Koa

next generation web framework for node.js

- 刘博文 @奇舞团
- 95后
- https://github.com/berwin

- 简介
- Koa如何使用
- 深入解析
- 总结

简介

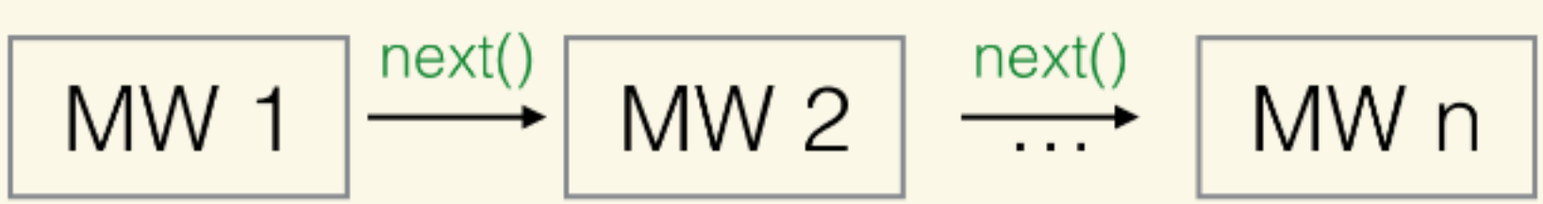
- 提供异步解决方案
- 更灵活的中间件
- 异常处理
- 更小，更优雅...

Express

```
var express = require('express');
var app = express();

app.use(function (req, res, next) {
  // ...
  next();
})

app.use(function (req, res) {
  res.send('Hello World!');
});

app.listen(3000);
```

req, res, next

express process

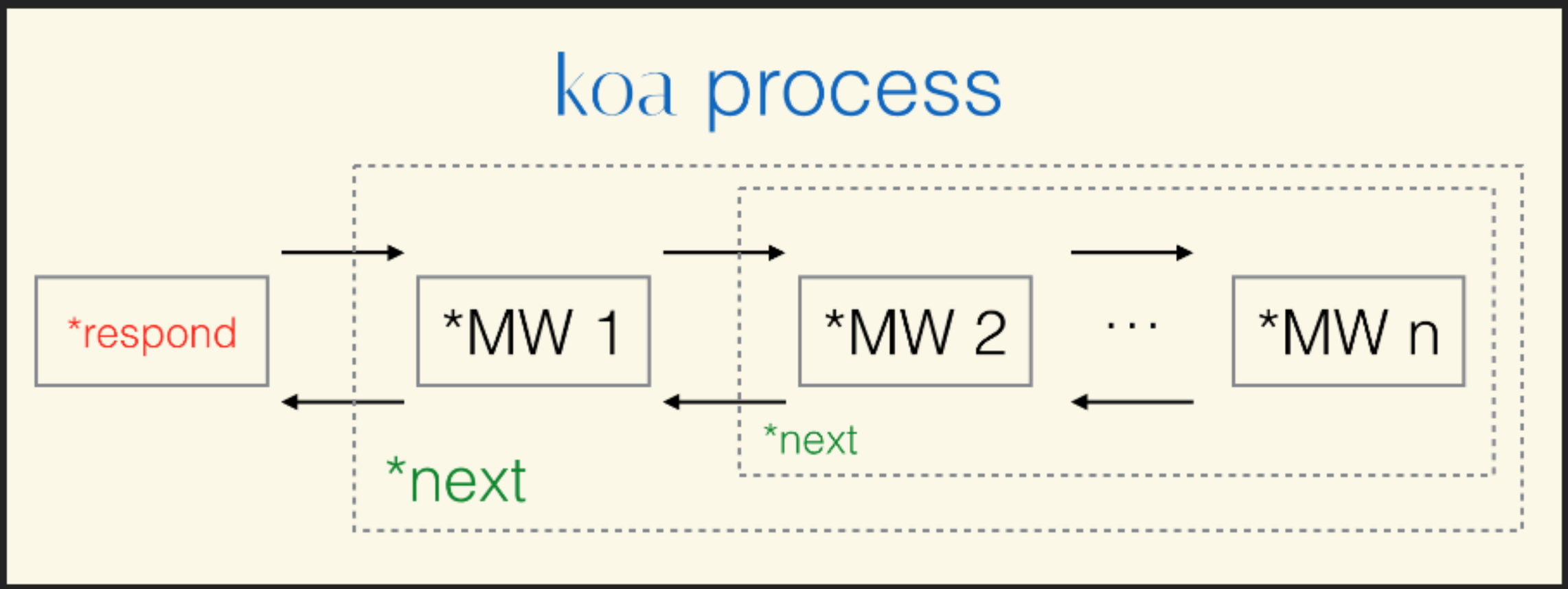


MW: middleware function

Koa

```
var koa = require('koa');
var app = koa();

// logger
app.use(function *(next){
  var start = new Date;
  yield next;
  var ms = new Date - start;
  console.log('%s %s - %s', this.method, this.url, ms);
});

// response
app.use(function *(){
  this.body = 'Hello World';
});

app.listen(3000);
```

