Express.js - Construção de Servidores Web

[**1. Introdução ao Express 2**](#_30j0zll)

[1.1 Métodos 3](#_1fob9te)

[Express.json 3](#_3znysh7)

[Express.raw 4](#_2et92p0)

[Express.router 4](#_tyjcwt)

[Express.static 5](#_3dy6vkm)

[Express.text 6](#_1t3h5sf)

[Express.urlencoded 7](#_4d34og8)

[**2. Aplicação 8**](#_2s8eyo1)

[2.1 Propriedades 9](#_17dp8vu)

[app.locals 9](#_3rdcrjn)

[app.mountPath 9](#_26in1rg)

[2.2 Eventos 10](#_lnxbz9)

[app.on 10](#_35nkun2)

[2.3 Métodos 10](#_1ksv4uv)

[app.all 10](#_44sinio)

[app.delete 11](#_2jxsxqh)

[app.disable 12](#_z337ya)

[app.engine 12](#_3j2qqm3)

[app.get 13](#_1y810tw)

[1. app.get(path, callback[, callback]) 13](#_4i7ojhp)

[app.listen 13](#_2xcytpi)

[1. app.listen(port, host, backlog, callback) 13](#_1ci93xb)

[app.method 14](#_3whwml4)

[app.params 15](#_2bn6wsx)

[app.post e app.put 17](#_qsh70q)

[app.render 18](#_3as4poj)

[app.route 18](#_1pxezwc)

[app.set 18](#_49x2ik5)

[1. Configuração do aplicativo 19](#_2p2csry)

[2. Configuração de trust proxy 20](#_147n2zr)

[3. Configuração de Etag 21](#_3o7alnk)

[app.use 21](#_23ckvvd)

[1. Middleware de tratamento de erros 22](#_ihv636)

[**3. Request 23**](#_32hioqz)

[3.1 Propriedades 24](#_1hmsyys)

[req.app 24](#_41mghml)

[req.baseUrl 24](#_2grqrue)

[req.body 24](#_vx1227)

[req.cookies 25](#_3fwokq0)

[req.fresh 25](#_1v1yuxt)

[req.hostname 26](#_4f1mdlm)

[rep.ip 26](#_2u6wntf)

[req.originalUrl 27](#_19c6y18)

[req.params 27](#_3tbugp1)

[req.path 28](#_28h4qwu)

[req.protocol 28](#_nmf14n)

[req.query 29](#_37m2jsg)

[req.route 29](#_1mrcu09)

[req.secure 30](#_46r0co2)

[req.signedCookies 30](#_2lwamvv)

[req.stale 30](#_111kx3o)

[req.subdomains 31](#_3l18frh)

[req.xhr 31](#_206ipza)

[3.2 Métodos 31](#_nwh4ungs21qw)

[req.accepts 31](#_ejqqnsoiyes3)

[req.get 32](#_s9h8td4erusj)

[**4. Response 32**](#_n81108qj4w8v)

[4.1 Propriedades 33](#_2koepysegsj0)

[res.headerSent 33](#_lmrvpyko1vn2)

[res.local 33](#_u830e7h0ut6o)

[4.2 Métodos 33](#_uv63ngflzpuh)

[res.append 33](#_5l1a92r31do9)

[res.cookie 34](#_vbxywk1zb2qi)

[res.download 35](#_epjb5r7sb5f4)

[res.end 35](#_b88zx5vemke3)

[res.json 36](#_3tyypfnmtj5f)

[res.location 36](#_wdad9ytt8hiv)

[res.redirect 36](#_llujmagox0su)

[res.render 36](#_rdjp46hujhgz)

[res.send 37](#_3vb78fo2b916)

[res.sendFile 38](#_1w3wwab3nngo)

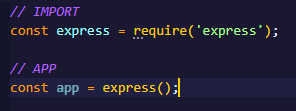
[res.set 38](#_lg1fo4pfnd7e)

[res.type 38](#_q51nwxghykgf)

[**5. Roteador 39**](#_s4dm1uomtc5n)

# 1. Introdução ao Express

**Exemplo**:



**express() é uma função de nível superior exportada pelo express módulo**.

## **1.1 Métodos**

### **Express.json**

**Middleware integrado** **que analisa solicitações recebidas com cargas JSON** e é baseado em body-parser.

* Este analisador aceita codificação Unicode do body.

BodyObjeto contendo os dados é preenchido no request objeto após o middleware ou objeto vazio caso não exista dados.

**OBS: todas as propriedades e valores neste objeto não são confiáveis e devem ser validados**.

**Propriedades opcionais**:

* **inflate** - ativa ou desativa o manuseio de bodys comprimidos (Padrão: true, tipo: booleano).
* **limit** - controla o tamanho máximo do corpo da solicitação (Padrão: 100kb).
* **reviver** - passada ao json.parser como segundo argumento (tipo: função).
* **strict** - aceita arrays e objetos (Padrão: true, tipo: booleano).
* **type** - determina o tipo de mídia que será analisada (Padrão: Aplicação json).
  + se for uma função, será chamada como fn(req) e a solicitação é analisada se retornar verdadeiro.
* **verify** - possui verify(req, res, buf e enconding) (tipo: função).

