异步非阻塞瓦片地图服务器的实现

李正学，许捍卫
（河海大学 地球科学与工程学院，南京 210098）

摘 要：针对 WebGIS 用户数量不断增大，导致瓦片地图服务器的负载压力越来越大不能满足高并发的需求的问题，该文提出了一种基于 NodeJS 的异步非阻塞地图服务器，通过提高读取瓦片的效率来提高服务器性能。深入研究了 ArcGIS 紧凑型地图瓦片数据存储格式，利用 NodeJS 解析 ArcGIS 紧凑型瓦片地图文件实现了非阻塞异步地图瓦片服务器，最后利用 WAST 对 NodeJS 和 ArcSever 瓦片服务进行压力测试。结果表明基于 NodeJS 的异步非阻塞瓦片地图服务器有更高的访问效率和并发性，以及更低的响应延迟，适合构建企业级大用户量的 WebGIS 应用。

关键词：瓦片地图；NodeJS；非阻塞；地图服务
【中图分类号】 P208 **【文献标识码】** A **【文章编号】** 1009-2307(2015)10-0128-05
DOI：10.16251/j.cnki.1009-2307.2015.10.026

0 引言

自从 Google 提出瓦片地图方案以来，瓦片技术被广泛应用到 WebGIS 开发中，成为在线地图标准，许多瓦片地图服务都是基于 Google Map 的地图模式提出的。瓦片地图服务将配置好的一定坐标范围的地图，按照固定的若干比例尺和图片的尺寸，在服务端切成若干行列的正方形图片，并按照一定的命名规则组织起来，客户端发送请求时只需返回指定的瓦片，不需要在服务端实时渲染地图，可以减少服务端的负荷，也提高了地图的加载速度[1-3]。

随着 GIS 的不断发展，GIS 正向普适化、大众化发展，GIS 应用也在向 Web、移动方向发展，地图应用用户基数不断增大，这对瓦片地图服务器的性能提出了更高的要求。地图服务器需要在几百个甚至更多的客户端同时连接的情况下，仍能够保持较高的并发处理能力。地图瓦片服务器高并发环境下需要频繁读取地图瓦片，传统的瓦片服务器都是采用阻塞式的 I/O 读写，大大影响系统性能，很难满足当前环境下高并发的需要。由于 I/O 读写速度低于 CPU 的运算速度，瓦片地图服务稳定性与并发性受限于瓦片的读取速度，造成阻塞导致服务器的效率低下。为了提高瓦片地图服务器的性能，来满足高并发性的需求，目前大多数的解决办法是通过改变数据的存储方式、优化空间索引机制、改进查询方法等技术[4-9]，然而这些技术虽然大大提高了访问效率，但最终受限于 I/O 自身的读写速率，不能满足当前实际应用中高并发的需求。百度个性化地图平台于 2014 年 5 月上线，实现了个性化配置功能，能够根据用户设置的地图样式实时渲染地图，其服务端采用了 NodeJS 的异步非阻塞技术进行地图的渲染，这是异步非阻塞技术企业级应用的示范。本文利用 NodeJS 的异步非阻塞技术，旨在最大化地减小由于 I/O 读写速度瓶颈造成的访问效率损失。

作者简介：李正学(1991—)，男，山东聊城人，硕士研究生，主要研究方向为 WebGIS 开发与应用。
E-mail：lzxue_gis@foxmail.com

收稿日期：2014-06-13
基金项目：国家自然科学基金项目(41101374)

1 Node.js 工作原理

Node.js 是一个在浏览器外部创建互联网应用程序的框架，它基于 Google 开发的 V8 JavaScript 引擎，具有轻量、高效、事件驱动、非阻塞 I/O 等特点，特别适合运行于跨平台分布式设备的实时数据处理程序。V8 引擎本身使用了一些最新的编译技术。这使得用 JavaScript 这类脚本语言编写出来的代码与用 C 这类高级语言写出来的代码性能相差无几，却节省了开发成本。Node 采用一系列“非阻塞”库来支持事件循环的方式。本质上就是为文件系统、数据库之类的资源提供接口。向文件系统发送一个

