[1. typescript项目架构 1](#_Toc500105312)

[1.2 package.json | tsconfig.json 1](#_Toc500105313)

[1.2.1 Directive (nodejs) 1](#_Toc500105314)

[2. typescript简介 2](#_Toc500105315)

[2.1 编译工具—tsc 2](#_Toc500105316)

[2.1.1 编译选项 2](#_Toc500105317)

[2.1.2 添加非.ts文件 2](#_Toc500105318)

[2.2 基本类型 3](#_Toc500105319)

[2.2.1 enum 3](#_Toc500105320)

[3. Import vs require 4](#_Toc500105321)

[3.1 包依赖的解析、引入状态 4](#_Toc500105322)

[3.1.1 简介 4](#_Toc500105323)

[3.1.2 nodejs的引入机制 4](#_Toc500105324)

[3.2 exports与module.exports的导出 4](#_Toc500105325)

[Class-based vs Prototype-base 4](#_Toc500105326)

[编译选项文件tsconfig 4](#_Toc500105327)

[编译生成的文件 4](#_Toc500105328)

Typescript项目webstorm的环境配置

File--> settings -->language & frameworks --> Typescript --> enable typescript compiler --> use tsconfig.json

# 前言

## 0.1机器码、真值、补码、反码、原码

机器码： 使用二进制表示，根据计算机字长， 最高位为符号位，其余位表示值。10000001 表示十进制中的-1.

真值： 机器码表示的实际数值。10000001 的真值为-1， 不是128.

原码： 即机器码

反码： 表示原码取反

补码： 表示反码加1

计算机不使用机器码（原码）表示二进制数据，亦不用反码。因为想要计算机只进行加法运算且不存在+0和-0的区别。

8bit字长的计算机：

原码范围： [-127, 127], 存在+0， -0

反码范围：[-127, 127]，存在+0， -0

补码范围：[-128, 127]， 不存在-0， 此时-0表示-128

0.2

# 1. typescript项目架构

<http://stackoverflow.com/questions/31173738/typescript-getting-error-ts2304-cannot-find-name-require>

对于单独的typescript文件，用于demo的用途，可以使用declare var require:any;来进行node的关键词require的使用。

便于使用let fs=require(“fs”)

Node中使用npm install依赖时，部分包存在.d.ts文件，如moment，当不存在时可以使用上述方法引入包依赖。;

使用Number进行字符转数字

## 1.2 package.json | tsconfig.json

package.json文件指明了项目的相关情况。

### 1.2.1 Directive (nodejs)

Main：指明了项目的入口文件。

Types: 指明typescript可以使用的说明文件。

Browser：指明了该module在被用于前端buddler打包的时，提示buddler应该加载的文件是哪些。当显示指定了browser时，当模块被打包成浏览器可用的版本时，该字段将会取代main，标志module的入口。相当于typescript的配置文件tsconfig中的指令paths

### 1.2.2 Directive (typescript)

baseUrl: 为非相对路径引入文件的机制，设置一个相对路径

---如果是通过命令行进行参数的传输的话，表明是当前目录。

---如果是通过tsconfig文件进行引入的话，则是tsconfig文件所在目录。

Paths: {}

该对象指明了相应的node\_modules的文件/或者程序的主入口， 该命令存在于compilerOptions对象中。同时paths指定的路径是相对baseUrl提供的路径的。

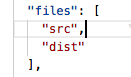
### 1.2.3 typescript的引入机制。

文件或者项目导入时，typescript的会按顺序进行相关文件的查找。首先是.ts文件，再次是.tsx文件，最后是.d.ts文件。

1.3 npm command

1.3.1 npm publish

当项目成功编译后，直接执行npm publish命令，会将项目直接push到npm仓库，当需要保存源码、和编译后的文件，此时需要在package.json中添加files属性（为数组类型）,该属性设置想要有的文件，



1.3.2 npm version patch

对于发布前，执行npm version patch会更新package文件中的version，自动加一

# 2. typescript简介

## 2.1 编译工具—tsc

参考: <https://zhongsp.gitbooks.io/typescript-handbook/content/doc/handbook/Compiler%20Options.html>