### **Express.raw**

**Analisa as cargas de solicitação** recebidas em buffe e é baseado em body-parse.

* retorna um middleware.
* apenas analisa solicitações onde contém typecabeçalho.

O analisador aceita qualquer codificação Unicode do corpo e suporta inflação gzipe.

Após o middleware, o novo body buffer contendo os dados é preenchido no request ou um objeto vazio se não houver nada para analisar.

**Propriedades opcionais**:

* **inflate** - ativa ou desativa o manuseio de bodys comprimidos (Padrão: true, tipo: booleano).
* **limit** - controla o tamanho máximo do corpo da solicitação (Padrão: 100kb).
* **type** - determina o tipo de mídia que será analisada (Padrão: Aplicação json).
  + se for uma função, será chamada como fn(req) e a solicitação é analisada se retornar verdadeiro.
* **verify** - possui verify(req, res, buf e enconding) (tipo: função).

### **Express.router**

**Exemplo: cria um novo objeto de roteador**.



**Parâmetro opcional: especifica o comportamento do roteador**.

* **caseSensitive** - distinção entre maiúsculas e minúsculas (Padrão: desativado).
* **mergeParams** - preserva os req.params do roteador pai (Padrão: false).
  + para nomes conflitantes entre pais e filhos, o nome do filho terá precedência.
* **strict** - habilita roteamento estrito (Padrão: desativado).

rotas de métodos HTTP e middlewares podem ser adicionadas, assim como é feita em aplicativos.

### **Express.static**

Serve arquivos estáticos e é baseado em serve static.

**Nota: a utilização de proxy reverse melhora o desempenho de fornecimento de ativos estáticos**.

**root** - específica diretório raiz que possui os arquivos estáticos.

**OBS: caso o arquivo não seja encontrado, ele chama next() e passa para o próximo middleware, permitindo o empilhamento e fallbacks (serve como plano de emergência em casos de falha)**.

**Propriedades**:

* **dotfiles** - determina como arquivos e diretórios que começam por um ponto são tratados (tipo: string, padrão: “ignore”).
* **etag** - habilita e desabilita a geração de etag (tipo: boolean, padrão: true).
* **extensions** - define substitutos de extensão de arquivo, caso um arquivo não seja encontrado (tipo: mixed, padrão: false).
* **fallthrough** - trata erros de clientes como solicitações não tratadas ou encaminha em erro ao cliente (tipo: boolean, padrão: true).
* **immutable** - impede que clientes com suporte façam solicitações condicionais durante a vida da maxage (tipo: boolean, padrão: false).
* **index** - envia arquivo de índice do diretório.
* **lastModified** - define com última data de modificação do arquivo (tipo: boolean, padrão: true).
* **maxAge** - define a propriedade maxage (tipo: number, padrão: 0).
* **redirect** - redireciona para “/” quando o nome do caminho for um diretório (tipo: booleano, padrão: true).
* **setHeaders** - configura cabeçalhos HTTP para servir com o arquivo (tipo: função).

**dotfiles: valores possíveis**.

1. **allow** - sem tratamento especial.
2. **deny** - nega solicitação de arquivo de ponto e retorno 403.
3. **ignore** - retorna 404.

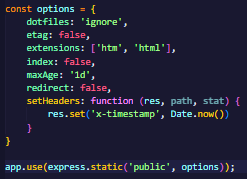
**fallthrough** - sempre definir como true para mapeamento de vários diretórios físicos para o mesmo endereço da web ou para que as rotas preencham os arquivos inexistentes.

**OBS: utilize false caso haja um único caminho projetado para ser estreitado o único diretório do sistema de arquivos**.

**setHeaders** - especifica função para definir cabeçalhos de resposta personalizada.

* Alterações nos cabeçalhos devem ocorrer de forma síncrona.
* fn(res, path, stat) - res (objeto de resposta), path (caminho do arquivo enviado) e stat (objeto do arquivo enviado).

**Exemplo**:



### **Express.text**

**Analisa as cargas de solicitações recebidas em uma string baseada em body-parse**.

* aceita qualquer codificação de Unicode do body.

Uma nova body string contendo os dados analisados é preenchida no request objeto após o middleware ou um objeto vazio é retornado.

**Propriedades opcionais**:

* **defualtCharset** - especifica o conjunto de caracteres padrão para o conteúdo de texto (Tipo: string, padrão: utf-8).
* **inflate** - ativa ou desativa o manuseio de bodys comprimidos (Padrão: true, tipo: booleano).
* **limit** - controla o tamanho máximo do corpo da solicitação (padrão: “100kb”).
* **type** - determina o tipo de mídia que será analisada (Padrão: text/plain).
  + se for uma função, será chamada como fn(req) e a solicitação é analisada se retornar verdadeiro.
* **verify** - possui verify(req, res, buf e enconding) (tipo: função).

