浙江大學

硕 士 研 究 生 读 书 报 告

题目基于 NodeJS 与 MongoDB 搭建

REST 风格的服务器

作者姓名 郁飞范

作者学号 21651073

指导教师 李启雷

学科专业 移动互联网与游戏开发

所在学院 软件学院

提交日期 二〇一七年 一 月

Build a REST-style Web Services Platform Based Node JS and MongoDB

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: Qilei Li

By

Feifan Yu

Zhejiang University, P.R. China

2017

摘要

NodeJS 目前是最火的后台技术之一，执行环境是 JavaScript，通过使用 JavaScript 可实现全栈式开发。NodeJS 的可以处理数万条的并发请求，且由于其小巧灵活的特点，特别适合用于构建 REST 风格的服务，MongoDB 是非关系型，基于分布式文件存储的数据库，具有高性能、易部署、易使用、存储数据非常方便的特点。本文主要介绍了 NodeJS 的背景、发展状况、缺点和一些改进之处，最后阐述了使用 Mongodb 来存储数据，用 NodeJS 平台搭建 REST 风格的服务器。

关键词：NodeJS;Mongodb;REST;服务器

Abstract

Nodejs now can be said to be one of the hottest Control technology,with the JavaScript execution environment.We can do Full stack development through JavaScript.NodeJS can handle tens of thousands of concurrent requests,and because of its small and flexible features, especially suitable for building REST-style services,MongoDB non-relational, distributed file storage based on the database, with high performance, easy to deploy,easy Use,data storage conveniently features. This paper introduces the background, development, shortcomings and some improvements of NodeJS. Finally, it describes how to Build a REST-style Web Services Platform Based NodeJS and Mongodb stored data.

Keywords: NodeJS;Mongodb;REST,Service.

1.TEST 简介

REST 是 Resentational State Transfer 的缩写，译为表述性状态转移。架构风格设计思想来源于 HTTP1.1 协议的设计原理。由 Roy Fielding 于 2000 年在自己的博士论文中首先提出来的。Fielding 将 REST 视为一种有助于传达蕴含于 web 中的基本概念的方式。其中最关键的几个概念就是资源（Resource）、表述（Representation）和状态（State）。资源可以是任何事物，既可以是一个实物，也可以是一个抽象的概念，只要这个事物需要被引用，就可以抽象为一个资源。表述是一个资源当前状态的有用的信息。REST 中状态分为资源状态和应用状态：资源状态是关于资源的信息，保存在服务端；应用状态是客户端在应用中所处状态的信息，由各个客户端自己维护。

RESTful Web 服务即为符合 REST 架构风格的 Web 服务。REST 风格强调以资源为中心，使用统一的操作接口和描述符，分层和无状态的交互，RESTful Web 服务具有一些 4 个特征。

（1）以通过 URL 定位的资源的方式提供服务。RESTful Web 服务以资源的方式对外发布服务，这些资源的 URL 是资源的唯一定位描述符。（2）使用 HTTP 标准方法操作资源。所有对资源的操作必须使用 HTTP 标准方法进行，主要是指 POST，PUT，GET，DELETE 方法，分别对应资源的新建、修改、查询和删除，每次对资源的访问只能进行增删改查四种操作之一，每个请求具有明确的行为语义。（3）资源通过自描述信息表示。为降低客户端和服务端的耦合，服务端可以提供多种资源描述方式来描述资源，如 XML 或 JSON，通过客户端请求报文包含的数据类型返回所需要的信息格式。(4)超媒体为应用状态的引擎(Hypermedia As The Engine Of Application State，HATEOAS）。服务端满足无状态性的要求，即客户端和服务端交互的信息必须包含上下文信息，服务端不保留客户端的任何状态信息，必须通过自描述的文件格式进行交互。客户端在获取资源的超媒体描述后，根据其中的超链接，决定进行进下一步的行为。应用的状态信息只与当前请求的资源地址有关，而与交互上下文无关。

