length + 1);

    const uint64Ptr = binding._malloc(8);
    raw.strToUint64Ptr(strPtr, uint64Ptr);
    binding._free(strPtr);

    return uint64Ptr;
}
```

代码片段

```
module.exports.crc64 = function(buff, prev) {
    if(!prev) prev = '0';
    if(typeof prev !== 'string' || !/\d+/.test(prev)) {
        throw new Error('Invlid previous value.');
    }

    const prevPtr = strToUint64Ptr(prev);
    const buffPtr = buffToPtr(buff);

    raw.crc64(prevPtr, buffPtr, buff.length);
    const ret = uint64PtrToStr(prevPtr);

    binding._free(prevPtr);
    binding._free(buffPtr);

    return ret;
};
```

6 结

看桃花，开出怎样的结果

FIN