|  |
| --- |
| buaa_1 |
| 改进设计文档 |
| 基于Node.js的分析与应用 |
|  |
|  |
| 北京航空航天大学 |
| 2017-05-10 |

版本变更历史

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本 | 提交日期 | 主要编制人 | 内容说明 | 审核人 | 版本说明 |
| V1.0.0 | 2017/5/6 | 温元祯 |  |  | 初稿 |
| V1.0.1 | 2017/5/10 | 王春柳 | 增加说明图 |  | 一稿 |
|  | 2017/5/12 | 温元祯 | 修改流程图 |  |  |
| V1.0.2 | 2017/5/12 | 王春柳 | 修改说明图 |  | 二稿 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目录

[1. 引言 3](#_Toc482209196)

[1.1 目的 3](#_Toc482209197)

[1.2 系统概述 3](#_Toc482209198)

[1.3 文档概述 4](#_Toc482209199)

[2. 基本框架搭建 4](#_Toc482209200)

[2.1 搭建Node.js 4](#_Toc482209201)

[2.2 安装npm 4](#_Toc482209202)

[2.3 安装mocha 4](#_Toc482209203)

[3. 程序设计说明 5](#_Toc482209204)

[3.1 程序1设计说明 5](#_Toc482209205)

[3.1.1 功能描述 5](#_Toc482209206)

[3.1.2 输入项目 5](#_Toc482209207)

[3.1.3 输出项目 5](#_Toc482209208)

[3.1.4 程序逻辑 5](#_Toc482209209)

[3.1.5 测试要点 6](#_Toc482209210)

[3.2 程序2设计说明 6](#_Toc482209211)

[3.2.1 功能 6](#_Toc482209212)

[3.2.2 输入项目 6](#_Toc482209213)

[3.2.3 输出项目 6](#_Toc482209214)

[3.2.4 程序逻辑 6](#_Toc482209215)

[3.2.5 测试要点 7](#_Toc482209216)

[4. 数据字典 8](#_Toc482209217)

[5. 程序接口 8](#_Toc482209218)

1. **引言**
   1. **目的**

在项目改进的过程中，对项目改进的方案、实现的目标进行介绍，以指导开发和测试工作的进行。

* 1. **系统概述**

Node.js是一个可以让JavaScript运行在服务器端的平台，它可以让JavaScript脱离浏览器的束缚运行在一般的服务器环境下。

Node.js在诞生之初就充分考虑了在事实响应、超大规模数据要求下架构的可扩展性，实现了诸如文件系统、模块、包、操作系统API、网络通信等Core JavaScript没有或者不完善的功能。历史上将JavaScript移植到浏览器外的计划不止一个，但Node.js是最出色的一个。Node.js的JavaScript引擎是V8,来自Google Chrome项目。V8号称是目前世界上最快的JavaScript引擎，它的即时编译执行速度已经快到了接近本地代码的执行速度。Node.js不运行在浏览器中，所以也不存在JavaScript的浏览器兼容性问题。

Node.js内建了HTTP服务器支持，用户可以轻松地实现一个网站和服务器的组合。这个服务器不仅可以用来调试代码，而且它本身就可以部署到产品环境。Node.js还可以部署到非网络应用的环境下，比如一个命令行工具。Node.js还可以调用C/C++的代码，这样可以充分利用已有的诸多函数库，也可以将对性能要求非常高的部分用C/C++来实现。

Node.js用异步式I/O和事件驱动代替多线程，带来了可观的性能提升。Node.js除了使用V8作为JavaScript引擎之外，还使用了高效的libev和libeio库支持事件驱动和异步式I/O。图1是Node.js架构的示意图。



图 1 Node.js架构图

* 1. **文档概述**

本文档是在Node.js的基础上，介绍了对Node.js的过载保护模块做出的改进。

1. 以流程图和rucm图的形式给出改进模块的设计和实现方案；
2. 描述了模块内部的数据定义和数据结构；
3. 给出初步的测试方案和测试的rucm说明。
4. **基本框架搭建**
   1. 搭建Node.js

Node.js是改进模块的基本框架，因此要先搭建Node.js。安装过程很简单，只需在官网上下载安装包。Windows系统下按照下一步的提示一步步安装即可。Linux环境下可在github上获取源码，之后编译即可。或直接用sudo apt-get install nodejs来安装。CentOS安装过程与Linux类似。

* 1. 安装npm

Npm是一个包管理工具，Node.js中也使用该工具，因此需要安装npm。Node.j中集成了npm，因此在安装Node.js时，npm已经安装好了。也可以单独安装npm。

