mkdir myapp

npm init -y # -y表示自动创建完成

检查nodejs对es6的支持：npm install es-checker ,运行es-checker查看结果

oneapm 性能监测，使用oneapm监控自己的项目，首先你需要去 oneapm.com注册一个开发者账号。资料地址http://cnodejs.org/topic/55dd2b68c9b0b38c7c5c8f72

**1、目录结构说明：**

/bin:用来启动应用（服务器）  
/public: 存放静态资源目录  
/routes：路由用于确定应用程序如何响应对特定端点的客户机请求，包含一个 URI（或路径）和一个特 定的 HTTP 请求方法（GET、POST 等）。每个路由可以具有一个或多个处理程序函数，这些 函数在路由匹配时执行。  
/views: 模板文件所在目录 文件格式为.ejs  
目录app.js：程序main文件 这个是服务器启动的入口

* demoProject目录结构如下

➜ demoProject tree

.

├── app.js #程序main文件 这个是服务器启动的入口

├── bin #应用启动bin目录

│ └── www

├── package.json #应用的依赖包信息

├── public #公共文件夹存放资源文件

│ ├── images

│ ├── javascripts

│ └── stylesheets

│ └── style.css

├── routes #路由文件夹，实际上可以看做Controller

│ ├── index.js

│ └── users.js

└── views #顾名思义，就是视图了，存放模板文件

├── error.jade

├── index.jade

└── layout.jade

1. directories, 9 files

1. 数据库连接说明

## test.js 文件代码：

var mysql = require('mysql'); //调用MySQL模块

//创建一个connection

var connection = mysql.createConnection({

host : 'localhost', //主机

user : 'root', //MySQL认证用户名

password : '123456', //MySQL认证用户密码

database : 'test' //端口号

});

//创建一个connection

connection.connect(function(err) {

if (err) {

console.error('error connecting: ' + err.stack);

return;

}

console.log('connected as id ' + connection.threadId);

console.log('[connection connect] succeed!');

});

//执行SQL语句

connection.query('SELECT 1 + 1 AS solution', function (error, results, fields) {

if (error) throw error;

console.log('The solution is: ', results[0].solution);

});

//关闭connection

connection.end(function(err){

// The connection is terminated now

});

//可以使用connection.destroy();代替end取关闭连接，destroy没有回调，调用该方法之后，会立即断开连接，连接不会在触发任何事件或回调

执行以下命令输出就结果为：

$ node test.js  
The solution is: 2

### **数据库连接参数说明：**

|  |  |
| --- | --- |
| **参数** | **描述** |
| host | 主机地址 （默认：localhost） |
| user | 用户名 |
| password | 密码 |
| port | 端口号 （默认：3306） |
| database | 数据库名 |
| charset | 连接字符集（默认：'UTF8\_GENERAL\_CI'，注意字符集的字母都要大写） |
| localAddress | 此IP用于TCP连接（可选） |
| socketPath | 连接到unix域路径，当使用 host 和 port 时会被忽略 |
| timezone | 时区（默认：'local'） |
| connectTimeout | 连接超时（默认：不限制；单位：毫秒） |
| stringifyObjects | 是否序列化对象（默认：'false' ；与安全相关*<https://github.com/felixge/node-mysql/issues/501>*） |
| typeCast | 是否将列值转化为本地JavaScript类型值 （默认：true） |
| queryFormat | 自定义query语句格式化方法,*[https://github.com/felixge/node-mysql#custom-format](https://github.com/felixge/node-mysql" \l "custom-format)* |
| supportBigNumbers | 数据库支持bigint或decimal类型列时，需要设此option为true （默认：false） |
| bigNumberStrings | supportBigNumbers和bigNumberStrings启用 强制bigint或decimal列以JavaScript字符串类型返回（默认：false） |
| dateStrings | 强制timestamp,datetime,data类型以字符串类型返回，而不是JavaScript Date类型（默认：false） |
| debug | 开启调试（默认：false） |
| multipleStatements | 是否许一个query中有多个MySQL语句 （默认：false） |
| flags | 用于修改连接标志,更多详情：*[https://github.com/felixge/node-mysql#connection-flags](https://github.com/felixge/node-mysql" \l "connection-flags)* |
| ssl | 使用ssl参数（与crypto.createCredenitals参数格式一至）或一个包含ssl配置文件名称的字符串，目前只捆绑Amazon RDS的配置文件 |