相比传统的 RPC 模式的 Web 服务，REST 风格提出了一组相互协作的架构约束，这些约束限制了架构元素的角色和功能，以及在任何一个遵循该风格的架构中的允许存在的元素之间的关系，这些约束给 REST 架构带来了一些架构属性，而这

些架构属性恰恰是 REST 较 RPC 模式在扩展性、耦合性、可寻址、连通性等方面更有优势的体现。

2. NodeJS

2.1 NodeJS 介绍[1]

NodeJS 是 Ryan Dahl 于 2009 年发起的开源项目，是一个基于 Chrome V8 引擎的 JavaScript 执行平台，它可以快速构建网络服务及相关应用。Chrome V8 引擎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运行速度获得了极大的提升。NodeJS 对一些特殊方法和模块进行细化和调优，提供了可替代的超越 Ruby on Rails，成为 GitHub 上关注度最高的项目。应用接口 API，简单高效。NPM 是 Node 的包管理器，管理着数万基于 Node 平台的第三方开发库，NPM 允许开发者自由上传自己编写的程序库。NodeJS 算是如今最火热的技术之一。

NodeJS 作为服务器端 JavaScript 的运行平台，弱类型、基于作用域和原型链依旧是其本身的特征，重点在于将 Web 前端中一些思想，比如事件机制等迁移到了服务端的环境。相较于其他的服务器端编程语言，NodeJS 的主要特点如下：

异步非阻塞 I/O。这是 NodeJS 的主要特性，是其处理高并发请求的秘诀所在。NodeJS 在底层构建了很多异步 I/O 的 API，可以很自然很方便的调用这些 API 进行异步操作，每个调用之间无需等待，操作结束后通过回调进行数据处理。相对于同步 I/O，异步编程模型可以提升效率。（2）单线程。NodeJS 和浏览器端的 JavaScript 一样保持了单线程的特点。单线程的好处是无需考虑多线程下的状态同步，上下文切换，死锁，线程安全等方面的问题。（3）事件机制。事件的编程方式已经在前端得到了广泛的应用和肯定，因此 NodeJS 将浏览器中常见且成熟的事件引入后端，配合异步 I/O 操作，具有轻量级，松耦合，只关注事务点等优势。

借助事件驱动，异步非阻塞 I/O 等特性，对于数据密集型，I/O 密集型的应用场景有着优秀的处理能力，广泛应用于当前流行的云计算平台。

2.2 NodeJS 的发展状况[2]

从 NodeJS 的发展历程来看，它的发展时间并没有多长。自 2009 年 2 月 Ryan

Dahl 宣布准备基于 V8 创建一个轻量级的 Web 服务器起，到现在也不过只有五年的时间，但 NodeJS 的发展速度很快。目前，NodeJS 的社区十分活越，有很多优秀的软件开发人员加入了社区，同时社区的发展也得到了众多公司的支 持。一些优秀的开发人员已经为社区贡献了大量实用、简洁和高效的 NodeJS 插件和框架，这些插件和框架有用于连接数据库的、有用于做测试的、也有用于轻博客的，其中比较优秀的有 Express、Socket．IO 等。NodeJS 原本只支持运行在 POSIX 环境下的 Linux 或 MacOSX 系统，但在 2011 年 7 月，在微软的支持下 NodeJS 社区发布了 Windows 版本，而且 NodeJS 也已经被不少公司所使用。 在国外，LinkedlnMobile 的服务器端完全是用 NodeJS 写的，Yahoo 有一部分项 目使用了 NodeJS，并且 Yahoo 有了自己的 NodeJS 云服务。在国内，阿里巴巴 一些新项目使用了 NodeJS，小米公司抢购服务的控制程序也是基于 NodeJS 实现。因此 NodeJS 虽然发展时间不长，却是一个使用成熟值得信任的技术，有着良好的发展前景。目前，NodeJS 的应用场景主要包括下面三个。