<https://www.typescriptlang.org/docs/handbook/compiler-options.html>

使用tsc，系统读取文件tsconfig.json文件进行系统配置，

Tsc file.ts必要时需要指定参数，来设置其运行环境

如: tsc --lib es6 ctrip-abroad-hotel.ts

### 2.1.1 编译选项-tsconfig.json

“target”: “es2015”, //语法规范版本

"declaration": true,

"declarationDir": "JLTypes",

“importHelpers”: true, //使用tslib库时，允许使用其中的各种helper function， 如\_\_extends、\_\_assign

### 2.1.2 添加非.ts文件

#### 2.1.2.1 添加js文件

"compilerOptions": {

"target": "es6",

"module": "commonjs",

"outDir": "dist",

"noEmitHelpers": true,

"importHelpers": true,

"sourceMap": true,

"alwaysStrict": true,

"experimentalDecorators": true,

"noImplicitUseStrict": false,

"allowSyntheticDefaultImports": true,

"baseUrl": ".",

"paths": {

"\*": ["\*", "\*/index"]

},

"lib": ["ES2015", "DOM"]

}

此时需要加入allowJs: true

#### 2.1.2.2 添加html、json等静态文件

### 2.1.3 vscode 调试typescript程序

reference: <https://code.visualstudio.com/docs/nodejs/nodejs-debugging>

.vscode文件夹中添加launch.json

#### 2.1.3.1 launch.json

{

"version": "0.2.0",

"configurations": [

{

"type": "node",

"request": "launch",

"name": "Launch Program",

"program": "${workspaceFolder}/index.js",

"preLaunchTask": "tsc: build - tsconfig.json",

"outFiles": [

"${workspaceFolder}/tmp/tsreq/\*\*/\*.js"

],

"protocol": "auto",

"trace": "all",

"sourceMaps": true

}

]

}

由于node作为engine，其执行的文件为.js文件，调试typescript语言时，需要将.ts文件编译成.js, 需要在编译时制定.js.map文件，用于.ts和.js 文件的映射。

#### 2.1.3.2 sourceMap

reference: <https://code.visualstudio.com/docs/nodejs/nodejs-debugging#_source-maps>

launch.json文件中的：

sourceMap属性： 控制是否生成.d.map文件用于制定.ts与.js文件的映射关系。

Vscode支持两种：

VS Code supports both the inlined source maps and the inlined source.

当生成的.d.map文件不与源码同目录，此时需要使用属性outFile制定编译后的文件

#### 2.1.3.3 smartStep

自动跳过不在.map文件中的代码

## 2.2 基本类型

function scope

block scope

### 2.2.1 enum

### 2.2.2 primitive type

Object、String、Boolean、Number为primitive类型， object、string、boolean、number为non-primitive类型

### 2.2.3 const vs let

Block scope variables that has temporal dead zone. If variable of const or let type is called before it’s declared, ***ReferenceError*** will be threw, however, variable of var type will be ***undefined*** in the same situation

Temporal dead zone: a period between entering scope and being declared

#### 2.2.3.1 Const

keyword used to declare constants. If a reference is assigned to a constant, the content of the reference could change.

**Note**

--- initialized when declaring

---can only be assigned once, only when declaring

---can’t make the assigned value immutable

So, for a variable of const and reference type, Object.defineProperty(param1, param2, param3) could be called on this variable.

### 2.2.4 Object

#### 2.2.4.1 attributes

##### 2.2.4.1.1 writable

to decide if the values of variables could be changed

##### 2.2.4.1.2 enumerable

for…in…、Object.key、JSON.stringify will access the enumrable properties.

##### 2.2.4.1.3 configurable

to decide if the properties of variable could be deleted or attributes like writable, enumerable, configurable can be changed.