请求时，无需等待硬盘(寻址并检索文件)，硬盘准备好的时候非阻塞接口会通知 Node[11-13]。

1.1　NodeJS 异步 I/O 特点

NodeJS 基于 V8 引擎和非阻塞 I/O 的特点决定其在构建读写频繁的 web 服务具有强大的优势，能够满足高并发性。非阻塞 I/O 在程序执行过程中，I/O 操作不会阻塞程序的执行，也就是在 I/O 操作的同时，继续执行其他代码(这得益于 Node 的事件循环机制，见图 1)。在当下 I/O 设备效率还远远低于 CPU 效率的时代，这种 I/O 模型(非阻塞 I/O)为程序带来的性能上的提高是非常可观的。

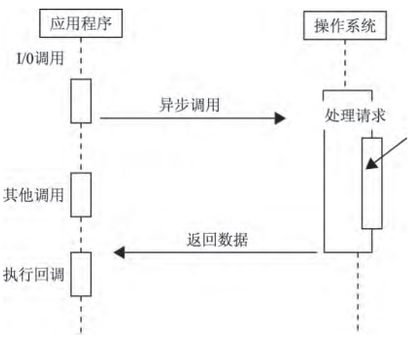

图 1　异步 I/O 执行流程

Fig. 1　Asynchronous I / O Execution Process

如图 1 所示，应用程序提交 I/O 异步调用请求，操作系统处理响应的 I/O 操作，同时应用程序无需等待操作系统返回结果，就开始执行其他处理操作(应用程序没有被 I/O 操作所阻塞)。当操作系统执行完毕并返回读取的数据，就会产生一个信号或执行一个基于线程的回调函数来完成这次 I/O 处理过程。例如执行两项任务，其中烧水 10min，洗菜 5min，异步非阻塞模型发送烧水命令不等待水烧开，直接执行洗菜命令，用时取决于耗费时间最长的任务。阻塞 I/O 模型发送烧水命令之后需等待烧水完成才可执行下一任务，时间耗费为每个任务的时间总和。非阻塞 I/O 模型在执行 I/O 密集的程序时有较高的速度和效率，瓦片地图服务是 I/O 密集、少量业务逻辑的 Web 服务，因此使用 NodeJS 构建瓦片服务器有较高的效率。

1.2　NodeJS 建立 web 服务器

NodeJS 提供 http 模块，可以较简单地构建 Http 服务器。

```
var http=require('http');
http.createServer(function (req, res) {
res.writeHead(200, {'Content-Type': 'text/plain'});
res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
```

通过上面简单的几行代码就可实现 web 服务，同样使用 NodeJS 读取瓦片文件并返回指定瓦片即可建立异步非阻塞的瓦片地图服务器，提供高性能的瓦片服务。

2　ArcGIS 紧凑型瓦片格式解析

在 ArcGIS 10.0 中出现了一种新的切片缓存文件格式：紧凑型存储(Compact)。紧凑型存储切片存储在紧凑、连续的文件流(bundle)中，具有占用磁盘空间小、快速地复制和迁移等特点。与松散型存储(Exploded)相比，紧凑型有迁移方便、创建更快、减少存储空间等诸多优点，已经成为了创建切片缓存的默认格式。本文以 ArcGIS 紧凑型瓦片文件作为瓦片服务的数据源构建瓦片地图服务器，但是 ArcGIS 并不提供使用第三方语言读取的接口，因此需要解析其文件的数据存储结构。

2.1　紧凑型存储的原理

紧凑型存储 Bundle 文件由 bundle 和 bundlx 文件组成，其中 bundle 文件用以存储切片数据，bundlx 是 bundle 文件中切片数据的索引文件。一个 bundle 文件中最多可以存储 128×128(16 384)个切片，但是创建切片缓存并不是一张张切片单独生成，而是以 4 096 像素(无抗锯齿)或 2 048 像素(有抗锯齿)为边长渲染的。如果选择的切片边长为 256 像素并开启了抗锯齿，那么每次 ArcSOC 进程创建的是一张以 8×8(64)个切片拼接成的大图，然后切割后存入 bundle 文件中。