this, yield, next


koa process
*respond
*MW 1
*MW 2
···
*MW n
*next
*next

- 简介
- Koa如何使用
- 深入解析
- 总结

初始化App

```
var app = koa();
```

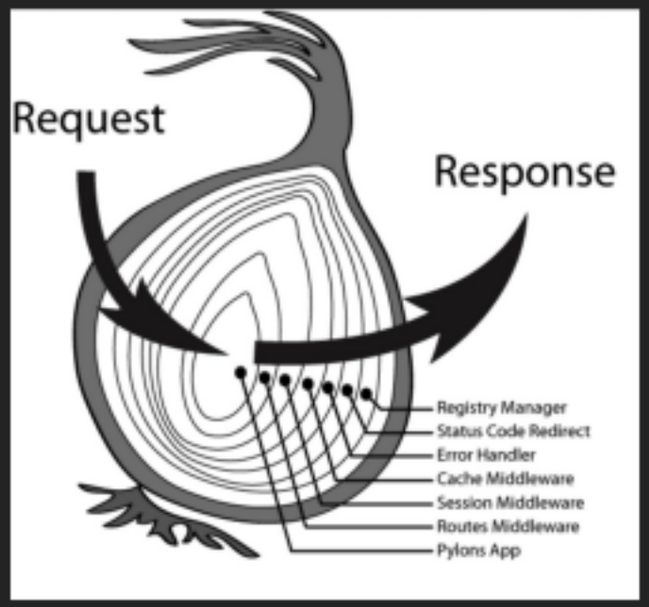
常用方法

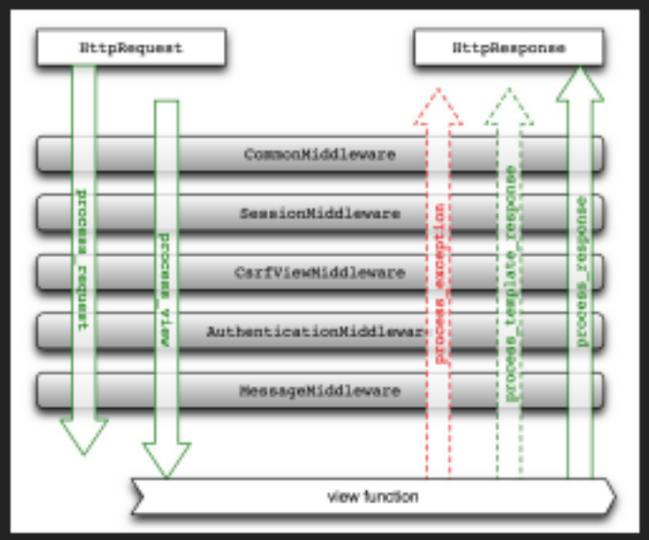
Hello World

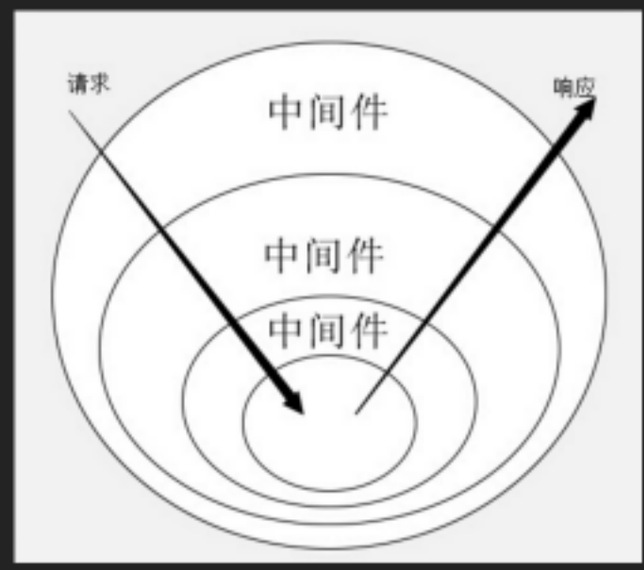
```
var koa = require('koa');
var app = koa();

// x-response-time

app.use(function *(next){
  var start = new Date;
  yield next;
  var ms = new Date - start;
  this.set('X-Response-Time', ms + 'ms');
});

// logger

app.use(function *(next){
  var start = new Date;
  yield next;
  var ms = new Date - start;
  console.log('%s %s - %s', this.method, this.url, ms);
});

// response

app.use(function *(){
```