Once the attribute configurable is set as non-configurable, only one behavior that Object.defineProperty triggers to set attribute writable from true to false can be allowed,

#### 2.2.4.2 properties

##### 2.2.4.2.1 Object.key

return all enumerable properties of a variable, including properties in it’s prototype in the order with that of for…in…

##### 2.2.4.2.2 Object.hasOwnProperties

return all own enumerable or non-enumerable properties of a variable

## 2.3 前端 vs 后端

### 2.3.1 后段模块引入模式

后段代码的引入机制是采用nodejs的模块机制，根据路径，查找本地的文件系统，从而运行完整程序。

### 2.3.2 前端模块引入模式

前端代码的引入机制是buddle所有的文件、模块，达到一种可以在浏览器中使用的形态（因为客户端，浏览器并不存在文件系统）。工具webpack、browserify等便是如此。Module bundler允许在浏览器中运行时解析外部模块。

对于angularjs，则假装外部模块并不存在。使用单一的namespace来限制对于global space的污染。Angularjs使用依赖注入的方式实现代码的正常工作。

## 2.4 new | constructor | prototype

2.4.1 new

static 申明的变量存在于对象本身属性，

原型链上的属性只有当使用new时，才会被初始化 ？

# 3. Import vs require

## 3.1 包依赖的解析、引入状态

### 3.1.1 简介

包的引入两种方式： 绝对路径引入、相对路径引入。已存在的解析策略有：classic、node。

ts中的module resolution有两种策略: Classic 和node

---前者根据文件所在目录依次向上寻找、为向后兼容。对于系统AMD、System、es2015的系统，默认使用classic的包解析策略。

---后者模仿nodejs的风格，进行包解析。

### 3.1.2 nodejs的引入机制

Nodejs中的包管理机制：

Nodejs中引入包的方式是通过调用函数require 来实现。

#### 3.1.2.1相对路径文件的引入

---当前所在目录进行目标文件的查找

---当前目录不存在，查找package.json文件，若存在，查找该文件中的main指令指明的文件路径。

---查找当前目录是否存在index.js文件，存在则会被隐式地作为main来进行。

#### 3.1.2.2 绝对路径的文件的查找

绝对路径方式导入外部文件，nodejs则会查找node\_modules的目录，在该目录下进行查找。

---当前所在文件的目录是否存在node\_modules

---一次上溯查找node\_modules目录，知道找到相应的外部依赖的包。

Node使用require来导入包依赖，对于参数为相对路径，进入包依赖后，根据文件中的package.json来进行查找main主要入口文件；若路径为绝对路径，则查找本目录的node\_modules，若不存在，则往上寻找。

<https://www.typescriptlang.org/docs/handbook/module-resolution.html>

## 3.2 exports与module.exports的导出

nodejs中的exports是module.exports的辅助方法。最终暴露给外部调用者的是module.exports。

exports中属性、方法最终会融合到module.exports中。但是当一个namespace或者module中显示指定了module.exports的对象赋值，那么exports中的属性无法融合到module.exports中，无法暴露给外部调用。

# Class-based vs Prototype-base

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model>

# 编译选项文件tsconfig

Include、exclude、files三个指令用于包含或移除至编译文件范畴。

但是files作用域更强，即使exclude指定了，若出现在files，那么仍然需要编译。

# 编译生成的文件

.js.map: 使用js文件与源代码.ts文件的映射文件。

# 4. New features

## 4.1 !!variable

var str = true;

!!str; //true, 表示 str || false;

## 4.2 expression1 && expression2

var str = expression && expression2; //此时在某些情况下为expression2

## 4.3 undefined vs null

undefined == null; //true

undefined === null; //falses

## 4.4 NaN

不等于任何值，包括NaN

## 4.5 module.exports & exports (nodejs) vs export (typescript)

### 4.5.1 nodejs environment

module.exports and exports are pointed to the same reference when initialization. If using exports = only in nodejs package mechanism, properties defined in one module can’t be exported. Expression “module.exports = exports = test” are needed.

### 4.5.2 Typescript environment

Export

Export default

Import str = require(“moment-timezone”);

## 4.6 bitwise operation (按位操作符)

### 4.6.1 bitwise and (&)

Returns a 1 in each bit position for which the corresponding bits of both operands are 1's.

Eg.

5 & 4 ----> 4

### 4.6.2 bitwise or

Returns a 1 in each bit position for which the corresponding bits of either or both operands are 1's.

Eg.

5 | 4 ----->5

### 4.6.3 bitwise XOR

Returns a 1 in each bit position for which the corresponding bits of either but not both operands are 1's.