### **Express.urlencoded**

**Analisa solicitações recebidas com cargas codificadas em URL** e é baseado em body-parser.

* retorna um middleware.
* aceita apenas a codificação UTF-8.

Uma novo body objeto contendo os dados analisados é preenchida no request objeto após o middleware ou um objeto vazio é retornado.

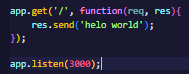
**Nota: este objeto conterá apenas pares chave-valor, onde o valor pode ser uma string ou array**.

**Propriedades opcionais**:

* **extended** - permite escolher analisar os dados como queryStringLib (false) ou queryStringLib (true) (Tipo: booleano, padrão true).
  + permite que objetos e matrizes ricos sejam codificados no formato codificado por URL.
* **inflate** - ativa ou desativa o manuseio de bodys comprimidos (Padrão: true, tipo: booleano).
* **limit** - controla o tamanho máximo do corpo da solicitação (padrão: “100kb”).
* parameterLimit.
* **type** - determina o tipo de mídia que será analisada (Padrão: application/x-www-form-urlencoded).
  + se for uma função, será chamada como fn(req) e a solicitação é analisada se retornar verdadeiro.
* **verify** - possui verify(req, res, buf e enconding) (tipo: função).

# 2. Aplicação

**Exemplo**:



**Algumas funcionalidade**:

* roteamento de solicitação HTTP.
* configuração de middleware.
* renderização de visualização html.
* Registro de mecanismo de modelo.

**Nota: esté objeto de aplicação pode ser referenciado a partir do objeto de solicitação e do objeto de resposta**.

## **2.1 Propriedades**

### **app.locals**

Variáveis locais dentro da aplicação que estarão disponíveis em templates renderizados com res.render.

**Exemplo**:



**O valor das propriedades persistem durante toda a vida da aplicativo**, diferentemente das propriedades de res.locals, que persiste até a vida útil da solicitação.

**req.app.locals** - disponibiliza variáveis locais.

* útil para o fornecimento de funções auxiliares aos modelos.
* bem como dados em nível de aplicativo.

### **app.mountPath**

Contém um ou mais padrões de caminho nos quais um sub-aplicativo foi montado.

**sub-aplicativo** - instância que pode ser usada para tratar a solicitação de uma rota.

**Exemplo**:



**req.baseURL retorna o caminho da url correspondente, em vez dos padrões correspondentes**.

Se existir vários padrões de caminhos, **mountpath retorna uma lista de padrões nos quais ele está montado**.

## **2.2 Eventos**

### **app.on**

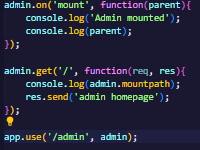
Disparado em um subaplicativo, quando montado em um aplicativo pai.

* aplicativo pai é passado para função de retorno.

**Nota: subaplicativos não herdam os valores das configurações que possuem o valor padrão**.

* valor deve ser definido no subaplicativo.

**Exemplo**:



## **2.3 Métodos**

### **app.all**

Semelhante aos métodos app.method() padrão, exceto que corresponde a todos os verbos HTTP.

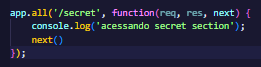
**Argumentos**:

1. **path** - caminho para qual a função middleware é chamada.
2. **callback** - funções de retorno de chamada.
   * pode ser um ou vários middlewares separados por vírgula.

**Nota: next() ignora os retornos de chamadas de rota restantes**.

* usado para passar controle para rotas subsequentes após analisar pré-condições.

**Exemplo: executado para solicitações /secret usando get, post, put e delete**.



**Útil para mapear a lógica global para prefixos de caminho específicos ou correspondências arbitrárias**.

**Nota: se colocado no topo de todas as outras definições, será necessário que todas as rotas daquele ponto em diante exijam autenticação e carreguem automaticamente o usuário**.

**loadUser** - executa uma tarefa e em seguida chama next() para continuar as correspondências de rotas.

**Exemplo: restrição de todos os caminhos**.



**Exemplo: restrição de caminhos específicos**.



### **app.delete**

Roteia solicitações HTTP Delete para caminho especificado com as funções de retorno de chamadas especificadas.

**Argumentos: mesma ideia de app.all**.

1. **path**.
2. **callback**.

**Exemplo**:



### **app.disable**

**Define a configuração name como false**.

* name - propriedade da tabela de configuração do aplicativo.

**Exemplo**:



**Nota: utilize app.disabled(name) para retornar true se a configuração booleana estiver desativada**.

**app.enable(name)** - define a configuração booleana como true.

**Nota: utilize app.enabled(name) para retornar true se a configuração booleana estiver desativada**.

### **app.engine**

Registra mecanismo de modelo fornecido como callback como ext.