* 1. 安装mocha

Mocha是现在最流行的js测试框架之一。在测试改进模块是否能正常运行时用到该框架，因此需要安装。Windows系统下安装，用git下载zip压缩包，解压后进入mocha-demo目录，安装依赖npm install，之后在全局安装mocha, npm install –global mocha

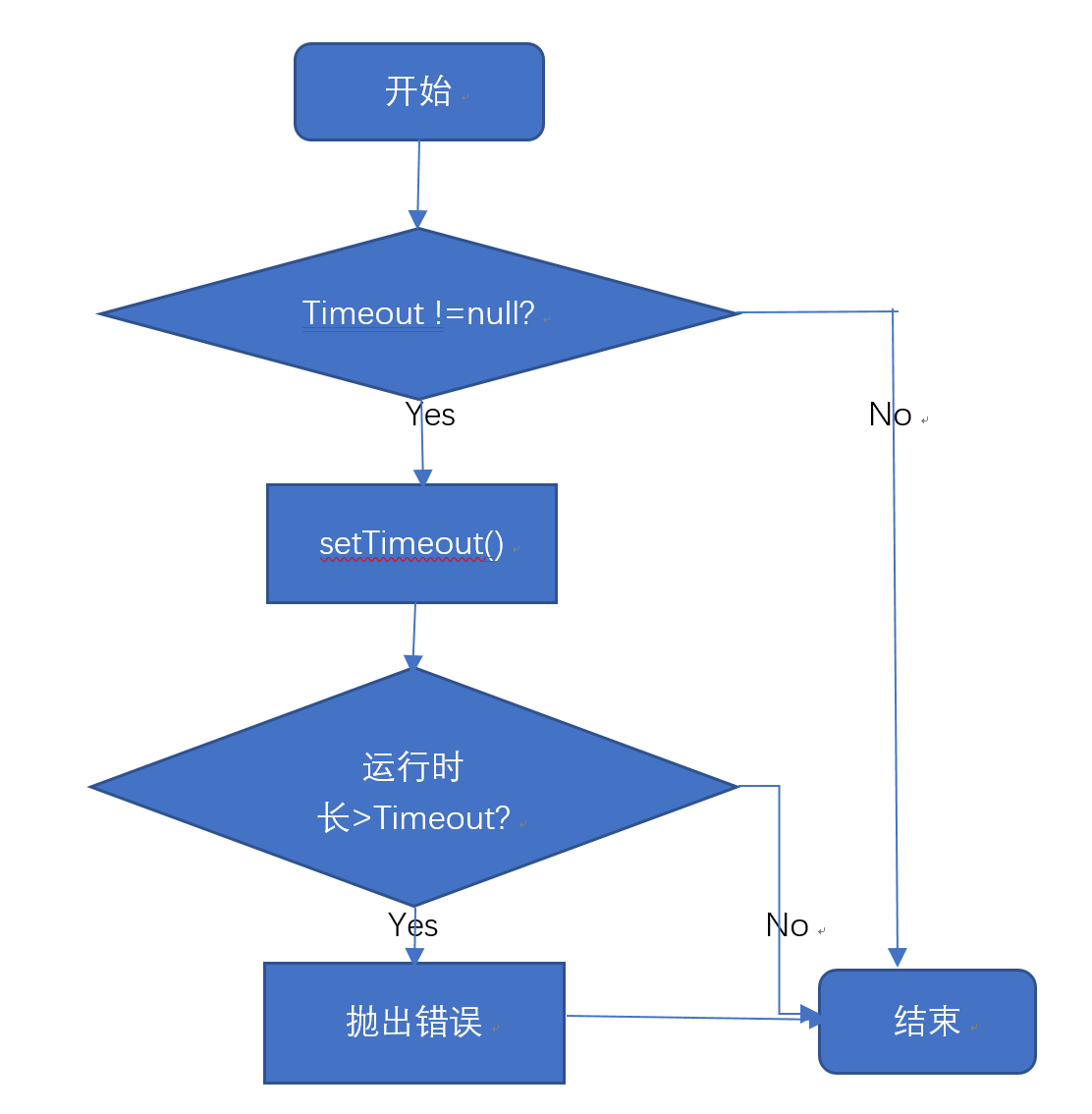
1. **程序设计说明**
   1. 程序1设计说明
      1. 功能描述

超时处理

* + 1. 输入项目

Timeout:超时时长

* + 1. 输出项目

抛出错误/正常执行

* + 1. 程序逻辑

图2 超时处理模块流程图

如图2所示,模块首先判断是否传入timeout参数，如果没有传入，默认值为Null，该模块结束。如果传入该参数值，调用setTimeout()函数来判断运行时长是否超过了设定最大时长。如果超过，抛出错误，如果没有超过，该模块结束，下图3为整个超时处理模块的rucm说明图。