如图 2 所示，外边框代表的是 bundle 文件，黑色小格子是生成切片时拼接的大图，具体的每个切片在黑色小格子中，图中并没有显示出来。

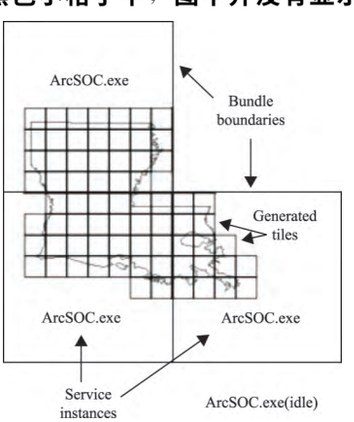

图 2　瓦片在 Bundle 文件中组织方式

Fig. 2　The Organization of Map Tiles in Bundle Files

2.2　存储格式分析

由图 3 可以看出，一个 bundlx 文件内部由文件头、文件内容、文件尾 3 部分组成。在 bundlx

文件中用固定的几个字节标识一个切片在 bundle 文件中的状态(瓦片数据在 bundle 中储存的偏移量)。ArcGIS 生成的 bundlx 文件，每个文件都是一样的大小：81 952 字节。每个 bundle 文件中最多存储 16 384 个切片，虽然 bundle 文件中可能并没有这么多切片，但是 bundlx 文件中保留了所有的 16 384 个切片的索引位置。每个切片会占据 5 个字节，16384×5=81920 字节，还多出 32 字节，这 32 个字节存储 bundlx 文件的标识信息。bundlx 中文件起始 16 字节和文件结束 16 字节与索引无关，剩余的 81 920 字节数据以 5 个字节的频率重复，构成了一个对 bundle 文件的索引。

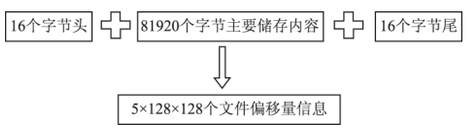

图 3 bundlx 文件结构

Fig. 3 The Structure of bundlx File

bundle 文件没有对数据压缩处理，同时每两个数据相隔了 4 个字节，这 4 个字节以低位到高位的方式标示了后续这个切片数据的长度。切片数据长度是在 bundle 文件中记录的，在 bundlx 文件中索引的只包括切片数据的偏移量，经过实验发现，bundlx 中的 5 个字节也是以低位到高位的方式标示了数据的偏移量。

了解了紧凑型文件的组织的方式，还需要知道其 bundlx 的数据偏移信息与 bundle 文件中的瓦片的映射关系，如图 4 所示。

图 4 展示了 bundlx 文件中的数据与每个瓦片的对应关系，表中的序号表示 bundle 中的瓦片序号，从左到右按列排序。根据图 4 的映射关系可以按照瓦片的序号获得瓦片的偏移信息。

1	129	…	…
2	130	…	…
3	131	…	…
…	…	…	…
…	…	…	…
128	256	…	16384

图 4 偏移量与瓦片的映射关系

Fig. 4 The Mapping Relationship between the Tile and the Offset

根据上文分析，依照切片的级别、行号、列号 3 个参数，就可以通过 bundlx 首先找到 bundle 中切片内容的偏移，然后从 bundle 文件中取出 4 个字节的长度数据，随后根据这个长度读取真实的切片数据。

2.3 Bundle 文件的命名格式

在大比例尺的切片层级下，地图图幅很大，而一个 bundle 只能存 128×128 个，所以会有多个 bundle 文件存在，这样就需要对其按所在区域分类命名。“R0080C0180”以 R+4 位 16 进制数字 C+4 位 16 进制数字组成。对于 4 个 16 进制计算规则，128×(row/128)，然后转换成 16 进制即可。举例：1～128 行的是 R0000，129～256 的是 R0080。列的计算同理。ArcGIS 的一个地图服务切片数据都存储在名为“_alllayers”的文件夹内，文件夹中包含每个层级的切面文件夹以“L+两位数层级号不足两位前面补零”命名，例如“L00”表示第 0 级地图瓦片。