Koa中最最重要的两个部分

中间件


Request
Response
Registry Manager
Status Code Redirect
Error Handler
Cache Middleware
Session Middleware
Routes Middleware
Pylons App


HttpRequest
HttpResponse
CommonMiddleware
SessionMiddleware
CsrfViewMiddleware
AuthenticationMiddlewar
MessageMiddleware
process_request
process_view
process_exception
process_template_response
process_response
view function


请求
中间件
响应
中间件
中间件

例子

```
var koa = require('koa');
var app = koa();

app.use(function* (next) {
  console.log('f1: pre next');
  yield next;
  console.log('f1: post next');
});

app.use(function* (next) {
  console.log('  f2: pre next');
  yield next;
  console.log('  f2: post next');
});

app.use(function* (next) {
  console.log('  f3: pre next');
  yield next;
  console.log('  f3: post next');
});

app.use(function* (next) {
  console.log('hello world')
  this.body = 'hello world';
```

上下文 - Context

Request & Response

Request aliases

The following accessors and alias Request equivalents:

- ctx.header
- ctx.headers
- ctx.method
- ctx.method=
- ctx.url
- ctx.url=
- ctx.originalUrl
- ctx.origin
- ctx.href
- ctx.path
- ctx.path=
- ctx.query
- ctx.query=
- ctx.querystring
- ctx.querystring=
- ctx.host
- ctx.hostname
- ctx.fresh
- ctx.stale
- ctx.socket
- ctx.protocol
- ctx.secure
- ctx.ip
- ctx.ips
- ctx.subdomains
- ctx.is()
- ctx.accepts()
- ctx.acceptsEncodings()
- ctx.acceptsCharsets()
- ctx.acceptsLanguages()
- ctx.get()

Response aliases

The following accessors and alias Response equivalents:

- ctx.body
- ctx.body=
- ctx.status
- ctx.status=
- ctx.message
- ctx.message=
- ctx.length=
- ctx.length
- ctx.type=
- ctx.type
- ctx.headerSent
- ctx.redirect()
- ctx.attachment()
- ctx.set()
- ctx.append()
- ctx.remove()
- ctx.lastModified=
- ctx.etag=

例子

```
var koa = require('koa');
var app = koa();

app.use(function *(){
  console.log(this);
  this.body = 'Hello World';
});

app.listen(3000);
```

异常处理

```
var koa = require('koa');
var app = koa();

app.use(function *(next){
  try {
    yield next;
  } catch (err) {
    this.status = err.status || 500;
    this.body = 'Something exploded';
  }
});

app.use(function *(){
  throw new Error('boom boom');
});

app.listen(3000);
```

异常处理2

```
var koa = require('koa');
var app = koa();

app.use(function *(){
  var userID = this.query.userID;

  try {
    this.body = yield user.findById(userID);
  } catch (err) {
    this.status = err.status || 403;
    this.body = 'incorrect parameter';
    this.app.emit('error', err, this);
  }
});

app.listen(3000);
```

异步

```
var koa = require('koa');
var app = koa();

app.use(function *(){
  function delay(interval) {
    return function (done) {
      setTimeout(function () {
        done(null, 'delay done');
      }, interval);
    };
  }

  var time = Date.now();

  console.log('收到请求');
  var body = yield delay(5000);
  console.log(Date.now() - time);

  this.body = body;
});

app.listen(3000);
```

