

ts项目架构

课程目标

- TypeScript实现类装饰器和方法装饰
- 搭建Node TS开发环境
- 基于装饰器的Router Validation Models

项目结构

1. package.json创建: `npm init -y`
2. 开发依赖安装: `npm i typescript ts-node-dev tslint @types/node -D`

`npm i sequelize-typescript@0.6.11`

`npm i sequelize@6.3.3`

1. 启动脚本

```
"scripts": {
  "start": "ts-node-dev ./src/index.ts -P tsconfig.json --no-cache",
  "build": "tsc -P tsconfig.json && node ./dist/index.js",
  "tslint": "tslint --fix -p tsconfig.json"
}
```

2. 加入tsconfig.json

```
{
  "compilerOptions": {
    "outDir": "./dist",
    "target": "es2017",
    "module": "commonjs", // 组织代码方式
    "sourceMap": true,
    "moduleResolution": "node", // 模块解决策略
    "experimentalDecorators": true, // 开启装饰器定义
    "allowsSyntheticDefaultImports": true, // 允许es6方式import
    "lib": ["es2015"],
    "typeRoots": ["./node_modules/@types"],
  },
  "include": ["src/**/*.ts"]
}
```

3. 创建入口文件./src/index.ts

```
console.log('hello');
```

4. 运行测试: `npm start`

项目基础代码

1. 安装依赖: `npm i koa koa-static koa-body koa-xtime -S`

2. 编写基础代码, index.ts

```
import * as Koa from 'koa';
import * as bodify from 'koa-body';
import * as serve from 'koa-static';
import * as timing from 'koa-xtime';

const app = new Koa();

app.use(timing());
app.use(serve(`${__dirname}/public`));

app.use(
  bodify({
    multipart: true,
    // 使用非严格模式, 解析 delete 请求的请求体
    strict: false,
  })
);

app.use((ctx: Koa.Context) => {
  ctx.body = 'hello'
})

app.listen(3000, () => {
  console.log('服务器启动成功');
});
```

3. 测试: `npm start`

- 实现一个装饰器
- 调用loader

路由定义及发现

1. 创建路由./src/routes/user.ts

```
import * as Koa from 'koa';

const users = [{ name: 'tom', age: 20 }, { name: 'tom', age: 20 }];
export default class User {
  @get('/users')
  public list(ctx: Koa.Context) {
    ctx.body = { ok: 1, data: users };
  }
}
```

开课吧web全栈架构师

```

    }

    @post('/users')
    public add(ctx: Koa.Context) {
        users.push(ctx.request.body);
        ctx.body = { ok: 1 }
    }
}

```

知识点补充：装饰器的编写，以@get('/users')为例，它是函数装饰器且有配置项，其函数签名为：

```

function get(path) {
    return function(target, property, descriptor) {}
}

```

另外需解决两个问题：

1. 路由发现
2. 路由注册

2. 路由发现及注册，创建./utils/route-decorcs.ts

```

import * as glob from 'glob';
import * as Koa from 'koa';
import * as KoaRouter from 'koa-router';

type HTTPMethod = 'get' | 'put' | 'del' | 'post' | 'patch';
type LoadOptions = {
    /**
     * 路由文件扩展名，默认值是`.js,ts`
     */
    extname?: string;
};
type RouteOptions = {
    /**
     * 适用于某个请求比较特殊，需要单独制定前缀的情形
     */
    prefix?: string;
    /**
     * 给当前路由添加一个或多个中间件
     */
    middlewares?: Array<Koa.Middleware>;
};
const router = new KoaRouter()
export const get = (path: string, options?: RouteOptions) => {
    return (target, property, descriptor) => {
        const url = options && options.prefix ? options.prefix + path : path
        router['get'](url, target[property])
    }
}
export const post = (path: string, options?: RouteOptions) => {
    return (target, property, descriptor) => {

```

```

    const url = options && options.prefix ? options.prefix + path : path
    router['post'](url, target[property])
  }
}

```

解决get post put delete方法公用逻辑

需要进一步对原有函数进行柯里化

```

const router = new KoaRouter()
const method = method => (path: string, options?: RouteOptions) => {
  return (target, property, descriptor) => {
    const url = options && options.prefix ? options.prefix + path : path
    router[method](url, target[property])
  }
}
export const get = method('get')
export const post = method('post')

```

router变量 不符合函数式编程引用透明的特点 对后面移植不利

所以要再次进行柯里化

```

const router = new KoaRouter()
const decorate = (method: HTTPMethod, path: string, options: RouteOptions = {}, router: KoaRouter) => {
  return (target, property: string) => {
    const url = options.prefix ? options.prefix + path : path
    router[method](url, target[property])
  }
}
const method = method => (path: string, options?: RouteOptions) =>
  decorate(method, path, options, router)

export const get = method('get')
export const post = method('post')

```

```

import * as glob from 'glob';
import * as Koa from 'koa';
import * as KoaRouter from 'koa-router';

type HTTPMethod = 'get' | 'put' | 'del' | 'post' | 'patch';
type LoadOptions = {
  /**
   * 路由文件扩展名, 默认值是`. {js,ts}`
   */
  extname?: string;
};
type RouteOptions = {
  /**
   * 适用于某个请求比较特殊, 需要单独制定前缀的情形
   */
  prefix?: string;
  /**
   * 给当前路由添加一个或多个中间件