1. 建立连接池

可以使用mysql.createPool(config)建立多个连接：

var mysql = require('mysql');

//创建连接池

var pool = mysql.createPool({

host : '192.168.0.200',

user : 'root',

password : 'abcd'

});

//监听connection事件

pool.on('connection', function(connection) {

connection.query('SET SESSION auto\_increment\_increment=1');

});

//直接使用

pool.query('SELECT 1 + 1 AS solution', function(err, rows, fields) {

if (err) throw err;

console.log('The solution is: ', rows[0].solution);

});

//共享

pool.getConnection(function(err, connection) { // connected! (unless `err` is set)

connection.query( 'SELECT \* FROM userinfo;', function(err, result) {

console.log(result);

connection.release();

});

connection.query( 'SELECT \* FROM userinfo;', function(err, result) {

console.log(result);

connection.release();

});

});

知识点：

**（1）结束数据库连接两种方法和区别**

前面的示例中我在结尾处都会调用一个connection.end()方法,这个方法connection.connect()对应，一个开始，一个结束！结束连接其实有两种方法end()，destory()；

end() 方法在queries都结束后执行，end()方法接收一个回调函数，queries执行出错，仍然后结束连接，错误会返回给回调函数err参数，可以在回调函数中处理！

destory() 比较暴力，没有回调函数，即刻执行，不管queries是否完成！

1. **在Mysql中取消外键约束**：

SET FOREIGN\_KEY\_CHECKS=0;

Mysql中如果表和表之间建立的外键约束，则无法删除表及修改表结构。

解决方法是在Mysql中取消外键约束:

SET FOREIGN\_KEY\_CHECKS=0;

然后将原来表的数据导出到sql语句，重新创建此表后，再把数据使用sql导入，然后再设置外键约束:

SET FOREIGN\_KEY\_CHECKS=1;

**（3）其它连接池配置选项**

waitForConnections　当连接池没有连接或超出最大限制时，设置为true且会把连接放入队列，设置为false会返回error

connectionLimit　连接数限制，默认：10

queueLimit　最大连接请求队列限制，设置为0表示不限制，默认：0

1. **释放连接池**：

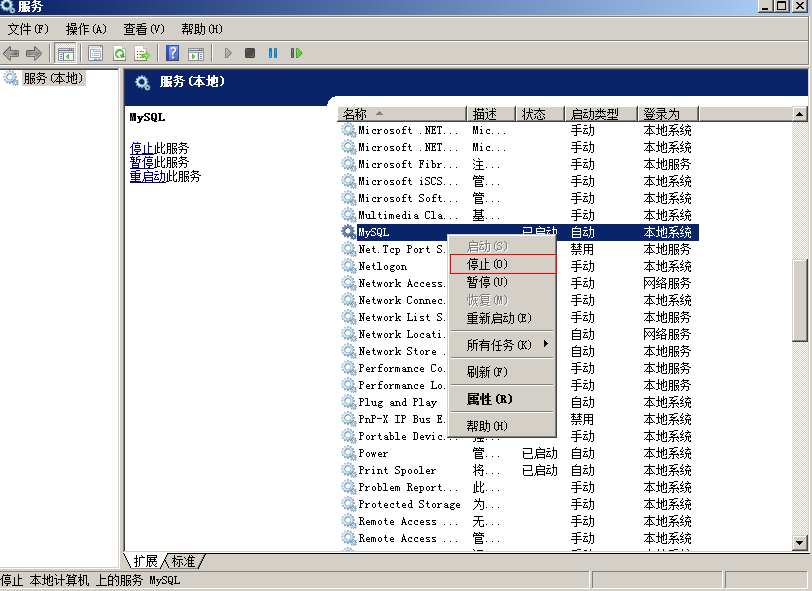
调用connection.release()方法，会把连接放回连接池，等待其它使用者使用!

