Express.js中文文档－Response对象

2016年03月14日 1052 声明

response 对象代表HTTP响应信息，响应信息在Express应用收到HTTP请求后发送给客户端。Express的 response 对象是对Node.js http.ServerResponse (http://itbilu.com/nodejs/core/VyfljHIh.html#response)对象扩展，在 ServerResponse 对象基础上增加了一些Web应用中常用的属性和方法。按照习惯，在本文档中 response 对象被表示为 res 。

1. Response 对象
2. 属性
   - 2.1 res.app
   - 2.2 res.headersSent
   - 2.3 res.locals
3. 方法
   - 3.1 响应HTTP头添加信息： res.append()
   - 3.2 设置附件响应头： res.attachment()
   - 3.3 设置Cookie： res.cookie()
   - 3.4 请除Cookie： res.clearCookie()
   - 3.5 请除Cookie： res.download()
   - 3.6 结束响应： res.end()
   - 3.7 格式化响应： res.format()
   - 3.8 获取响应头： res.get()
   - 3.9 响应JSON格式： res.json()
   - 3.10 响应JSONP格式： res.jsonp()
   - 3.11 设置 Link 头： res.links()
   - 3.12 设置 Location 头： res.location()
   - 3.13 URL重定向： res.redirect()
   - 3.14 渲染视图： res.render()
   - 3.15 发送响应： res.send()
   - 3.16 发送文件： res.sendFile()
   - 3.17 设置响应状态： res.sendStatus()
   - 3.18 设置HTTP响应头： res.set()
   - 3.19 设置 Content-Type ： res.type()
   - 3.20 设置 Vary ： res.vary()

1. Response 对象

Response 对象继承自Node.js的 ServerResponse 对象， ServerResponse 中的属性、事件、方法，在 Response 对象中都可以使用。它在收到用户请求时被自动创建，通过这个对象可以实现对不同用户请求的响应，我们可以在路由处理器（即：回调函数）中访问这个对象。

```
app.get('/user/:id', function(request, response){
  response.send('user ' + request.params.id);
});
```

按习惯我们可以将 response 简写为 res ：

```
app.get('/user/:id', function(req, res) {
  res.send('user ' + req.params.id);
});
```

2. 属性

2.1 res.app

res.app 是对Expressapplication (http://itbilu.com/nodejs/npm/VJ5TIyRnl.html)实例的引用，通过个属性我们可以访问 app 对象中的设置、属性、中间件等。

res.app 与 request 对象的 req.app (http://itbilu.com/nodejs/npm/41wCi_C3e.html#req-prop-app)属性完全一样。

2.2 res.headersSent

一个表示HTTP头信息是否已发送的布尔值。

```
app.get('/', function (req, res) {
  console.log(res.headersSent); // false
  res.send('OK');
  console.log(res.headersSent); // true
});
```

2.3 res.locals

一个针对当前求的包含在响应信息中的局部变量，这个变量是提供给要渲染的视图(s)使用的（如果存在）。这个属性与app.locals (http://itbilu.com/nodejs/npm/VJ5TlyRnl.html#app-properties-locals)完全相同。

res.locals 属性很适合在顶层请求中暴露一些公用信息，如：验证用户信息、用户设置等。

```
app.use(function(req, res, next){
  res.locals.user = req.user;
  res.locals.authenticated = ! req.user.anonymous;
  next();
});
```

3. 方法

3.1 响应HTTP头添加信息： res.append()

```
res.append(field [, value])
```

- field：{String}－HTTP头字段，如：'Link'、'Set-Cookie'等
- value：{String}，可选－要添加的值

向指定HTTP头字段 field 中添加值 value 。如果指定字段不存在，会创建该字段值。

```
res.append('Link', ['', '']);
res.append('Set-Cookie', 'foo=bar; Path=/; HttpOnly');
res.append('Warning', '199 Miscellaneous warning');
```

3.2 设置附件响应头： res.attachment()

```
res.attachment([filename])
```

设置HTTP响应头 Content-Disposition 字段值为'attachment'（附件）。如果已给出文件名，那么设置 Content-Type 字段值为 res.type() 获取的文件扩展名，并在 Content-Disposition 字段中设置'filename=...'参数。

```
res.attachment();
// Content-Disposition: attachment

res.attachment('path/to/logo.png');
// Content-Disposition: attachment; filename="logo.png"
// Content-Type: image/png
```

