计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(6):226−230 [doi: 10.15888/j.cnki.csa.007966]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

针对载荷/载荷舱监控的固定码率数据非阻塞模式实时解析与展示方法①

李雪松, 李子扬, 朱家佳, 窦　帅, 杨　光, 陈宾宾, 李传荣

(中国科学院 空天信息创新研究院, 北京 100094)
通讯作者: 李传荣, E-mail: crli@aoe.ac.cn

摘　要: 临近空间探测科学实验是以浮空器、无人机等平台为手段, 获取多尺度、多层次、多类型的探测数据, 为临近空间认知体系构建提供支撑. 因为临近空间探测的复杂性和探测数据的多样性及重大的应用价值, 所以建立临近空间探测数据实时可视化展示, 有利于提升临近空间探测任务的安全、稳定开展. 由于传统的同步阻塞 I/O 处理模式在对数据流进行处理时, 搭建异步事件驱动架构复杂, 数据处理效率难以提升, 因此本文设计了一种基于 NodeJS+ElectronJS+EchartsJS 的适用于临近空间探测科学实验固定码率数据非阻塞模式实时解析与展示方法. 该方法采用事件驱动、异步编程的实现方式, 实现了临近空间探测科学试验固定码率数据的实时数据解码以及下传数据高效准确的可视化展示.

关键词: Javascript; NodeJS; ElectronJS; 非阻塞; 线程池; 异步; 固定码率; 临近空间

引用格式: 李雪松,李子扬,朱家佳,窦帅,杨光,陈宾宾,李传荣.针对载荷/载荷舱监控的固定码率数据非阻塞模式实时解析与展示方法.计算机系统应用,2021,30(6):226–230. http://www.c-s-a.org.cn/1003-3254/7966.html

Real Time Analysis and Display Method of CBR Data Non-Blocking Mode for Payloads/Payloads' Chamber Monitoring

LI Xue-Song, LI Zi-Yang, ZHU Jia-Jia, DOU Shuai, YANG Guang, CHEN Bin-Bin, LI Chuan-Rong

(Aerospace Information Research Institute, Chinese Academy of Sciences, Beijing 100094, China)

Abstract: The scientific experiment of near-space exploration is to obtain multi-scale, multi-level, and multi-type detection data by aerostats, UAVs, and other platforms, which supports the construction of near-space cognitive systems. In view of the complexity of near-space exploration, the diversity of detection data, and the great application value, the real-time visualized display of near-space detection data is conducive to the safe and stable development of near-space exploration. During the processing of data streams, the traditional synchronous-blocking I/O mode is difficult in constructing the driver architecture of asynchronous events and thus the data processing efficiency is low. For this reason, this study introduces the real-time analysis and display method of non-blocking mode for Constant-Bit-Rate (CBR) data of near-space exploration based on NodeJS+ElectronJS+EchartsJS. This event driven method is of asynchronous programming and can achieve real-time decoding of CBR data in the scientific experiment of near-space exploration, as well as efficient and accurate visualized display of the downward transmission data.

Key words: JavaScript; NodeJS; ElectronJS; non-blocking; thread-pool; asynchronous; Constant-Bit-Rate (CBR); near-space

① 基金项目: 中国科学院战略性先导科技专项 (A 类)(XDA17040303)
Foundation item: Category A Strategic Priority Research Program of Chinese Academy of Sciences (XDA17040303)
收稿时间: 2020-10-10; 修改时间: 2020-11-16; 采用时间: 2020-12-01; csa 在线出版时间: 2021-06-01

