Node.js 受益于它的事件驱动和异步的特征，已经很快了。但是，在现代网络中只是快是不行的。如果你打算用 Node.js 开发你的下一个Web 应用的话，那么你就应该无所不用其极，让你的应用更快，异常的快。本文将介绍 10 条，经过检验得知可大大提高 Node 应用的技巧。废话不多说，让我们逐条来看看。

**1. 并行**

创建 Web 应用的时候，你可能要多次调用内部 API 来获取各种数据。比如说，假设在 Dashboard 页面上，你要执行下面这几个调用:

* 用户信息 -getUserProfile().
* 当前活动 -getRecentActivity().
* 订阅内容 -getSubscriptions().
* 通知内容 -getNotifications().

为了拿到这些信息，你应该会为每个方法创建独立的中间件，然后将它们链接到 Dashboard 路由上。不过问题是，这些方法的执行是线性的，上一个没结束之前下一个不会开始。可行解决案是并行调用它们。

如你所知由于异步性，Node.js 非常擅长并行调用多个方法。我们不能暴殄天物。我上面提到的那些方法没有依赖性，所以我们可以并行执行它们。这样我们可以削减中间件数量，大幅提高速度。

我们可以用 async.js 来处理并行，它是一个专门用来调教 JavaScript 异步的 Node 模块。下面代码演示怎样用 async.js 并行调用多个方法的:

function runInParallel() {

async.parallel([

getUserProfile,

getRecentActivity,

getSubscriptions,

getNotifications

], function(err, results) {

//This callback runs when all the functions complete

});

}

如果你想更深入了解 async.js ，请移步它的 GitHub 页面。

**2. 异步**

根据设计 Node.js 是单线程的。基于这点，同步代码会堵塞整个应用。比如说，多数的文件系统 API 都有它们的同步版本。下面代码演示了文件读取的同步和异步两种操作:

// Asynchronous

fs.readFile('file.txt', function(err, buffer) {

var content = buffer.toString();

});

// Synchronous

var content = fs.readFileSync('file.txt').toString();

不过要是你执行那种长时间的阻塞操作，主线程就会被阻塞到这些操作完成为止。这大大降低你应用的性能。所以，最好确保你的代码里用的都是异步版本 API，最起码你应该在性能节点异步。而且，你在选用第三方模块的时候也要很小心。因为当你想方设法把同步操作从你代码中剔除之后，一个外部库的同步调用会让你前功尽弃，降低你的应用性能。

**3. 缓存**

如果你用到一些不经常变化的数据，你应该把它们缓存起来，改善性能。比如说，下面的代码是获取最新帖子并显示的例子:

var router = express.Router();

router.route('/latestPosts').get(function(req, res) {

Post.getLatest(function(err, posts) {

if (err) {

throw err;

}

res.render('posts', { posts: posts });

});

});

如果你不经常发贴的话，你可以把帖子列表缓存起来，然后一段时间之后再把它们清理掉。比如，我们可以用 Redis 模块来达到这个目的。当然，你必须在你的服务器上装 Redis。然后你可以用叫做 node\_redis 的客户端来保存键/值对。下面的例子演示我们怎么缓存帖子:

var redis = require('redis'),

client = redis.createClient(null, null, { detect\_buffers: true }),

router = express.Router();

router.route('/latestPosts').get(function(req,res){

client.get('posts', function (err, posts) {

if (posts) {

return res.render('posts', { posts: JSON.parse(posts) });

}

Post.getLatest(function(err, posts) {

if (err) {

throw err;

}

client.set('posts', JSON.stringify(posts));

res.render('posts', { posts: posts });

});

});

});

看到了吧，我们首先检查 Redis 缓存，看看是否有帖子。如果有，我们从缓存中拿这些帖子列表。否则我们就检索数据库内容，然后把结果缓存。此外，一定时间之后，我们可以清理 Redis 缓存，这样就可以更新内容了。