**（5）断开重连：**

数据库可以因为各种原因导致连接不上，这种就必须有重连接机制！主要判断：

errorcode:PROTOCOL\_CONNECTION\_LOST

1. 首先去数据库服务器停止MySQL服务



2.运行断线重连代码

var mysql = require('mysql');

var db\_config = {

host : '192.168.0.200',

user : 'root',

password : 'abcd',

port: '3306',

database: 'nodesample'

};

var connection;

function handleDisconnect() {

connection = mysql.createConnection(db\_config);

connection.connect(function(err) {

if(err) {

console.log("进行断线重连：" + new Date());

setTimeout(handleDisconnect, 2000); //2秒重连一次

return;

}

console.log("连接成功");

});

connection.on('error', function(err) {

console.log('db error', err);

if(err.code === 'PROTOCOL\_CONNECTION\_LOST') {

handleDisconnect();

} else {

throw err;

}

});

}

handleDisconnect();

代码中在error事件中，判断返回errorcode是否为：PROTOCOL\_CONNECTION\_LOST ，如果是用setTimeout定时2秒重连！

从执行结果可以看出，大约2-3秒会输出重连输出信息，不断在尝试重新连接！

1. 去数据为服务器，开启mysql服务器，再看看执行结果

当数据库服务器mysql服务重新启动后，执行结果输出连接成功，不再输出断线重连日志^\_^!

**（6）mysql安全方法**

1.escape()　防止SQL注入，可以使用pool.escape()和connect.escape()

var mysql = require('mysql');

var pool = mysql.createPool({

host: '192.168.0.200',

user: 'root',

password:'abcd',

port:'3306',

database:'nodesample'

});

pool.getConnection(function(err,connection){

connection.query('SELECT \* FROM userinfo WHERE id = ' + '5 OR ID = 6',

function(err,result){

//console.log(err);

console.log(result);

connection.release();

});

connection.query('SELECT \* FROM userinfo WHERE id = ' + pool.escape('5 OR ID = 6') ,

function(err,result){

//console.log(err);

console.log(result);

connection.release();

});

})

　2.mysql.escapeId(identifier)

　如果不能信任由用户提示的SQL标识符（数据库名，列名，表名），可以使用此方法，官方提供有示例（最常见的是通过列名来排序什么的...）

　3.mysql.format准备查询，该函数会选择合适的转义方法转义参数

　还有一些安全相关的方法可以自行查看一下官方说明

**（7）创建数据库**

CREATE DATABASE IF NOT EXISTS nodesample CHARACTER SET UTF8;

USE nodesample;

SET FOREIGN\_KEY\_CHECKS=**0**;

DROP TABLE IF EXISTS `userinfo`;CREATE TABLE `userinfo` (

`Id` int(**11**) NOT NULL AUTO\_INCREMENT COMMENT '主键',

`UserName` varchar(**64**) NOT NULL COMMENT '用户名',

`UserPass` varchar(**64**) NOT NULL COMMENT '用户密码',

PRIMARY KEY (`Id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='用户信息表';

(8) SQLException: access denied for @'localhost' (using password: no)

解决办法   grant all privileges on \*.\* to [joe@localhost](mailto:joe@localhost" \t "http://blog.csdn.net/wengyupeng/article/details/_blank) identified by '1';

flush privileges;

例如：user: joe   pwd: 1 登陆mysql

附：

mysql> grant 权限1,权限2,…权限n on [数据库](http://blog.csdn.net/wengyupeng/article/details/javascript:;" \t "http://blog.csdn.net/wengyupeng/article/details/_self)名称.表名称 to 用户名@用户地址 identified by ‘连接口令’;  
权限1,权限2,…权限n代表

select, insert, update, delete, create, drop, index, alter, grant, references, reload, shutdown, process, file 等14个权限。  
当权限1,权限2,…权限n被all privileges或者all代替，表示赋予用户全部权限。  
当数据库名称.表名称被\*.\*代替，表示赋予用户操作服务器上所有数据库所有表的权限。  
用户地址可以是localhost，也可以是ip地址、机器名字、域名。也可以用’%'表示从任何地址连接。  
‘连接口令’不能为空，否则创建失败。

mysql>grant select,insert,update,delete,create,drop on vtdc.employee to joe@10.163.225.87 identified by ‘123′;  
给来自10.163.225.87的用户joe分配可对数据库vtdc的employee表进行select,insert,update,delete,create,drop等操作的权限，并设定口令为123。  
  
mysql>grant all privileges on vtdc.\* to joe@10.163.225.87 identified by ‘123′;  
给来自10.163.225.87的用户joe分配可对数据库vtdc所有表进行所有操作的权限，并设定口令为123。  
  
mysql>grant all privileges on \*.\* to joe@10.163.225.87 identified by ‘123′;  
给来自10.163.225.87的用户joe分配可对所有数据库的所有表进行所有操作的权限，并设定口令为123。  
  
mysql>grant all privileges on \*.\* to joe@localhost identified by ‘123′;  
给本机用户joe分配可对所有数据库的所有表进行所有操作的权限，并设定口令为123。