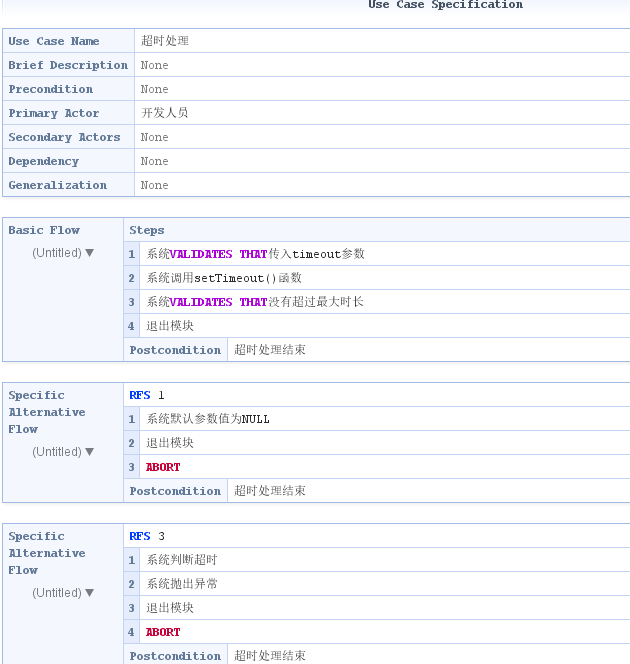


图3 超时处理模块说明（rucm）

* + 1. 测试要点
* 超时是是否会抛出错误
* 不超时是否能正常运行



图4 测试超时处理模块流程说明（rucm）

* 1. 程序2设计说明
     1. 功能

异步并发数量控制

* + 1. 输入项目
* Limit：并发数的限制额
* queueLength：队列的长度
* Disabled：是否禁用限流
  + 1. 输出项目

抛出错误/抛出事件（full等）/正常执行

* + 1. 程序逻辑

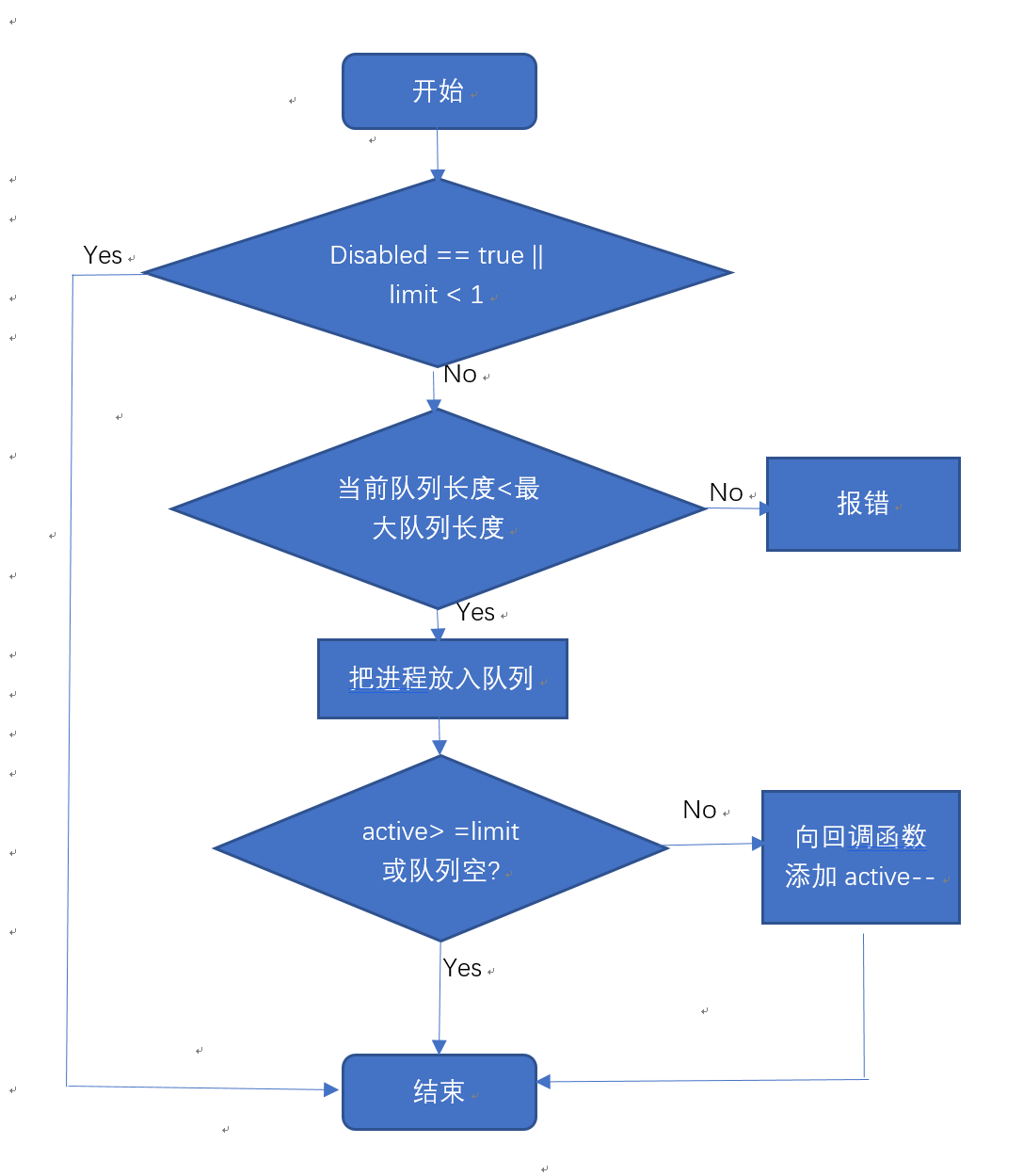


图5 异步并发数量控制模块流程图

如图3所示，首先判断传入的Disabled是否为true，或limit值是否等于0。若满足其中一个条件则不进行异步并发控制。如果均不满足，则判断当前队列中进程数是否超过队列的最大长度，如果超过的话，抛出错误。如果没有超过，就把当前进程放入队列中。然后再次判断limit是否大于1，若大于1,则证明队列中有进程，emit一个full事件。之后继续判断active是否超过limit或队列是否为空,若满足其中一个，则结束当前模块，进入下一个进程的Push。若不满足，则在回调函数中，将active的值减1。下图6为异步并发数量控制的rucm说明图