异步2

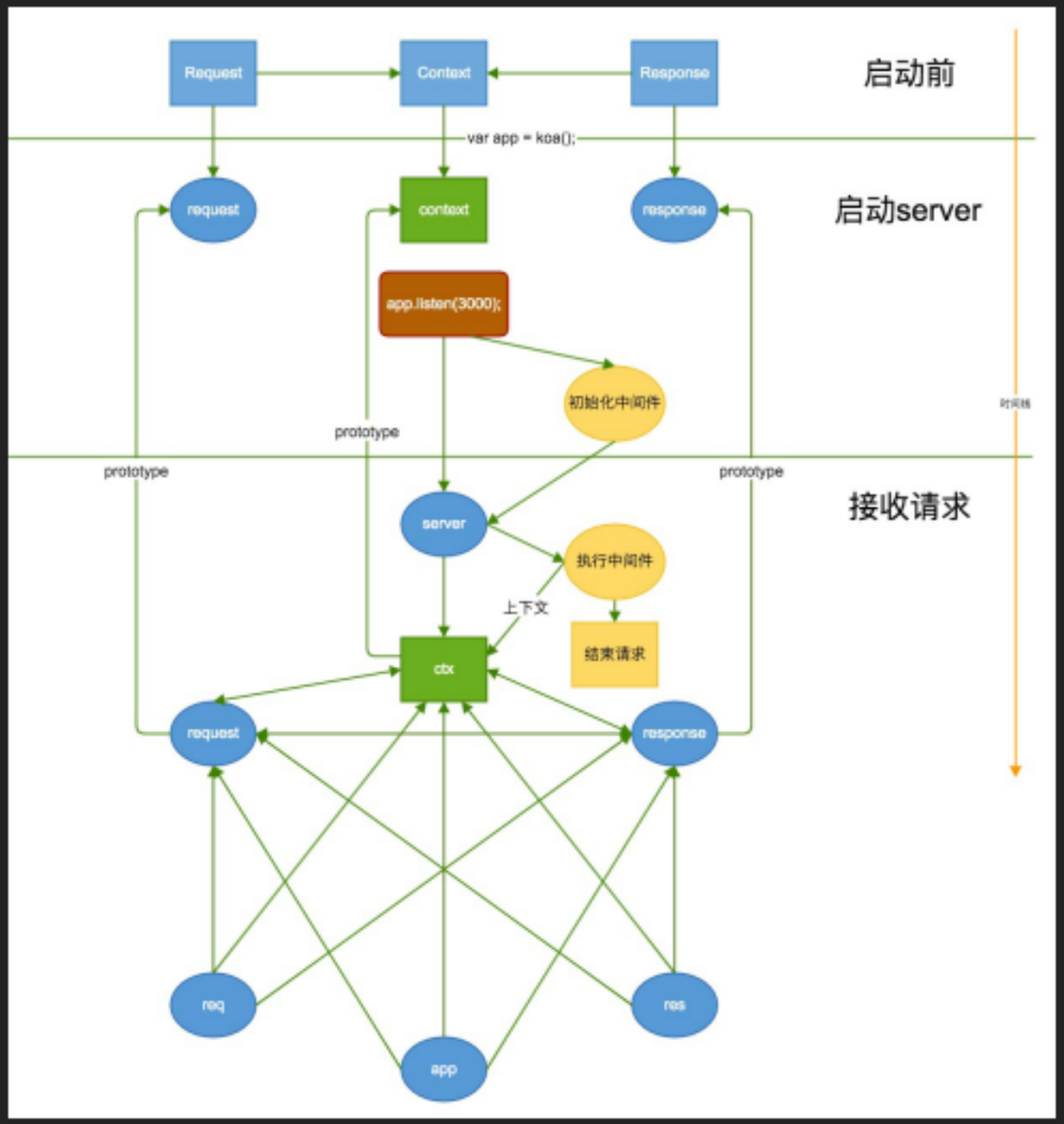
```
app.use(function *(){
  // 先获取购物车中的数据列表
  var cart = yield cart.findByUserId(userID);

  // 在获取购物车中第一个物品的详细信息
  this.body = yield commodity.findById(cart[0].id);
});
```

并发

```
app.use(function *(){
  var userInfo = cart.findByUserId(userID);
  var cart = cart.findByUserId(userID);
  this.body = yield {userInfo: userInfo, cart: cart};
});
```

