NodeJS介绍

SY1606117 李岳檑
SY1606118 温元祯
SY1606413 谭伟良
PT1600283 王春柳

CONTENT

1 Node简介

2 Node特点

3 项目目标

4 工作计划

5 本周工作

6 需求初步

Node简介

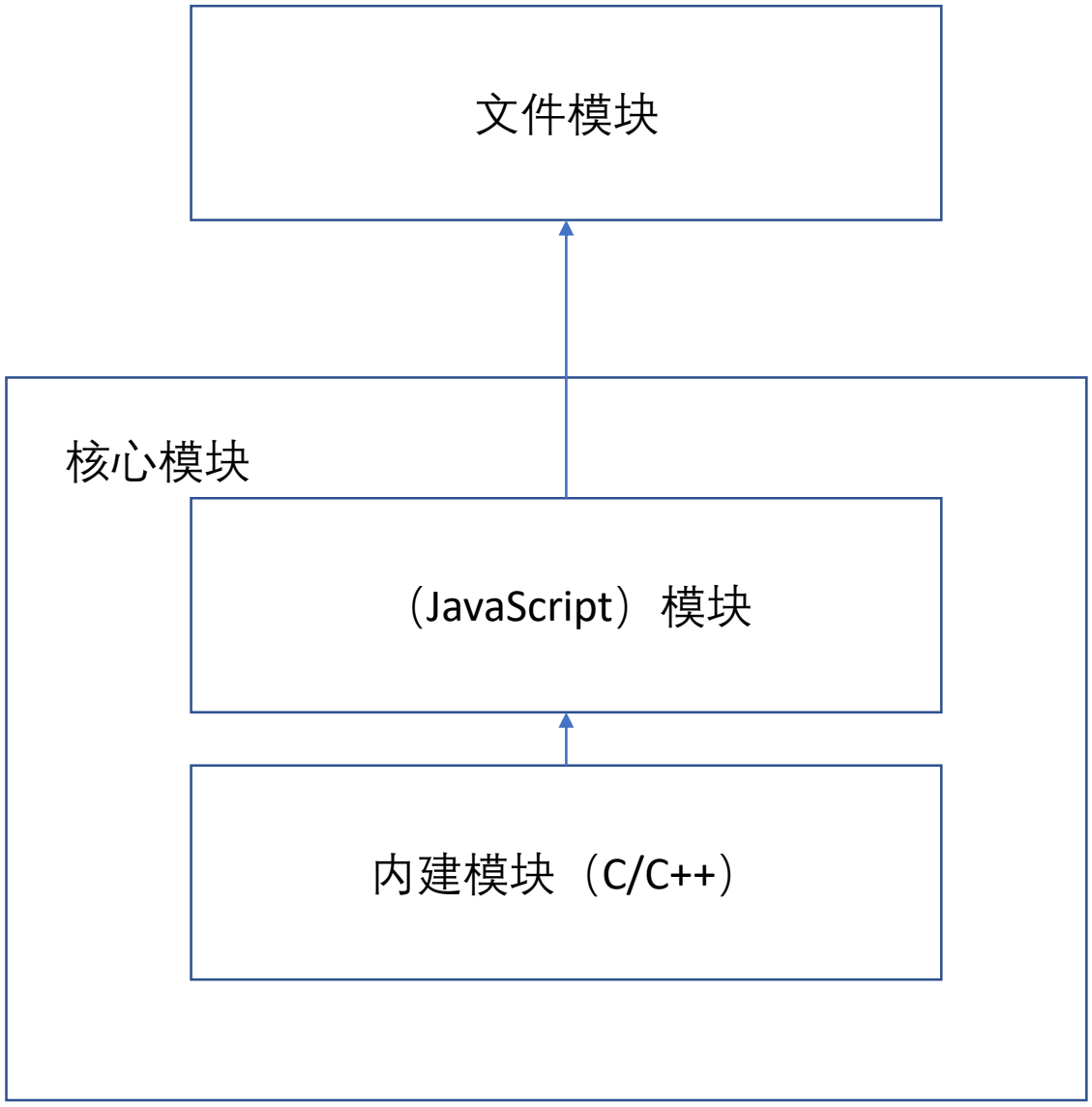
Node简介

Nodejs不是一个js应用、而是一个js运行平台。其最核心的模块由C++编写而成。

Node简介

Node自身非常简单，通过通信协议来组织许多node，非常容易通过扩展来达成构建大型网络应用的目的。

Node简介

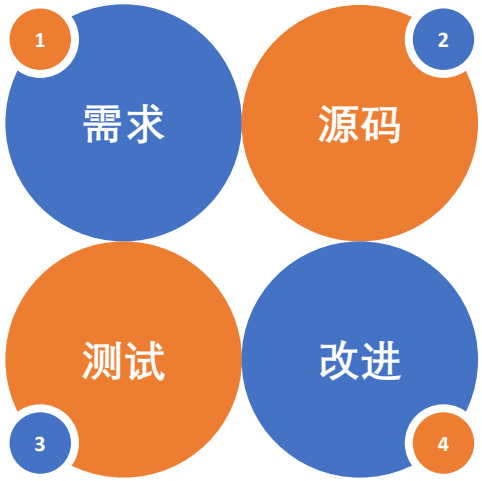


Node特点

Node特点

项目目标

项目目标



需求

从node功能出发，编写其需求规格说明书。

源码

阅读node核心模块源码，从代码理解项目实现细节

测试

为模块制定调优策略编写测试规格说明书并进行测试

改进

对node核心模块中的js模块进行改进，有必要可以改成C++内建模块

4 工作计划

工作计划

- 通过邮件微信群及时线上沟通
- 每周2-3次见面讨论，分配任务，解决问题
- 根据项目计划书来进行任务
- 根据组内成员的情况合理分配

本周工作内容

本周工作内容

- 评审B,C组工作，根据课上及讨论区的评审改进上周工作
- 安装node，配置环境
- 学习node，做一些简单的demo出来