**Nodejs两件mysql具体操作**：<http://blog.fens.me/nodejs-mysql-intro/>

**Mysql创建索引：**

SQL CREATE INDEX 语句：

CREATE INDEX 语句用于在表中创建索引。

在不读取整个表的情况下，索引使数据库应用程序可以更快地查找数据。

索引

您可以在表中创建索引，以便更加快速高效地查询数据。

用户无法看到索引，它们只能被用来加速搜索/查询。

注释：更新一个包含索引的表需要比更新一个没有索引的表更多的时间，这是由于索引本身也需要更新。因此，理想的做法是仅仅在常常被搜索的列（以及表）上面创建索引。

SQL CREATE INDEX 语法：

在表上创建一个简单的索引。允许使用重复的值：

CREATE INDEX index\_name ON table\_name (column\_name)

注释："column\_name" 规定需要索引的列。

SQL CREATE UNIQUE INDEX 语法：

在表上创建一个唯一的索引。唯一的索引意味着两个行不能拥有相同的索引值。

CREATE UNIQUE INDEX index\_name ON table\_name (column\_name)

Node模块debug：

"start": "node ./bin/www",

"dev":"set DEBUG=znode:\* && node ./bin/www ",

"build":"set DEBUG=null && node ./bin/www",

"sup":"set DEBUG=znode:\* && supervisor ./bin/www"

用法：var debug = require('debug')('znode:server');

server.on('listening', function(){debug(‘listening on port..’)});

**Node中间件：**

1. **body-parser**

**（ http://blog.csdn.net/yanyang1116/article/details/54847560）**

Node.js(Express) HTTP请求体解析中间件

在HTTP请求中，POST、PUT和PATCH三种请求方法中包含请求体，Node.js 原生HTTP模块中，请求体要基于流的方式接收和解析。body-parser是一个HTTP请求体解析中间件，使用这个模块可以解析JSON、Raw、文本、URL-encoded格式的请求体，[Express](http://itbilu.com/nodejs/npm/4k-rFZt3l.html" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)框架中就是使用这个模块做为请求体解析中间件。

1. [请求体解析](https://itbilu.com/nodejs/npm/EkDXWklVb.html" \l "parse" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)
   * [1.1 原生环境中的解析](https://itbilu.com/nodejs/npm/EkDXWklVb.html" \l "parse-node" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)
   * [1.2 使用body-parser解析请求体](https://itbilu.com/nodejs/npm/EkDXWklVb.html" \l "body-parse" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)
2. [请求体解析](https://itbilu.com/nodejs/npm/EkDXWklVb.html" \l "parse" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)
   * [2.1 bodyParser.json() - 解析JSON格式](https://itbilu.com/nodejs/npm/EkDXWklVb.html" \l "api-json" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)
   * [2.2 bodyParser.raw() - 解析二进制格式](https://itbilu.com/nodejs/npm/EkDXWklVb.html" \l "api-raw" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)
   * [2.3 bodyParser.text() - 解析文本格式](https://itbilu.com/nodejs/npm/EkDXWklVb.html" \l "api-text" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)
   * [2.4 bodyParser.urlencoded() - 解析文本格式](https://itbilu.com/nodejs/npm/EkDXWklVb.html" \l "api-urlencoded" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)

### 1. 请求体解析

#### 1.1 原生环境中的解析

Node.js 原生HTTP模块中，是将用户请求数据封装到了用于请求对象req中，该对象是一个[IncomingMessage](http://itbilu.com/nodejs/core/N1okQ7Eh.html" \l "incomingMessage" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)，该对象同时也是一个[可读流](http://itbilu.com/nodejs/core/4y3SPWbS.html" \l "readable" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)对象。在原生HTTP服务器，或不依赖第三方解析模块时，可以像下面这样接收并解析请求体：

const http = require('http');

//用http模块创建一个http服务端

http.createServer(function(req, res) {

if (req.method.toLowerCase() === 'post') {

var body = '';

req.on('data', function(chunk){

body += chunk;