- 简介
- Koa如何使用
- **深入解析**
- 总结

深入解析

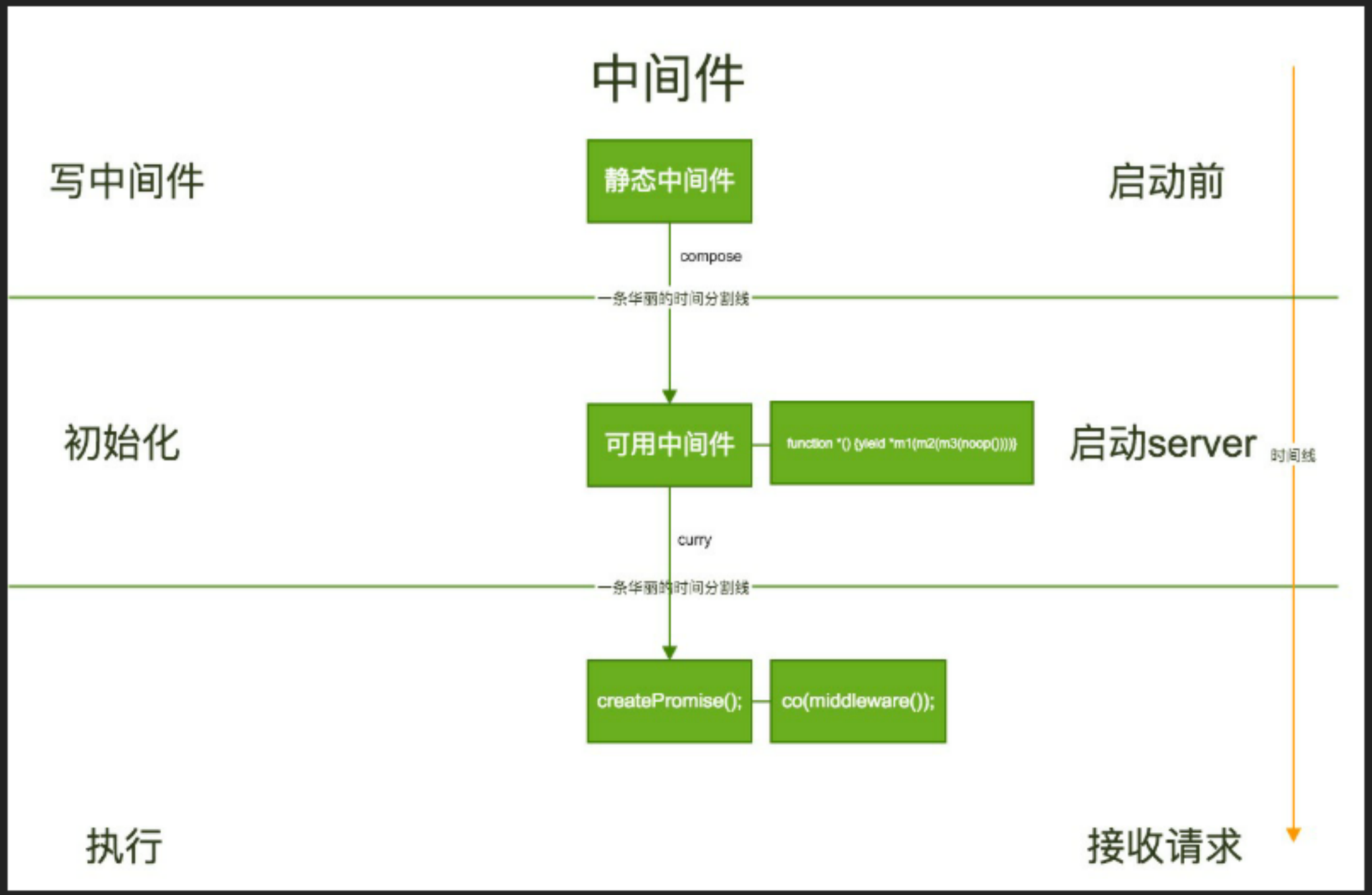

Request
Context
Response
启动前
var app = koa();
request
context
response
启动server
app.listen(3000);
初始化中间件
prototype
prototype
prototype
server
执行中间件
上下文
结束请求
ctx
request
response
req
res
app
接收请求

两个重要的部分

中间件 - Middleware

- Generator
- Promise
- Co

中间件 - 流程图


中间件
写中间件
静态中间件
启动前
compose
一条华丽的时间分割线
初始化
可用中间件
function *() {yield *m1(m2(m3(noop())))}
启动server
时间线
curry
一条华丽的时间分割线
createPromise();
co(middleware());
执行
接收请求