```

```

    */
    middlewares?: Array<Koa.Middleware>;
  };
  const router = new KoaRouter()
  const decorate = (method: HTTPMethod, path: string, options: RouteOptions = {}, router: KoaRouter) => {
    return (target, property: string) => {
      const url = options.prefix ? options.prefix + path : path
      router[method](url, target[property])
    }
  }
  const method = method => (path: string, options?: RouteOptions) =>
  decorate(method, path, options, router)
  export const get = method('get')
  export const post = method('post')
  export const put = method('put')
  export const del = method('del')
  export const patch = method('patch')

  export const load = (folder: string, options: LoadOptions = {}): KoaRouter
  => {
    const extname = options.extname || '.{js,ts}'
    glob.sync(require('path').join(folder,
    `./**/*${extname}`)).forEach((item) => require(item))
    return router
  }

```

3. 使用

routes/user.ts

```
import { get, post } from '../utils/decorators'
```

index.ts

```
import { load } from './utils/decorators';
import { resolve } from 'path'
const router = load(resolve(__dirname, './routes'));
app.use(router.routes())
```

4. 数据校验: 可以利用中间件机制实现

添加校验函数, ./routes/user.ts

```

//异步校验接口
const api = {
  findByName(name) {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        if (name === 'xia') {
          reject('用户名已存在')
        } else {
          resolve()
        }
      }, 500);
    })
  }
}

```

```

    })
  }
}

export default class User {
  // 添加中间件选项
  @post('/users', {
    middlewares: [
      async function validation(ctx: Koa.Context, next: () =>
Promise<any>) {
        // 用户名必填
        const name = ctx.request.body.name
        if (!name) {
          throw "请输入用户名";
        }
        // 用户名不能重复
        try {
          await api.findByName(name);
          // 校验通过
          await next();
        } catch (error) {
          throw error;
        }
      }
    ]
  })
  public async add(ctx: Koa.Context) {}
}

```

更新decorators.ts

```

export const load = function(prefix: string, folder: string, options:
LoadOptions = {}): KoaRouter {
  // ...
  route = function(method: HTTPMethod, path: string, options: RouteOptions =
{}) {
    return function(target, property: string, descriptor) {
      // 添加中间件数组
      const middlewares = [];

      // 若设置了中间件选项则加入到中间件数组
      if (options.middlewares) {
        middlewares.push(...options.middlewares);
      }

      // 添加路由处理器
      middlewares.push(target[property]);
      const url = (options.prefix || prefix) + path;
      // router[method](url, target[property]);
      router[method](url, ...middlewares);
    };
  };

  // ...
  return router;
}

```

```
};
```

5. 类级别路由守卫

使用, routes/user.ts

```
@middlewares([
  async function guard(ctx: Koa.Context, next: () => Promise<any>){
    console.log('guard', ctx.header);

    if(ctx.header.token) {
      await next();
    } else {
      throw "请登录";
    }
  }
])
export default class User {}
```

增加中间装饰器, 更新route-decors.ts

```
//增加中间装饰器
export const middlewares = function middlewares(middlewares:
Koa.Middleware[]) {
  return function(target) {
    target.prototype.middlewares = middlewares;
  };
};

//修改load方法
export const load = function(prefix: string, folder: string, options:
LoadOptions = {}): KoaRouter {

  route = function(method: HTTPMethod, path: string, options: RouteOptions
= {}) {
    return function(target, property: string, descriptor) {
      // 晚一拍执行路由注册: 因为需要等类装饰器执行完毕
      process.nextTick(() => {
        let mws = [];
        // 获取class上定义的中间件
        if (target.middlewares) {
          middlewares.push(...target.middlewares);
        }
        // ...
      });
    };
  };

  return router;
};
```

数据库整合

1. 安装依赖: `npm i -S sequelize sequelize-typescript reflect-metadata mysql2`
2. 初始化, index.ts

```
import { Sequelize } from 'sequelize-typescript';

const database = new Sequelize({
  port: 3306,
  database: 'kaikeba',
  username: 'root',
  password: 'example',
  dialect: 'mysql',
  modelPaths: [`${__dirname}/model`],
});
database.sync({force: true})
```

3. 创建模型

```
// model/user.js
import { Table, Column, Model, DataType } from 'sequelize-typescript';

@Table({modelName: 'users'})
export default class User extends Model<User> {
  @Column({
    primaryKey: true,
    autoIncrement: true,
    type: DataType.INTEGER,
  })
  public id: number;

  @Column(DataType.CHAR)
  public name: string;
}
```

4. 使用模型, routes/user.ts

```
import model from '../model/user';

export default class User {

  @get('/users')
  public async list(ctx: Koa.Context) {
    const users = await model.findAll()
    ctx.body = { ok: 1, data: users };
  }
}
```

框架不足

- Restful接口

- model可以自动加载到ctx中
- server层自动加载