1 引言

鉴于临近空间探测的复杂性和探测数据的多样性及重大的应用价值, 中国科学院设立了临近空间探测科学实验项目. 临近空间探测数据实时可视化展示是临近空间探测项目的重要组成部分, 通过实时监控探测数据的变化, 为飞行任务决策提供支撑, 有利于提升临近空间探测任务安全、稳定的开展. JavaScript 是目前最流行的客户端编程语言, 自 2016 年起就有逐渐取代 Java 和 PHP 的趋势[1]. NodeJS 本质上是采用了谷歌浏览器 V8 引擎的 JavaScript 运行环境, 其非阻塞模式的 I/O 处理带来了相对低系统资源耗下的高性能与出众的负载能力, 非常适合用作依赖其他 I/O 资源的中间服务. 因此, NodeJS 被认为是数据密集型分布式部署环境下的实时应用系统的完美解决方案[2,3]. 载荷/载荷舱监控系统实现需要考虑在有限带宽的高比特率条件下系统与临空飞行器实时通信, 同时保障数据展示的实效性. 使用 UDP 无状态协议连接, 使得资源消耗小, 数据传递快, 同时可以分配到不同机器处理, 做到完全无状态的横向处理[4]. 临近空间探测试验是高寒野外环境下开展, 其具有网络可靠性不高、数据流大的特点, 传统的同步阻塞 I/O 处理模式在对数据流进行处理时, 数据下传效率与数据处理能力很难同时得到效率上的提升, 线程很容易在做数据处理过程中阻塞数据传输的进行, 造成系统内资源的浪费, 进而导致在过程中无法处理其他任务, 例如数据块写入文件, 或者读取其他数据[5,6]. 因此需要设计一种改进的数据解析与展示方法, 在不阻塞数据下传的同时提高软件的数据处理能力.

2 基于非阻塞模式的数据解析与展示方法

2.1 NodeJS I/O 与 Java I/O 处理模式比较

理论上在处理计算方面, NodeJS 单条主线程并没有太多优势, 然而如果涉及 IO 密集型任务, 由于 NodeJS 在每个级别强制支持非阻塞 API 规范, 因此能更高效地利用 CPU 和内存资源. 表 1 对比了 NodeJS 与 Java 在 I/O 处理中的特点.

表 1 NodeJS 与 Java I/O 处理模式特点比较[6–11]

I/O模式	特点	优点	缺点
Java I/O阻塞模式	多线程且线程之间相互独立, 客户端与服务端线程间始终连接.	适用于同步通信及错误控制.	CPU利用率低、线程效率低, 手工处理多线程, 通信时线程增加或切换性能降低较快.
Java NIO非阻塞模式	异步事件监听, 缓冲区处理数据流, 通过线程池维护多线程, servlet规范的基本范例包含阻塞.	自动处理多线程, 线程通信不会阻塞, 应用程序开销较小, 高并发.	应用依赖于操作系统的I/O实现, 依赖于阻塞式API, 构建事件驱动模型不容易.
NodeJS I/O非阻塞模式	主应用为单线程, 强制支持非阻塞API规范, 事件驱动, 轻量化, 跨平台, libuv维护线程池, 版本更新迭代快.	自动处理多线程, 不会被阻塞, 适用I/O密集型业务, 可高效地使用CPU和内存资源, 对操作系统依赖小.	多线程模式中, 主线程对应用性能影响较大, 异步非阻塞编程嵌套过于复杂.

近年来不少知名互联网公司包括阿里、腾讯、苏宁, 都有尝试用 NodeJS 代替 Java 应用于自身 I/O 密集型业务场景的成功案例[9].

2.2 NodeJS 异步非阻塞 I/O 处理基本原理

一般情况下, 异步非阻塞 I/O 处理流程是发送方向接收方发送请求后, 不等待响应可以继续其他工作. 接收方收到请求后, 进行 I/O 操作时如果不能返回结果. 其不是等待, 而是立刻返回去做其他工作. 当 I/O 操作完成时, 再将完成状态或结果通知接收方, 接收方再响应发送方[5–15].

异步非阻塞处理是基于应用程序重复调用 I/O 操作轮询线程池来完成[8–11]. NodeJS 使用的是单线程异步非阻塞模型, 对所有的 I/O 都采用异步请求方式, 这种设计的底层依托的是 C++高性能事件驱动 libuv 库, 该驱动库提供了线程池, 事件池, 跨平台, 异步 I/O 处理能力等[1]. 如图 1 所示, NodeJS 的主程序只有一个主线程作为执行栈执行程序代码. 主线程通过 libuv 库对事件队列与系统线程池进行维护, 从事件队列取出相应的事件, 再从线程池中分配一个线程去执行该事件. 主线程会通过 libuv 库不断对事件队列轮询从而检查未执行的事件. 当子线程对事件执行完毕, 主线程执行回调, 子线程被释放归还给线程池. 因此基于 NodeJS 异步非阻塞 I/O 处理操作实际上是通过 libuv 维护的线程池完成的. 由于主线程将所有的事件通过 libuv 分配给了线程池中的子线程, 因此主线程基本只负责事件调度, 没有进行真正的 I/O 操作, 从而实现了异步非阻塞 I/O.