3.3 设置Cookie： res.cookie()

```
res.cookie(name, value [, options])
```

设置Cookie名为'name'的值为'value'。值可以是一个字符串，或是转换为JSON形式的对象。

option 是一个可选对象，对象中可以有以下值：

- domain：{String}－Cookie的有效域名。默认为应用的域名
- encode：{Function}－用户设置Cookie编码方式的函数（同步）。默认为encodeURIComponent (http://itbilu.com/javascript/js/4J2LKMVO.html)
- expires：{Date}－Cookie超时时间（GTM）。默认为会话Cookie
- httpOnly：{Boolean}－是否只能通过服务器访问Cookie
- maxAge：{String}－Cookie最大存在时间， expires 的方便写法
- secure：{Boolean}－是否仅在 HTTPS 下有效
- signed：{Boolean}－Cookie是否签名

res.cookie() 本质上是按照RFC 6265 (http://tools.ietf.org/html/rfc6265)标准设置 Set-Cookie 头，了解更多请参考Http Cookie机制及Cookie的实现原理 (http://itbilu.com/other/relate/4J4n8fIPe.html)。

```
res.cookie('name', 'tobi', { domain: '.example.com', path: '/admin', secure: true });
res.cookie('rememberme', '1', { expires: new Date(Date.now() + 900000), httpOnly: true });
```

3.4 请除Cookie： `res.clearCookie()`

```
res.clearCookie(name [, options])
```

请除指定名称的Cookie。可选值 `options` 与 `res.cookie()` 的 `options` 一样。

```
res.cookie('name', 'tobi', { path: '/admin' });
res.clearCookie('name', { path: '/admin' });
```

3.5 请除Cookie： `res.download()`

```
res.download(path [, filename] [, fn])
```

传输'attachment'路径对应的文件，通常浏览器会提示用户下载。默认情况下，会使用 `Content-Disposition` 头字段中'filename=...'中的路径及文件名，如果需要重写下载文件名可以使用 `filename` 可选参数。

如果下载发生错误，会调用 `fn` 可选参数。这个方法传输文件会调用res.sendFile()方法。

```
res.download('/report-12345.pdf');

res.download('/report-12345.pdf', 'report.pdf');

res.download('/report-12345.pdf', 'report.pdf', function(err){
  if (err) {
    // 处理错误，但响应可能已经部分发送
    // 这时应该检查 res.headersSent
  } else {
    // 下载……
  }
});
```

3.6 结束响应： `res.end()`

```
res.end([data] [, encoding])
```

发送数据后结束响应，这个方法来自于Node HTTP模块 `http.ServerResponse` 中的 `response.end()` 方法。不添加任何参数时，会在不响应任何数据的情况下结束响应。

```
res.end();
res.status(404).end();
```

3.7 格式化响应： `res.format()`

```
res.format(object)
```

根据HTTP请求头中的 `content-negotiation` 按类型响应请求。它使用 `req.accepts()` (http://itbilu.com/nodejs/npm/41wCi_C3e.html#req-method-accepts)选择响应处理程序，如果没有指定头字段将使用第一个进行响应。如果匹配不到，会响应'406'（不接受）或使用默认的回调。

下面是一个响应 `{ "message": "hey" }` 信息的示例，当请求头中的请求类型不同，所响应的格式也有所不同：

```
res.format({
  'text/plain': function(){
    res.send('hey');
  },
  'text/html': function(){
    res.send('<p>hey</p>');
  },
  'application/json': function(){
    res.send({ message: 'hey' });
  },
  'default': function() {
    // 记录请求并响应406
    res.status(406).send('Not Acceptable');
  }
});
```

除了规范化的MIME类型外，还可以使用扩展进行设置：

```
res.format({
  text: function(){
    res.send('hey');
  },
  html: function(){
    res.send('
hey

');
  },
  json: function(){
    res.send({ message: 'hey' });
  }
});
```

3.8 获取响应头： res.get()

```
res.get(field)
```

获取指定的HTTP响应头。匹配时不区分大小写。

```
res.get('Content-Type');
// => "text/plain"
```

3.9 响应JSON格式： res.json()

```
res.json([body])
```

发送一个JSON格式的响应。这个方法等同于使用 res.send() 响应一个对象或数组。

```
res.json(null);
res.json({ user: 'tobi' });
res.status(500).json({ error: 'message' });
```

3.10 响应JSONP格式： res.jsonp()

```
res.jsonp([body])
```

向一个支持JSONP的请求发送JSON响应。这个方法等同于 res.json() ，但会加入对JSONP的支付。

```
res.jsonp(null);
// => null

res.jsonp({ user: 'tobi' });
// => { "user": "tobi" }

res.status(500).jsonp({ error: 'message' });
// => { "error": "message" }
```

3.11 设置 Link 头： res.links()

```
res.links(links)
```

向HTTP响应头的 Link 字段，加入提供 links 参数。

```
res.links({
  next: 'http://api.example.com/users?page=2',
  last: 'http://api.example.com/users?page=5'
});
```

将生成以下结果：

```
Link: <http://api.example.com/users?page=2>; rel="next",
  <http://api.example.com/users?page=5>; rel="last"
```

3.12 设置 Location 头： res.location()

```
res.location(path)
```

以 path 参数设置 Location HTTP响应头。

```
res.location('/foo/bar');
res.location('http://example.com');
res.location('back');
```