1．包含状态信息传输的应用

这种应用的过程是指，提供包含状态信息的 Web 服务接收多个参数请求，对参数进行解析组合成一个响应，并返回该响应(通常是小数据)给用户。这是适合 NodeJS 处理的理想情况，这种情况 NodeJS 可以处理数万条连接。这样的应用不需要大量逻辑处理，它实质上只是从数据库中查找一些数据作为结果返回给用户。由于返回用户的数量较小，用户的请求数据也小，所以流量不是很大，一台普通的服务器就可以处理当前世界上大部分组织这样的应用需求。在本文系统中，数字标牌终端与服务器端的实时通信，就相当符合只包含状态 信息传输的应用。终端发送的请求信息，只包括简单的状态信息和请求的参数信息，服务器端返回给终端最大信息量的响应是播放策略信息，该信息量的大小最多只有两三百字节。因此，本文的服务器使用 NodeJS 无疑是一个很好的选择。

2．高并发的短消息存储发布应用

像 Twitter 这样提供社交网络的公司，它的主要业务中包括一直不问断的接收短消息并将其写入数据库。事实上，一秒的时间内可能会有数千条短消息同时到达，这导致了数据库无法及时处理高峰时期所需要写入的大量的短消息。NodeJS 则能够成为这个问题的解决方案的重要一个环节。NodeJS 具有处理数万

条短消息同时访问的能力，它能快速而又方便地将这些短消息写进一个内存排队机制(例如 memcached)的内存数据库，另一个独立的进程再把这些短消息写入磁盘上的数据库。NodeJS 所要做的工作就是快速接收这些短消息队列，然后告诉另一个负责把短消息写入数据库的进程把所有信息写入数据库。如果这件工作由常规的PHP服务器来做，通常是由PHP服务器自己来处理把这些消息写入数据库，这样每个短消息都会在写入数据库时导致一个短暂的延迟，因为写入数据库的操作都会引发阻塞。由于数据库的延迟，一台普通的服务器每秒可能只会处理 2000 条入站的短消息。如果 Twitter 每秒要处理接收 100 万条的短消息，总共就需要 500 台服务器。但是，NodeJS 能不产生阻塞地处理每个连接，因此相同的服务器在一秒钟时间内 NodeJS 处理的短消息能达到 50000 条，这样一共只需要 20 台服务器就可以处理 100 万条入站的短消息。NodeJS 恰好对高并发量的问题有着高效又简单的处理能力。

3．网络游戏等数据统计的应用。

如果您玩过一些大型的在线网络游戏，当您查看自己的游戏统计数据时，可能会发现这样的一个问题：在大型网络游戏中要想生成所有用户的统计数据，就必须跟踪海量信息。这样，如果有数百万游戏玩家同时在线玩游戏，而且他们处于游戏中的不同位置，那么短时间内就会产生海量的信息。NodeJS 给这种问题提供了一个不错的解决方法，因为它能快速收集游戏产生的数据，对数据进行少量的逻辑运算，然后对生成的数据结果排队，便于将这些数据写入数据 库。例如，服务器需要跟踪玩家在游戏中发射了多少子弹，如果我们使用 Apache 这样的服务器来会存在不小的问题；不过，如果专门使用一个运行 NodeJS 的服务器来跟踪一个游戏的所有统计数据，这不会是一个很难的问题，也不会投 入很高的成本。在本文系统的设计中，如果还需要数字标牌终端收集大众的一些行为特征时，使用 NodeJS 能够简单快速的完成这样的应用。

3.MongoDB

数据库技术发展至今产生了 SQL 数据库和 NoSQL（对所有非关系数据库的统称）数据库两大类型。非关系型数据库不支持关系模型，不支持连接操作，易使用和部署，在实现海量数据分布式存储和快速方法技术上取得了一定的成果。MongoDB 是一个基于分布式文件存储的 NoSQL 数据库，由 C++编写的，存储数据方便、性