**Exemplo**:



**método para mecanismos que não fornecidos .\_\_express imediatamente**.



**ou se desejar mapear uma extensão diferente para o mecanismo do modelo**.

* este forma cria um alias para ejs.\_\_express internamente.
* fornece mesmo assinatura que o express (path, options, callback).

**consolidate.js** - mapeia o mecanismo do modelo node para seguir as convenções.

**Exemplo**:



### **app.get**

Retorna um valor de name, onde name é uma string na tabela de configuração do aplicativo.

#### **1. app.get(path, callback[, callback])**

roteia solicitações http get para caminho especificado com as funções de retorno de chamada especificada.

**Argumentos**:

1. **path**.
2. **callback**.

**Exemplo**:



### **app.listen**

Inicia um soquete Unix e escuta conexões no caminho especificado.

#### **1. app.listen(port, host, backlog, callback)**

Vincula e escuta conexões no host e na porta especificada.

**Nota: se port for omitido ou 0, o sistema operacional atribuirá uma porta arbitrária não utilizada**.

* útil para tarefas automatizadas.

**Exemplo**:



**listen() - método que retorna um objeto http.server**.

**app** - função javascript, projeta para ser passada aos servidores http do node como um retorno de chamada para lidar com solicitações.

### **app.method**

roteia uma solicitação http, onde method é o método http (GET, POST, PUT, DELETE) da solicitação.

**Argumentos**:

1. **path**.
2. **callback**.

**Métodos de roteamento**:

* checkout, copy, delete, get, head, lock e merge.
* mkactivity, mkcol, move, m-search, notify, options e patch.
* post, purge, put, report, search, subscribe, trace e unlock.
* unsubscribe.

**Nota: para rotear métodos inválidos, use a notação de colchetes, por exemplo: app[ ‘ m-search ’ ](path, callback)**.

**OBS: app.get é chamado automaticamente para o método head**.

### **app.params**

Gatilhos de retorno de chamadas aos parâmetros de rota, onde name é o nome parâmetro ou matriz e callback é a função de retorno de chamada.

**Sequências de parâmetros de função**:

1. objeto de solicitação (resquest).
2. objeto de resposta (response).
3. próximo middleware (next).
4. valor do parâmetro.
5. nome do parâmetro.

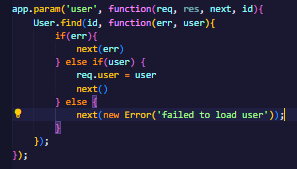
**OBS: se name for array, callback trigger é registrado para cada parâmetro nele cadastrado**.

**Para cada parâmetro (exceto o último)**, next é chamado dentro do retorno da chamada para o próximo retorno da chamada do próximo parâmetro declarado.

* para o último parâmetro, next() chamará o próximo middleware.

**Se user estiver no caminho de uma rota**, pode-se mapear a lógica de carregamento do usuário para fornecer automaticamente req.user à rota ou realizar validações na entrada do parâmetro.

**Exemplo**:



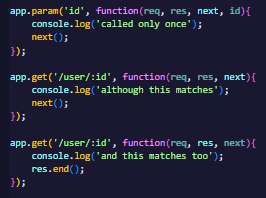
**funções de retorno de chamada de parâmetro** - locais para o roteador no qual estão definidas.

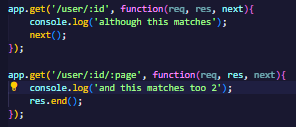
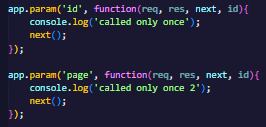
* não herdados por roteadores ou aplicativos montados.
* somente acionados apenas por parâmetros de rotas definidos em app.routes.

**OBS: retorno de chamada de parâmetro são chamados antes de qualquer manipulador de qualquer rota em que o parâmetro ocorra**.

* chamados apenas uma vez (ciclo de solicitação e resposta).

**Exemplo**:

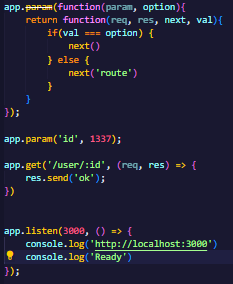




**Nota: uma função de alteração pode ser passada ao param para alterar o método, onde ela pode receber dois parâmetros e retornar um middleware**.

* parâmetros: primeiro - nome do parâmetro de URL; e o segundo - objeto javascript usado para retornar a implementação do middleware.
* middleware determina o comportamento depois que o parâmetro é capturado.

**Exemplo: assinatura é modificada para app.param(name, accessID)**.



**OBS: app.path() pode ser usado para retornar o caminho canônico do aplicativo, mas não é tão indicado a utilização em virtual de sua complexidade**.