5 ^ 4 ------> 1

### 4.6.4 Left shift

位移运算实用补码形式

Shifts a in binary representation b (< 32) bits to the left, shifting in 0's from the right.

5 << 1 ----> 10

### 4.6.5 Right Shift—arithmetic shift right

Shifts a in binary representation b (< 32) bits to the right, discarding bits shifted off.

5 >> 1 ----->2;

Noted:

Arithmetic shift right preserve the signedness of the number.

-5 >> 1

-5的原码形式：10000101， 补码形式：11111011， represented as -5.

After right shift, 补码右移： 11111101 ， 该机器码为10000011， represented as -3

### 4.6.6 Right Shift With Zero-fill ----logical shift right

Shifts a in binary representation b (< 32) bits to the right, discarding bits shifted off, and shifting in 0's from the left.

Noted:

Arithmetic shift right doesn’t preserve the signedness of the number. Just fill the bit as 0.

-5 >>> 1

10000101 represented as -5.

After right shift, 00000010 represented as

### 4.6.7 ~ not

~ 表示每bit取反（包括符号位）。（注意： 是一个数的补码形式进行取反, 获得的结果也是补码形式，需要转换成源码）

如-5的补码为111111011， ～（-5）为4

## 4.7 array.indexOf(index)

var array = [‘hello’, ‘world’]

普通做法

if(array.indexOf(‘hello’) >=0) return true;

快速做法：

if(~array.indexOf(‘hello’)) return true;

特点： 对于indexOf, 不存在的元素，一律返回-1， 否则返回相应的下标，0，1，2，3…

not calculation, 见4.6.7

～0，为-1，

～-1， 为0， 即false

# 5. Memory Allocation

## 5.1 introduction

## 5.2 usage of memory allocation

### 5.2.1 strong reference

GC engine won’t garbage-collect the variables of strong reference type. Like var str = new Array();

### 5.2.2 soft reference

GC engine will garbage-collect the variables of soft reference type when there is no enough memory to be used.

### 5.2.3 weak reference

GC engine will garbage-collect the variables of soft reference type after first round of garbage-collection.

### 5.2.4 phantom reference

如果一个对象仅持有虚引用，在任何时候都可能被垃圾回收，虚引用与软引用和弱引用的一个区别在于：虚引用必须和引用队列联合使用，虚引用主要用来跟踪对象 被垃圾回收的活动。

# 6. Background Mechanism

## 6.1 event-loop & worker pool

### 6.1.1 introduction

reference: <https://nodejs.org/en/docs/guides/dont-block-the-event-loop/>

Node uses a small number of threads to handle many clients. In Node there are two types of threads: one Event Loop (aka the main loop, main thread, event thread, etc.), and a pool of k Workers in a Worker Pool (aka the threadpool).

Node uses the Event-Driven Architecture: it has an Event Loop for orchestration and a Worker Pool for expensive tasks.

#### 6.1.1.1 Event Loop

Event Loop is used to responding to incoming client requests by executing the appropriate callback. In summary, the Event Loop executes the JavaScript callbacks registered for events, and is also responsible for fulfilling non-blocking asynchronous requests like network I/O.

In truth, the Event Loop does not actually maintain a queue. Instead, it has a collection of file descriptors that it asks the operating system to monitor, using a mechanism like epoll (Linux), kqueue (OSX), event ports (Solaris), or IOCP.aspx) (Windows). These file descriptors correspond to network sockets, any files it is watching, and so on. When the operating system says that one of these file descriptors is ready, the Event Loop translates it to the appropriate event and invokes the callback(s) associated with that event.