- 完成初步的需求分析

需求初步

- 功能需求
- 非功能需求
- 应用场景

功能需求

- 模块和包
- 网络通信
- 文件系统

模块和包

Node.js摒弃了js在全局定义上的缺点，引入了模块和包机制，使得用户完全不必考虑变量污染、命名空间等问题。

- 创建和加载模块

  在Node.js中创建一个模块非常简单，因为一个文件就是一个模块。创建模块时需要一个对外公开的接口exports，外部加载模块时利用require获取该模块的接口。

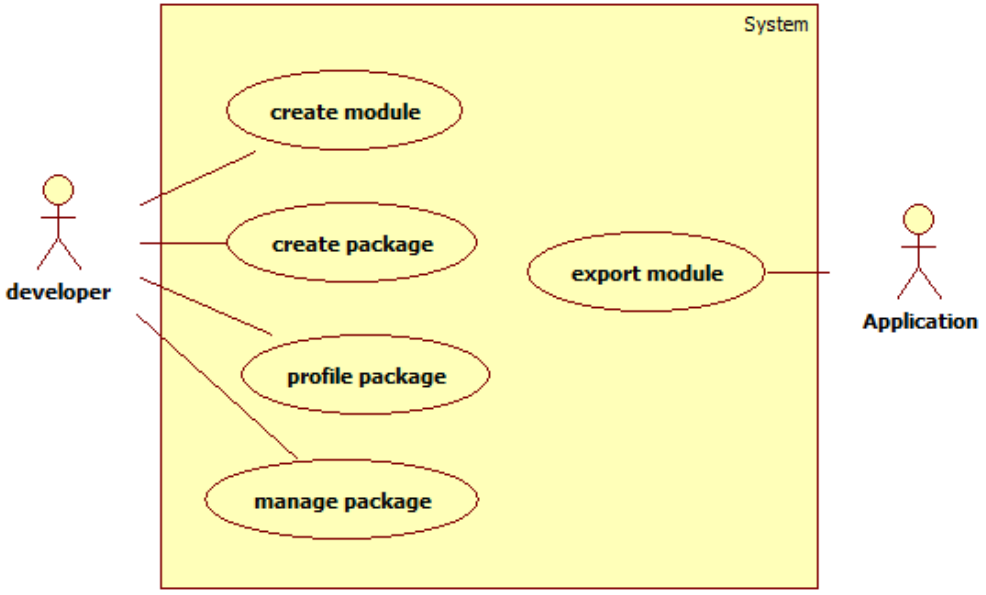
- 创建和调用包

  创建包时需要一个一有模块，或者在创建包之后在包内创建一个新的模块。调用时依旧使用require方法来获取包的接口。

- 管理**包**

  Npm是Node.js发布的包管理工具，用于包的发布、传播、依赖控制。Npm提供了命令行工具，使用户方便地下载、安装、升级、删除包，也可以让开发者发布并维护包。

模块和包



模块和包机制用例图

网络通信

在使用PHP等脚本语言开发网站时，必须先搭建一个Apache之类的HTTP服务器，然后通过HTTP服务器的模块加载或CGI调用，才能将PHP脚本执行结果呈现给用户。这样打破了过去js只能在浏览器运行的局面，前后端实现了统一。Node提供了net、dgram、http、https这4个模块，分别用于处理TCP、UDP、HTTP、HTTPS，适用于服务端和客户端。