* rep.BaseUrl pode ser usado para obter o caminho mais facilmente.

**Exemplo**:



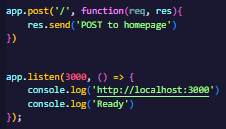
### **app.post e app.put**

Roteia solicitações HTTP post ou put para o caminho especificado com as funções de retorno de chamada especificados.

**Argumentos**:

1. path - caminho para a qual a função é chamada.
2. callback - funções de retorno de chamada.

**Exemplo**:



### **app.render**

Retorna um html renderizado de uma visualização por meio de um callback.

* aceito objeto contendo variáveis locais para a visualização.

**OBS: diferentemente de res.render, esté método não pode mandar a visualização renderizada por conta própria ao cliente**.

**Exemplo**:

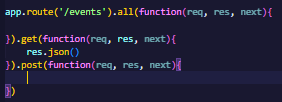


### **app.route**

Retorna instância de rota única, usada para lidar com verbos http com middleware opcional.

* usado para evitar nomes de rotas duplicadas.

**Exemplo**:



### **app.set**

**Nota: name pode ser usado para configurar o comportamento do servidor**.

**Exemplo: recupera o valor de uma configuração com app.get()**.



#### **1. Configuração do aplicativo**

**OBS: subaplicativos não herdam o valor das configurações padrões**.

* exceto por trust proxy que será herdado com o valor padrão.

**Propriedades**:

* case sensitive routing - distinção entre maiúsculas e minúsculas.
  + tipo: booleano.
* env - modelo de ambiente.
* etag - define cabeçalhos de resposta etag.
  + tipo: variado.
* jsonp callback name - nome de retorno da chamada JSON padrão.
* json escape - habilite o escape de resposta JSON das APIs, escapando dos caracteres <> e &.
  + tipo: booleano.
* json replace.
* query parser - analisador de consulta.
  + pode ser ativado para simples ou estendido ou função de análise de string.
  + analisador simples - baseado no Node.
  + analisador estendido - baseado em qs.
  + função de análise - recebe string de consulta completa e retorna um objeto de chaves de consultas e seus valores.
* strict routing - roteamento restrito.
  + tipo: booleano.
* subdomain offset.
* trust proxy.
* views - diretório ou matriz de diretórios para a visualização de aplicativos.
  + tipo: string ou matriz.
* view cache - cache de compilação do modelo de visualização.
* view engine - extensão do mecanismo padrão a ser usado quando omitida.
* x-powered-by.

#### **2. Configuração de trust proxy**

**Tipo**:

* booleano.
  + se true - endereço IP do cliente entendido como entrada mais a esquerda do x-forwarded.
  + se false (Padrão) - app entendido como voltado para internet e endereço ip do cliente é voltado para req.connection.remoteAddress.
* string ou matriz de strings - pode ser endereço IP, sub-rede ou uma matriz de endereços IP e sub-redes a confiar.
  + sub-redes pré-configuradas: loopback, linklocal e local único.

**Exemplo: formas de configuração dos endereços IP**.  


**OBS: quando especificado, IPs e sub-redes são excluídos do processo de determinação de endereço e endereço IP não confiável mais próximo do app é determinado como endereço IP do cliente**.

* number.
* function - implementação de confiança personalizada.

**Exemplo**:  


#### **3. Configuração de Etag**

**OBS: essas configurações funcionam apenas em arquivos dinâmicos, arquivos estáticos ignoram essas configurações**.

**OBS: funcionalidade implementada usando o pacote etag**.

**Tipo**:

1. booleano.
2. string.
3. function

### **app.use**

Monta função ou funções de middlewares especificadas no caminho especificado.

* middleware executado quando base de caminho solicitado corresponde a path.

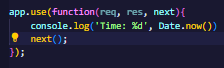
**Argumentos**:

1. path.
2. callback.

Uma rota corresponderá a qualquer caminho que siga o seu caminho imediatamente com um “/”, por exemplo: app.use(‘/apple’, …), corresponde a “/apple”, “/apple/image” e assim por diante.

* Um middleware sem caminho será solicitado para cada solicitação.

**Exemplo: middleware executado a cada solicitação ao aplicativo**.

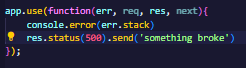


**Nota: as funções de middleware são executadas sequencialmente, portanto a ordem de inclusão é importante**.

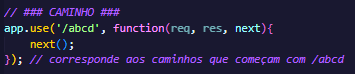
#### **1. Middleware de tratamento de erros**

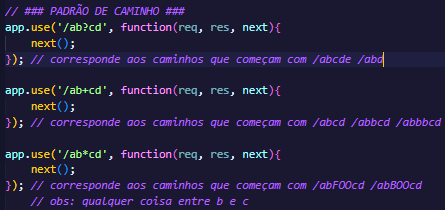
Este tipo sempre deve levar quatro argumentos para ser identificado como função para tratamento de erros (obrigatório).

