Introducing Express

Objectives

- Review basics of Express.js
- Review design philosophy of the Express.js framework.

Show Demo: Node.js simple static server

- Attempt to not "interpret" URL's but simply serve files found in a directory.
- Try examples:
- 01_example_table.html
- 01_example_table_css.html
- 01_example_table_css_internal.html

- Notice one of them does not work.
- What is going wrong and how do we program around that?
- One of many typical problems that needs to be solved when building a Node.js app from scratch

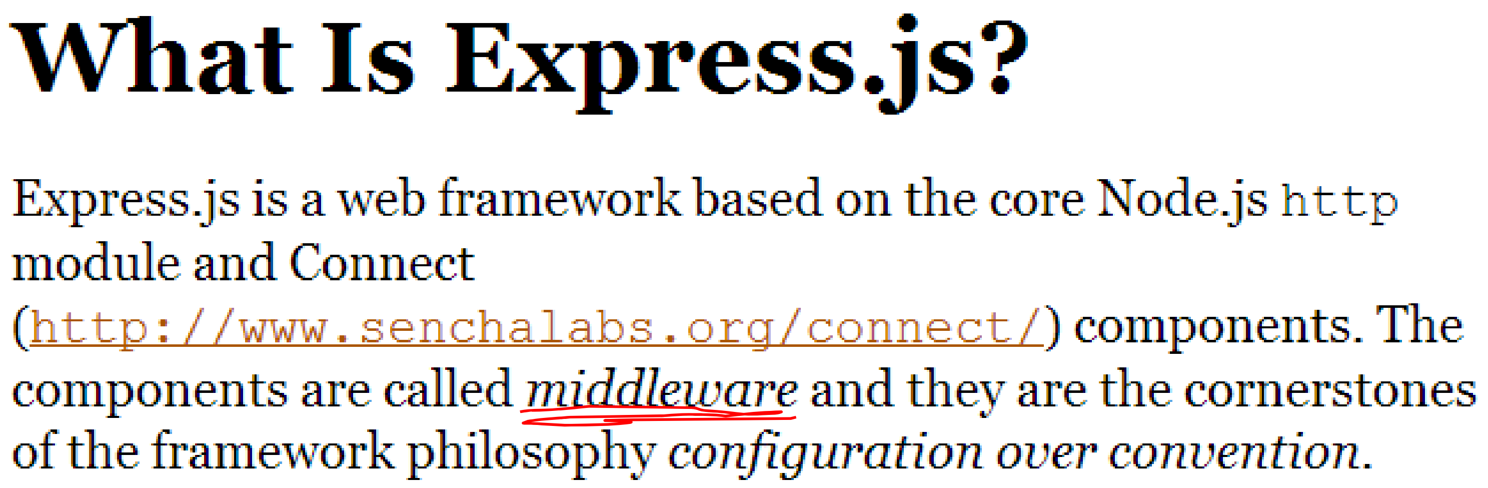
We could:

- Learn how to solve these problems in Node.js
- Use a framework of existing solutions

The Node.js community have contributed many small solutions to problems in the form of modules.

Express.js is one of the most popular frameworks (aggregation of modules with intended architecture or goal)

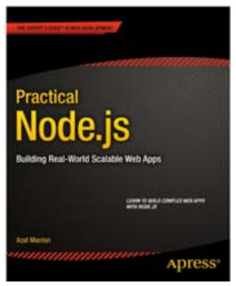
This area is very new and subject to much change



What Is Express.js?

Express.js is a web framework based on the core Node.js `http` module and Connect (`http://www.senchalabs.org/connect/`) components. The components are called *middleware* and they are the cornerstones of the framework philosophy *configuration over convention*.

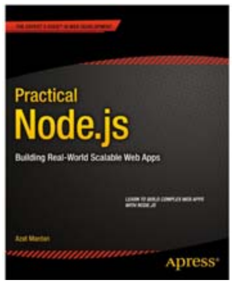
source: Azat Mardan "Practical Node.js" Apress 2014



Common Re-occurring Problems in Node.js

- Parsing of HTTP request bodies
- Parsing of cookies
- Managing sessions
- Organizing routes with a chain of `if` conditions based on URL paths and HTTP methods of the requests
- Determining proper response headers based on data types

source: Azat Mardan "Practical Node.js" Apress 2014



We could:

Study things like cookies and sessions in Node.js by itself or within the context of a framework like Express?

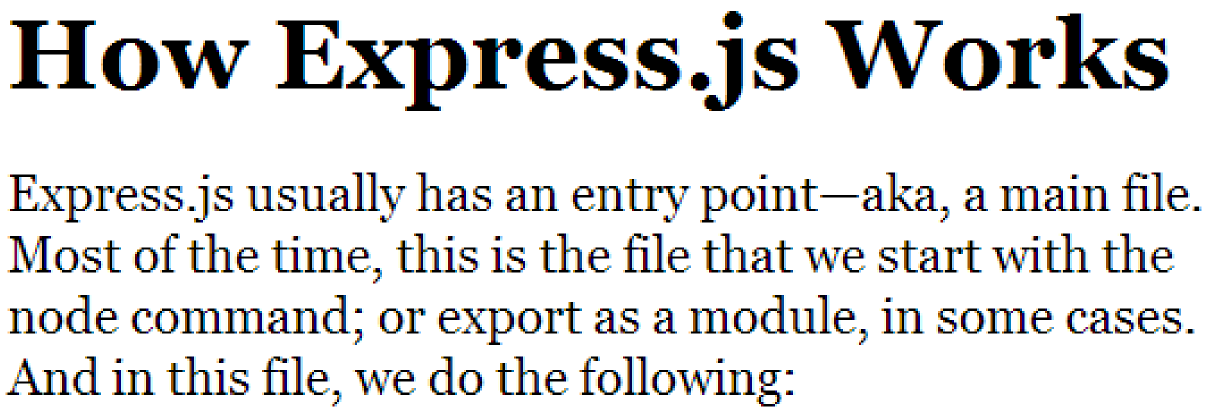
Done by themselves we might see them better in isolation

But

Many problems are overcome by the frameworks

Which is best?

How Express Works

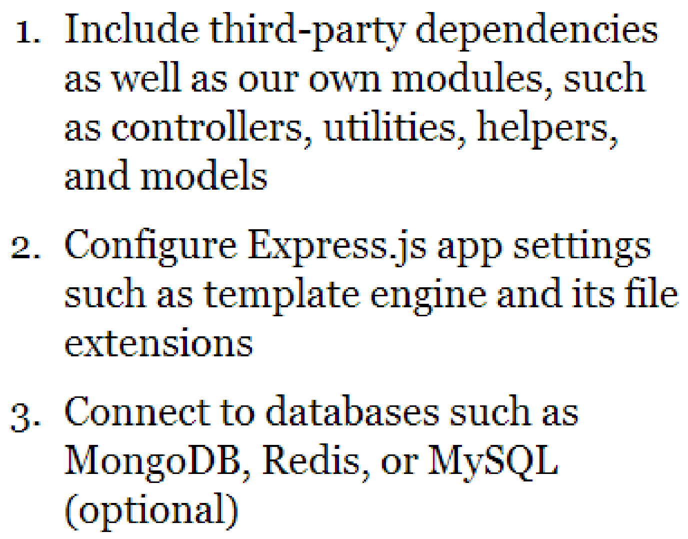


How Express.js Works

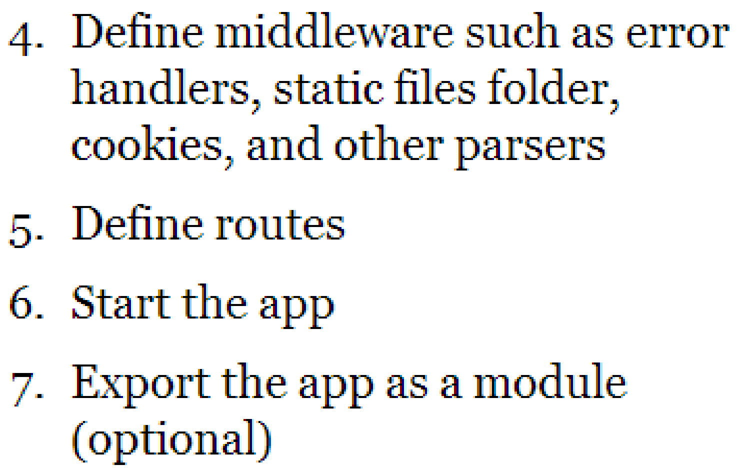
Express.js usually has an entry point—aka, a main file. Most of the time, this is the file that we start with the node command; or export as a module, in some cases. And in this file, we do the following:

source: Azat Mardan "Practical Node.js" Apress 2014

How Express Works



1. Include third-party dependencies as well as our own modules, such as controllers, utilities, helpers, and models
2. Configure Express.js app settings such as template engine and its file extensions
3. Connect to databases such as MongoDB, Redis, or MySQL (optional)



4. Define middleware such as error handlers, static files folder, cookies, and other parsers
5. Define routes
6. Start the app
7. Export the app as a module (optional)

source: Azat Mardan "Practical Node.js" Apress 2014

How Express Works

When the Express.js app is running, it's listening to requests.

Each incoming request is processed according to a defined chain of middleware and routes, starting from top to bottom. This

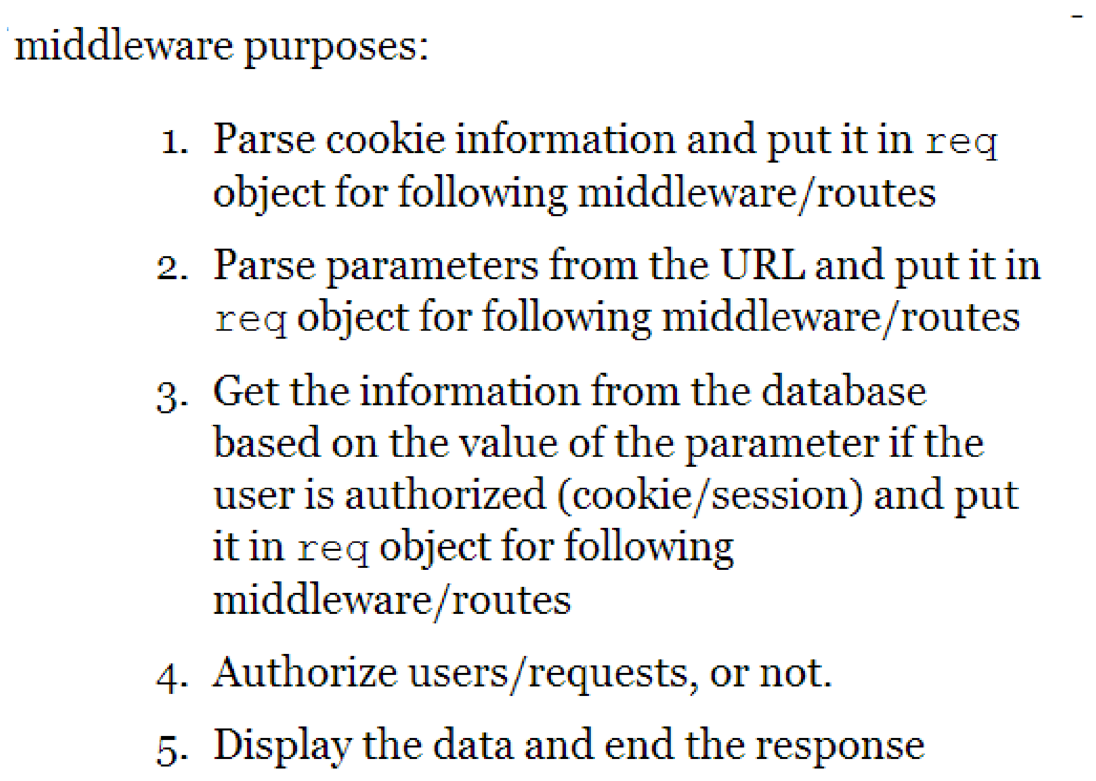
routes/middleware that are higher in the file have precedence over the lower definitions.

This aspect, among others, has changed from Express 3.x to 4.x

We will be illustrating 4.x

source: Azat Mardan "Practical Node.js" Apress 2014

What Does Middleware Do?



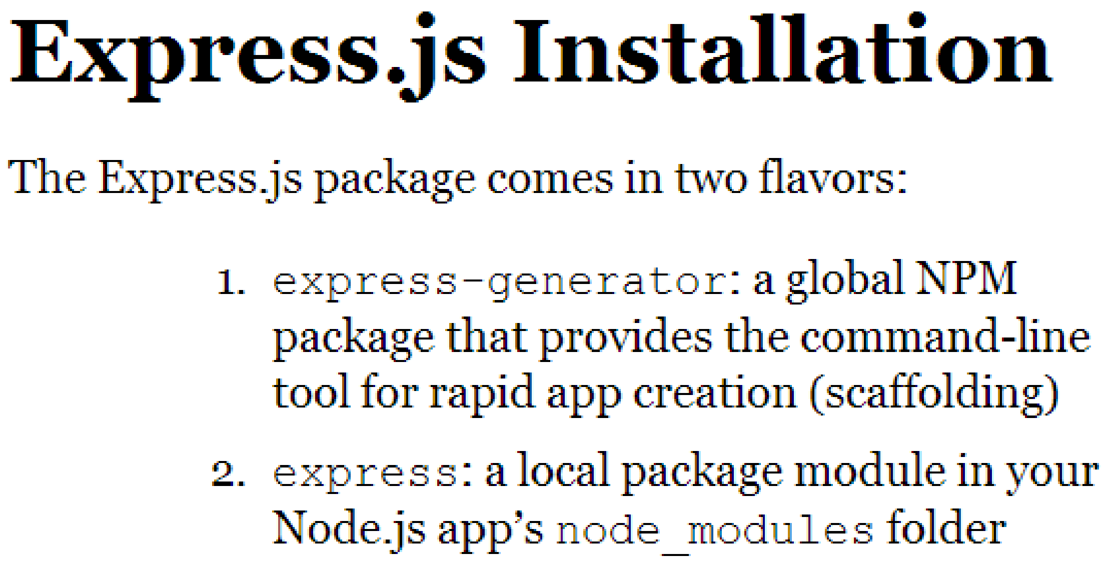
middleware purposes:

1. Parse cookie information and put it in `req` object for following middleware/routes
2. Parse parameters from the URL and put it in `req` object for following middleware/routes
3. Get the information from the database based on the value of the parameter if the user is authorized (cookie/session) and put it in `req` object for following middleware/routes
4. Authorize users/requests, or not.
5. Display the data and end the response

source: Azat Mardan "Practical Node.js" Apress 2014

How Express Works

Express 3.x only had one "bundled" flavour
Express 4.x has express module and express-generator module



Express.js Installation

The Express.js package comes in two flavors:

1. `express-generator`: a global NPM package that provides the command-line tool for rapid app creation (scaffolding)
2. `express`: a local package module in your Node.js app's `node_modules` folder

We will do this install as a tutorial

source: Azat Mardan "Practical Node.js" Apress 2014

Express –The good parts

Minimal: basic features to create web applications, such as

- routing based on URL paths (it has DSL to describe routes),
- support for template engines,
- cookie and session management,
- parsing of incoming requests

Without these features, we would need to create custom solutions in Node HTTP.

