Koa2源码解读

课前准备

1. koa2 https://github.com/koajs/koa

课堂目标

1. 手写koa
2. 手写static中间件

课堂主题

1. koa 原理
2. context
3. Application剖析
4. 中间件机制
5. 常见中间件

知识点

koa 原理：

```
// 创建kkb.js
const http = require("http");

class KKB {
```

```
  listen(...args) {
    const server = http.createServer((req, res) => {
      this.callback(req, res);
    });
    server.listen(...args);
  }
  use(callback) {
    this.callback = callback;
  }
}
module.exports = KKB;

// 调用，app.js
const KKB = require("./kkb");
const app = new KKB();

app.use((req, res) => {
  res.writeHead(200);
  res.end("hi kaikeba");
});

app.listen(3000, () => {
  console.log("监听端口3000");
});
```

context

- 封装request、response和context

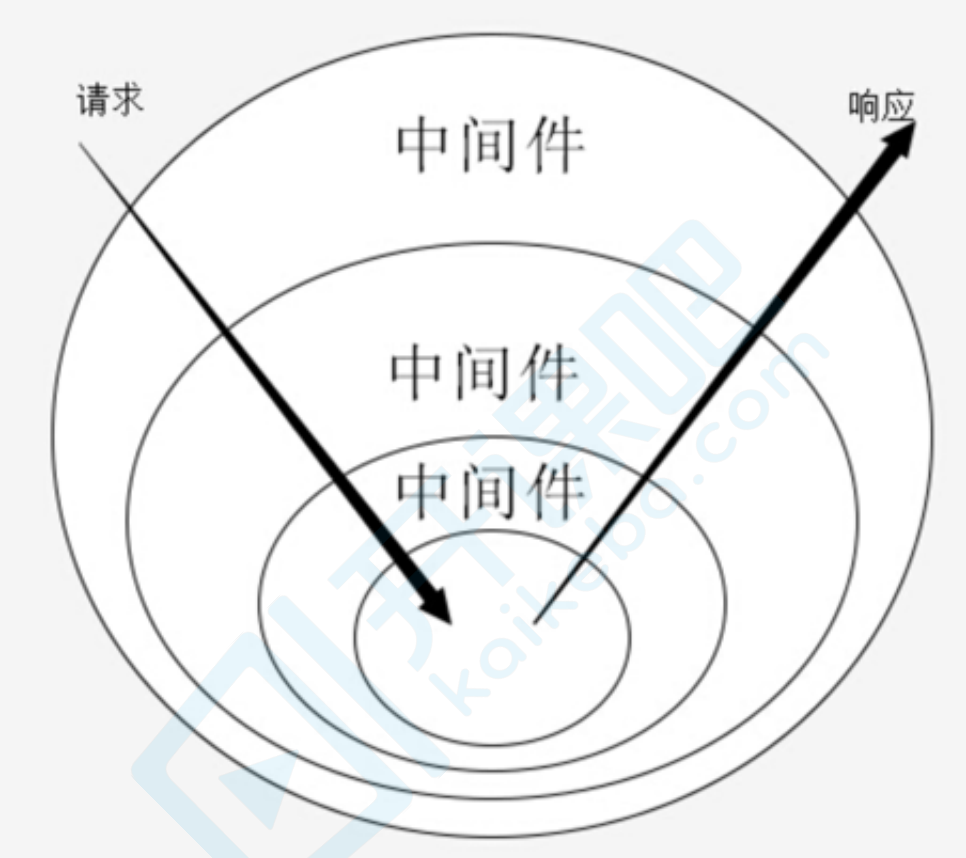
```
// request.js
module.exports = {
  get url() {
    return this.req.url;
  }
};


// response.js
module.exports = {
  get body() {
    return this._body;
  },
  set body(val) {
    this._body = val;
  }
};
```

```
// context.js
module.exports = {
  get url() {
    return this.request.url;
  },
  get body() {
    return this.response.body;
  },
  set body(val) {
    this.response.body = val;
  }
};


// kkb.js
// 导入这三个类
const context = require("./context");
const request = require("./request");
const response = require("./response");

class KKB {
  listen(...args) {
    const server = http.createServer((req, res) => {
      // 创建上下文
      let ctx = this.createContext(req, res);
      // 响应
      res.end(ctx.body);
    });
    // ...
  }
  // 构建上下文，把res和req都挂载到ctx之上，并且在ctx.req和ctx.request.req同时保存
  createContext(req, res) {
    const ctx = Object.create(context);
    ctx.request = Object.create(request);
    ctx.response = Object.create(response);

    ctx.req = ctx.request.req = req;
    ctx.res = ctx.response.res = res;
    return ctx;
  }
}
```