WebGIS 客户端向服务端发送瓦片请求时，将发送一个切片的层级、行、列 3 个参数。服务端即可根据 3 个参数，读取指定瓦片地图文件夹的相应 Bundle 文件，将瓦片数据返回到客户端。

3 NodeJS 构建瓦片地图服务器

3.1 服务器构建

瓦片地图服务是能够响应前端瓦片请求并返回指定瓦片的服务。地图服务器根据客户端请求瓦片的行列号，在服务端读取地图瓦片文件然后返回到客户端显示。服务端主要是进行频繁的 I/O 读取操作，速度的瓶颈在于 I/O 的读写速度不能满足请求速度的需要。传统的地图瓦片服务器采用阻塞式 I/O 操作，服务处理局限于磁盘的读写速度。异步非阻塞技术能解决部分由于的 IO 读写效率带来的效率低下的问题，提高性能。ArcGIS 紧凑型瓦片储存技术、数据结构、索引技术等方面已经十分高效，因此本文在其基础上采用 NodeJS 异步非阻塞技术进一步提高瓦片的读取效率。如何采用异步技术读取 ArcGIS 紧凑型文件是本文的关键技术。Node. js 作为服务端的异步编程语言可快速搭建一个高性能 Web 服务器，利用 NodeJS 的“fs”模块采用异步方式读取解析地图瓦片文件。通过上文对 bundle 和 bundlx 文件存储方式的解析，可以编写读取解析瓦片文件的方法，并根据请求返回指定的行列号和缩放等级下的地图瓦片[14-16]。图 5 展示了构建异步非阻塞服务器的运行流程。

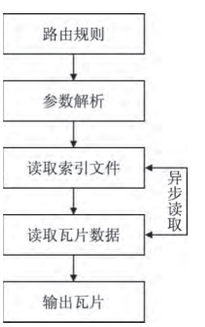

图 5 异步非阻塞流程图

Fig. 5 The Flow Chart of Asynchronous Non-blocking

如图 5 所示，通过路由规则和参数解析可以获取前端请求瓦片地图的行、列、等级号，其中读

取索引文件和读取瓦片是 I/O 读取操作，会占用较多的时间，该流程使用异步非阻塞技术读取可以充分发挥 NodeJS 异步优势，减少网络请求时间。获取地图瓦片需要先读取 bundlx 文件中的索引数据才能在 bundle 中读取瓦片数据，读取流程中包含了较多异步操作。NodeJS 的异步方式代码读取风格不同于同步的读取代码风格，在程序编写中需要进行回调控制。读取和解析 ArcGIS 紧凑型文件用到了多层嵌套而且都是异步执行，完成算法程序必须理清逻辑才能保证程序高效稳定地运行，如图 6 所示展示读取瓦片的嵌套逻辑。

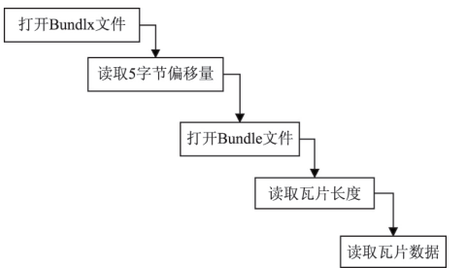

图 6　异步读取 ArcGIS 瓦片流程图

Fig. 6　The Flow Chart of Asynchronous Reading ArcGIS Tiles

图 6 的 5 个步骤均为异步操作，上一步骤执行完成之后才能进行下一步。5 个步骤由 5 个异步函数嵌套运行完成数据的读取。首先需要以异步的方式打开 bundlx 文件，打开完成之后异步读取 5 字节偏移量信息。偏移量信息读取完成之后才能异步打开 Bundle 文件，然后异步读取瓦片长度信息，最终异步读取瓦片数据。5 个步骤中函数嵌套函数，每个函数异步执行。瓦片文件打开采用 fs. open 函数、文件数据读取采用 fs. read 函数按字节进行读取，最终返回瓦片数据。