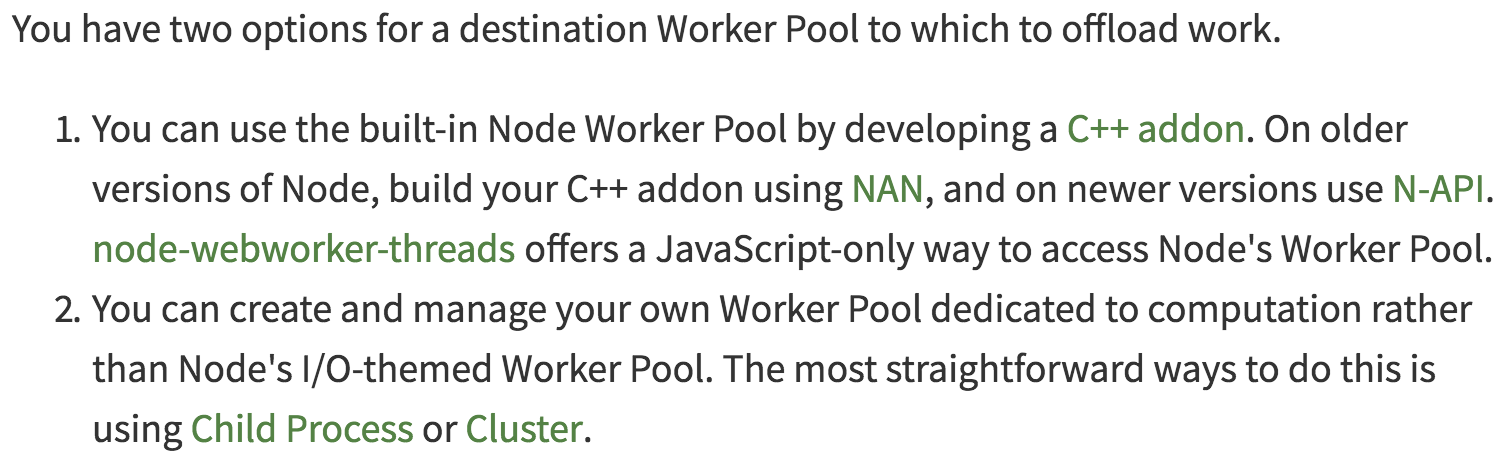
#### 6.1.1.2 Worker Loop

Node's Worker Pool is implemented in libuv (docs), which exposes a general task submission API. Node uses the Worker Pool to handle "expensive" tasks. This includes I/O for which an operating system does not provide a non-blocking version, as well as particularly CPU-intensive tasks.

In contrast, the Worker Pool uses a real queue whose entries are tasks to be processed. A Worker pops a task from this queue and works on it, and when finished the Worker raises an "At least one task is finished" event for the Event Loop.

Note:

the Event Loop should orchestrate client requests, not fulfill them itself. For a complicated task, move the work off of the Event Loop onto a Worker Pool.