能高。使得来源于不同用户、不同类型、构建不同数据集的网络素材信息的管理提供了技术支撑。

4.基于 NodeJS 与 MongoDB 构建 REST 风格服务器

本文将介绍使用 NodeJS 和 MongoDB 来构建一个新闻管理系统。

首先在 NodeJS 中使用 MongoDB 需要安装中间件 mongoose，通过 npm install mongoose 安装，然后下面一段代码 MongoDB 的配置。

```
var mongoose= require('mongoose');
module.exports=function () {
    var db=mongoose.connect('mongodb://localhost/news');
    require('./New');
    return db;
}
```

然后是新闻的模型。

```
var mongoose = require('mongoose');
var NewSchema = new mongoose.Schema({
    _id:{
        type:String,
        unique:true,
  },
    title:String,
    content:String
});
var News mongoose.model('News',NewSchema);
```

NodeJS 是一种后端的 JavaScript 语言，和 Java、python、ruby 语言相似，都可以构建 Web 应用，不同点是构建的 Web 应用的实现模型不一样，NodeJS 本身是基于异步非阻塞 I/O 模型，通过单线程的事件轮询机制实现高并发请求处理，这点非常适合于构建大型面向用户端的高并发 Web 应用，更重要的是 NodeJS 本身就可以创建 HTTP 服务器，而不需要借助像 Apache、IIS 这类辅助的执行容

器，所以使用 NodeJS 来构建 Web 服务，可以有很棒的灵活性，伸缩性和扩展性，这点非常适合用来构建基于 HTTP 协议的 REST 风格 Web 服务。

时下设计 REST 风格 Web 服务最常用的 2 种数据结构即 XML 和 JSON。相比较而言 JSON（Javascript Object Notation）是一种更轻量级的数据交换格式，具有良好的可读性和便于快速编写的特性。JSON 采用兼容性很高的、完全独立于语言文本格式，同时也具备类似于 C 语言的习惯（包括 C,C++,C#,Java,Python 等）体系的行为，并可在不同平台之间进行数据交换，这使得 JSON 是成为时下最为理想的数据交换语言。JSON 是基 JavaScript Programming Language,Standard ECMA-262 3rd Edition-December 1999 的一个子集，而 NodeJS 拥有完全的 Java Script 语言特性，对 JavaScript 语法结构的完美支持，这使得 NodeJS 中的对象实例可以与 JSON 进行无缝对接转化，在构建 RESTful Web 服务方面有着天然的优势。下面对 Web 服务进行模块分析。

（1）基于 NodeJS 构建 Web 服务，需要构建一个 HTTP 服务器。（2）对于不同的 HTTP 请求，不同的请求方法，Web 服务接口需要响应不同的结果，需要设计一个路由（Router），对请求进行转发，交给不同的请求处理程序（Request Handler）。（3）当 HTTP 请求被 Router 接收并转发后，需要对具体请求进行具体的处理，所以需要处理程序（Request Handler）。（4）同时为了更好的维护整个 Web 服务的启动和停止，需要设计启动文件(app.js)，为程序的入口。

通过对 REST 设计原则的理解，可以确立 RESTful 服务设计原则有如下几点：

（1）URL 是网络资源的唯一标识。（2）URL 应该全部包含名词，不能包含动词。（3）URL 应该具有语义化。（4）同一 URI，不同的 HTTP 请求方法对应不同的资源处理操作。

依据这几点原则，结合分析具体的应用场景，对 RESTful 服务接口设计进行如表 1 所示。

HTTP 方法	URL	功能
GET	/news	返回所有新闻
POST	/news	创建某新闻
GET	/news/{_id}	返回某新闻的信息
DELETE	/news/{_id}	删除某新闻

put	/news/{_id}	更新某条新闻

(1)/app.js。程序启动模块，创建 Web 服务器进行端口监听，进行全局变量配置。

```
var express = require('express'),
    routes = require('./routes');
var app = express();
routes(app); //配置路由程序
app.listen(3000); //进行端口监听
```