**4. gzip 压缩**

开启 gzip 压缩对你的 Web 应用会产生巨大影响。当一个 gzip 压缩浏览器请求某些资源的时候，服务器会在响应返回给浏览器之前进行压缩。如果你不用 gzip 压缩你的静态资源，浏览器拿到它们可能会花费更长时间。

在 Express 应用中，我们可以用内建 express.static() 中间件来处理静态内容。此外，还可以用 compression 中间件压缩和处理静态内容。下面是使用例:

var compression = require('compression');

app.use(compression()); //use compression

app.use(express.static(path.join(\_\_dirname, 'public')));

**5. 如果可以，在用客户端渲染**

现在有超多功能强劲的客户端 MVC/MVVM 框架，比如说 AngularJS, Ember, Meteor, 等等，构建一个单页面应用变得非常简单。基本上，你只要公开一个 API，返回 JSON 响应给客户端就可以了，而不需要在服务端渲染页面。在客户端，你可以用框架来组织 JSON 然后把它们显示在 UI 上。服务端只发送 JSON 响应可以节省带宽，改善性能，因为你不需要在每个响应里面都返回布局标记了，对吧，你只需要返回纯 JSON，然后在客户端渲染它们。

看下我的这个教程，它是关于怎样用 Express 4 公开一个 RESTful APIs的。我还写了另一篇教程，演示了怎样把这些 APIs 和 AngularJS 结合起来。

**6. 不要在 Sessions 存储太多数据**

典型的 Express 页面应用， Session 数据默认是保存在内存中的。当你把太多数据保存在 Session 的时候，会导致服务器开销显著增大。所以，要么你切换到别的储存方式来保存 Session 数据，要么尽量减少存储在 Session 中的数据量。

比如说，当用户登录到你的应用的时候，你可以只在 Session 中保存他们的 ID 而不是整个用户数据对象。还有，对于那些你能够从 id 拿到对象的查询，你应该会喜欢用 MongoDB 或者 Redis 来存储 session 数据。

**7. 优化查询**

假设你有个博客，你要在主页上显示最新帖子。你可能会通过 Mongoose 这样取数据:

Post.find().limit(10).exec(function(err, posts) {

//send posts to client

});

不过问题是 Mongoose 的 find() 方法会把对象的所有字段都查询出来，而许多字段在主页上并不要求。比如说，commentsis 保存的是特定帖子的回复。我们不需要显示文章回复，所以我们可以在查询的时候把它给剔除掉。这无疑会提高速度。可以像这样优化上面那条查询:

Post.find().limit(10).exclude('comments').exec(function(err, posts) {

//send posts to client

});

**8. 用标准的 V8 方法**

集合上的一些操作，比如 map，reduce，和 forEach 不一定支持所有浏览器。我们可以通过前台的库解决部分浏览器兼容性问题。但对于 Node.js，你要确切知道 Google 的 V8 JavaScript 引擎支持哪些操作。这样，你就可以在服务端直接用这些内建方法来操作集合了。

**9. 在 Node 前面用 Nginx**

Nginx 是个微小型轻量 Web 服务器，用它可以降低你的 Node.js 服务器的负载。你可以把静态资源配置到 nginx 上，而不是在 Node 上。你可以在 nginx 上用 gzip 压缩响应，让所有的响应都变得更小。所以，如果你有个正在营运的产品，我觉得你应该会想用 nginx 来改善运行速度的。

**10. 打包 JavaScript**

最后，你还可以大大提高页面应用速度，通过把多个 JS 文件打包。当浏览器在页面渲染中碰到 <script> 元素的时候会被堵塞，直到拿到这个脚本才继续运行(除非设置了异步属性)。比如，如果你的页面有五个 JavaScript 文件，浏览器会发出五个独立的 HTTP 请求来获取他们。如果把这五个文件压缩打包成一个，整体性能将可以大幅提升。CSS 文件也是一样。你可以用诸如 Grunt/Gulp 这样的编译工具来打包你的资源文件。

**结论**

上面 10 条技巧肯定可以提高你的 Web 应用的速度的。不过，我知道还有改善和优化的空间。如果你有任何改善性能的技巧的话，在回复里告诉我。