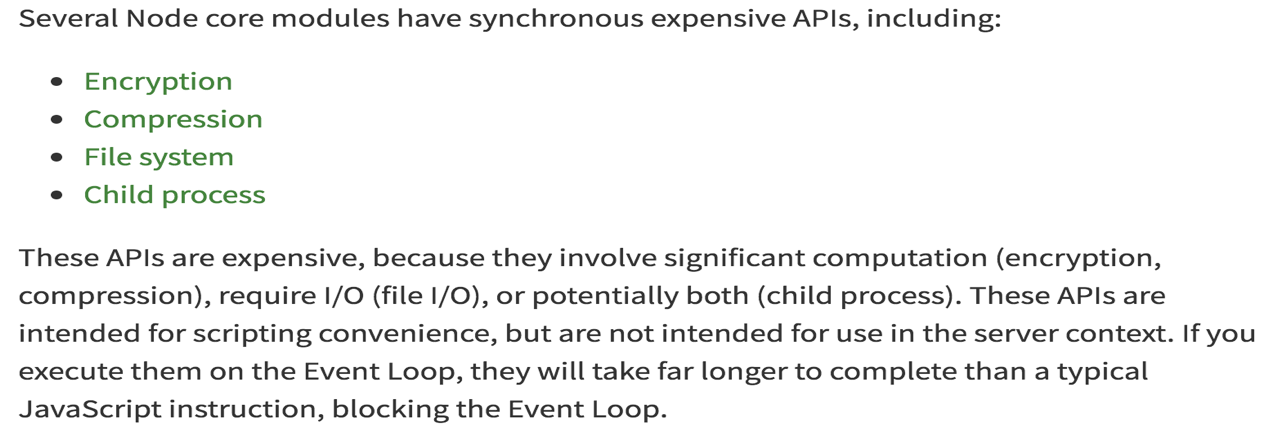


### 6.1.2 Avoidance of blocking event-loop

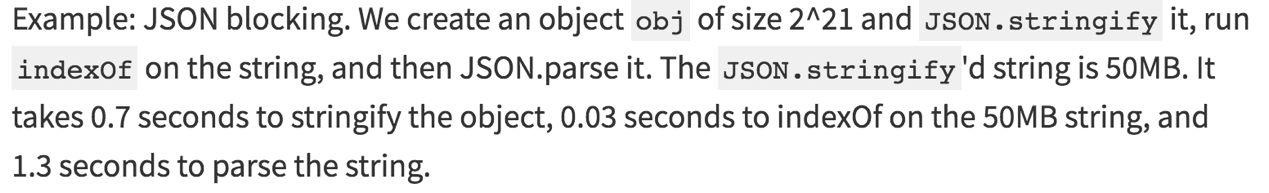
#### 6.1.2.1 REDoS

见7.1

#### 6.1.2.2 Avoidance of invoking core modules synchronously



#### 6.1.2.3 avoidance of json-data



### 6.1.3 Event-loop



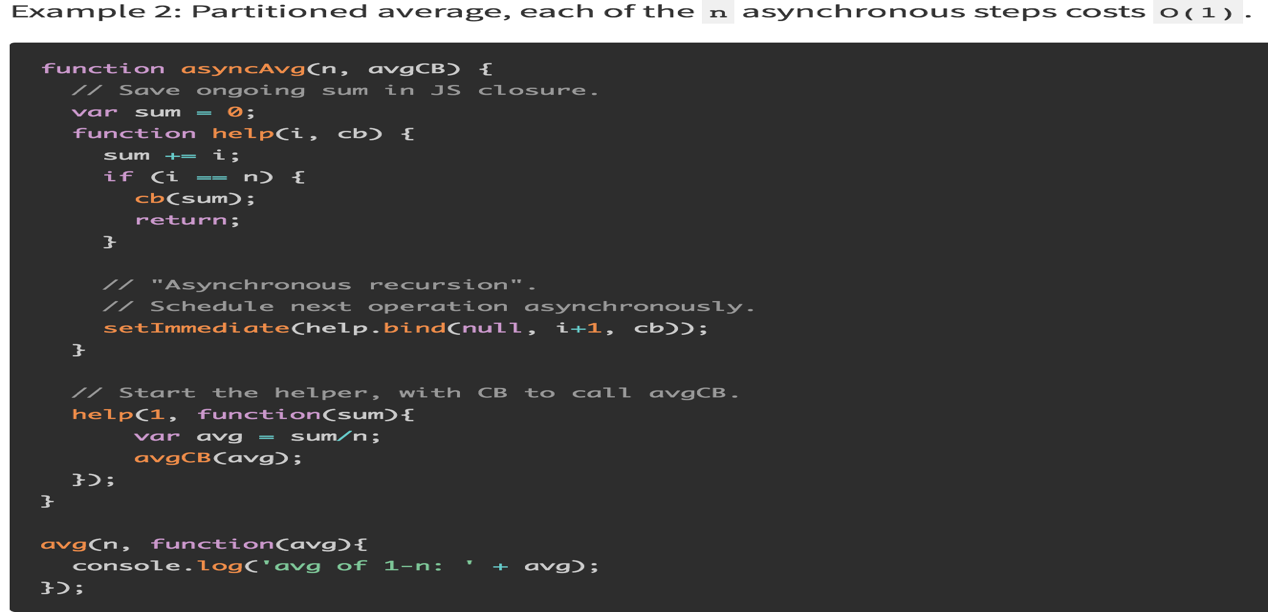