（2）/routes/index.js。Router 具体实现模块，进行请求 URL 匹配和转发。

```
var news = require('../controller/news.js');
module.exports = function(app) {
    app.route('/news')
        .get(news.list)
    //匹配“/news” URL 路径，并将请求转发给具体的 controller 处理
        .post(news.add);

    app.route('/news/:id')
        .get(news.get)
        .delete(user.delete)
        .put(user.update);

    app.param('id', News.getById);
}
```

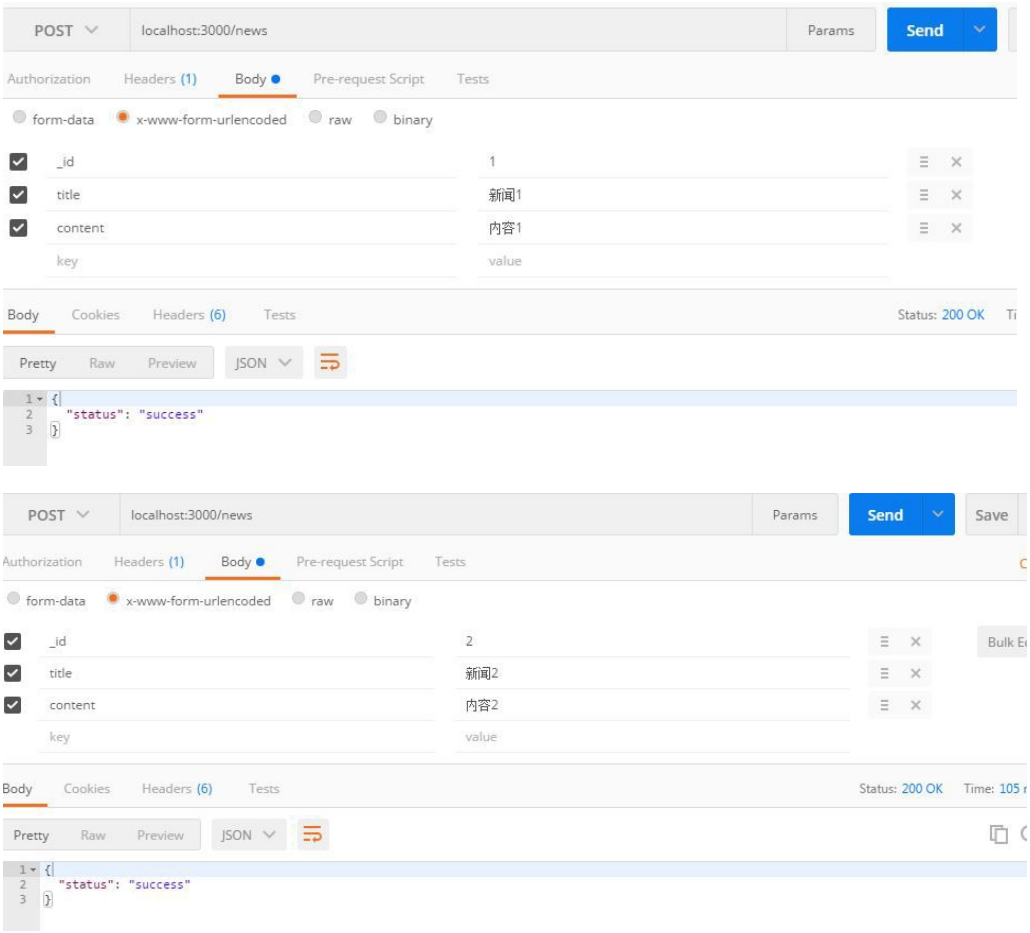
（3）/controller/news.js。具体处理 Request 模块，是程序功能的具体实现。

```
var mongoose = require('mongoose');
var News = mongoose.model('News');
```