});

req.on('end', function(){

if(req.headers['content-type'].indexOf('application/json')!==-1){

// JSON 格式请求体解析

JSON.parse(body);

} else if(req.headers['content-type'].indexOf('application/octet-stream')!==-1){

// Raw 格式请求体解析

// ……

} else if(req.headers['content-type'].indexOf('text/plain')!==-1){

// text 文本格式请求体解析

// ……

} else if(req.headers['content-type'].indexOf('application/x-www-form-urlencoded')!==-1){

// URL-encoded 格式请求体解析

// ……

} else {

// 其它格式解析

}

})

} else {

res.end('其它提交方式');

}

}).listen(3000);

#### 1.2 使用body-parser解析请求体

body-parser模块是一个Express/Connect中间件，它使用非常简单且功能强大，可以像下面这样用这个模块解析请求体：

**Express/Connect 项层处理**

Express框架默认使用body-parser做为请求体解析中间件，创建[Express项目](http://itbilu.com/nodejs/npm/EJUJrGVsg.html" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)后，可以在app.js文件中看到如下代码：

/\* 引入依赖项 \*/

var express = require('express');

// ……

var bodyParser = require('body-parser');

var routes = require('./routes/index');

var users = require('./routes/users');

var app = express();

// ……

// 解析 application/json

app.use(bodyParser.json());

// 解析 application/x-www-form-urlencoded

app.use(bodyParser.urlencoded());

这样就在项目的[Application](http://itbilu.com/nodejs/npm/VJ5TlyRnl.html" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)级别，引入了body-parser模块处理请求体。在上述代码中，模块会处理application/x-www-form-urlencoded、application/json两种内容格式的请求体。经过这个中间件处理后，就可以在所有路由处理器的req.body中访问请求参数。

**解析Express具体路由**

在实际应用中，不同路径（路由）可能会要求用户使用不同的内容类型，body-parser还支持为单个[Express路由](http://itbilu.com/nodejs/npm/Vk96Hcepe.html" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)添加请求体解析：

var express = require('express')

var bodyParser = require('body-parser')

var app = express()

// 创建 application/json 解析

var jsonParser = bodyParser.json()

// 创建 application/x-www-form-urlencoded 解析

var urlencodedParser = bodyParser.urlencoded({ extended: false })

// POST /login 获取 URL编码的请求体

app.post('/login', urlencodedParser, function (req, res) {

if (!req.body) return res.sendStatus(400)

res.send('welcome, ' + req.body.username)

})

// POST /api/users 获取 JSON 编码的请求体

app.post('/api/users', jsonParser, function (req, res) {

if (!req.body) return res.sendStatus(400)

// create user in req.body

})

**指定请求类型**

body-parser还支持为某一种或一类内容类型的请求体指定解析方式，指定时可以通过在解析方法中添加type参数修改指定Content-Type的解析方式。

如，可以对text/plain内容类型使用JSON解析：

app.use(bodyParser.json({ type: 'text/plain' }))

这一选项更多是用在非标准请求头的解析中，如下：

// 解析自定义的 JSON

app.use(bodyParser.json({ type: 'application/\*+json' }))

// 解析自定义的 Buffer

app.use(bodyParser.raw({ type: 'application/vnd.custom-type' }))

// 将 HTML 请求体做为字符串处理

app.use(bodyParser.text({ type: 'text/html' }))

### 2. body-parser模块的API

通过npm install body-parser命令安装模块后，可以通过以下方式获取模块引用：

var bodyParser = require('body-parser')

bodyParser变量是对中间件的引用。请求体解析后，解析值都会被放到req.body属性，内容为空时是一个{}空对象。

#### 2.1 bodyParser.json() - 解析JSON格式

bodyParser.json(options)

返回一个仅解析json格式数据的中间件。这个方法支持任意Unicode编码的请求体，且支持gzip和deflate编码的数据压缩。

**Option**是一个包含以下可选值的对象

* inflate - 设置为true时，deflate压缩数据会被解压缩；设置为true时，deflate压缩数据会被拒绝。默认为true。
* limit - 设置请求的最大数据量。默认为'100kb'
* reviver - 传递给JSON.parse()方法的第二个参数，详见[JSON.parse()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse" \l "Example.3A_Using_the_reviver_parameter" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)
* strict - 设置为true时，仅会解析Array和Object两种格式；设置为false会解析所有JSON.parse支持的格式。默认为true
* type - 该选项用于设置为指定MIME类型的数据使用当前解析中间件。这个选项可以是一个函数或是字符串，当是字符串是会使用[type-is](https://www.npmjs.com/package/type-is" \l "readme" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)来查找MIMI类型；当为函数是，中间件会通过fn(req)来获取实际值。默认为application/json。
* verify - 这个选项仅在verify(req, res, buf, encoding)时受支持