使用

```
// app.js
app.use(ctx=>{
    ctx.body = 'hehe'
})
```

中间件



- 异步中间件：要支持async + await的中间件，要等异步结束后，再执行下一个中间件。

```
function compose(middlewares) {
  return function() {
    return dispatch(0);
    // 执行第0个
    function dispatch(i) {
      let fn = middlewares[i];
      if (!fn) {
        return Promise.resolve();
      }
      return Promise.resolve(
        fn(function next() {
          // promise完成后，再执行下一个
          return dispatch(i + 1);
        })
```

```
      );
    }
  };
}

async function fn1(next) {
  console.log("fn1");
  await next();
  console.log("end fn1");
}

async function fn2(next) {
  console.log("fn2");
  await delay();
  await next();
  console.log("end fn2");
}

function fn3(next) {
  console.log("fn3");
}

function delay() {
  return new Promise((reslove, reject) => {
    setTimeout(() => {
      reslove();
    }, 2000);
  });
}

const middlewares = [fn1, fn2, fn3];
const finalFn = compose(middlewares);
finalFn();
```

- compose用在koa中，kkb.js

```
const http = require("http");
const context = require("./context");
const request = require("./request");
const response = require("./response");

class KKB {
  // 初始化中间件数组
  constructor() {
    this.middlewares = [];
  }
  listen(...args) {
    const server = http.createServer(async (req, res) => {
      const ctx = this.createContext(req, res);
```

```
      // 中间件合成
      const fn = this.compose(this.middlewares);
      // 执行合成函数并传入上下文
      await fn(ctx);
      res.end(ctx.body);
    });
    server.listen(...args);
  }
  use(middleware) {
    // 将中间件加到数组里
    this.middlewares.push(middleware);
  }
  // 合成函数
  compose(middlewares) {
    return function(ctx) { // 传入上下文
      return dispatch(0);
      function dispatch(i) {
        let fn = middlewares[i];
        if (!fn) {
          return Promise.resolve();
        }
        return Promise.resolve(
          fn(ctx, function next() {// 将上下文传入中间件，mid(ctx,next)
            return dispatch(i + 1);
          })
        );
      }
    };
  }
  createContext(req, res) {
    let ctx = Object.create(context);
    ctx.request = Object.create(request);
    ctx.response = Object.create(response);

    ctx.req = ctx.request.req = req;
    ctx.res = ctx.response.res = res;
    return ctx;
  }
}
module.exports = KKB;
```

使用，app.js

```
function sleep() {
  return new Promise((reslove, reject) => {
    setTimeout(() => {
      reslove();
    }, 2000);
  });
```

```
}

app.use(async (ctx, next) => {
  ctx.body = "1";
  setTimeout(() => {
    ctx.body += "2";
  }, 2000);
  await next();
  ctx.body += "3";
});

app.use(async (ctx, next) => {
  ctx.body += "4";
  await delay();
  await next();
  ctx.body += "5";
});

app.use(async (ctx, next) => {
  ctx.body += "6";
});
```

> koa-compose的源码

常见koa中间件的实现

- 实现logger中间件

```
module.exports = async function(ctx, next) {
    // 拦截操作请求 request
    const start = new Date().getTime()
    console.log(`start: ${ctx.url}`);
    await next();
    const end = new Date().getTime()
    console.log(`请求${ctx.url}, 耗时${parseInt(end-start)}ms`)
};
```