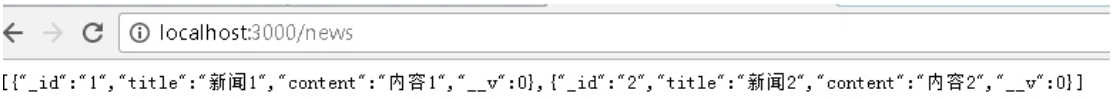
```
module.exports = {
    list: function(req, res, next){
     var pagesize = parseInt(req.query.pagesize, 10) || 10;
     var pagestart = parseInt(req.query.pagestart, 10) || 1;
     News
         .find()
         // 搜索时，跳过的条数
         .skip( (pagestart - 1) * pagesize )
         // 获取的结果集条数
         .limit( pagesize)
         .exec(function(err, docs){
             if(err) return res.json({'err'});

             return res.json(docs);
         });
},
create: function(req, res, next){
    var news = new News(req.body);
    news.save(function(err){
         if(err) return res.json({status:'err'});
         return res.json({status:'success'});
    });
},
getById: function(req, res, next, id){
    if(!id) return next(new Error('News not Found'));
    News
        .findOne({_id: id})
        .exec(function(err, doc){
           if(err) return res.json({status:'err'});
```

```
            if(!doc) return next(new Error('News not Found'));
            req.news = doc;
            return next();
        });
    },
    get: function(req, res, next) {
        return res.json(req.news);
    },
    delete:function(req,res,next){
        req.news.remove(function(err)){
            if(err){
                return res.json({status:'err'});
            }
            return res.json({status:'success'});
        }
    },
    update:function(req,res,next){
        req.news = req.body;
        req.news.save(function(err)){
            if(err){
                return res.json({status:'err'});
            }
            return res.json({status:'success'});
        }
    }
    }
```

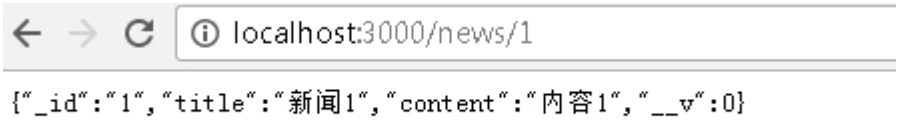
首先通过postman以POST的方式访问localhost:3000/news 来添加2条新闻。



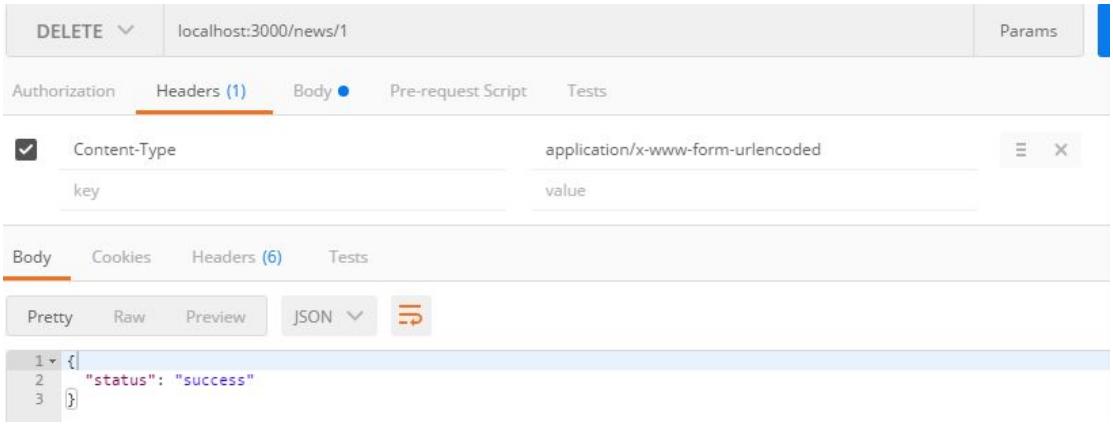
然后通过浏览器访问 localhost:3000/news 获取所有新闻：



通过浏览器访问 localhost:3000/news/1 可以获取_id 为 1 的新闻



通过 postman 以 DELETE 的方式访问 localhost:3000/news/1 来删除_id 为 1 的新闻。



此时 localhost:3000/news 就只有一条新闻了

5. Nodejs 的不足和解决方法

NodeJS 作为一项新兴的技术，虽然在异步非阻塞 I/O、事件驱动特性方面给人们眼前一亮，但也存在着许多不足的地方，比如在异步编程方式于传统的同步编程习惯有很大差异，让人一开始难适应，代码嵌套、异常捕获方面也有很大的不足。