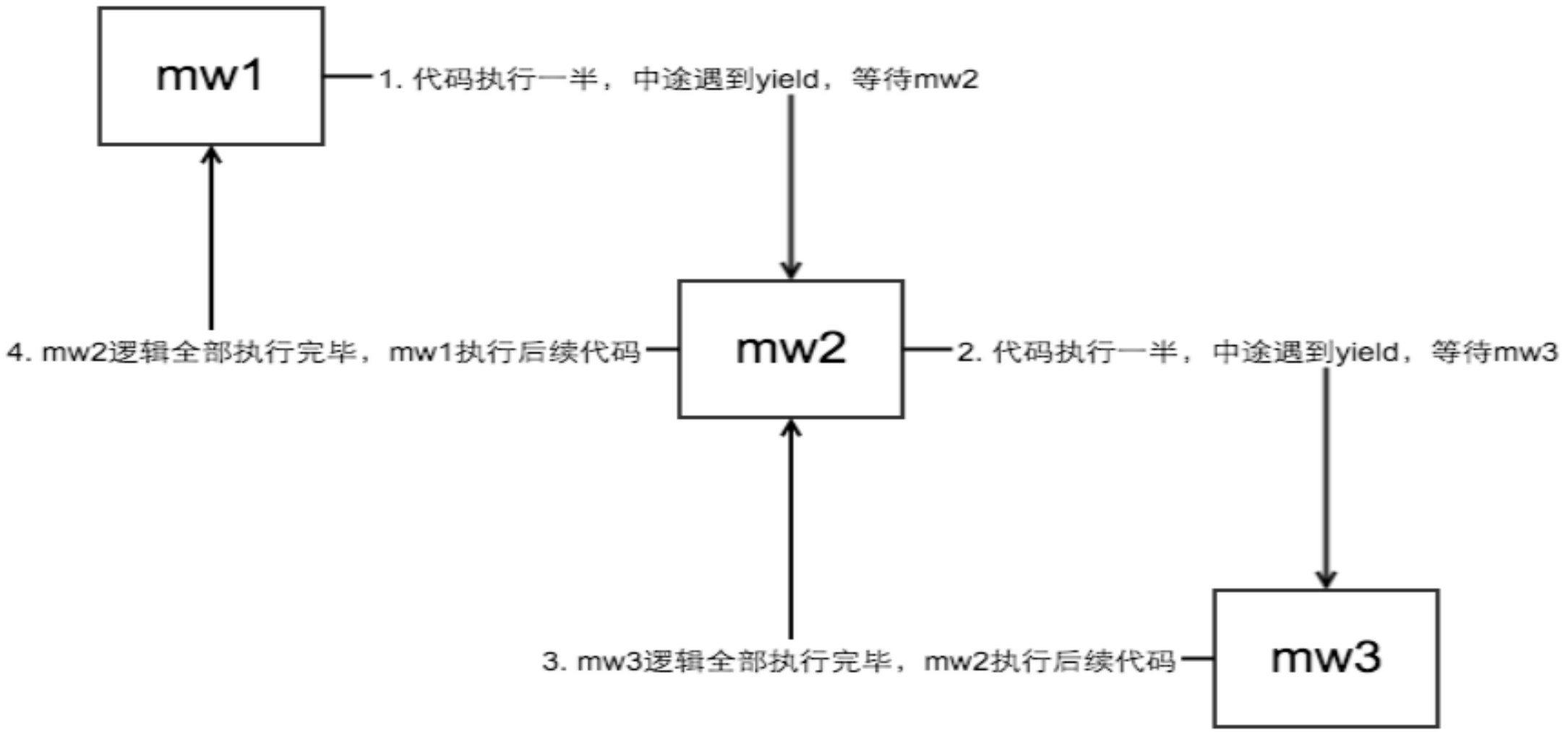
Co

处理异步（与async有点像）

koa充分利用co特性，通过3个步骤实现中间件逻辑

yield链必须满足两个条件

yield 链


mw1
1. 代码执行一半，中途遇到yield，等待mw2
mw2
2. 代码执行一半，中途遇到yield，等待mw3
mw3
3. mw3逻辑全部执行完毕，mw2执行后续代码
4. mw2逻辑全部执行完毕，mw1执行后续代码

伪代码

```
new Promise(function(resolve, reject) {
  // 我是中间件1
  yield new Promise(function(resolve, reject) {
    // 我是中间件2
    yield new Promise(function(resolve, reject) {
      // 我是中间件3
      yield new Promise(function(resolve, reject) {
        // 我是body
      });
      // 我是中间件3
    });
    // 我是中间件2
  });
  // 我是中间件1
});
```

co的原理

接收一个参数

```
function *() {
  yield *m1(m2(m3(noop())))
}
```

next是下一个中间件

```
app.use(function* f1(next) {
  console.log('f1: pre next');
  yield next;
  console.log('f1: post next');
});

app.use(function* f2(next) {
  console.log('  f2: pre next');
  yield next;
  console.log('  f2: post next');
});

app.use(function* f3(next) {
  console.log('  f3: pre next');
  yield next;
  console.log('  f3: post next');
});

app.use(function* (next) {
  this.body = 'hello world';
});
```

co源代码片段

```
function co(gen) {
  return new Promise(function(resolve, reject) {
    onFulfilled();

    function onFulfilled(res) {
      var ret;
      try {
        ret = gen.next(res);
      } catch (e) {
        return reject(e);
      }
      next(ret);
    }

    function next(ret) {

      if (ret.done) return resolve(ret.value);
      var value = toPromise.call(ctx, ret.value);

      if (value && isPromise(value)) return value.then(onFulfilled, onRejected);

      return onRejected(new TypeError('You may only yield a function, promise, generator, a
        + 'but the following object was passed: "' + String(ret.value) + '"'));
    }
```