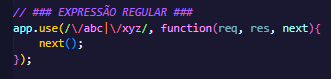
* caso os 4 argumentos não sejam inseridos, o middleware será considerado como regular.

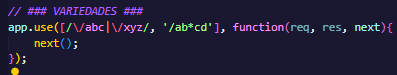
**Exemplo: tratamento de erros**.  


**Exemplo: caminhos válidos para middleware**.

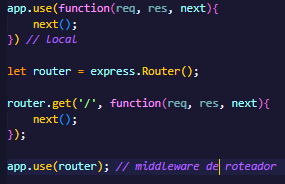








**Exemplo: funções de retorno de chamada de middleware**.



**OBS: mais de uma função de middleware pode ser especificada no mesmo caminho, por exemplo: app.use(router1, router2) - série de middleware ou app.use([router1, router2]) - variedade**.

* todas as formas podem ser usadas para montar um middleware.

**Exemplo: uso do middleware express.static**.



caso existe outros middleware após este, o primeiro a ser executado será o public em virtude de precedência.

**Exemplo: desativando o recurso do logger**.



# 3. Request

Representa a solicitação http e possui propriedades para strings de consulta da solicitação, parâmetros, corpo e cabeçalhos http.

* Retorno de chamada.

**Exemplo**:



## **3.1 Propriedades**

### **req.app**

Contém a referência à instância do aplicativo Express que está usando o middleware.

**Exemplo: middleware acessando instância Express via req.app**.



### **req.baseUrl**

Caminho da URL no qual a instância do roteador foi montada.

**Nota: tem a mesma funcionalidade de mountpath, a única diferença é que mountpath retorna o padrão do caminho correspondente**.

**Exemplo**:





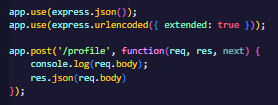
Mesmo com padrão de caminho ou conjunto de padrões, a baseUrl retornará a string correspondente.

### **req.body**

Contém pares de chave-valor de dados enviados no corpo da solicitação.

**Nota: undefined é preenchido quando você usa um middleware de análise corporal, como express.json() ou express.urlencoded()**.

**Exemplo: middleware de análise corporal**.



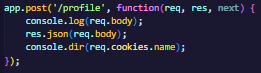
### **req.cookies**

Propriedade que representa um objeto que contém cookies enviados pela solicitação.

**OBS: se não contiver cookies o padrão será vazio**.

**Nota: se o cookie foi assinado, utilize req.signedCookies**.

**Exemplo**:



### **req.fresh**

Retorna um valor booleano dependendo da situação do cache do client, ou seja, retorna true caso haja resposta no cache do client e false indica que o cache está obsoleto e a resposta completa deve ser enviada.

No envio de cache-control para indicar uma solicitação de recarga ponta a ponta, um valor false será retornado para tornar transparente o tratamento da solicitação.

* envio feito no cache do cabeçalho.

**Exemplo**:



### **req.hostname**

Contém o nome do host derivado do host cabeçalho http.

Se trust proxy não for false, a propriedade obterá o valor x-forwarded-host campo de cabeçalho.

* cabeçalho pode ser definido pelo cliente ou pelo proxy.
  + se houver mais de um cabeçalho, será utilizado o valor do primeiro cabeçalho.

**Exemplo**:



### **rep.ip**

Contém o endereço IP remoto da solicitação.

Se a configuração trust proxy não for false, o valor do propriedade é derivada da entrada mais à esquerda no x-forwarded-for cabeçalho.

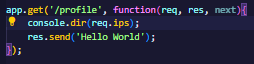
* pode ser definido pelo cliente ou pelo proxy.

**exemplo**:



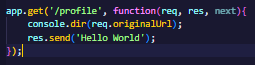
**Nota: req.ips pode ser utilizado para obter uma matriz de IPs especificados no x-forwarded-for cabeçalho da solicitação**.

**Exemplo**:



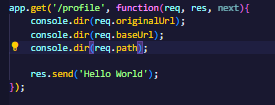
### **req.originalUrl**

Mantém o URL de solicitação original, permitindo reescrever req.url livremente para fins de roteamento interno.

**Exemplo**:  


**Nota: esta propriedade de requisição está disponível em objetos de middleware e roteador e é uma combinação de req.baseUrl e req.url**.

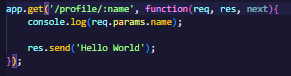
**Exemplo**:



### **req.params**

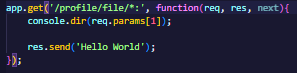
Objeto que contém propriedades mapeadas para os parâmetros da rota nomeada.

**Exemplo: rota será definida como /user/:name**.



**Nota: quando a utilização de expressão regulares na definição da rota, os grupos de captura são fornecidos na matriz usando req.params[n]**.