#### 2.2 bodyParser.raw() - 解析二进制格式

bodyParser.raw(options)

返回一个将所有数据做为[Buffer](http://itbilu.com/nodejs/core/4y8qlhOml.html" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)格式处理的中间件。这个方法支持gzip和deflate编码的数据压缩。解析后，其后的所有的req.body中将会是一个Buffer数据。

**Option**是一个包含以下可选值的对象

* inflate - 设置为true时，deflate压缩数据会被解压缩；设置为true时，deflate压缩数据会被拒绝。默认为true。
* limit - 设置请求的最大数据量。默认为'100kb'
* type - 该选项用于设置为指定MIME类型的数据使用当前解析中间件。这个选项可以是一个函数或是字符串，当是字符串是会使用[type-is](https://www.npmjs.com/package/type-is" \l "readme" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)来查找MIMI类型；当为函数是，中间件会通过fn(req)来获取实际值。默认为application/octet-stream。
* verify - 这个选项仅在verify(req, res, buf, encoding)时受支持

#### 2.3 bodyParser.text() - 解析文本格式

bodyParser.text(options)

返回一个仅处理字符串格式处理的中间件。这个方法支持gzip和deflate编码的数据压缩。解析后，其后的所有的req.body中将会是一个字符串值。

**Option**是一个包含以下可选值的对象

* defaultCharset - 如果Content-Type后没有指定编码时，使用此编码。默认为'utf-8'
* inflate - 设置为true时，deflate压缩数据会被解压缩；设置为true时，deflate压缩数据会被拒绝。默认为true。
* limit - 设置请求的最大数据量。默认为'100kb'
* type - 该选项用于设置为指定MIME类型的数据使用当前解析中间件。这个选项可以是一个函数或是字符串，当是字符串是会使用[type-is](https://www.npmjs.com/package/type-is" \l "readme" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)来查找MIMI类型；当为函数是，中间件会通过fn(req)来获取实际值。默认为application/octet-stream。
* verify - 这个选项仅在verify(req, res, buf, encoding)时受支持

#### 2.4 bodyParser.urlencoded() - 解析文本格式

bodyParser.urlencoded(options)

返回一个处理urlencoded数据的中间件。这个方法默认使用UTF-8编码，且支持gzip和deflate编码的数据压缩。解析后，其后的所有的req.body中将会是一个键值对对象。

**Option**是一个包含以下可选值的对象

* extended - 当设置为false时，会使用[querystring](http://itbilu.com/nodejs/core/Ny8rxFMU.html" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)库解析URL编码的数据；当设置为true时，会使用[qs](https://www.npmjs.com/package/qs" \l "readme" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)库解析URL编码的数据。后没有指定编码时，使用此编码。默认为true
* inflate - 设置为true时，deflate压缩数据会被解压缩；设置为true时，deflate压缩数据会被拒绝。默认为true。
* limit - 设置请求的最大数据量。默认为'100kb'
* parameterLimit - 用于设置URL编码值的最大数据。默认为1000
* type - 该选项用于设置为指定MIME类型的数据使用当前解析中间件。这个选项可以是一个函数或是字符串，当是字符串是会使用[type-is](https://www.npmjs.com/package/type-is" \l "readme" \t "http://blog.csdn.net/yanyang1116/article/details/_blank)来查找MIMI类型；当为函数是，中间件会通过fn(req)来获取实际值。默认为application/octet-stream。
* verify - 这个选项仅在verify(req, res, buf, encoding)时受支持

1. morgan日志：<https://segmentfault.com/a/1190000007769095>
2. Cookie-parser：<https://segmentfault.com/a/1190000004139342?_ea=504710>
3. debug模块

|  |
| --- |
| **var** debug = require('debug')('myapp:main');  debug('现在的时间是 %s' , **new** Date()); |

如果是开发环境，只需要设置环境变量

IMG_256

这样就可以输出调试信息

IMG_257

如果需要关闭调试信息

IMG_258