- 请求拦截：黑名单中存在的ip访问将被拒绝

```
// interceptor.js
module.exports = async function(ctx, next) {
  const { res, req } = ctx;
  const blackList = ['127.0.0.1'];
  const ip = getClientIP(req);

  if (blackList.includes(ip)) {//出现在黑名单中将被拒绝
    ctx.body = "not allowed";
```

```
  } else {
    await next();
  }
};
function getClientIP(req) {
  return (
    req.headers["x-forwarded-for"] || // 判断是否有反向代理 IP
    req.connection.remoteAddress || // 判断 connection 的远程 IP
    req.socket.remoteAddress || // 判断后端的 socket 的 IP
    req.connection.socket.remoteAddress
  );
}

// app.js
app.use(require("./interceptor"));
app.listen(3000, '0.0.0.0', () => {
  console.log("监听端口3000");
});
```

- 静态文件服务koa-static

```
// static.js
const fs = require("fs");
const path = require("path");

module.exports = (dirPath = "./public") => {
  return async (ctx, next) => {
    if (ctx.url.indexOf("/public") === 0) {
      // public开头 读取文件
      const url = path.resolve(__dirname, dirPath);
      const fileBaseName = path.basename(url);
      const filepath = url + ctx.url.replace("/public", "");
      console.log(filepath);
      // console.log(ctx.url,url, filepath, fileBaseName)
      try {
        stats = fs.statSync(filepath);
        if (stats.isDirectory()) {
          const dir = fs.readdirSync(filepath);
          // const
          const ret = ['<div style="padding-left:20px">'];
          dir.forEach(filename => {
            console.log(filename);
            // 简单认为不带小数点的格式，就是文件夹，实际应该用statSync
            if (filename.indexOf(".") > -1) {
              ret.push(
                `<p><a style="color:black" href="${
                  ctx.url
                }/${filename}">${filename}</a></p>`
              );
```

```
            } else {
              // 文件
              ret.push(
                `<p><a href="${ctx.url}/${filename}">${filename}</a></p>`
              );
            }
          });
          ret.push("</div>");
          ctx.body = ret.join("");
        } else {
          console.log("文件");

          const content = fs.readFileSync(filepath);
          ctx.body = content;
        }
      } catch (e) {
        // 报错了 文件不存在
        ctx.body = "404, not found";
      }
    } else {
      // 否则不是静态资源，直接去下一个中间件
      await next();
    }
  };
};
```

```
// 使用
const static = require('./static')
app.use(static('./public'));
```

- 路由 router

routes()的返回值是一个中间件，由于需要用到method，所以需要挂载method到ctx之上，修改request.js

```
module.exports = {
  get url(){
    return this.req.url
  },
  get method(){
    return this.req.method.toLowerCase()
  }
}
```

```
class Router {
    constructor() {
```

```
    this.stack = [];
  }
  register(path, methods, middleware) {
    let route = {path, methods, middleware}
    this.stack.push(route);
  }
  // 现在只支持get和post，其他的同理
  get(path,middleware){
    this.register(path, 'get', middleware);
  }
  post(path,middleware){
    this.register(path, 'post', middleware);
  }
  routes() {
    let stock = this.stack;
    return async function(ctx, next) {
      let currentPath = ctx.url;
      let route;

      for (let i = 0; i < stock.length; i++) {
        let item = stock[i];
        if (currentPath === item.path &&
item.methods.indexOf(ctx.method) >= 0) {
          // 判断path和method
          route = item.middleware;
          break;
        }
      }

      if (typeof route === 'function') {
        route(ctx, next);
        return;
      }

      await next();
    };
  }
}
module.exports = Router;
```

使用

```
const Koa = require('./kkb')
const Router = require('./router')
const app = new Koa()
const router = new Router();

router.get('/index', async ctx => {
  console.log('index,xx')
```

```
  ctx.body = 'index page';
});
router.get('/post', async ctx => { ctx.body = 'post page'; });
router.get('/list', async ctx => { ctx.body = 'list page'; });

router.post('/index', async ctx => { ctx.body = 'post page'; });

// 路由实例输出父中间件 router.routes()
app.use(router.routes());

app.listen(3000,()=>{
  console.log('server runing on port 9092')
})
```