setImmediate()的定时器在check phase,

执行顺序是：

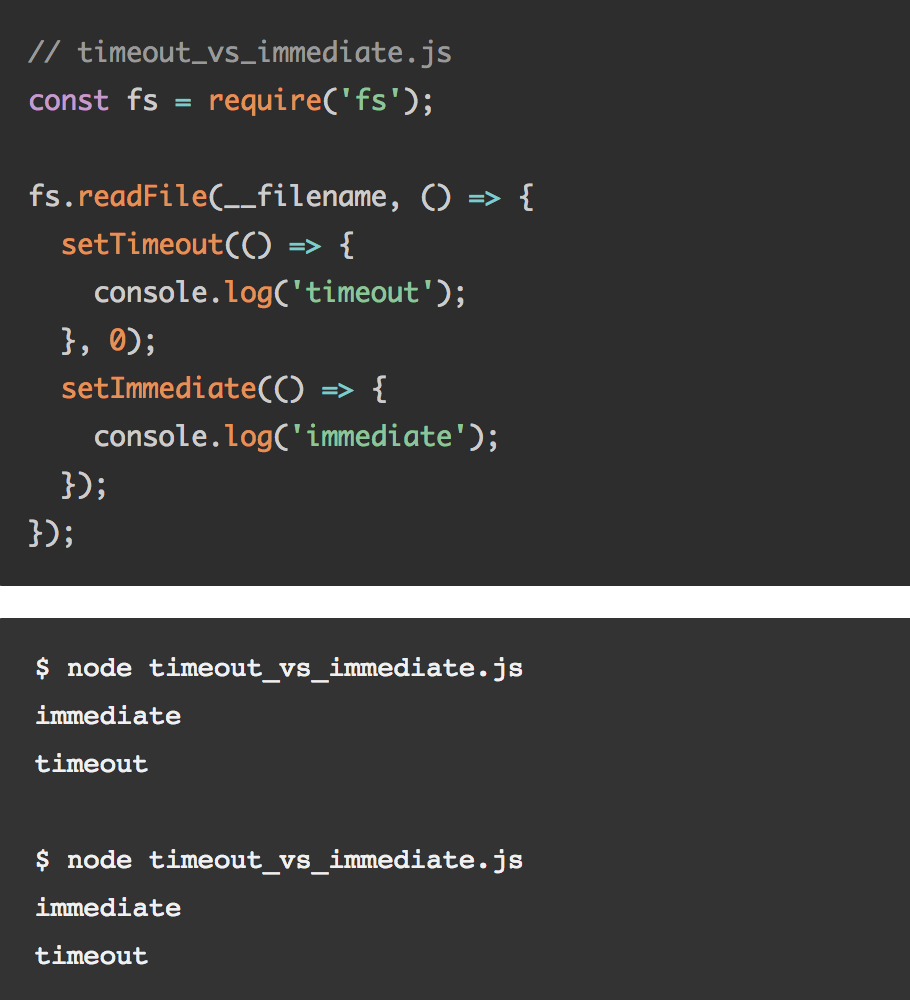
poll phase： 执行注册的callback，触发timers phase（主要包括setTimeout, setInterval）

setImmediate: when poll phase finished， execute functions registered by setImmediate