The source code for Express is small and comprehensible –unlike big frameworks

Express –The good parts

- Framework does not impose much application structure or database layers etc. (good and bad?)
- Not all middleware is included by default – some must be installed as npm modules
- Unlike other big, monolithic frameworks, you explicitly include only what you want.
- Not quite an Model-View-Controller (MVC) framework, but can easily be made to be one. (does not include the model)

Express –The good parts

With Express you can build different kinds of (web) applications:

- **REST APIs (Representable State Transfer apps)**
- **Single-page and multipage applications,**
- **Real-time applications,**
- **Applications that spawn external processes and output their result**
- **and many others …**

Express –The good parts

Express is popular for rapid prototyping.

Said to be easy to learn.

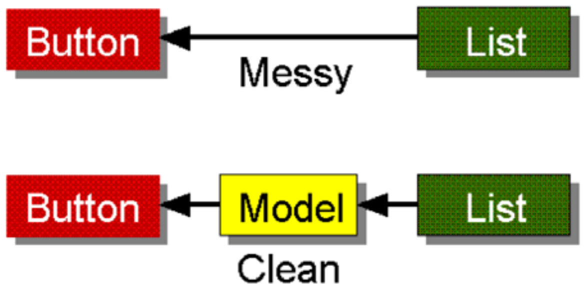
Functions built into Node are still available.

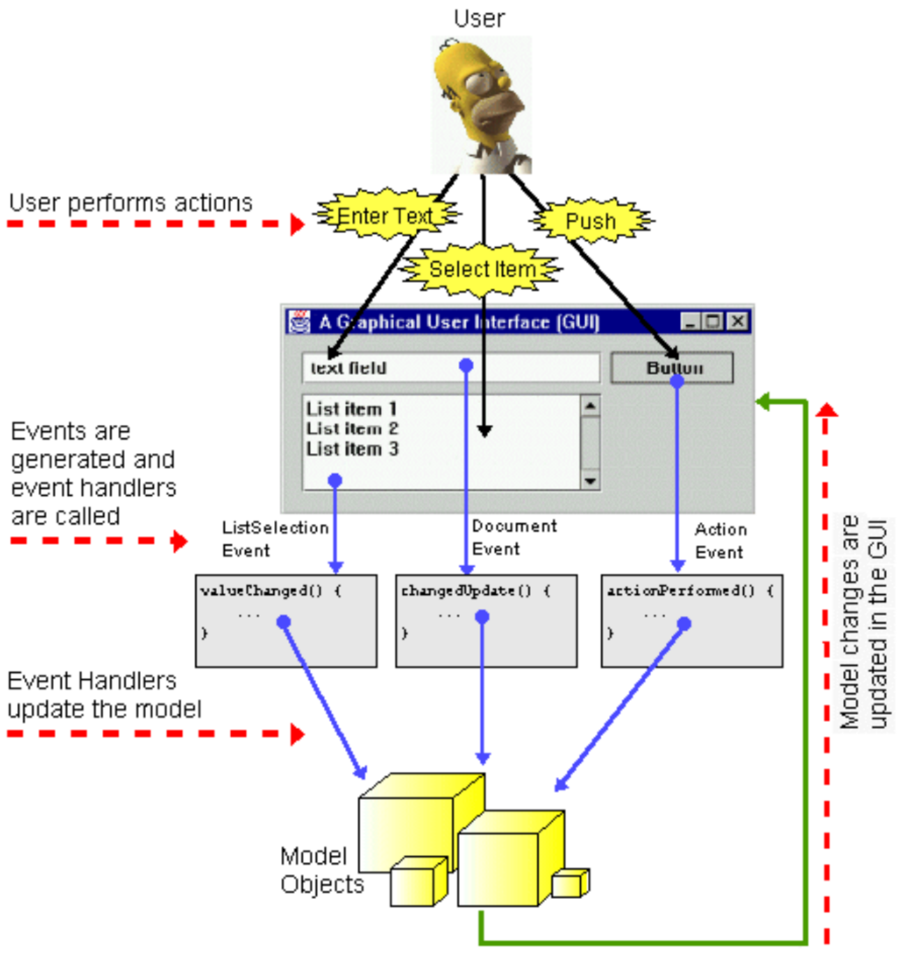
Out-of-the-box performance of Express is very good -thousands of concurrent connections per second.

Express –The good parts

- Express is the most popular web framework for Node.js
- First commit June 2009. (still young)
- Its repository is one of the most watched on GitHub.
- Companies using Express include: MySpace, eBay, Uber, and Mozilla, Paypal, Yahoo, …
- Still very new.

Review: Model-View-Control Pattern from COMP 1406





MVC Often Implemented as "Observer Pattern"

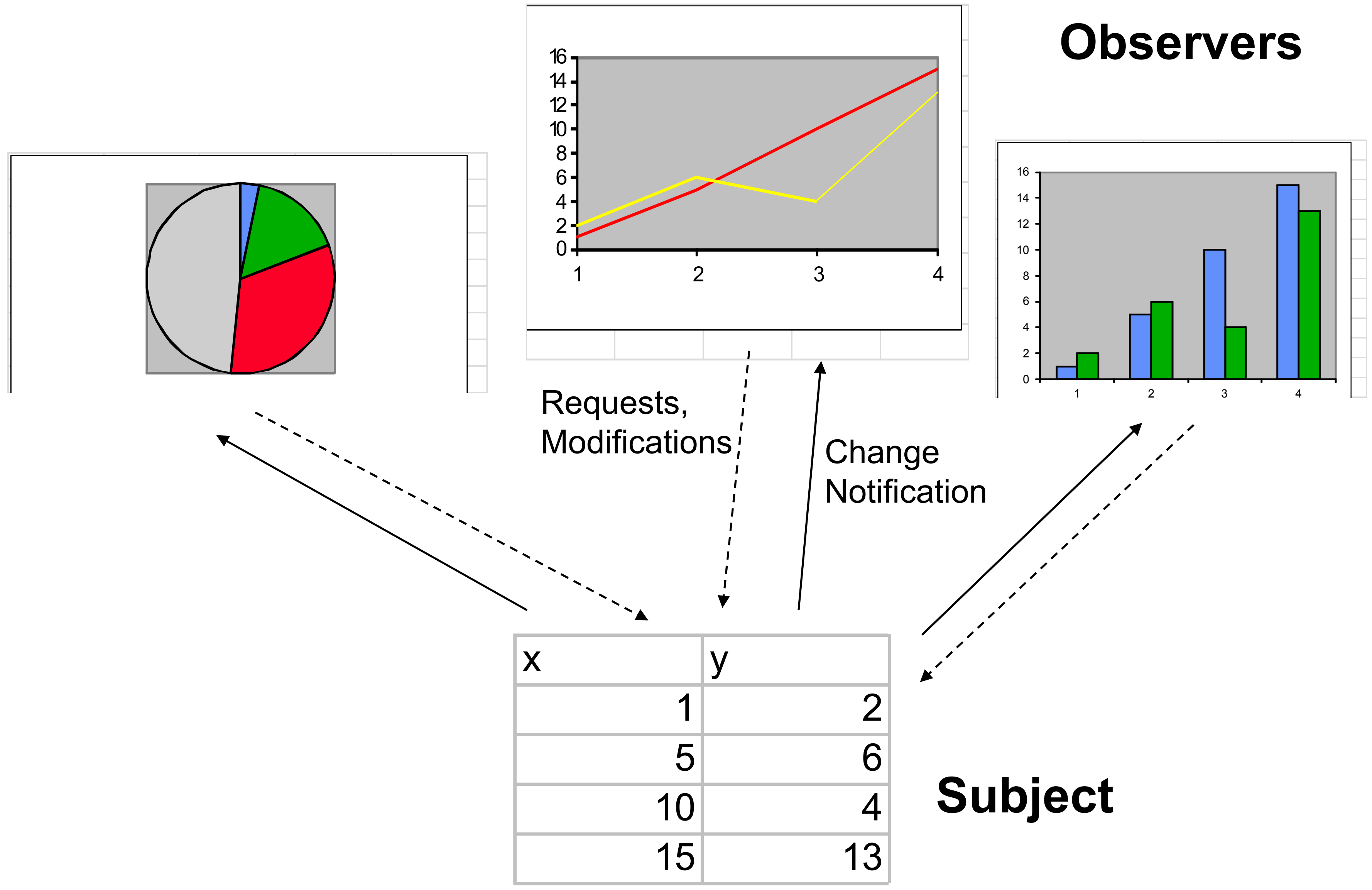
Observer Pattern:

Intent: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically

(Notice the obvious usefulness to GUI and Event driven applications)

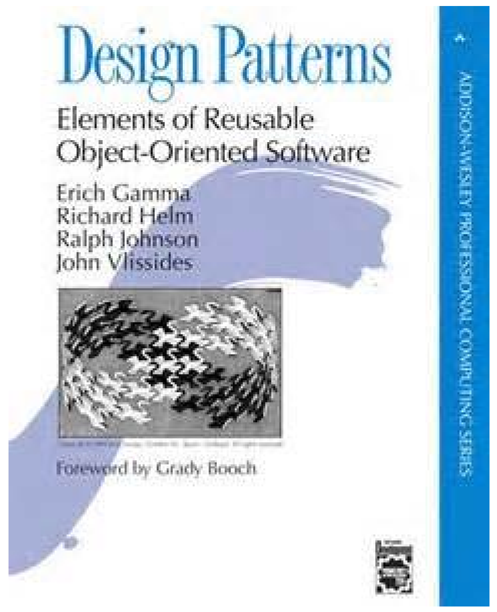
Also known as: Dependents, Publish-Subscribe

Observers



x	y
1	2
5	6
10	4
15	13

One of the Patterns of Gamma et al. [1994]

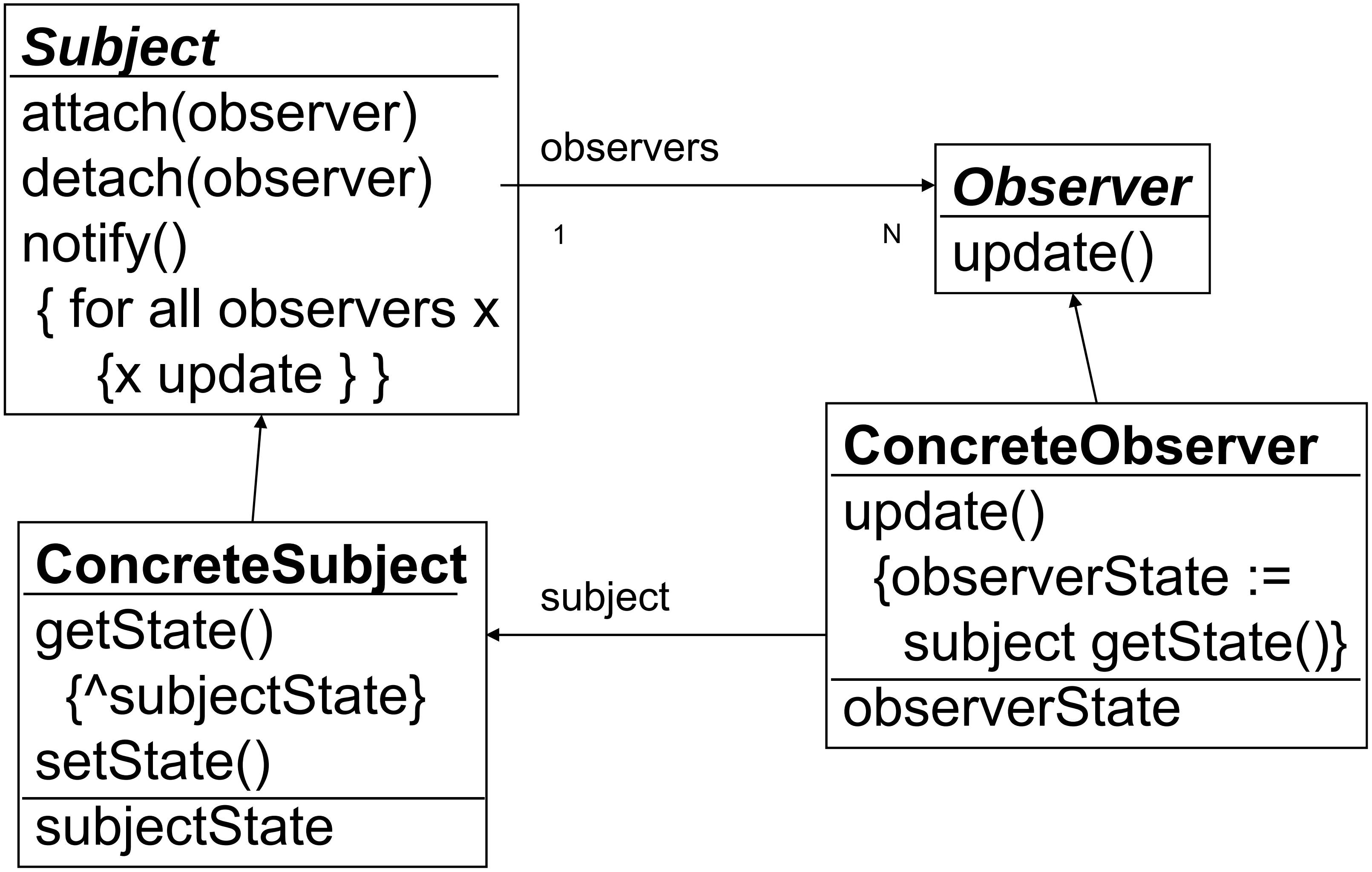

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

- 23 Design Patterns that record good solutions to recurring OO programming problems.
- Patterns meant to capture design expertise.
- Gamma's Patterns deal with general OO design problems and proven solutions.
- Patterns solve small, typical, construction problems not large architecture problems
- Not domain specific
- Have anti-patterns