图6 异步并发数量控制说明（rucm）

* + 1. 测试要点
* Disabled为True时，是否禁止了限流
* 当前队列长度超过最大队列长度时是否会抛出错误
* Limit大于1时是否会触发full事件

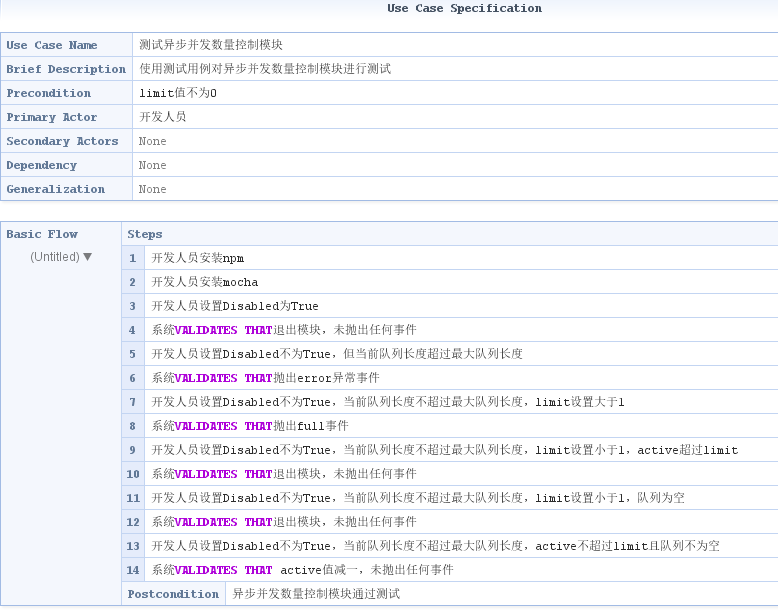


图7a 测试异步并发数量控制模块说明（rucm）



图7b 测试异步并发数量控制模块说明（rucm）

1. **数据字典**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 默认值 | 允许为空 | 备注 |
| Limit |  | 无 | 不允许 | 异步并发的最大值 |
| active |  | 0 | 允许 | 当前活跃的进程数 |
| Disabled | boolean | false | 允许 | 是否禁用异步并发控制 |
| timeout |  | null | 允许 | 超时的最大时长 |
| QueueLength |  | Limit | 不允许 | 队列的长度 |
| Timer |  | null | 允许 | 用于超时控制 |
| Called | boolean | false | 允许 | 用于判断是否超时 |

1. **程序接口**

模块：asyncLimit

方法：constructor(),push()