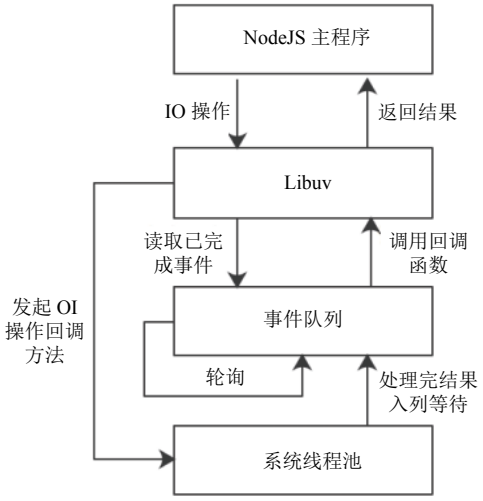

图 1 NodeJS 异步非阻塞 IO 处理流程

2.3 数据实时解析与展示

在临近空间科学探测试验过程中, 载荷/载荷舱通过专用测控链路与地面测控舱通信, 数据通过服务端存储并通过 UDP 广播分发数据. 本文针对载荷/载荷舱下传的单帧数据的特征, 设计了针对载荷/载荷舱监控的异步非阻塞 I/O 模式处理方法[16–20]. 处理流程如图 2 所示.

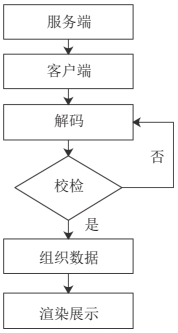

图 2 固定码率数据非阻塞解析与展示流程

客户端通过 UDP 协议与数据发送端建立连接, 并接收数据包数据流. 在该套接字模块中使用了数据报 dgram 模块, dgram 提供了 UDP 数据包实时通信 Socket 的实现方法. 该模块的 API 包括事件和方法两大类, 事件类在 UDP 连接状态发生改变时触发, 包含关闭 (close), 错误 (error), 监听 (listening), 消息 (message); 方法类包含绑定端口主机 (bind), 返回对象地址 (address), 关闭实时通信 (close), 广播发送数据报 (send) 等[21].

程序套接字处理部分通过引用 NodeJS 中的 dgram 模块建立基于 UDP 的网络通信, 模块通信流程如图 3 所示. 该模块可对服务端的发送地址和端口进行绑定, 注册监听事件. 在客户端连接服务端监听状态发生改变时, 客户端可接收服务端发送的数据. 客户端获取数据后,会触发 NodeJS 的异步非阻塞 I/O 处理流程, 主程序会通过 libuv 使用事件队列维护接收的数据[1].

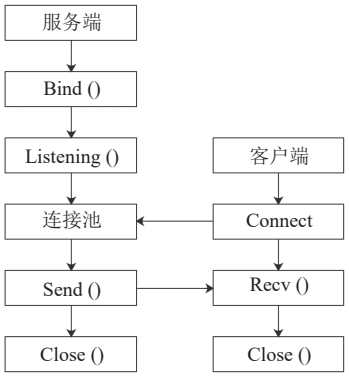

图 3 dgram 模块通信流程

非阻塞 I/O 模式处理方法利用 NodeJS 中的 Buffer 类来创建专用的二进制数据缓存区, 并利用其提供的一系列 API 接口对数据进行操作. 如图 3 中服务端 send() 与客户端 recv() 方法中传递的数据类型. Buffer 类的内存申请并不是通过 NodeJS V8 引擎完成, 而是由 C++实现. 底层是通过 slab 机制对碎片进行管理. 当数据小于 8 KB 时, 通过 new Buffer(8000) 申请一个可用空间为 8000 字节的 slab. 在对服务端发送端口监听的过程中, 可直接对内存进行密集的 I/O 操作. 在这个过程中 slab 机制进行了预先申请和事后分配, 这使得 JavaScript 到操作系统之间没有过多的关于内存申请的系统调用, 提高了数据处理的效率.