Observer Pattern Roles: Subjects and Observers

- **Subjects will notify their observers (dependents) whenever the subject changes state; subjects don’t know, or care, who the observers are and what they are watching for.**
- **Observers will register interest in the subject, and query the subject for current state information when notified of a state change**

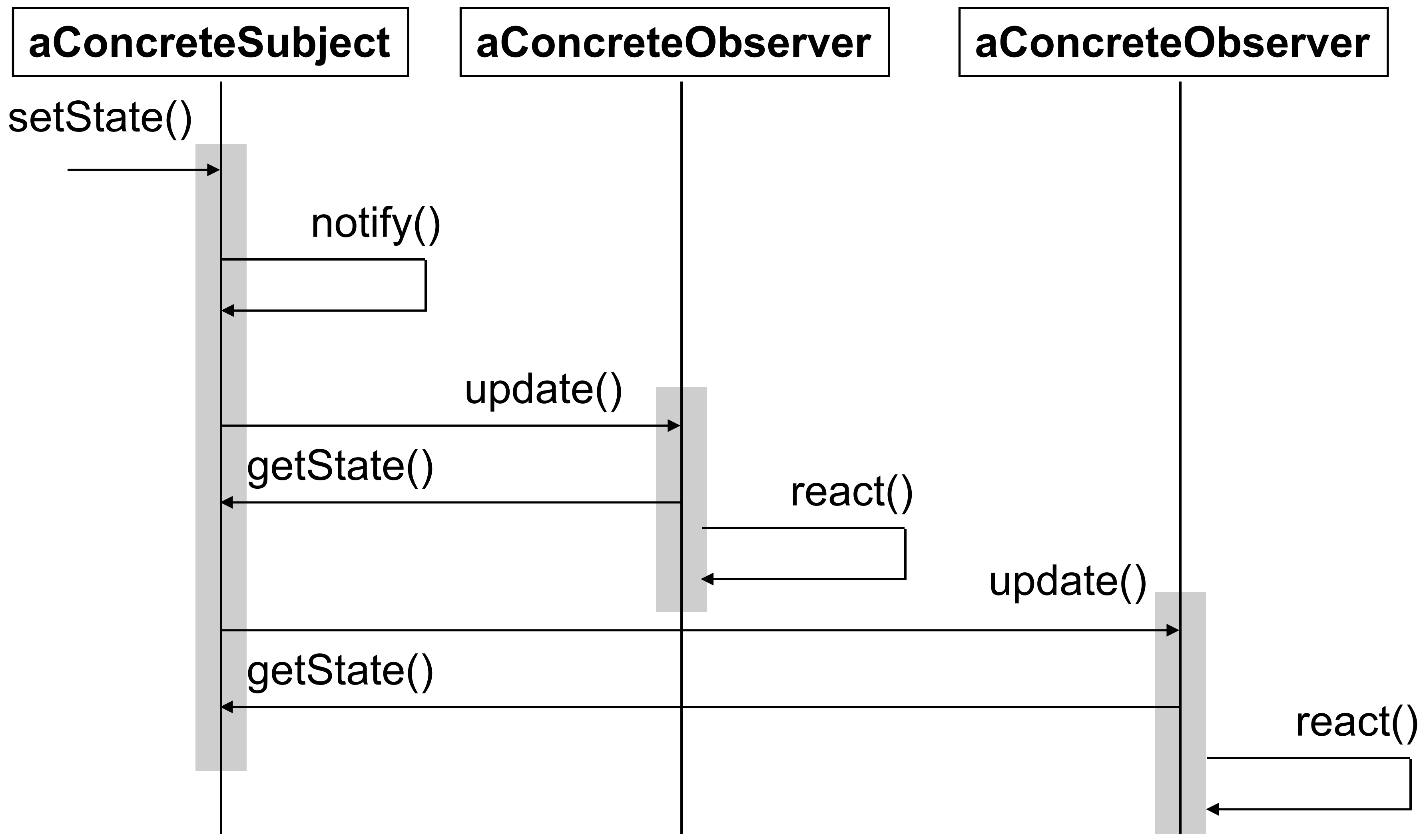
Observer Pattern (OMT Structure)



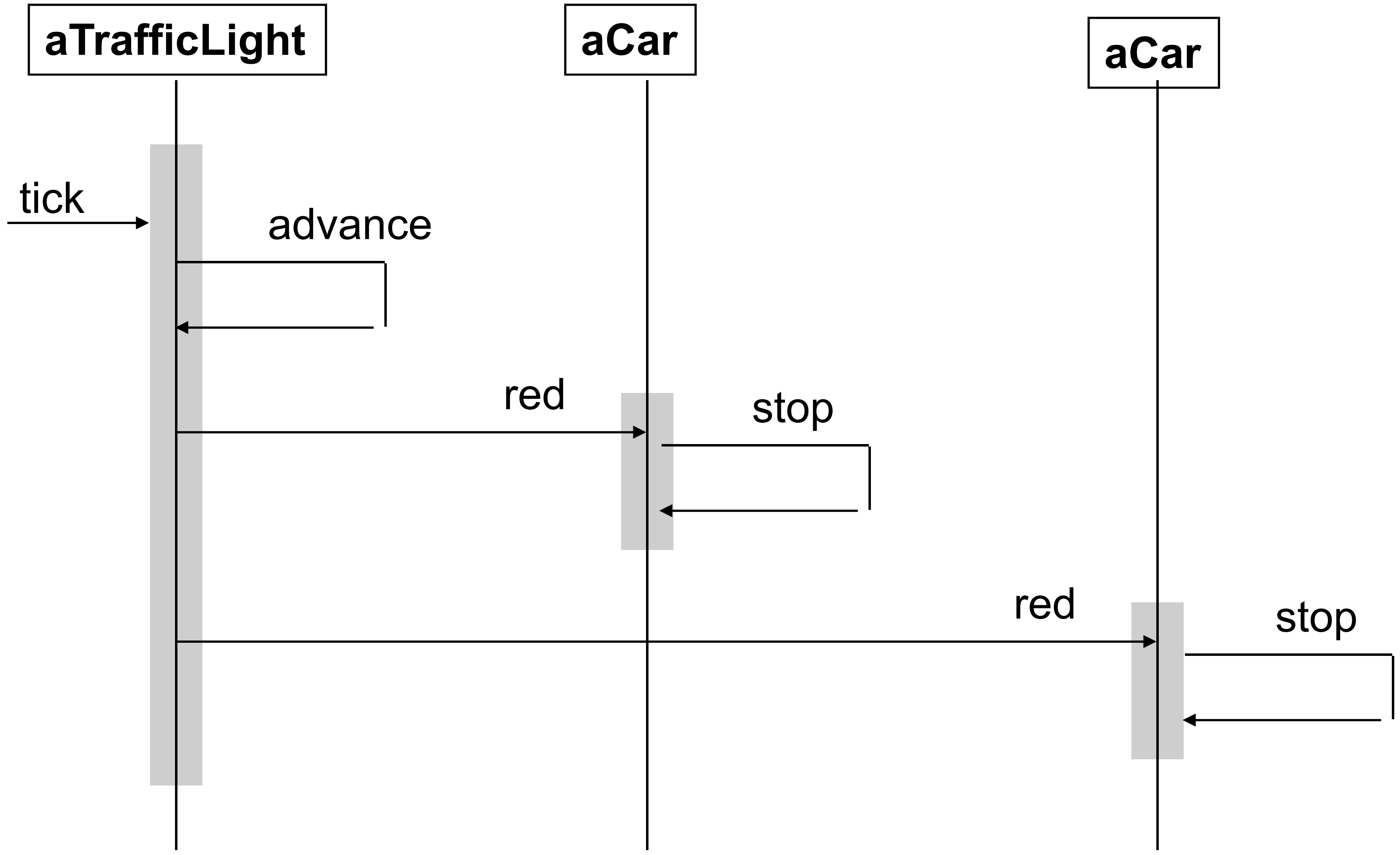
Observer Pattern -Participants

- Subject (Abstract Super Class)
  - -knows it has some number of observers
- Observer (Abstract Super Class)
  - -defines an updating interface for objects that should be notified of changes in a subject
- Concrete Subject
  - -stores state of interest
  - -notifies observers whenever a change occurs that could leave observer inconsistent
- Concrete Observer
  - -maintains reference to concrete subject
  - -stores state at should be consistent with subjects
  - -implements the Observer updating interface to keep its state consistent with the subjects