NodeJS 代码如下：

```
http. createServer(
  function(request, response) {
   response. writeHead (200, {"Content-Type":"image/png"});
   var qString=url. parse(request. url). query;
   var value=querystring. parse(qString)
   l=value. level;
   c= value. col;
   r=value. row;
   bundle(l, c, r, function(data) {//解析瓦片函数
    response. write(data);
    response. end();
   });
  }). listen(3000);
```

代码中 bundle()函数是一个异步读取解析瓦片函数，函数内部嵌套了上文所诉的 5 个嵌套的执行的异步函数，具体代码此处不再列出。服务端将请求返回文件类型“Content-Type”设置为“image/png”，前端得到数据将以图片显示。在客户端通过 ArcGIS JavaScript API 可以调用 Node. js 瓦片服务提供的地图服务，用户体验、地图展示和其他地图服务有相同的效果。

3.2　性能优化

服务能够提供稳定高效的瓦片服务，需要进行系统配置和代码的优化进一步提高性能。下面几点方案可以提高系统的性能：

1）Node 为单进程，开启 Cluster 多核机器可以很好地利用 CPU 的使用率。

2）使用 NodeJS Async 库进行异步流程控制。

3）NodeJS 服务器管理模块采用 forever 库，保证服务器的稳定运行。

4）设置瓦片缓存，对于访问频率较高的瓦片存储在缓存中。

4　性能测试

压力测试工具采用 Web Application Stress Tool（WAST），其由微软的网站测试人员开发，是专门用来进行实际网站压力测试的一套工具。通过这套强大的压力测试工具，可以使用少量的 Client 端进行实际网站仿真，减少大量用户上线对网站服务可能造成的影响。

为了更准确地进行压力对比测试，使用一台电脑部署服务环境提供地图服务，另外一台电脑作为客户端进行压力测试。服务端环境配置为 Win8 64 位操作系统、8GB 内存、CPU3. 4 GHZ。安装 ArcServer10. 1 并发布一个地图切片服务。同时将 NodeJS 编写的地图服务器部署在该服务器，与 ArcServer 发布的地图服务共用一个地图瓦片文件。在测试端机器上安装 WAST 测试软件进行压力测试。WAST 测试主要设置 3 个测试参数：Stress Level（Threads）是指定程序在后台用多少线程进行请求，也就是相当于模拟多少个客户机的连接；Stress Multiplier（Sockets per Thread）表示每个线程产生多少请求；Test Run Time 设定测试时间。基本公式为：用户数（线程数）= Stress Level × Stress Multiplier。本次压力测试参数时间为 5min，Stress Level（Threads）值为 100，Stress Multiplier 值设置为 10 来模拟 1 000 个用户并发，测试结果如表 1 所示。

表 1 压力测试结果

Tab. 1 The Stress Test Results

项目		ArcServer 测试结果	NodeJS 测试结果
Result	Number of hits	389423	442651
	Requests per Second	1279.14	1473.38
Socket Statistics	Socket Connects	390314	443495
	Total Bytes Sent(in KB)	154561.53	126680.80
	Bytes Sent Rate(in KB/s)	507.69	421.66
	Total Bytes Recv(in KB)	635215.49	745320.17
	Bytes Recv Rate(in KB/s)	2086.50	2480.83

由表 1 可以看出，300s 的测试时间内 ArcServer 的访问量为 389 423 次，平均 1 279.14 次/s；NodeJS 的访问量为 442 651 次，平均 1 473.38 次/s。由以上数据可以得出，NodeJS 服务器在 1 000 并发下比 ArcServer 的效率高 15%。本次测试所用 NodeJS 服务端是最简单的单线程 web 服务，没有充分利用多核 CPU 的特点，在服务端增加多核多进程处理可以大幅提升 NodeJS 瓦片服务器的稳定性和并发性。