非阻塞 I/O 模式处理方法利用 dgram 模块会将接收的二进制字节流封装成 Buffer 类, 根据预先约定的协议, 利用数据处理函数, 对 Buffer 按位解码. 再将十进制明码数据集合逐个赋给原型 JSON 数组. 表 2 是 JSON 数组的主要字段描述.

表 2 解码后的 JSON 数组字段表

字段	描述
id	字段标识, 由0递增
key	预先约定的关键字标识
value	存放解码后的十进制数
length	预先约定的字段所占字节数

非阻塞 I/O 模式处理方法对二进制字节流解码时会读取原型 JSON 数组中的 length 属性, 该属性标识了二进制字节流中每个字段所占字节长度, 通过该长

度对字节流逐位解码. 再将解码后的值返回给原型 JSON 数组. 最后将获得一维数组供后续模块使用.

页面渲染模块在获取一维数组前, 数据流传入验证函数, 数据通过有效性验证后, 再利用 ECharts 组件对数据进行可视化图表展示. 这里 JavaScript 类库 ECharts 其底层依赖轻量级的 Canvas 类库 ZRender, 可以流畅的运行在 PC 和移动设备上, 并兼容当前绝大部分浏览器, 可提供直观的数据可视化图表[22]. 由于第三方开源组件 ECharts 对 DOM 的渲染效率不可控, 同时系统解码速率远高于图形渲染速率. 所以在获取等待展示的一维数组后, 系统采用了响应式处理方法, 使解码模块与 DOM 渲染模块解耦并通过异步数据流构建处理关系. 当渲染模块出现阻塞时, 不影响数据获取与解析, 以保障下传数据的完整和准确性.

载荷/载荷舱监控客户端的开发还引入了由 Github 发布的跨平台桌面应用工具 ElectronJS. 该应用工具封装了谷歌的 Chromium 浏览器引擎作为图形应用界面, 允许基于 NodeJS 的载荷/载荷舱监控客户端通过 ElectronJS 在没有部署 NodeJS 开发环境的任意操作系统中使用 NodeJS API, 从而实现了 NodeJS 网页应用到桌面应用的转换. 图 4 为载荷/载荷舱监控客户端在 MacOS 系统下的软件界面.