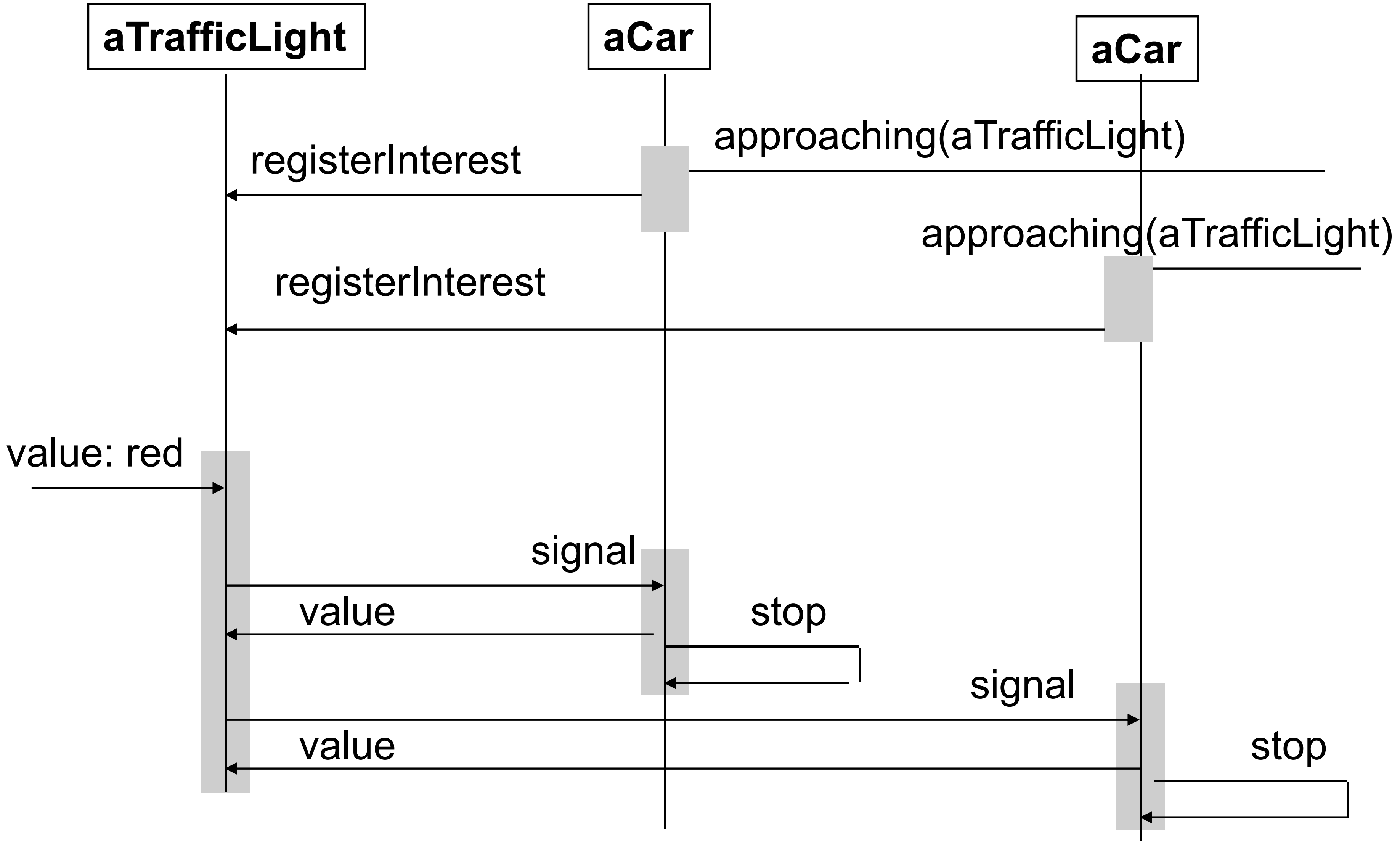
Observer Pattern Collaborations



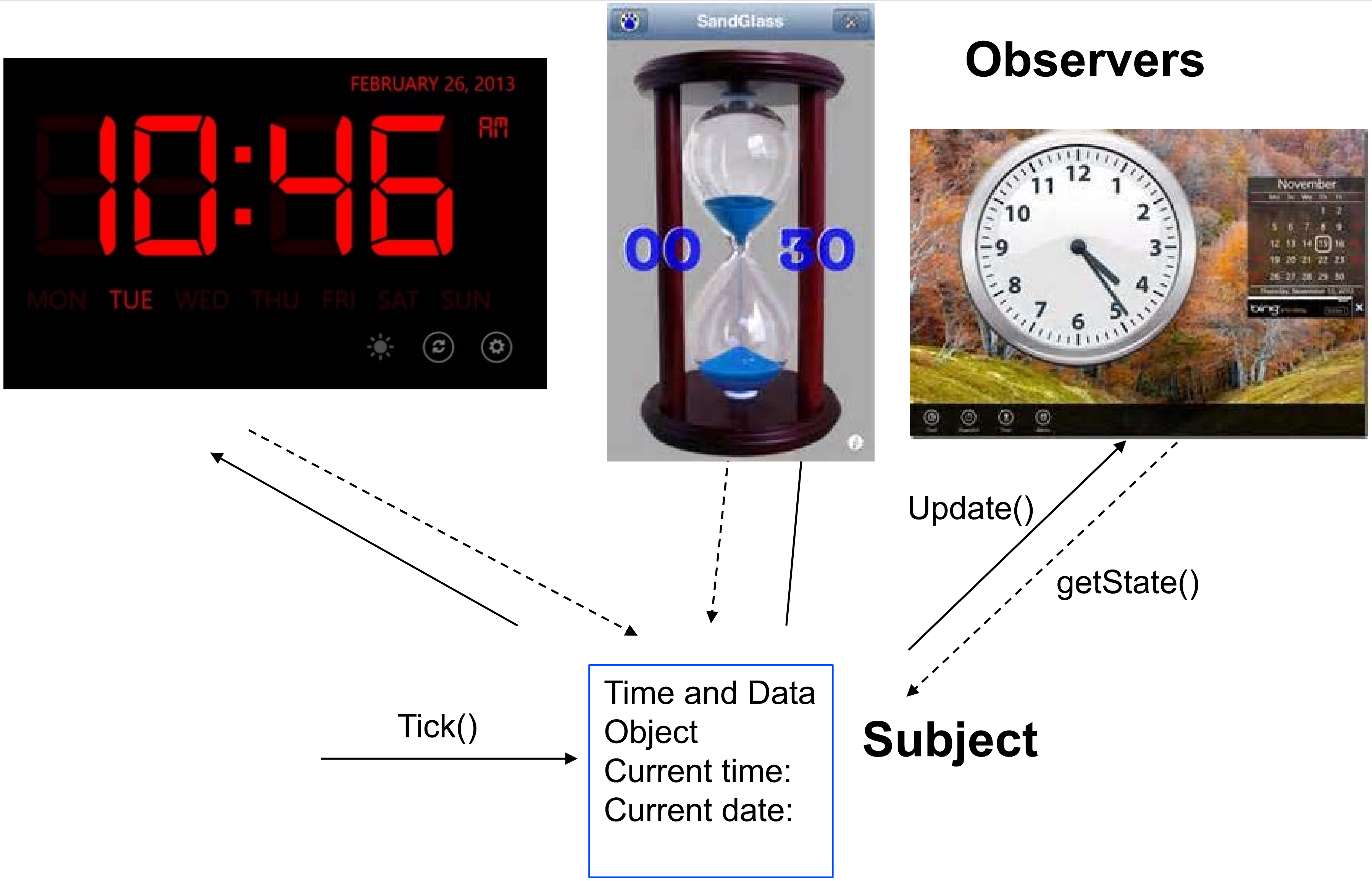
Application Stopping for Traffic Light



Stopping for Traffic Light (as Observer Pattern)



Gamma Example



Observer Pattern example in C++ (From Gamma et al)

```
class Subject;
class Observer {
public:
  virtual ~Observer();
  virtual void Update(Subject *theChangedSubject) = 0;
protected:
  Observer();
};
```

...Observer Pattern example in C++ (From Gamma et al)

```
class Subject {
public:
  virtual ~Subject();
  virtual void Attach(Observer *);
  virtual void Detach(Observer *);
protected:
  Subject();
private:
  List<Observer*> *_observers;
};
void Subject::Attach(Observer * o) {
  _observers->Append(o);
}
void Subject::Detach(Observer * o) {
  _observers->Remove(o);
}
void Subject::Notify(){
  ListIterator<Observer*> i(_observers);
  for (i.First(); !i.IsDone(); i.Next()) {
       i.CurrentItem()->Update(this);
  }
}
```

...Observer Pattern example in C++ (From Gamma et al)

```
class ClockTimer: public Subject {  //concrete subject
public:
  ClockTimer();
  virtual int GetHour();
  virtual int GetMinute();
  virtual int GetSecond();
  void Tick(); //called regularly by internal timer
};

void ClockTimer::Tick() {
  //update internal time-keeping state
  //. . .
  Notify();
}
```

...Observer Pattern example in C++ (From Gamma et al)

```
class DigitalClock: public Widget, public Observer {
public:
  DigitalClock(ClockTimer *);
  virtual  ~DigitalClock();
  virtual void Update(Subject *); //overrides Observer
  virtual void Draw(); //overrides Widget
private:
  ClockTimer* _subject;
};
DigitalClock::DigitalClock(ClockTimer * s){
  _subject = s;
  _subject->Attach(this);
}
DigitalClock::~DigitalClock() {
  _subject->Detach(this);
}
```

...Observer Pattern example in C++ (From Gamma et al)

```
void DigitalClock::Update(Subject * theChangedSubject) {
  if (theChangedSubject == _subject) {
       Draw();
   }
}
void DigitalClock::Draw() {
  //get new values from subject
  int hour = _subject->GetHour();
  int minute = _subject->GetMinute();
  int second = _subject->GetSecond();
  // . . .
  //draw the digital clock
}
```

...Observer Pattern example in C++ (From Gamma et al)

```
Class AnalogClock : public Widget, public Observer {
public:
  AnalogClock(ClockTimer*);
  virtual void Update(Subject *);
  virtual void Draw();
  //. . .
};

//Code to create Digital and Analog Clock
ClockTimer *timer = new ClockTimer;
AnalogClock* analogClock = new AnalogClock(timer);
DigitalClock* digitalClock = new DigitalClock(timer);

//Whenever the timer ticks(), the analog and digital
  clock will be updated and redraw themselves
```

Observer Pattern -Consequences

- Subjects and observers can be varied or reused independently of each other
- Abstract coupling between subject & observer
  - -subject only knows it has some observers
  - -subject doesn't know what kind they are
- Unexpected Updates
  - -observers don't know about each other
  - -seemingly innocent action on a subject can cause a cascade of updates
  - -sensitive to spurious updates
  - -aggravated because update protocol does not indicate what changed

Observer Pattern -Implementation Issues

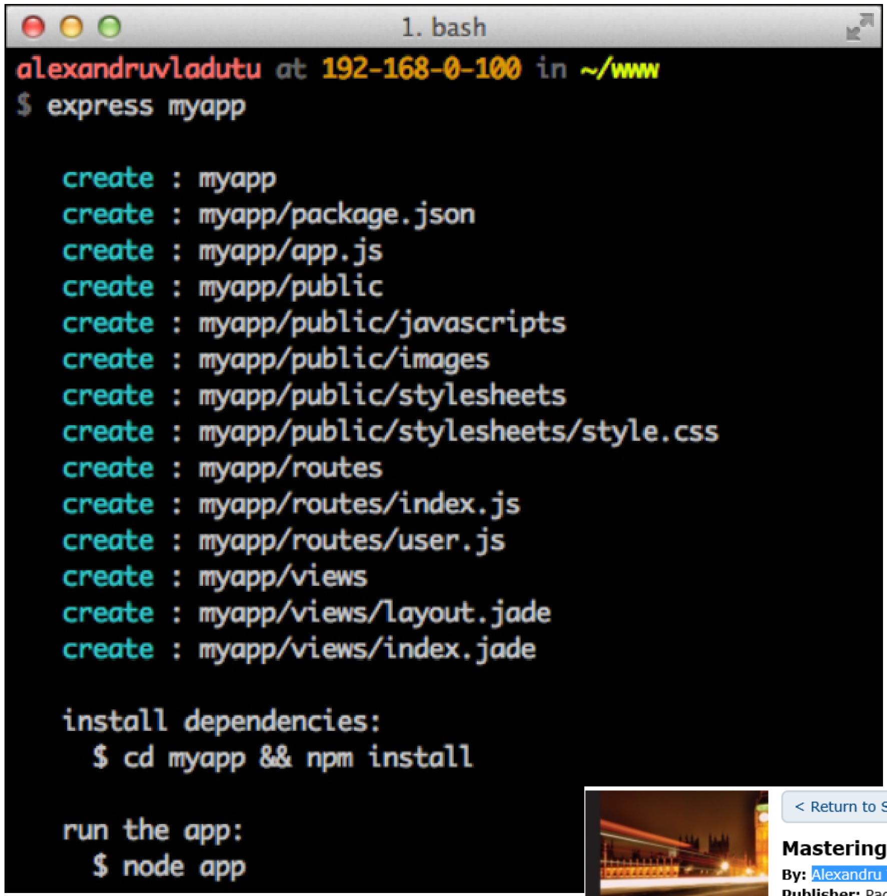
- Deleting subject must not leave dangling references in observers
  - -have subject notify observers first
  - -cannot just delete observer

- What if observer detaches as a result of being notified?

Model-View-Control Pattern in Express CLI -App

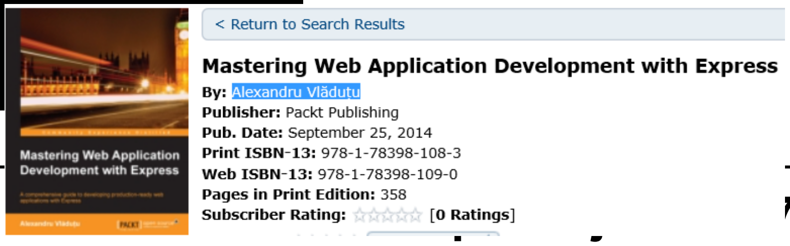
Model-View-Controller

The most common technique to structure web applications with Express is MVC. When generating a project using the Express CLI, it almost provides an MVC structure, omitting the `models` folder. The following screenshot lists all the files and folders generated for a sample application using the CLI tool:



```
1. bash
alexandruvladutu at 192-168-0-100 in ~/www
$ express myapp

   create : myapp
   create : myapp/package.json
   create : myapp/app.js
   create : myapp/public
   create : myapp/public/javascripts
   create : myapp/public/images
   create : myapp/public/stylesheets
   create : myapp/public/stylesheets/style.css
   create : myapp/routes
   create : myapp/routes/index.js
   create : myapp/routes/user.js
   create : myapp/views
   create : myapp/views/layout.jade
   create : myapp/views/index.jade

   install dependencies:
     $ cd myapp && npm install

   run the app:
     $ node app
```

source: Alexandru Vlăduțu



< Return to Search Results

Mastering Web Application Development with Express

By: Alexandru Vlăduțu
Publisher: Packt Publishing
Pub. Date: September 25, 2014
Print ISBN-13: 978-1-78398-108-3
Web ISBN-13: 978-1-78398-109-0
Pages in Print Edition: 358
Subscriber Rating: ☆☆☆☆☆ [0 Ratings]

Model-View-Control Pattern in Express CLI

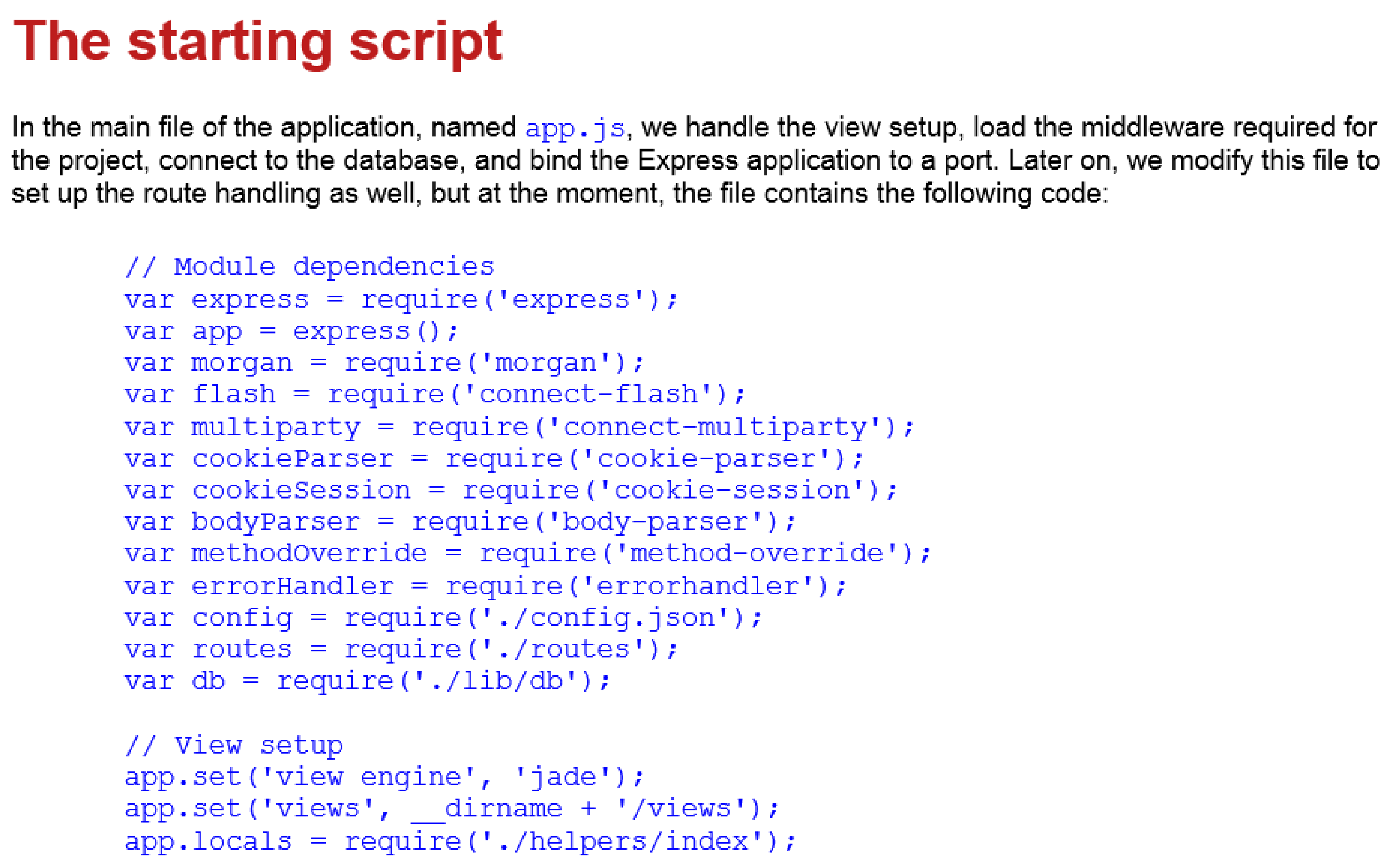
The `package.json` file is automatically populated with the name of the application, the dependencies, the private attribute, and the starting script. This starting script is named `app.js` and loads all the middleware, assigns the route handlers, and starts the server. There are three folders in the root:

- `public`: This folder contains the static assets
- `views`: This folder is populated with Jade templates by default
- `routes`: This folder includes the routes (these are the equivalent controllers)

Apart from these existing folders and the `models` folder, which we need to create ourselves, we might also create folders for tests, logs, or configuration. The best thing about this structure is that it's easy to get started with and is known to most developers.

source: Alexandru Vlăduțu "Mastering Web Application Development with Express, Packt, 2014

Express CLI generated default App



The starting script

In the main file of the application, named `app.js`, we handle the view setup, load the middleware required for the project, connect to the database, and bind the Express application to a port. Later on, we modify this file to set up the route handling as well, but at the moment, the file contains the following code:

```
// Module dependencies
var express = require('express');
var app = express();
var morgan = require('morgan');
var flash = require('connect-flash');
var multiparty = require('connect-multiparty');
var cookieParser = require('cookie-parser');
var cookieSession = require('cookie-session');
var bodyParser = require('body-parser');
var methodOverride = require('method-override');
var errorHandler = require('errorhandler');
var config = require('./config.json');
var routes = require('./routes');
var db = require('./lib/db');

// View setup
app.set('view engine', 'jade');
app.set('views', __dirname + '/views');
app.locals = require('./helpers/index');
```

source: Alexandru Vlăduțu "Mastering Web Application Development with Express, Packt, 2014

…Express CLI default App

```
// Loading middleware
app.use(morgan('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(methodOverride(function(req, res){
  if (req.body && typeof req.body === 'object' && '_method' in req.body) {
    // look in url - encoded POST bodies and delete it
    var method = req.body._method;
    delete req.body._method;
    return method;
  }
}));
app.use(cookieParser());
app.use(cookieSession({
  secret: config.sessionSecret,
  cookie: {
    maxAge: config.sessionMaxAge
  }
}));
app.use(flash());
```

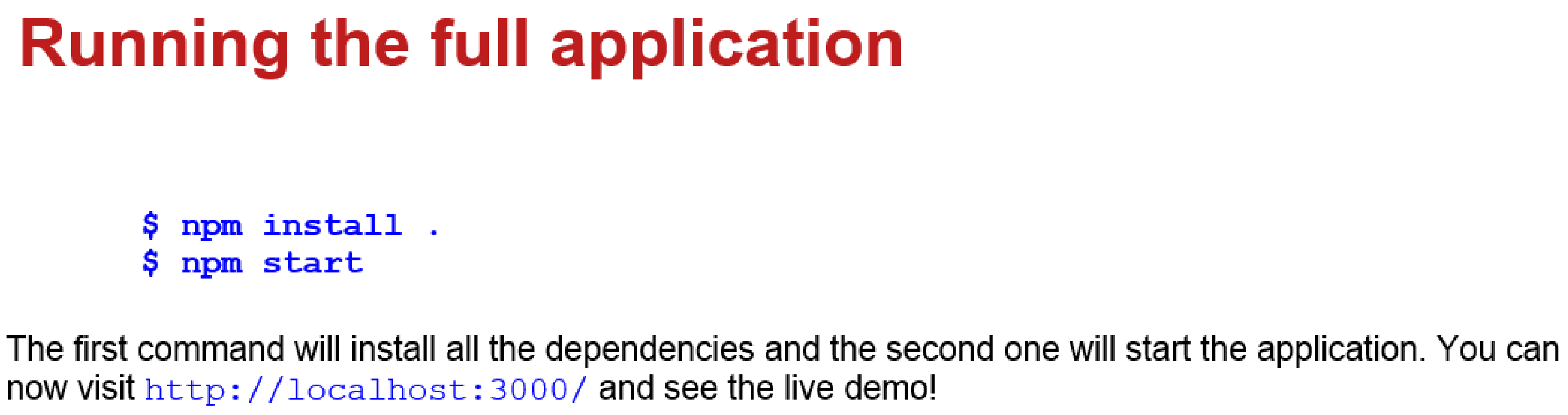
source: Alexandru Vlăduțu "Mastering Web Application Development with Express, Packt, 2014

…Express CLI default App

```
if (app.get('env') === 'development') {
  app.use(errorHandler());
}

// static middleware after the routes
app.use(express.static(__dirname + '/public'));

// Establishing database connection and binding application to specified
port
db.connect();
app.listen(config.port);
console.log('listening on port %s', config.port);
```

source: Alexandru Vlăduțu "Mastering Web Application Development with Express, Packt, 2014

…Express CLI default App



Running the full application

```
$ npm install .
$ npm start
```

The first command will install all the dependencies and the second one will start the application. You can now visit `http://localhost:3000/` and see the live demo!

source: Alexandru Vlăduțu "Mastering Web Application Development with Express, Packt, 2014

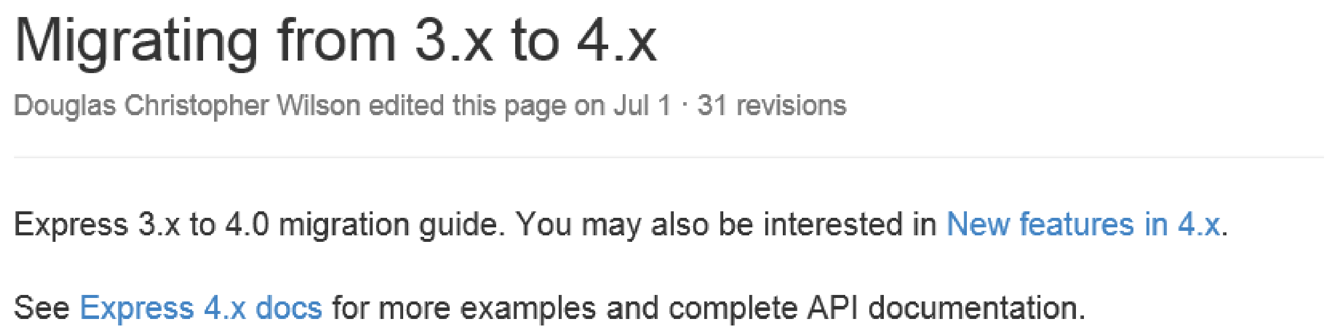
Migrating from Express 3.x to 4.x

- Connect has been removed from Express, so with the exception of the `static` middleware, you will need to install the appropriate packages (namely, `connect`). At the same time, Connect has been moving some of its middleware into their own packages, so you might have to do some searching on npm to figure out where your middleware went.
- `body-parser` is now its own package, which no longer includes the `multipart` middleware, closing a major security hole. It's now safe to use the `body-parser` middleware.
- You no longer have to link the Express router into your application. So you should remove `app.use(app.router)` from your existing Express 3.0 apps.
- `app.configure` was removed; simply replace calls to this method by examining `app.get(`*`env`*`)` (using either a `switch` statement or `if` statements).

source: "Web Development with Node and Express" Brown, Ethan (2014-07-01).

Migrating from Express 3.x to 4.x

Official Migration Guide: Express 3.x to 4.x
https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x



Migrating from 3.x to 4.x

Douglas Christopher Wilson edited this page on Jul 1 · 31 revisions

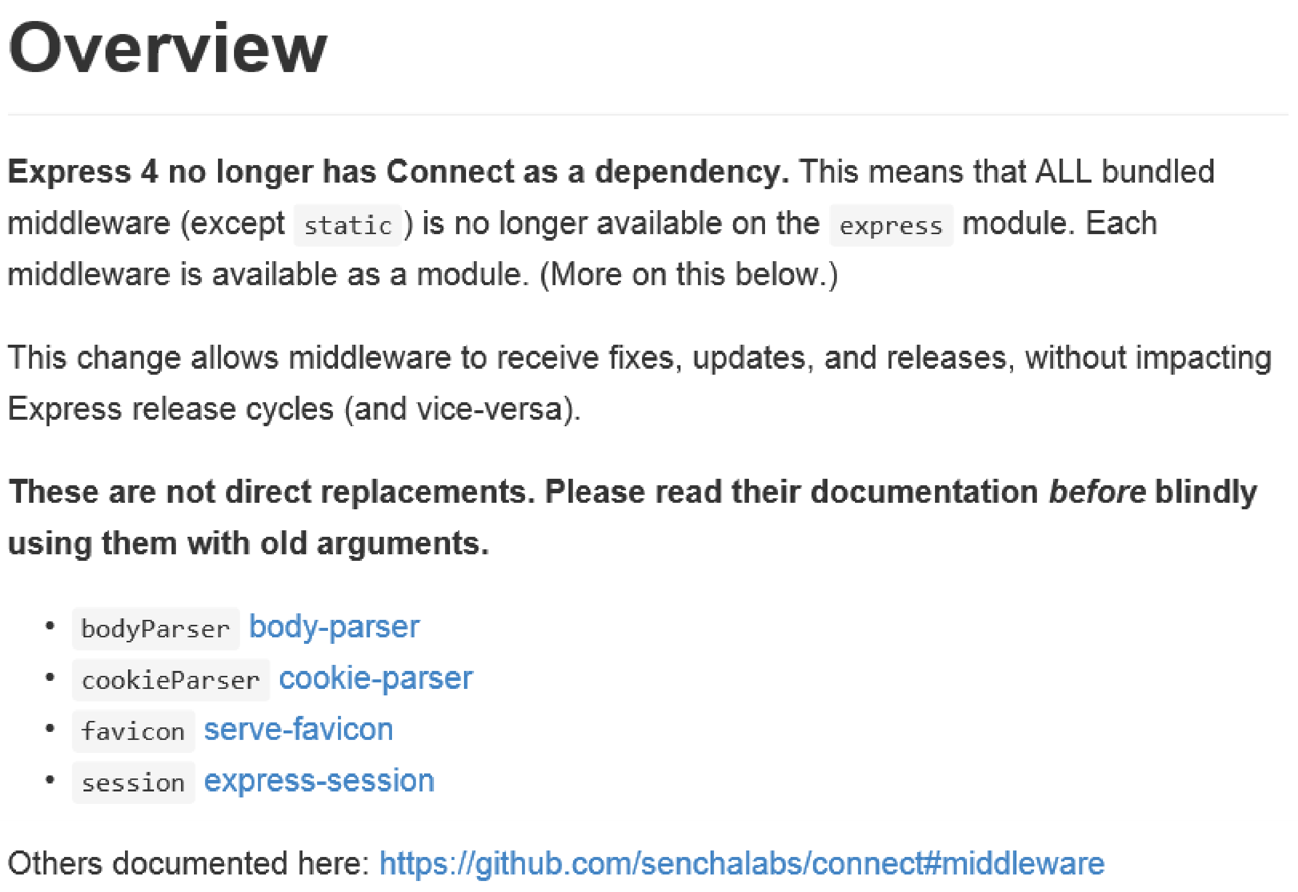
Express 3.x to 4.0 migration guide. You may also be interested in New features in 4.x.

See Express 4.x docs for more examples and complete API documentation.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

Official Migration Guide: Express 3.x to 4.x
https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x



Overview

Express 4 no longer has Connect as a dependency. This means that ALL bundled middleware (except `static`) is no longer available on the `express` module. Each middleware is available as a module. (More on this below.)

This change allows middleware to receive fixes, updates, and releases, without impacting Express release cycles (and vice-versa).

These are not direct replacements. Please read their documentation *before* blindly using them with old arguments.