5 结束语

传统的阻塞式多线程服务器在面对长连接、大数据量、高并发请求时常常遇到性能方面的局限。NodeJS 事件驱动、异步非阻塞 I/O 的特点能够满足瓦片地图服务高并发的需求，突破这一局限。本文利用 Esri 紧凑型瓦片数据作为瓦片数据源，建立了基于 NodeJS 的异步非阻塞 I/O 瓦片地图服务器提供地理底图服务，并通过压力测试工具进行性能比较。结果表明 NodeJS 实现的地图服务不但效率高，而且占用资源少，可节约服务器资源。利用 NodeJS 异步非阻塞 I/O 技术构建瓦片地图服务器可用于企业级地图应用，占用最少的服务器资源，满足大量用户并发访问的需求。

参考文献

[1] 耿庆斋，缪纶，段媛媛，等. 基于 GoogleMapsAPI 的 Web 地图服务系统研究及应用[J]. 中国水利水电科学研究院学报，2009(1)：62-66.

[2] 苏旭明，谭建成. WebGIS 中瓦片地图关键技术研究[J]. 北京测绘，2012(2)：9-12.

[3] 唐中实，朱贤泽，饶顺斌. 基于 AJAX 的 Internet 地图服务方法初探[J]. 测绘科学，2007(3)：156-159，198.

[4] 徐卓揆. 基于 HTML5、Ajax 和 WebService 的 WebGIS 研究[J]. 测绘科学，2012(1)：145-147.

[5] 王娟. GIS 普适化发展趋势凸显[N]. 中国计算机报，2012-06-18(17).

[6] 路东林，智广玉. 地图发布平台下瓦片金字塔技术研究[J]. 数字技术与应用，2013(3)：99-101.

[7] 汪龙，黄德志. ArcGIS Server 与 GeoServer 瓦片生成对比研究[J]. 科技视界，2013(27)：156-157.

[8] 刘冰，谢轲，陈小乐，等. 基于 GIS 的瓦片式地图切图算法的设计与实现[J]. 科技信息，2011(7)：60-61.

[9] 陈超，王亮，闫浩文，等. 一种基于 NoSQL 的地图瓦片数据存储技术[J]. 测绘科学，2013(1)：142-143，159.

[10] 童雪娟. 基于 WebGIS 的地图发布技术研究[D]. 武汉：武汉大学，2005.

[11] 刘静沙. 基于瓦片地图技术的车辆信息服务系统的设计[D]. 武汉：武汉理工大学，2012.

[12] 程超，杨风召. 基于 Java 非阻塞 I/O 开发高性能网络应用程序[J]. 电子工程师，2006(10)：71-73.

[13] 高原. 服务器端 javascript 技术研究[J]. 信息与电脑：理论版，2012(1)：78-80.

[14] 李晶. NodeJS——服务器端 JavaScript 运行环境[J]. 程序员，2010(12)：29.

[15] 关颖. 基于 REST 的瓦片地图 Web 服务研究[D]. 赣州：江西理工大学，2013.

[16] 周沛. 智能交通系统中的瓦片地图技术研究与应用[D]. 上海：同济大学，2008.

[17] 邬群勇，王钦敏，王焕炜. 一种 Web 地图服务搜索器的设计[J]. 微计算机应用，2009(2)：35-39.

An asynchronous non-blocking tile map server based on NodeJS

Abstract: With the continuous development of WebGIS, WebGIS application penetration into all areas of social life, the number of users is increasing, the pressure of tile map server cannot meet the growing demand for high concurrency. This paper proposed an asynchronous non-blocking map server based on NodeJS by improving the efficiency of reading tiles to improve server performance. The ArcGIS map tiles compact data storage format was studied, and using NodeJS parsing ArcGIS compact tiles map file to implement the asynchronous non-blocking tile map server. Finally, WAST was used to stress test for NodeJS and ArcSever map server. Experimental results showed that non-blocking asynchronous tile map server based on NodeJS was with higher access efficiency, higher concurrency and lower response delay for building enterprise-class large amount of users WebGIS applications.

Key words: tile map; NodeJS; non-blocking; map server

LI Zheng-xue, *XU Han-wei*(School of Earth Sciences and Engineering, Hohai University, Nanjing 210098, China)