3.13 URL重定向： res.redirect()

```
res.redirect([status,] path)
```

将URL重定向到 path ，并设置指定的状态码 status （如果指定）。如果不指定状态码时，默认为"302 Found"：

```
res.redirect('/foo/bar');
res.redirect('http://example.com');
res.redirect(301, 'http://example.com');
res.redirect('../login');
```

更多关于 res.location() 与 res.redirect() 请参考：Express URL跳转（重定向）的实现：res.location()与res.redirect() (http://itbilu.com/nodejs/npm/EJD5cyg3l.html)

。

3.14 渲染视图： res.render()

```
res.render(view [, locals] [, callback])
```

渲染视图并将渲染后的HTML发送到客户端。可选参数：

- locals ：{Object}，视图使用的本地变量对象
- callback ：{Function}，视图沉静染完成后的回调函数，格式为： fn(err, html)

```
// 发送渲染后的视图
res.render('index');

// 如果指定了回调函数，则渲染后的HTML需要显式的发送
res.render('index', function(err, html) {
  res.send(html);
});

// 向视图传递一个本地变量
res.render('user', { name: 'Tobi' }, function(err, html) {
  // ...
});
```

3.15 发送响应： res.send()

```
res.send([body])
```

发送HTTP响应

body 参数可是一个Buffer、字符串或数组。如：

```
res.send(new Buffer('whoop'));
res.send({ some: 'json' });
res.send('
some html

');
res.status(404).send('Sorry, we cannot find that!');
res.status(500).send({ error: 'something blew up' });
```

这个方法自动做了很多简单有效的工作，如：自动添加 Content-Length 响应头，并自动添加HTTP缓存新鲜支持。

当发送一个 Buffer 数据时， Content-Type 响应头字段为"application/octet-stream"，除非手动指定：

```
res.set('Content-Type', 'text/html');
res.send(new Buffer('
some html

'));
```

当发送一个 字符吕 数据时， Content-Type 响应头字段会自动设置为"text/html"：

```
res.send('
some html

');
```

如果发送一个对象或数组时，会自动按JSON格式响应：

```
res.send({ user: 'tobi' });
res.send([1,2,3]);
```

3.16 发送文件： res.sendFile()

```
res.sendFile(path [, options] [, fn])
```

这是一在Expres v4.8.0 中添加的方法。它会发送 path 中指定的文件，并自动使用文件扩展名设置 Content-Type 响应头。 path 必须是绝对路径，除非在 options 选项中指定。

options 对象可选参数如下：

- maxAge： 设置缓存 Cache-Control 头的 max-age 属性。默认0
- root： 相对文件名的根目录
- lastModified： 设置 Last-Modified 头（文件最后修改时间）。默认为 Enabled， Express 4.9.0+
- headers： 文件相关的头信息
- dotfiles： 是否支持'.'文件，可选值有：“allow”、“deny”、“ignore”，默认：“ignore”

3.17 设置响应状态： res.sendStatus()

```
res.sendStatus(statusCode)
```

设置 statusCode HTTP状态码，并响应一个状态描述信息：

```
res.sendStatus(200); // equivalent to res.status(200).send('OK')
res.sendStatus(403); // 等于 res.status(403).send('Forbidden')
res.sendStatus(404); // 等于 res.status(404).send('Not Found')
res.sendStatus(500); // 等于 res.status(500).send('Internal Server Error')
```

如果是未知的HTTP状态码，则发送的状态码与状态描述一样：

```
res.sendStatus(2000); // 等于 res.status(2000).send('2000')
```

3.18 设置HTTP响应头： res.set()

```
res.set(field [, value])
```

设置HTTP响应头 field 的值为 value 。如果一次性设置多个头，可以传入一个设置对象：

```
res.set('Content-Type', 'text/plain');

res.set({
  'Content-Type': 'text/plain',
  'Content-Length': '123',
  'ETag': '12345'
});
```

这个方法是 res.header(field [, value]) 的别名。

3.18 设置状态码： res.status()

```
res.status(code)
```

设置HTTP响应状态码：

```
res.status(403).end();
res.status(400).send('Bad Request');
res.status(404).sendFile('/absolute/path/to/404.png');
```

3.19 设置 Content-Type： res.type()

```
res.type(type)
```

使用mime.lookup() (http://itbilu.com/nodejs/core/VJYaAfKrl.html)返回的 type 值设置 Content-Type 响应头。

```
res.type('.html');              // => 'text/html'
res.type('html');               // => 'text/html'
res.type('json');               // => 'application/json'
res.type('application/json');   // => 'application/json'
res.type('png');                // => image/png:
```

3.20 设置 Vary： res.vary()

```
res.vary(field)
```

将 Vary 字段添加到响应头，如果指定的字段不存在时。

```
res.vary('User-Agent').render('docs');
```

下一篇：Express.js中文文档－Router对象 (/nodejs/npm/Vk96Hcepe.html)

上一篇：Express.js中文文档－Request对象 (/nodejs/npm/41wCi_C3e.html)