**Exemplo**:



app.param pode ser usado para alterar uma chave em req.params.

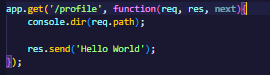
* estás alterações são aplicáveis apenas nos parâmetros já definidos no caminho da rota.

**OBS: qualquer manipulação feita em req.param seja no middleware ou manipulador de rotas, será redefinida**.

### **req.path**

Contém a parte do caminho da URL da solicitação.

**Exemplo**:

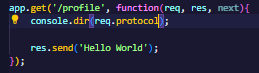


### **req.protocol**

Contém a string do protocolo de solicitação, seja ele http ou https.

**OBS: se a configuração trust proxy não for false, a propriedade usará o valor do x-forwarded-proto**.

**Exemplo**:



### **req.query**

Objeto que contém uma propriedade para cada parâmetro de string de consulta na rota.

* Se o analisador de consulta estiver desativado, o objeto estará vazio.

**OBS: como está propriedade é baseada na entrada do usuário, todas as propriedades e valores neste objeto devem ser validadas**.

**Nota: valor de propriedade pode ser configurado na configuração do aplicativo analisador de consulta**.

* qsmodule - analisador de string de consulta usado por padrão.

**Exemplo**:

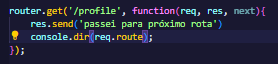




### **req.route**

Contém a rota atualmente correspondida, uma string.

**Exemplo**:



### **req.secure**

Propriedade booleana que será verdadeira se uma conexão TLS for estabelecida.

**Exemplo**:



### **req.signedCookies**

Propriedade que contém cookies assinados enviados pela solicitação, não assinados e prontos para uso.

* somente ao utilizar um middleware de cookies.

**OBS: os cookies assinados residem em um objeto diferente para mostrar a intenção do desenvolvedor, para evitar ataques maliciosos na requisição de cookies e valores**.

Nota: se nenhum cookie assinado for enviado, o padrão da propriedade será { … }.

**Exemplo**:



### **req.stale**

Indica se a solicitação está obsoleta.

**Exemplo**:



### **req.subdomains**

Representa uma matriz de subdomínios no nome do domínio da solicitação.

**Exemplo**:



subdomain offset (padrão 2) - usada para determinar o início dos segmentos do subdomínio.

* o comportamento pode ser alterado usando app.set.

### **req.xhr**

Propriedade booleana que pode indicar se o campo de cabeçalho x-request-with for XMLHttpRequest, retornando true.

* Indica que a solicitação for emitida por uma biblioteca cliente.

**Exemplo**:



## **3.2 Métodos**

### req.accepts

Verifica se os tipos de conteúdo especificados são aceitáveis a partir do campo accept do cabeçalho http da solicitação.

**Tipos disponíveis**:

1. string MIME.
2. nome de extensão.
3. lista.

**Exemplo**:

| req.accepts('html'); //text/html req.accepts(['json', 'text']); //text/html req.accepts('application/json'); //json |
| --- |

**Requisições Accepts**:

* acceptsCharsets - retorna o primeiro charset aceito dos conjuntos de caracteres especificados.
* acceptsEncoding - retorna a primeira codificação aceita das codificações especificadas.
* acceptsLanguages - retorna o primeiro idioma aceito dos idiomas especificados.

### req.get

Retorna o campo de cabeçalho de solicitações HTTP especificado.

**Exemplo**:

| req.get('Content-type') //"text/plain" |
| --- |

# 4. Response

Resposta enviada quando app recebe uma solicitação HTTP.

**Exemplo**:

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

## **4.1 Propriedades**

### res.headerSent

Propriedade booleana que indica se foi enviado cabeçalhos HTTP do app para resposta.

**Exemplo**:

| app.get('/', **function** (req, res) {  console.dir(res.headerSent); // false  res.send('ok');  console.dir(res.headerSent); //true }) |
| --- |

### res.local

Define variável acessível em modelos renderizados com res.render.

**OBS: variáveis não compartilhadas entre solicitações**.

**Nota: pode ser útil para apresentar o usuário autenticado e configurações do usuário**.

## **4.2 Métodos**

### res.append

Especifica o cabeçalho de resposta HTTP.

* Caso não esteja definido o cabeçalho, ele será criado com o valor especificado.

**Exemplo**:

| res.append('Link', ['<http://localhost/>', '<http://localhost:3000/>']) |
| --- |

### res.cookie

Define cookie name como value.

* o parâmetro pode ser uma string ou objeto JSON.

**Propriedades para options**:

* domain - nome de domínio para o cookie.
* encode - função usada para codificação de valor de cookie.
* expires - data de expiração do cookie.
  + se não definido, cria um cookie de sessão.
* httpOnly - sinaliza o cookie para ser acessível somente pelo servidor web.
* maxAge - define tempo de expiração em relação ao tempo atual.
* path - caminho.
* partitioned - indica que deve ser armazenado usando armazenamento particionado.
* priority - valor do atributo Set-Cookie.
* secure - informa que deve ser usado somente com HTTPS.
* signed - indica se deve ser assinado o cookie.