co 主要其实干了两件事

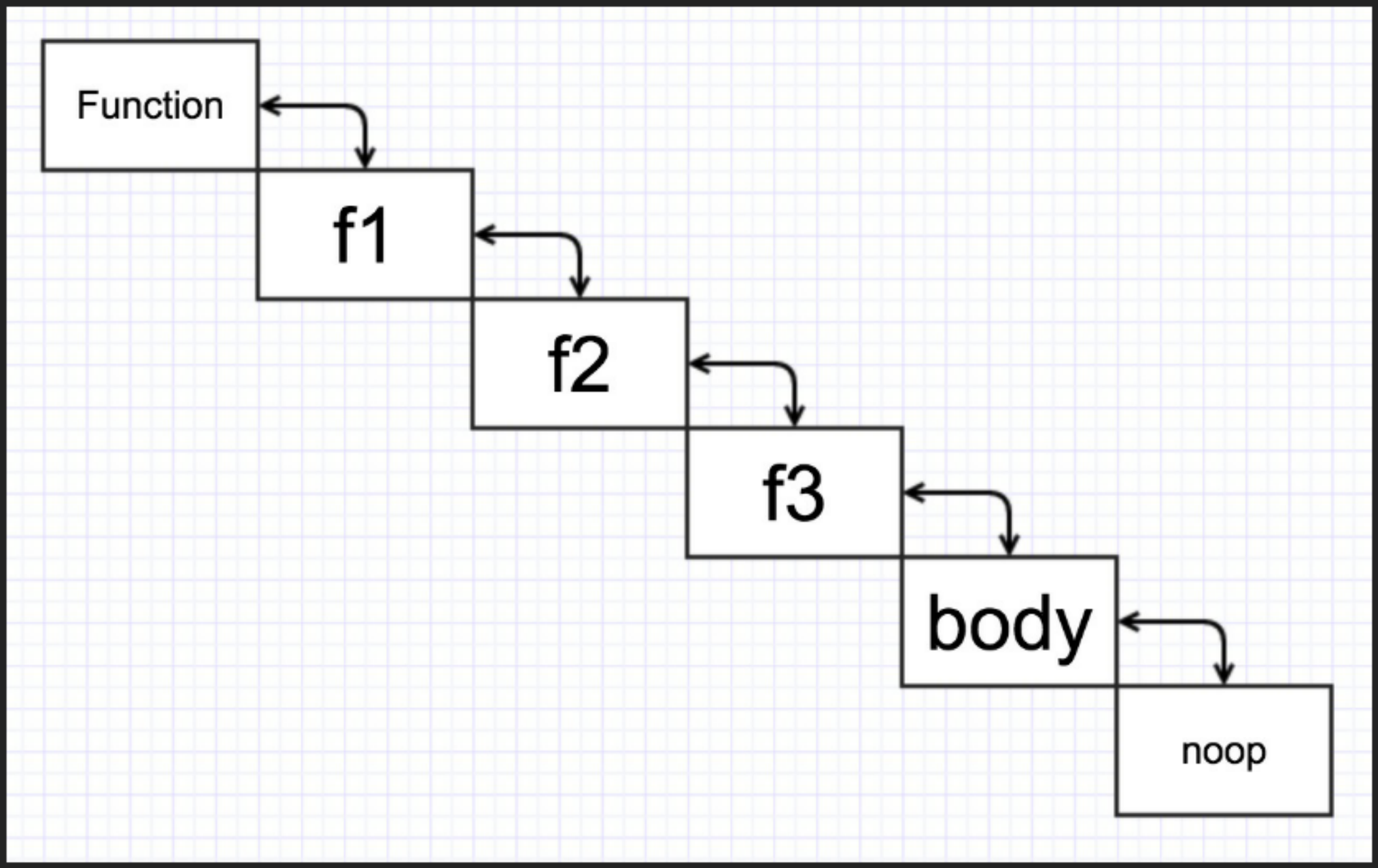
- 执行gen.next
- 把yield返回的内容转换成promise并监听

一个重要的细节 - onFulfilled的执行机制

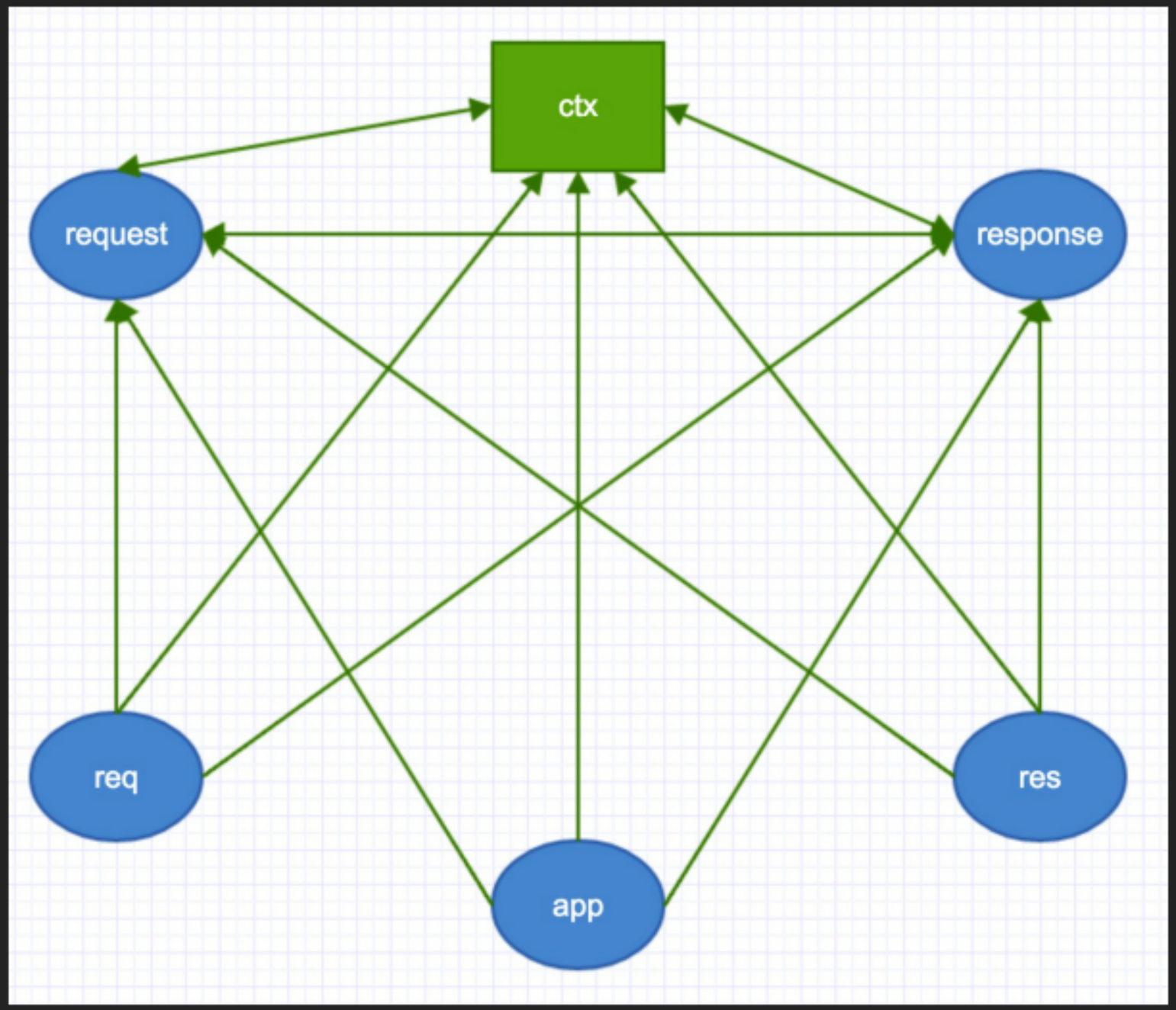
例子

```
var koa = require('koa');
var app = koa();

app.use(function* f1(next) {
  console.log('f1: pre next');
  yield next;
  console.log('f1: post next');
  yield next;
  console.log('f1: done');
});

app.use(function* f2(next) {
  console.log('  f2: pre next');
  yield next;
  console.log('  f2: post next');
  yield next;
  console.log('  f2: done');
});

app.use(function* f3(next) {
  console.log('  f3: pre next');
  yield next;
  console.log('  f3: post next');
  yield next;
```

co流程图



上下文 - Context


ctx
request
response
req
res
app

koa源代码片段

```
function(req, res){
  res.statusCode = 404;
  var ctx = self.createContext(req, res);

  fn.call(ctx).then(function () {
    respond.call(ctx);
  }).catch(ctx.onerror);
}

/**
 * Initialize a new context.
 *
 * @api private
 */

app.createContext = function(req, res){
  var context = Object.create(this.context);
  var request = context.request = Object.create(this.request);
  var response = context.response = Object.create(this.response);
  context.app = request.app = response.app = this;
  context.req = request.req = response.req = req;
  context.res = request.res = response.res = res;
  request.ctx = response.ctx = context;
  request.response = response;
```

谢谢

- 深入浅出 Koa：https://github.com/berwin/Blog/issues/8
- 深入浅出 Koa2：https://github.com/berwin/Blog/issues/9