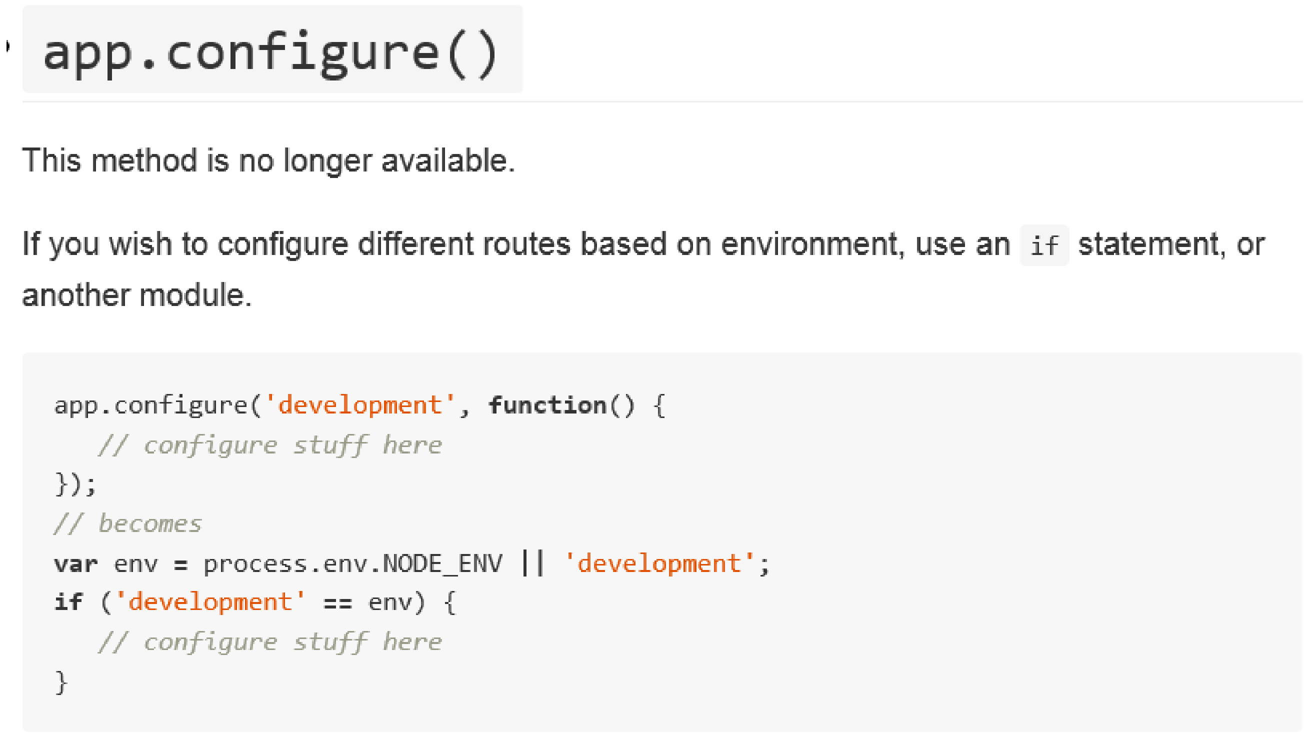
- `bodyParser` body-parser
- `cookieParser` cookie-parser
- `favicon` serve-favicon
- `session` express-session

Others documented here: https://github.com/senchalabs/connect#middleware

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

REMOVED IN EXPRESS 4.x



`app.configure()`

This method is no longer available.

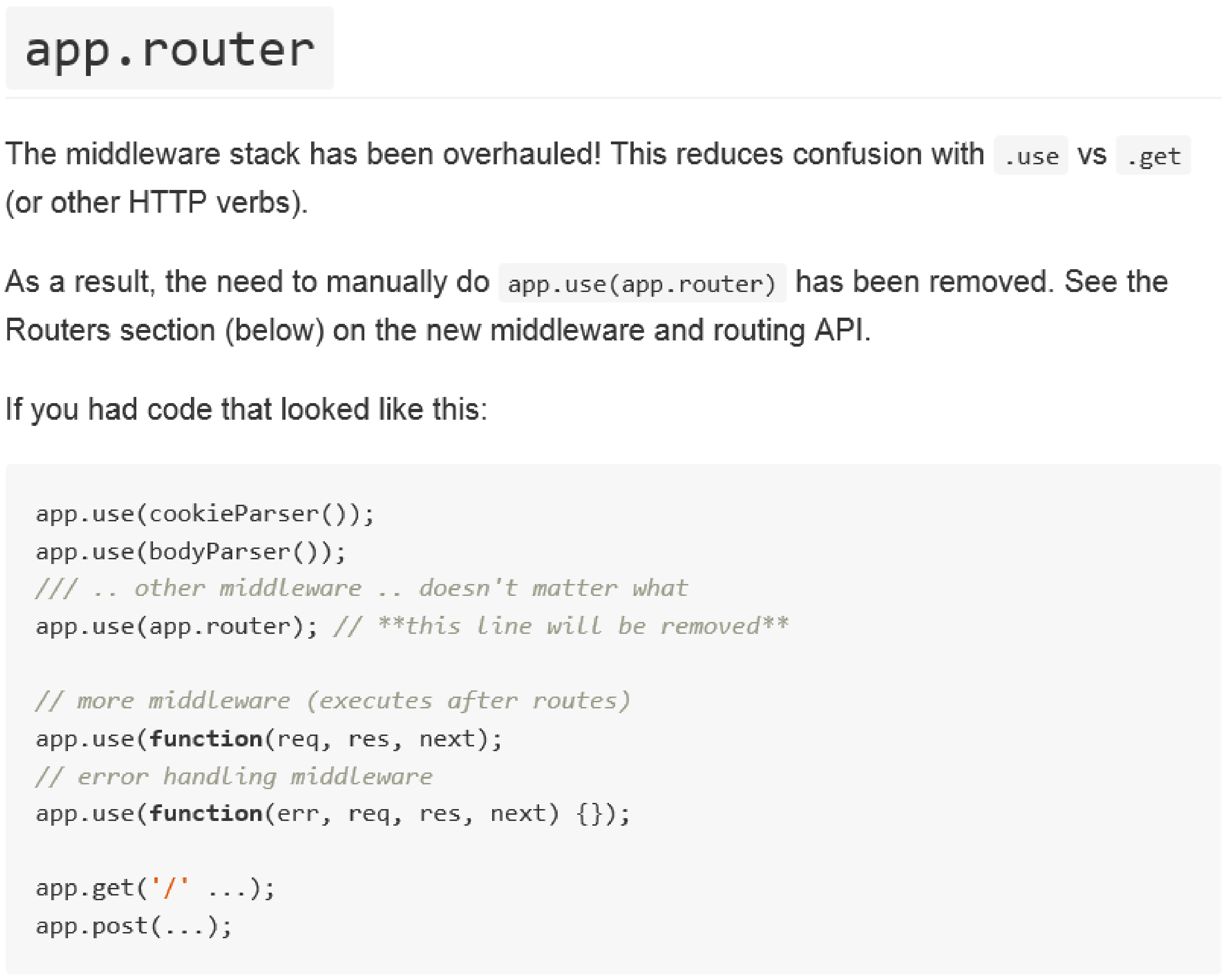
If you wish to configure different routes based on environment, use an `if` statement, or another module.

```
app.configure('development', function() {
   // configure stuff here
});
// becomes
var env = process.env.NODE_ENV || 'development';
if ('development' == env) {
   // configure stuff here
}
```

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

REMOVED IN EXPRESS 4.x



`app.router`

The middleware stack has been overhauled! This reduces confusion with `.use` vs `.get` (or other HTTP verbs).

As a result, the need to manually do `app.use(app.router)` has been removed. See the Routers section (below) on the new middleware and routing API.

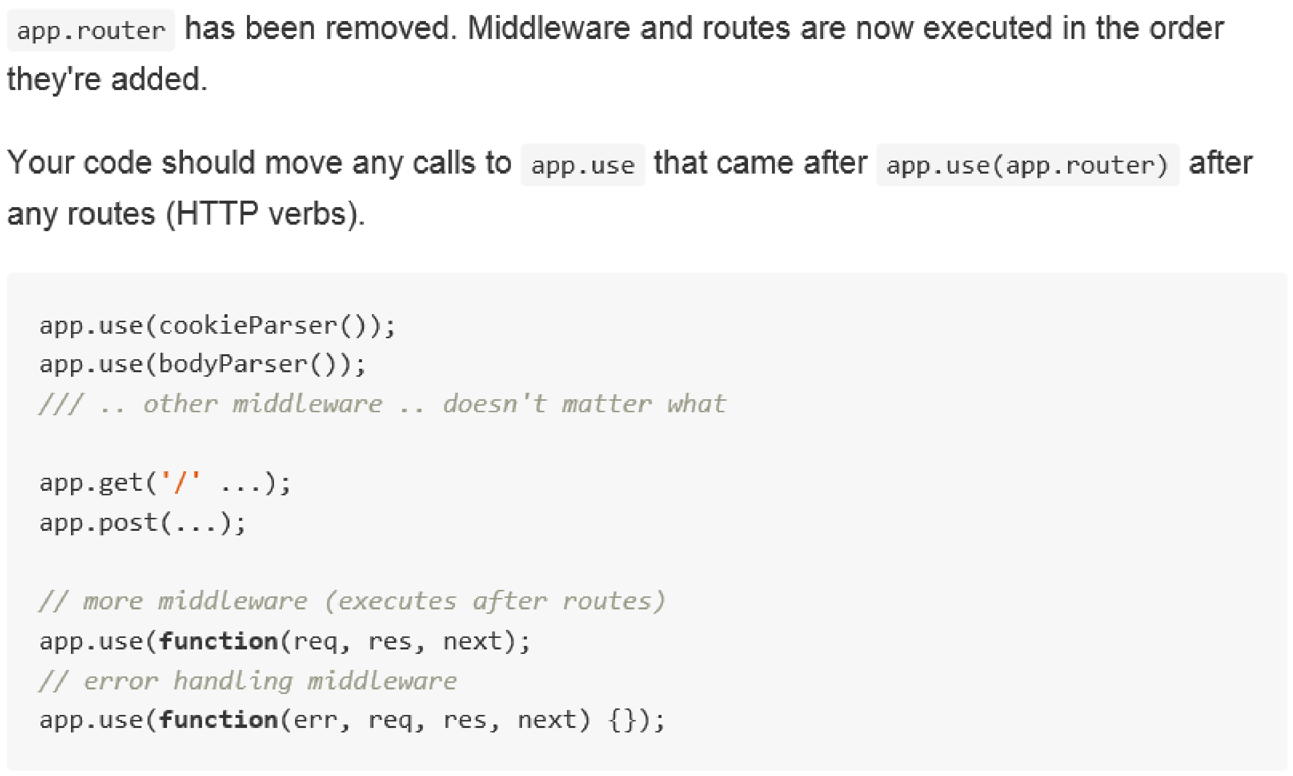
If you had code that looked like this:

```
app.use(cookieParser());
app.use(bodyParser());
/// .. other middleware .. doesn't matter what
app.use(app.router); // **this line will be removed**

// more middleware (executes after routes)
app.use(function(req, res, next);
// error handling middleware
app.use(function(err, req, res, next) {});

app.get('/' ...);
app.post(...);
```

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

REMOVED IN EXPRESS 4.x



`app.router` has been removed. Middleware and routes are now executed in the order they're added.

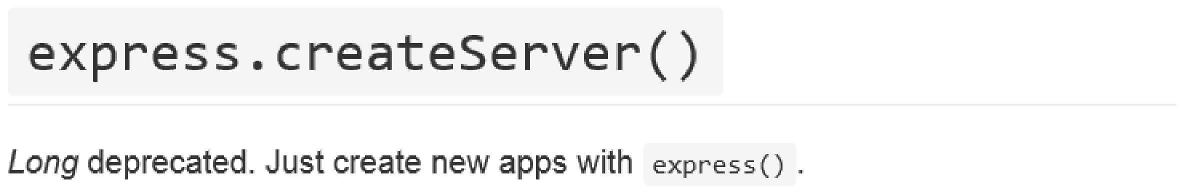
Your code should move any calls to `app.use` that came after `app.use(app.router)` after any routes (HTTP verbs).

```
app.use(cookieParser());
app.use(bodyParser());
/// .. other middleware .. doesn't matter what

app.get('/' ...);
app.post(...);

// more middleware (executes after routes)
app.use(function(req, res, next);
// error handling middleware
app.use(function(err, req, res, next) {});
```

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

REMOVED IN EXPRESS 4.x



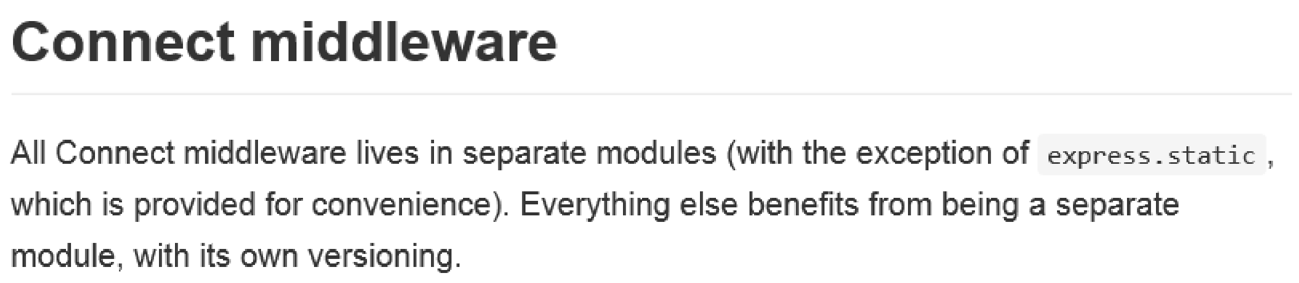
```
express.createServer()
```

Long deprecated. Just create new apps with `express()`.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