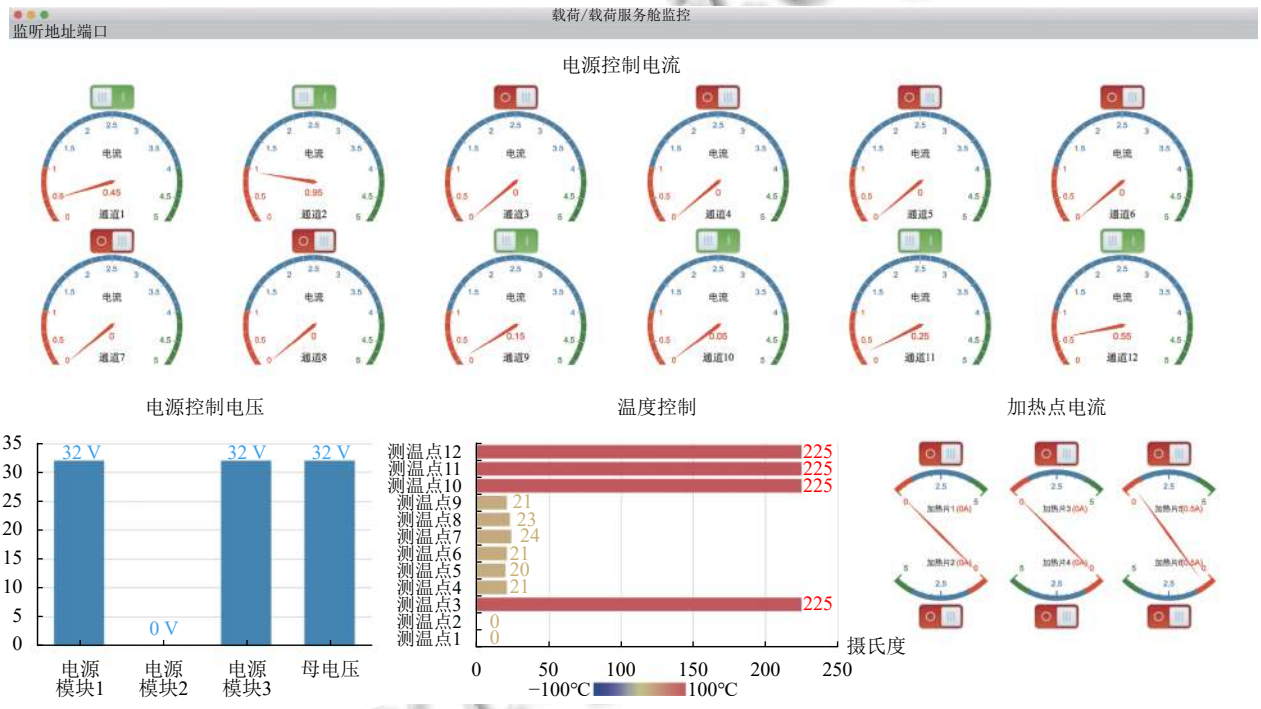

图 4 载荷/载荷舱监控客户端

3 性能测试

性能测试编译工具使用的是 ElectronJS 内置的控制台. 实验机器为 MacBook Pro (Retina, 15-inch, Mid 2015) CPU2.2 GHz intel corei7, 显卡 Intel Iris Pro 1536 MB. 通过在试验机器搭建模拟服务, 通过 UDP 协议连接客户端并发送数据, 同时在数据解析模块与 Echarts 数据渲染模块中设置计时函数, 对软件解码及渲染的能力进行测试统计.

如图 5 所示, 服务端以 22 B/s 的速率向客户端发送二进制数据流. 数据解析模块完成二进制字节流转换一维数组耗时 0.27 ms, 代入公式 $PA \approx \frac{V}{T}$, 计算解码模块单位时间内的工作效率大约是 80 MB/s. 数据渲染模块对一维数组渲染耗时 72.27 ms, 代入公式 $FPS \approx \frac{1}{T}$, 计算每秒可渲染的帧数大约 14 帧. 使用 MacOS 系统的任务管理器对软件 CPU 使用率监控, CPU 使用率在 3%~5% 之间, 无内存泄露, 满足试验指标要求. 由此可以推断当数据流在 80 MB/s 以下时, 解码模块工作无压力, 当大于 80 MB/s 时由于 EchartsJS 渲染速率较慢, NodeJS 的事件队列长度会随时间增加而增加, 因

此会造成执行栈阻塞与内存泄露.

该系统参加了临空 2019 年 7~9 月在青海省海西蒙古族藏族自治州大柴旦地区的外场试验. 系统通过 Electron-Package 打包为 EXE 可执行文件, 并部署在 CPU2.5 GHz intel corei7 Windows10 系统的移动工作站中, 数据传输率为 100 Mb/s. 在试验期间系统运行 34 小时 19 分, 获取、解析并存储数据 12 万包, 各项功能工作状态稳定, 为各试验载荷及载荷舱系统的监视与控制提供了可靠的数据支持.

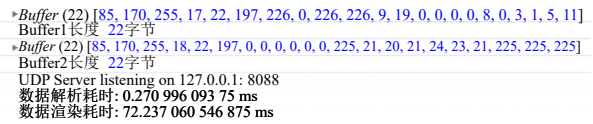

图 5 模拟数据测试

4 总结

本文使用 NodeJS+ElectronJS+EchartsJS 架构, 采用固定码率数据非阻塞模式实时解析与展示方法, 实现了对临空探测试验载荷/载荷舱的实时监控, 实时解析并展示数据理论可达 80 MB/s. 由于利用封装好的 libuv 库维护线程池, 因此相对于手动维护多线程的方法能有效地节省开发成本, 在保障下传数据准确性的同时, 能有效利用 CPU 和内存资源. 经过临空野外试验的验证, 该解析与展示方法适用于资源受限环境下的野外科学试验. 在需要 I/O 密集型场景下进行可视化数据展示架构中, 本模式非常值得推广.