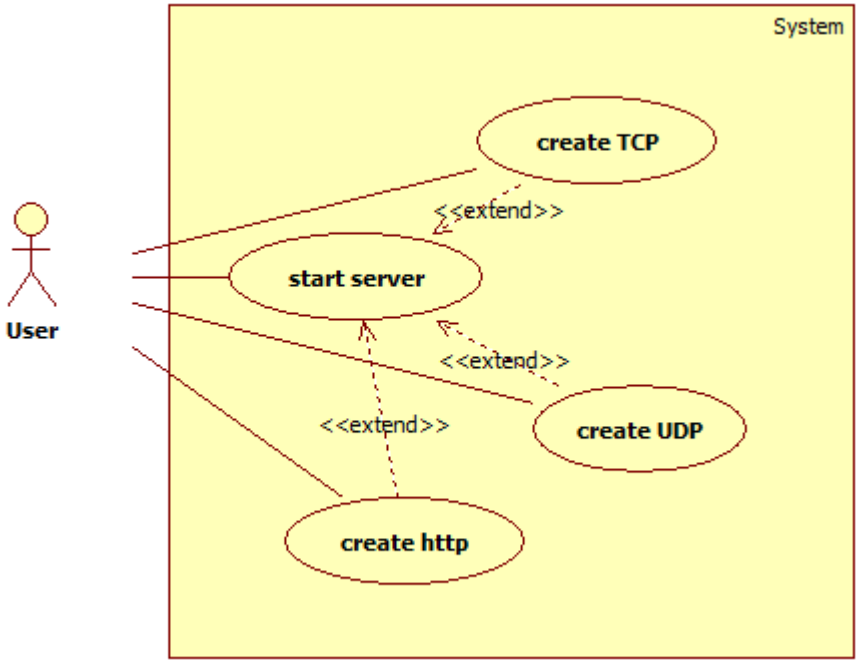
构建HTTP服务

通过http. createServer()方法即可实现一个HTTP服务器。对于TCP连接的读操作，http模块将其封装为ServerRequest对象。HTTP的相应机制封装了对底层连接的写操作，将其看成一个可写的流对象。对于报头的写入，分为setHeader()和writeHead()两个步骤。报文体部分则是调用res.write()和res.end()方法实现。客户端的实现同服务器端。

构建WebSocket服务

首先new WebSocket()新建一个WebSocket,然后用socket.onopen()方法在浏览器与服务器端创建WebSocket协议请求，并规定向服务器端发送数据的时间。同时可以通过onmessage()方法接收服务器端传来的数据。

网络通信



网络通信用例图

文件系统

文件I/O 是由简单封装的标准 POSIX 函数提供的，提供了文件的读取、写入、更名、删除等文件操作。

文件操作

Fs提供文件的读取、写入、更名、删除、遍历目录、链接等文件操作。每种操作都有异步和同步两种形式。

Buffer

Buffer类被引入作为 Node.js API 的一部分，使其可以在 TCP 流和文件系统操作等场景中处理二进制数据流。

非功能需求

- 兼容性
- 高效性
- 容错性
- 可扩展性

兼容性

浏览器兼容

Node.js不运行在浏览器中，所以不存在JavaScript的浏览器兼容性问题，可以放心使用JavaScript语言的所有特性。

操作系统兼容

Node.js兼容Windows、Linux、Mac OS X操作系统。

其它非功能需求

高效性

Node.js非阻塞模式的IO处理给Node.js带来在相对低系统资源耗用下的高性能与出众的负载能力，非常适合用作依赖其它IO资源的中间层服务。

容错性

JavaScript提供uncaughtException容错机制。早期Node.js 中提出Domain 机制，用于隔离错误域。但是Nodejs在容错方面仍存在一些不足。例如，当异步回调中出现异常，不管相应的 error 的事件有没有被订阅，整个进程都会挂掉。