REMOVED IN EXPRESS 4.x



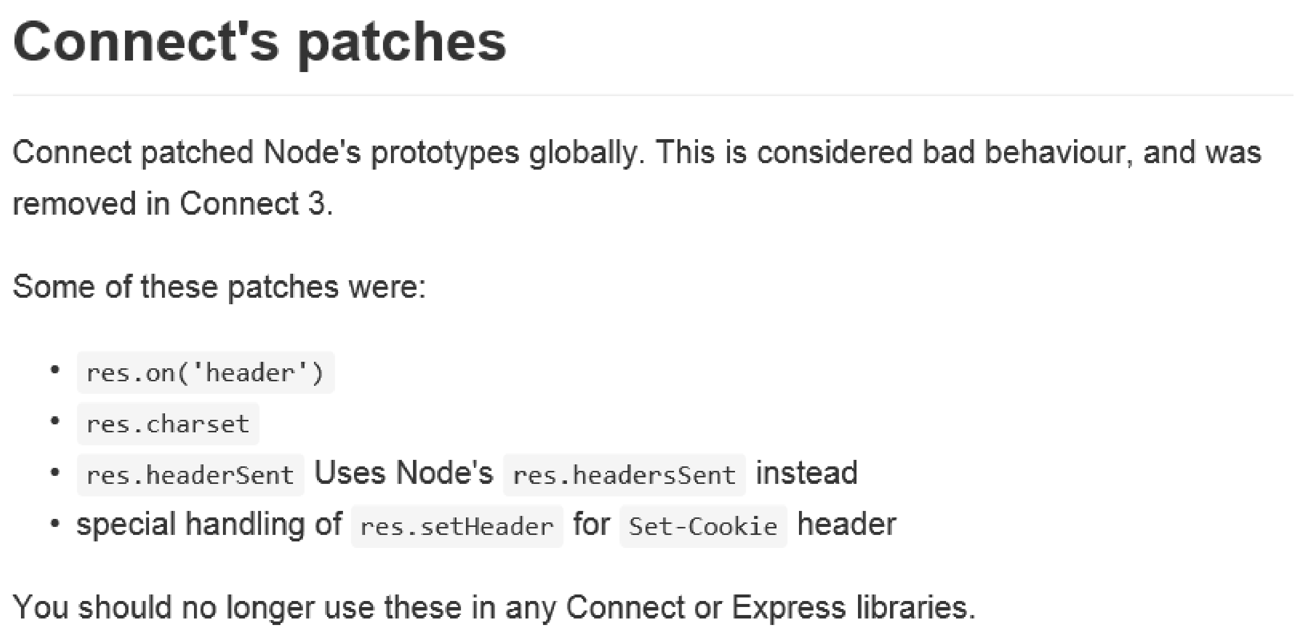
Connect middleware

All Connect middleware lives in separate modules (with the exception of `express.static`, which is provided for convenience). Everything else benefits from being a separate module, with its own versioning.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

REMOVED IN EXPRESS 4.x



Connect's patches

Connect patched Node's prototypes globally. This is considered bad behaviour, and was removed in Connect 3.

Some of these patches were:

- `res.on('header')`
- `res.charset`
- `res.headerSent` Uses Node's `res.headersSent` instead
- special handling of `res.setHeader` for `Set-Cookie` header

You should no longer use these in any Connect or Express libraries.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

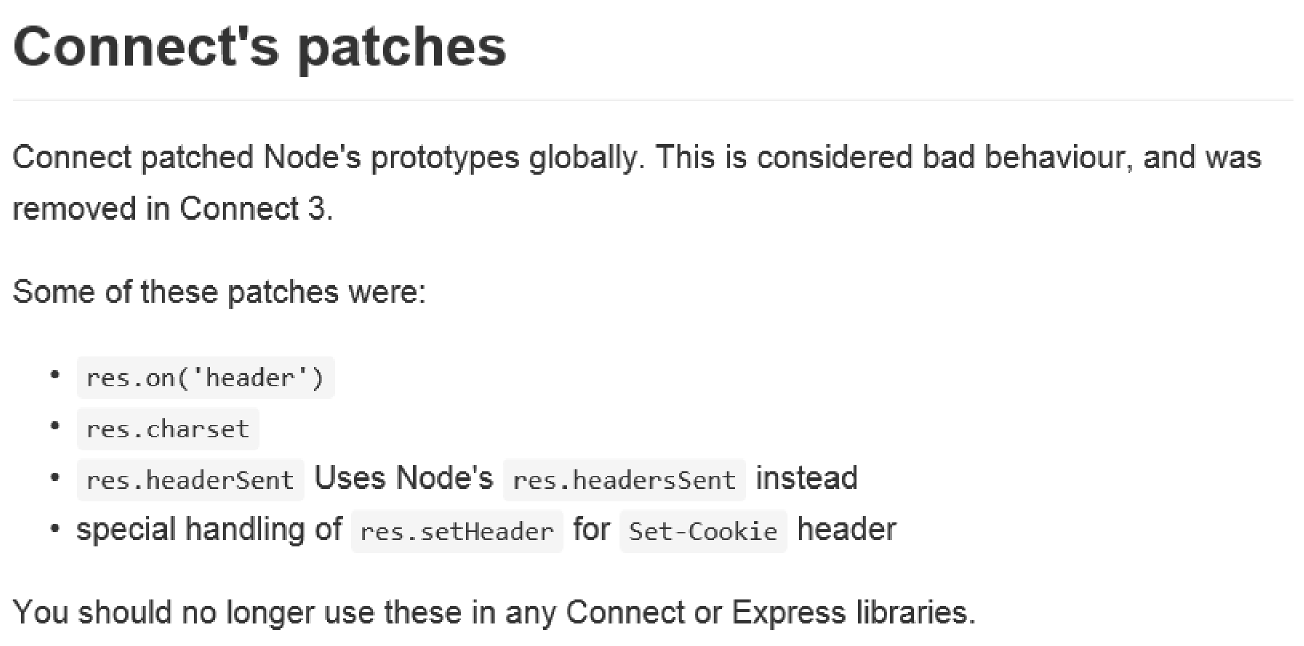
Migrating from Express 3.x to 4.x

REMOVED IN EXPRESS 4.x

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

REMOVED IN EXPRESS 4.x



Connect's patches

Connect patched Node's prototypes globally. This is considered bad behaviour, and was removed in Connect 3.

Some of these patches were:

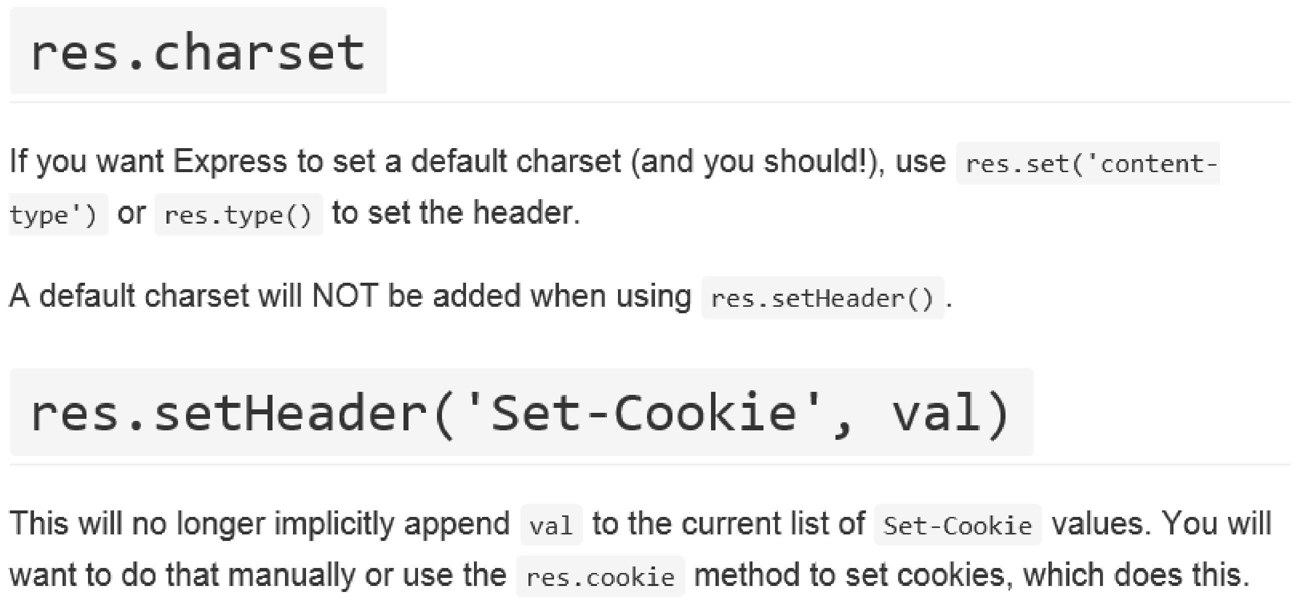
- `res.on('header')`
- `res.charset`
- `res.headerSent` Uses Node's `res.headersSent` instead
- special handling of `res.setHeader` for `Set-Cookie` header

You should no longer use these in any Connect or Express libraries.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

REMOVED IN EXPRESS 4.x



`res.charset`

If you want Express to set a default charset (and you should!), use `res.set('content-type')` or `res.type()` to set the header.

A default charset will NOT be added when using `res.setHeader()`.

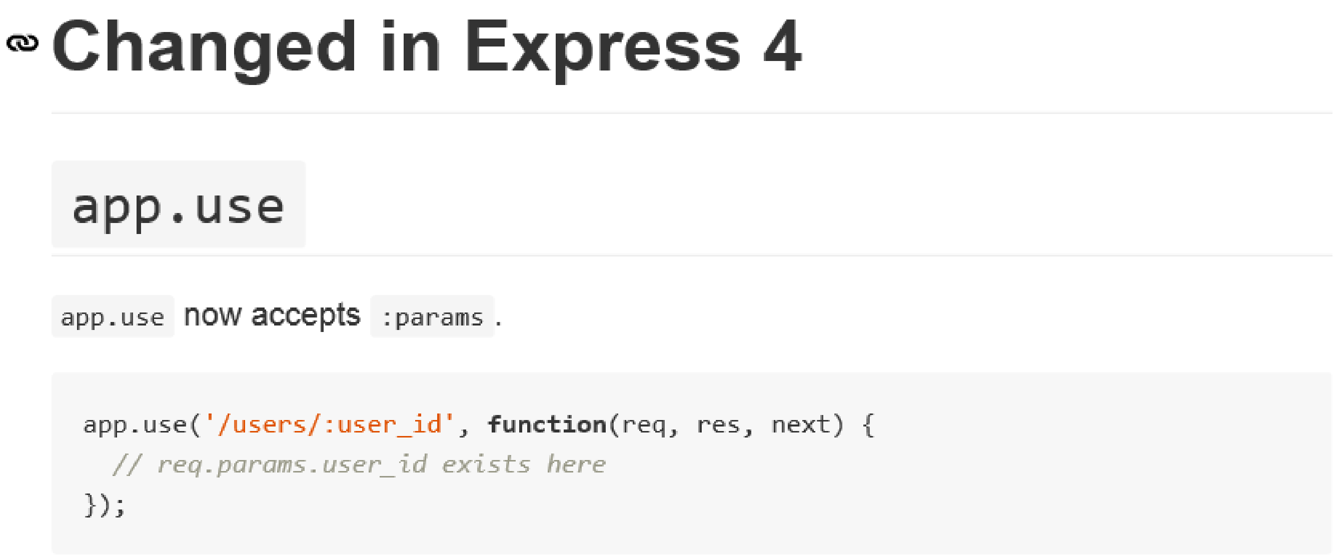
`res.setHeader('Set-Cookie', val)`

This will no longer implicitly append `val` to the current list of `Set-Cookie` values. You will want to do that manually or use the `res.cookie` method to set cookies, which does this.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

CHANGED IN EXPRESS 4.x



Changed in Express 4

`app.use`

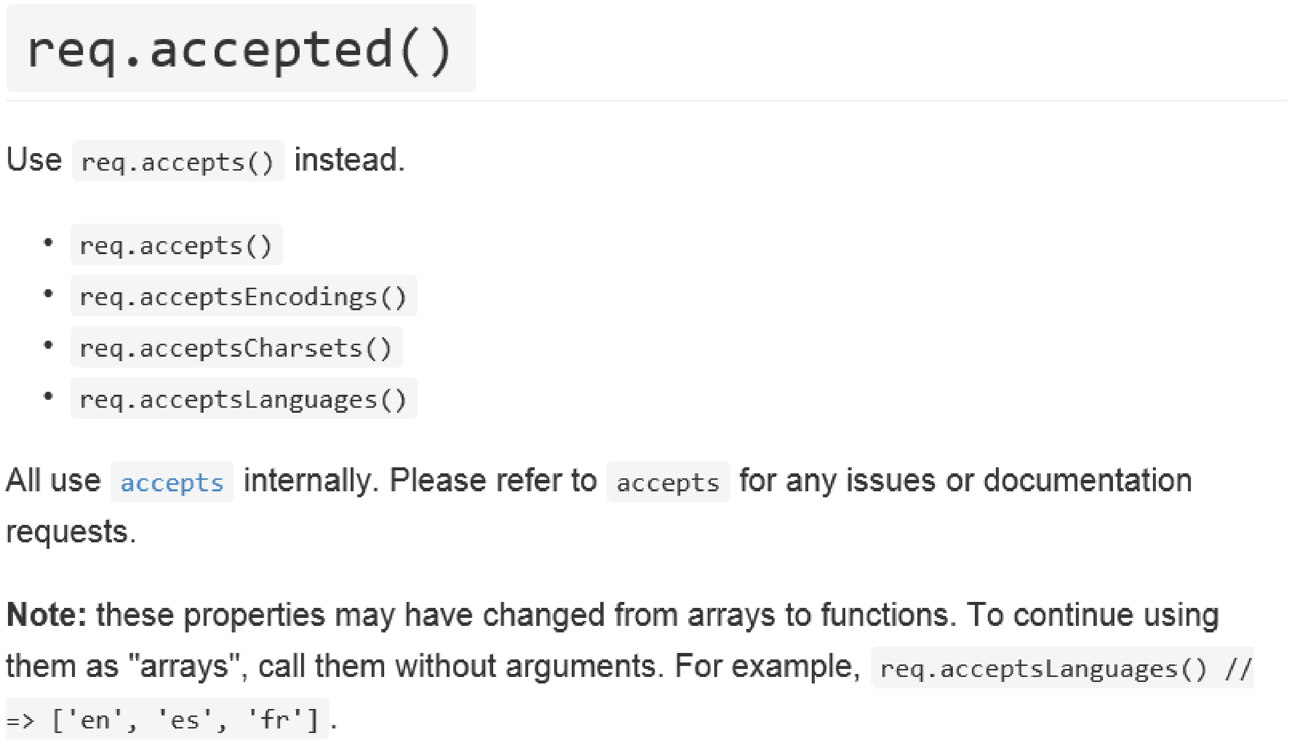
`app.use` now accepts `:params`.

```
app.use('/users/:user_id', function(req, res, next) {
  // req.params.user_id exists here
});
```

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

CHANGED IN EXPRESS 4.x



`req.accepted()`

Use `req.accepts()` instead.