process.nextTick(): 创建子堆栈，当父堆栈完成时，执行。在timers phase和check phase之前，poll phase之后执行



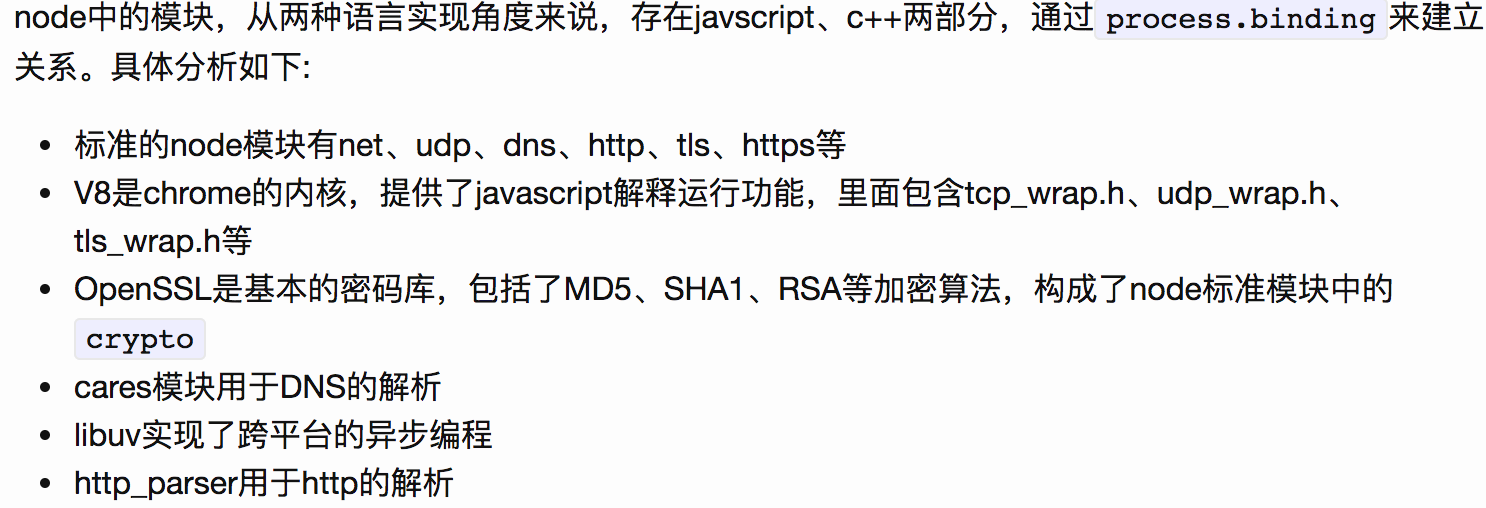
The main advantage to using setImmediate() over setTimeout() is setImmediate() will always be executed before any timers if scheduled within an I/O cycle, independently of how many timers are present.



## 6.2 Implementation of Core

### 6.2.1 Introduction

nodejs library is divided into to parts: javascript modules and c++ modules. In javascript, c++ library is imported through function process.binding().



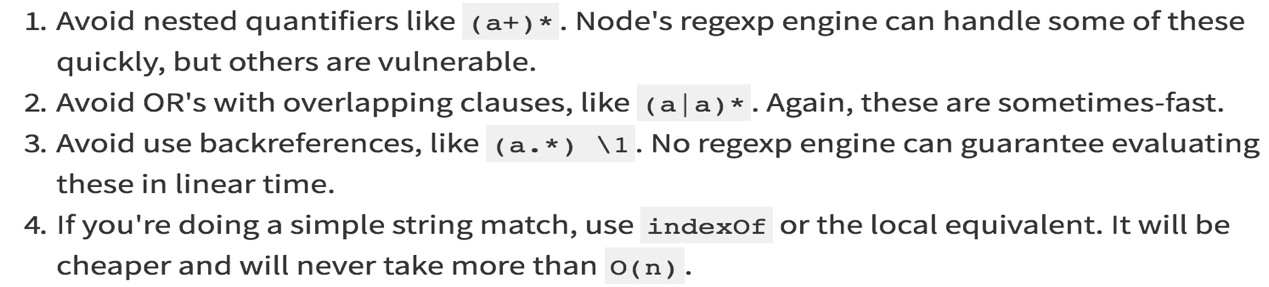
### 6.2.1 process.binding

# 7.Pitfall & hazard

## 7.1 REDoS (regular expression denial of service)

reference: <https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS>

<https://nodejs.org/en/docs/guides/dont-block-the-event-loop/>



## 7.2 fork bomb (rabbit virus or wabbit)

<https://en.wikipedia.org/wiki/Fork_bomb>

细节误区

对于函数参数使用？标记，获取后，若调用时未传递，则为undefined，

但是if() 和？：两者不同

Promise的reject，在调用时需要添加try catch

Try{} catch{}

异常抛出

# 8. nodejs内置库

## 8.1 crypto

### 8.1.1 简介

不可逆的hash算法可以保证登陆密码的安全；通过非对称的加密算法，可以保证数据存储的安全性；通过数字签名，可以验证数据在传输过程中是否被篡改。主要提供了加密、解密、签名、验证等功能。Crypto利用OpenSSL库来实现它的加密技术，它提供OpenSSL中的一系列哈希方法，包括hmac、cipher、decipher、签名和验证等方法的封装。

### 8.1.2 hash

#### 8.1.2.1 introduction

哈希算法，是指将任意长度的二进制值映射为较短的固定长度的二进制值，这个小的二进制值称为哈希值。