可扩展性

Node.js支持C/C++扩展模块的编写。用户可根据自己的需求编写相应的功能模块放入核心模块中，以提高效率。

应用场景

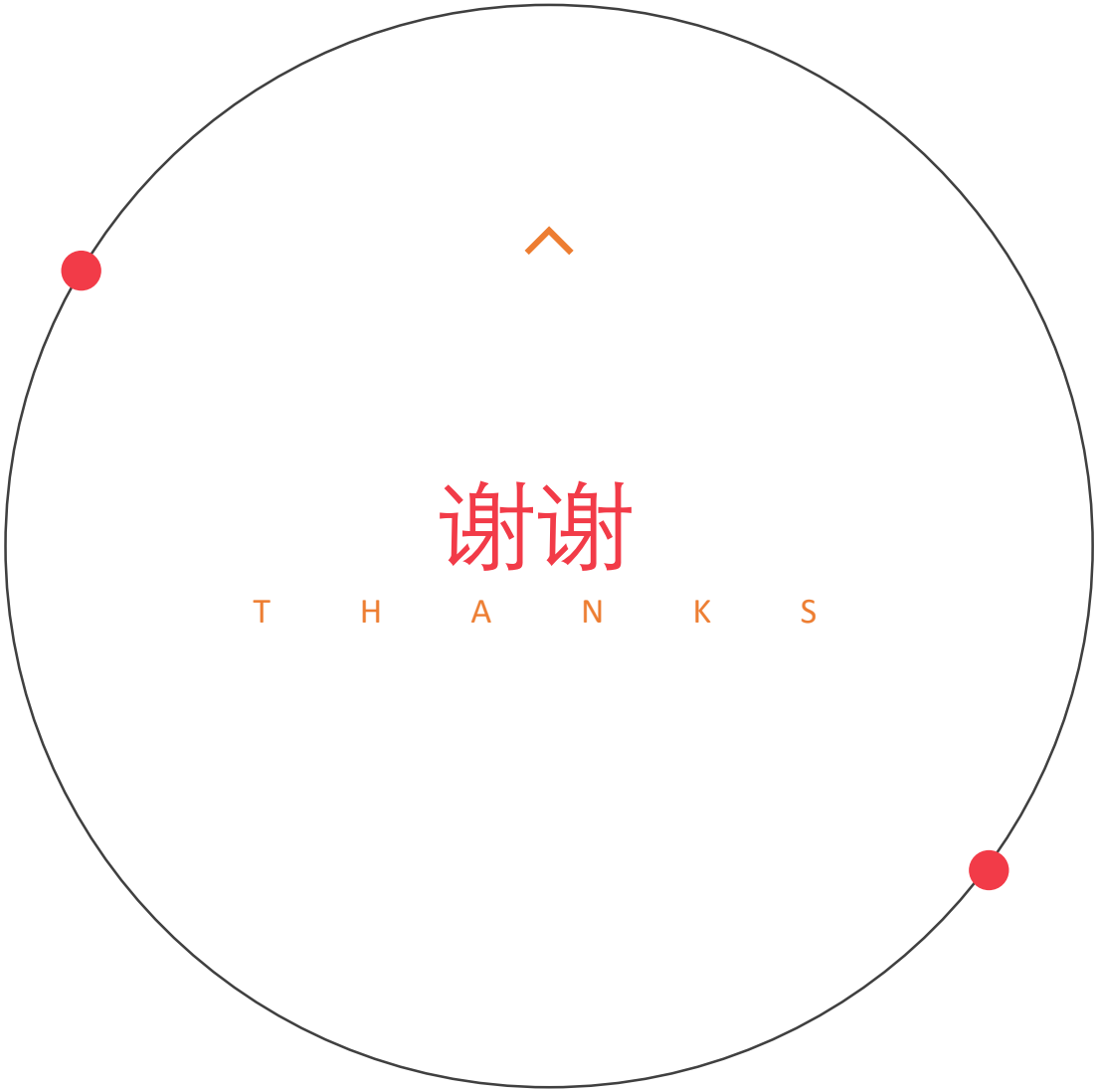
I/O密集型

Node面向网络且擅长并行I/O，能够有效地组织起更多的硬件资源，从而提供更多好的服务。I/0密集的优势主要在于Node利用事件循环的处理能力，而不是启动每一个线程为每一个请求服务，资源占用极少。

CPU密集型

Node.js的引擎是V8，实际上V8的执行效率是非常高的。关于CPU密集型应用，Node的异步I/O已经解决了在单线程上CPU与I/O之间阻塞无法重叠利用的问题，I/O阻塞造成的性能浪费远比CPU的影响小。

对于长时间的运算或者纯运算的场景，Node.js可以通过编写C/C++扩展的方式更高效的利用CPU,将一些V8不能做到性能极致的地方通过C/C++来实现。


谢谢
T H A N K S