- `req.accepts()`
- `req.acceptsEncodings()`
- `req.acceptsCharsets()`
- `req.acceptsLanguages()`

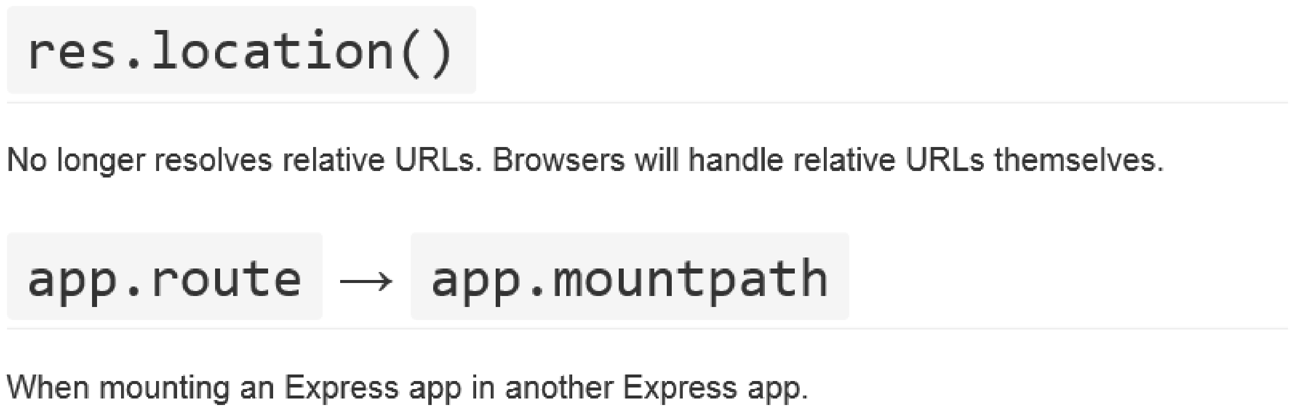
All use `accepts` internally. Please refer to `accepts` for any issues or documentation requests.

Note: these properties may have changed from arrays to functions. To continue using them as "arrays", call them without arguments. For example, `req.acceptsLanguages() // => ['en', 'es', 'fr']`.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

CHANGED IN EXPRESS 4.x



`res.location()`

No longer resolves relative URLs. Browsers will handle relative URLs themselves.

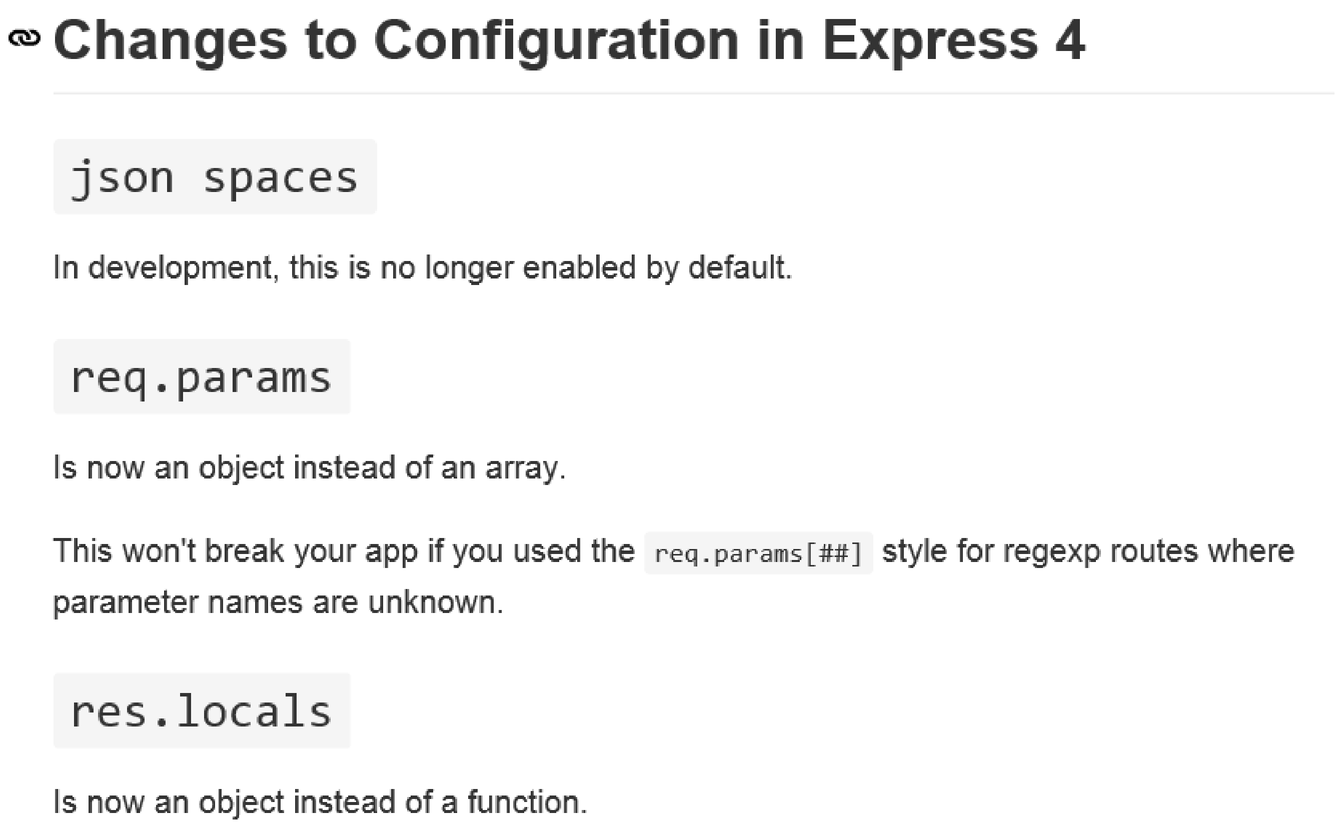
`app.route` → `app.mountpath`

When mounting an Express app in another Express app.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

CHANGED IN EXPRESS 4.x



Changes to Configuration in Express 4

`json spaces`

In development, this is no longer enabled by default.

`req.params`

Is now an object instead of an array.

This won't break your app if you used the `req.params[##]` style for regexp routes where parameter names are unknown.

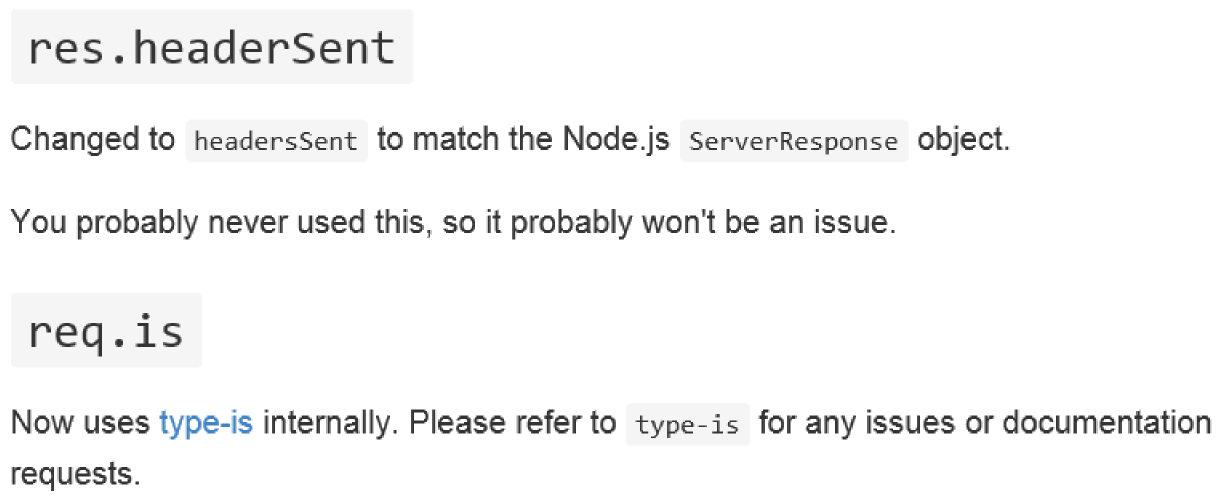
`res.locals`

Is now an object instead of a function.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

CHANGED IN EXPRESS 4.x



`res.headerSent`

Changed to `headersSent` to match the Node.js `ServerResponse` object.

You probably never used this, so it probably won't be an issue.

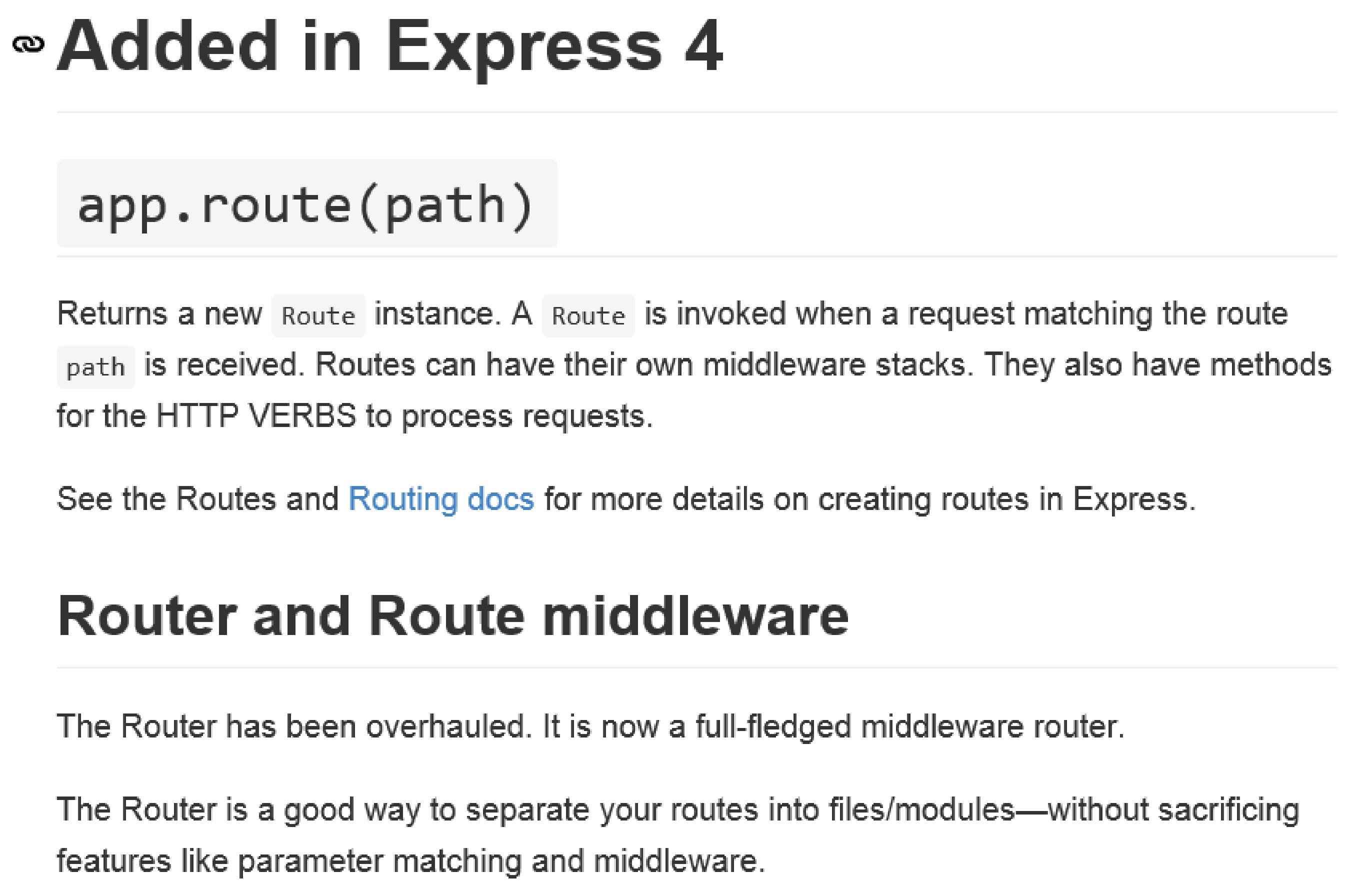
`req.is`

Now uses type-is internally. Please refer to `type-is` for any issues or documentation requests.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x

Migrating from Express 3.x to 4.x

ADDED IN EXPRESS 4.x



Added in Express 4

`app.route(path)`

Returns a new `Route` instance. A `Route` is invoked when a request matching the route `path` is received. Routes can have their own middleware stacks. They also have methods for the HTTP VERBS to process requests.

See the Routes and Routing docs for more details on creating routes in Express.

Router and Route middleware

The Router has been overhauled. It is now a full-fledged middleware router.

The Router is a good way to separate your routes into files/modules—without sacrificing features like parameter matching and middleware.

source: https://github.com/strongloop/express/wiki/Migrating-from-3.x-to-4.x