而且异步虽然能够极大的提升 node.js 处理请求的性能，但是同时也让错误处理变得复杂。接下来通过两个简单的 web 应用来演示 NodeJS 中的异步流程控制和错误处理[3]。

1、 generator function（co）

```
var koa = require("koa");
var app = koa();// 调用异步函数
app.use(function *(next) {
    var getPromise = yield promise();
    var getThunks = yield thunks();
    this.body = getPromise + getThunks;});// promise 推荐 function promise() {
    return new Promise(function (resolve, reject) {
        //throw Error('error') //[@1](/user/1) 可捕获的异常
```

```
        setTimeout(function () {
            //throw Error('error') //[@2](/user/2) 异步中的异常,不可捕获的异常
            resolve("Hello");
        }, 1);
    })}// thunks function 不推荐 function thunks() {
    return function (fn) {
        //throw Error('error') //[@3](/user/3) 可捕获的异常
        setTimeout(function () {
            fn(null, " world");//[@4](/user/4) 正常
            //fn(Error('error'), 1); //[@5](/user/5) 异常
        })
    }}
app.listen(3000, function () {
    console.log('node listen localhost:3000');});
```

yield 支持多种形式，这里近演示 promise 和 thunks function，推荐 promise。其中@1，@3，@5 异常可以捕获，@2 会造成程序的 crash。

大家可以看到，在这种方式中，通过 yield 来调用异步函数，和大家平常写同步代码类似代码，但代码执行却依旧是异步的，怎么实现的呢，这里主要是通过 generator function 的特性来实现的，感兴趣的也可以去了解下，目前 node 4.0 及以上版本已经实现。 而且只要是支持返回 promise 的，都可以直接使用，唯一的不足时，需要使用到 generator，和一般的 function 并不兼容，并且通过 generator 处理异步，有种 hack 的感觉。

2、async await function（es2016 规范）

```
var async=require('asyncawait/async');
var await = require('asyncawait/await');
var yibu = async(
    function (s) {
        return new Promise(function (resolve, reject) {  //这里面方的是异步代码  结果通过 resolve 返回,   出错用 reject 返回
            if(s) resolve(1);
```

```
            if(!s)  reject("is err");
        });

    }
);
async(
    function () {
        try {
            var a= await(yibu(0));
            console.log(a);
        }catch (err){
            console.log(err);
        }
    }
)();
```

为了更好的处理异步，es2016 加入了关键词 async await，和上面的 generator+promise 很类似，因为 async await 实现方式本身也是基于 generator+promise。

我们上面的例子中演示的，不管采用那种方式，我们都不能处理异步中抛出的异常，异步中的异常只能通过传递来传递给调用者，所以，在异步中应该尽可能的少做逻辑，只是作为必要的操作等，比如，我们可以通过异步来获取数据，但数据的格式化等，则放到异步之外（也就是上面演示中的 setTimeout 之外），这样的话，绝大部分的异常都能被捕获，应用 crash 的几率也会大大的减少。

6.总结

REST 风格 Web 服务由于完全的基于 HTTP 协议，充分利用 Web 的特性，与传统 RPC 服务模式相比，在扩展性、安全性、数据耦合性方面有着独特的优势。利用 NodeJS 的一些优秀的特性来搭建 REST 风格的 Web 服务是一种新的尝试，相信未来会被越来越多的采用，逐渐成为构建 Web 服务的主流技术。

参考文献

[1]黄扬子.基于 NodeJS 平台搭建 REST 风格 Web 服务[J].无线互联科技.2015(16):57-59

[2]柯肇丰,曾霞霞.基于 HTML5+nodeJS+MongoDB 构建在线图像编辑器系统[J].福建电脑.2015(6):42-44

[3]elover.node.js 的异步和错误处理.CNode 技术社区