**Exemplo**:

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

**Nota: vários cookies podem ser definidos para uma única resposta**.

“**clearCookie**” - limpa o cookie especificado por name.

**Exemplo**:

| res.clearCookie('name', { path: '/admin' }); |
| --- |

* deve ter o mesmo valor fornecido para cookie().

### res.download

Transfere o arquivo “path” como um anexo.

**Propriedades para Options**:

* maxAge - define propriedade do cache-control do cabeçalho.
* root - dir raiz para nomes de arquivos relativos.
* lastModified - define o cabeçalho para última data modificada do arquivo no SO.
* headers - contém cabeçalhos HTTP para servir com o arquivo.
* dotfiles - serve dotfiles, com: allow, deny e ignore.
* acceptRanges - aceitação de solicitações de alcance.
* cacheControl - configuração de cabeçalho de resposta.
* immutable - impedi solicitações condicionais durante o maxAge.

**Nota: o método invoca uma função de retorna de chamada para tratamento de erro**.

* deve ser usada para encerrar o ciclo de solicitação ou passar o controle para próxima rota.

**Exemplo**:

| res.download('/report-4321.pdf', 'report.pdf', **function** (err) {  **if** (err) { ... } **else** { ... } }); |
| --- |

### res.end

Finaliza o processo de resposta.

* Usado normalmente para terminar uma resposta rapidamente sem nenhum dado.

**Exemplo**:

| res.send(); res.status(404).end(); |
| --- |

### res.json

Envia uma resposta JSON.

* o parâmetro pode ser qualquer tipo JSON.

**Exemplo**:

| res.json(**null**); res.json({ user: 'tobi' }); res.status(500).json({ err: 'message' }); |
| --- |

### res.location

Define o location cabeçalho HTTP de resposta para path parâmetro especificado.

**Exemplo**:

| res.location('/foo/bar'); res.location('http://exemple.com'); res.location('back'); // value default '/' |
| --- |

### res.redirect

Redireciona para a URL derivada do especificado path, com especificado status.

**Exemplo**:

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

**Nota: os redirecionamento podem ser relativos à raiz do nome do host**.

### res.render

Renderiza um view e envia uma a string HTML renderizada para o cliente.

* **Parâmetros**: locals - objeto que define variáveis locais para visualização; e callback - método retorna o erro possível e a string renderizada.

**Argumento View** - caminho do arquivo de visualização a ser renderizado.

* aceita caminho absoluto ou relativo à views configurações.

**OBS: se path não contiver uma extensão de arquivo, a view engine determinará a extensão do arquivo**.

**Exemplo**:

| // recebendo error res.render('index', **function** (err, html) {  res.send(html); });  //enviando dados res.render('user', { name: 'Tobi' }); |
| --- |

### res.send

Envia uma resposta HTTP.

* O parâmetro enviado pode ser um bufferObjeto, string, objeto, boolean ou array.

**Exemplo**:

| res.send({ some: 'json' }); res.send('<p>some html</p>'); res.status(404).send('Sorry, we cannot find that'); |
| --- |

Executa tarefas úteis para resposta simples, **atribuindo o content-length campo de cabeçalho de resposta HTTP e fornece suporte de atualização**.

### res.sendFile

Transfere o arquivo no dado path.

**Content-Type do cabeçalho HTTP** é definido com base na extensão do nome do arquivo.

* fornece acesso a dados no sistema de arquivos em execução.

**Exemplo**:

| res.sendFile(fileName, options, **function**(err) {  **if**(err){  next(err);  } **else** console.log('Sent: ', fileName); }); |
| --- |

**Nota: possui as mesma propriedades de options existentes em res.download**.

### res.set

Define o cabeçalho HTTP da resposta field como value.

* um objeto pode ser passado para definir vários campos de uma só vez.

**Exemplo**:

| res.set({  'Content-Type': 'text/plain',  'Content-Length': '123' }) |
| --- |

### res.type

Define o Content-Type para o tipo MIME conforme determinado pelo especificado.

**Exemplo**:

| res.type('.html'); // 'text/html' res.type('json'); // 'application/json' |
| --- |

# 5. Roteador

Objeto de instância de middlewares e rotas.

* pode ser passado como argumento para app.use() ou use() de outro roteador.

**Exemplo**:

| // rota de middleware router.use(**function** (req, res, next) {  next(); })  // rota de método HTTP router.get('/events', **function** (req, res, next) { ... }) |
| --- |

## **5.1 Métodos**

### router.all

Corresponde a todos os métodos HTTP.

**Nota: útil para mapear a lógica global para prefixos de caminhos específicos**.

**Exemplo**:

| router.all('/api/\*', requiredAuthentication); |
| --- |