参考文献

1 Ciszewski B. Node. js Performance: How Your Web App Can Benefit from Node. Js? https://www.netguru.co/blog/nodejs-performance-web-application-benefit. [2020-09-21].

2 Peterson C, Cook V, Dechev D. Practical progress verification of descriptor-based non-blocking data structures. Proceedings of 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. Rennes, France. 2019. 83–93. [doi: 10.1109/MASCOTS.2019.00019]

3 Kiessling M. The Node Beginner Book. https://www.nodebeginner.org. [2020-09-25].

4 陈冈, 夏火松. 基于异步文件通道的 Java Web 多任务分块文件上传. 河南理工大学学报 (自然科学版), 2015, 34(3): 400–405.

5 李汝佳, 胡婧. 基于 Nodejs 的异步非阻塞服务器研究. 计算机科学与应用, 2013, 3(3): 191–194.

6 贺杰. 非阻塞输入输出在信息交换系统中的应用. 软件导刊, 2011, 10(5): 135–137.

7 许会元, 何利力. NodeJS 的异步非阻塞 I/O 研究. 2014 年全国工业控制计算机技术年会论文集. 重庆, 中国. 2014. 151–156.

8 龚黔芬. 解析 Java 中的阻塞 I/O 与非阻塞 I/O 控制. 重庆文理学院学报 (自然科学版), 2006, 5(1): 18–20. [doi: 10.3969/j.issn.1673-8012.2006.01.007]

9 林力文. 探究 Node. js 技术特性在电信 IT 支撑领域的应用. 计算机产品与流通, 2019, (12): 32–33.

10 袁劲松, 马旭东. 基于阻塞与非阻塞 I/O 网络模型的 Java 语言实现. 计算机系统应用, 2008, 17(9): 98–101. [doi: 10.3969/j.issn.1003-3254.2008.09.026]

11 程超, 杨风召. 基于 Java 非阻塞 I/O 开发高性能网络应用程序. 电子工程师, 2006, 32(10): 71–73. [doi: 10.3969/j.issn.1674-4888.2006.10.024]

12 陈刚. 非对称数据率网络下的无阻塞连接算法. 计算机时代, 2013, (7): 11–12, 16. [doi: 10.3969/j.issn.1006-8228.2013.07.004]

13 刘晓建, 吴庆波, 戴华东, 等. 一种用于并行系统的非阻塞消息队列机制. 计算机工程与科学, 2011, 33(4): 75–80. [doi: 10.3969/j.issn.1007-130X.2011.04.014]

14 边耐政, 刘玄. 基于非阻塞的分布式事务提交协议的实现. 计算机应用与软件, 2014, 31(7): 89–92, 104. [doi: 10.3969/j.issn.1000-386x.2014.07.024]

15 熊安萍, 唐巍, 蒋溢. 一种非阻塞读文件系统的实现方法. 计算机工程, 2011, 37(5): 71–73, 76. [doi: 10.3969/j.issn.1000-3428.2011.05.024]

16 钟德福, 张良国, 艾红, 等. 基于 NodeJS 的渔业资源调查数据采集系统框架重构. 渔业现代化, 2019, 46(6): 104–109.

17 张海龙, 张萌, 王杰, 等. 基于 MPI 和 Taurus 高性能计算系统的 Jacobi 并行迭代算法. 吉林大学学报 (工学版), 2019, 49(2): 606–613.

18 李正学, 许捍卫. 异步非阻塞瓦片地图服务器的实现. 测绘科学, 2015, 40(10): 128–132.

19 冉从敬, 徐晓飞. 基于 NodeJS+ECharts 的专利权人引证关系可视化方法研究. 情报科学, 2018, 36(8): 77–83.

20 徐瑞. 基于 WebSocket 与 WebGL 的城市三维协同规划. 测绘地理信息, 2019, 44(6): 96–98, 102.

21 雷津, 赵寅. VC++环境下的 UDP 网络通信实现. 电子测试, 2017, (13): 36–38. [doi: 10.3969/j.issn.1000-8519.2017.13.016]

22 Tsigas P, Zhang Y. Non-blocking data sharing in multiprocessor real-time systems. Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications. Hong Kong, China. 1999. 247–254. [doi: 10.1109/RTCSA.1999.811240]