kkb-honor-admin

基于OAS的接口定义和数据mock

OpenAPI Specification简称OAS，是一套接口定义规范，使用它定义接口可以用于未来的文档生成和代码生成。

> [swagger官网]
>
> [OpenAPI Specification]

我们项目中的数据mock

存储于mock文件夹下，入口mock-server.ts

```
// 读取api配置文件
const apiDefinition = yaml.load(path.resolve(__dirname, 'swagger.yml'))
// 基于配置文件创建数据mock函数
const options = {
  security: {
    AccessTokenAuth: accessTokenAuth
  }
}
// api是路由定义
const connectSwagger = connector(api, apiDefinition, options)
connectSwagger(app)
// Print swagger router api summary
const apiSummary = summarise(apiDefinition)
console.log(apiSummary)
```

api定义文件是swagger.yml

```
openapi: 3.0.0 # 版本号
info: # 描述信息
  ...
servers: # 接口服务器地址，也用于咱们路由声明前缀
  - url: /mock-api/v1
```

```
tags: # 接口文档中的标签名称
  - name:articles
    description: ...
path: # 接口路径和定义
  /articles:
    get:
      ...
    post:
      ...
components: # 可复用组件，数据结构、响应定义、安全认证定义等
  securitySchemas:
    ...
  responses:
    ...
  schemas:
    ...
```

路由定义文件api.ts，集中导出路由定义

```
export * from './articles'
export * from './role'
export * from './transactions'
export * from './users'
```

每个文件具体实现

```
export const getArticles = (req: Request, res: Response) => {}
```

> 可通过http://localhost:9528/mock-api/v1/articles体验

编写一个接口：getPlayers

能够根据用户名过滤，能够分页的获取玩家列表。

定义接口，swagger.yml

> 为了便于后续接口定义、编写和查看，请先安装如下VSCode扩展：
>
> - YAML
> - OpenAPI(Swagger)
> - Swagger Viewer

```
/players: # path前缀
```

```yaml
    get: # method类型
      tags: # 所属标签
        - players
      summary: 获取玩家信息列表 # 概要
      description: 返回玩家信息列表 # 描述
      operationId: getPlayers # 操作id，必要，映射到实现函数
      parameters: # 参数
        - name: acountname
          in: query
          description: 账号用于查询用户
          schema:
            type: string
        - name: nickname
          in: query
          description: 昵称用于查询用户
          schema:
            type: string
        - name: page
          in: query
          description: 页码
          schema:
            type: integer
            format: int32
            minimum: 1
        - name: limit
          in: query
          description: 每页数量
          schema:
            type: integer
            format: int32
            minimum: 0
            maximum: 50
      responses:
        200:
          description: OK
          content:
            application/json:
              schema:
                type: object
                properties:
                  code:
                    type: integer
                    format: int32
                  data:
                    type: object
                    properties:
                      total:
                        type: integer
                        format: int32
                      items:
                        type: array
                        items:
                          $ref: '#/components/schemas/Player'
```

实现接口，创建players.ts

```
import { Response, Request } from 'express'

export const getArticle = (req: Request, res: Response) => {
  res.json([1,2,3])
}
```

导出，api.ts

```
export * from './players'
```

> 测试一下：http://localhost:9528/mock-api/v1/players

玩家数据mock

类型声明，src/api/types.d.ts

```
export interface Player {
  id: number,
  acountname: string,
  nickname: string,
  avatar: string,
  level: number, // 用户等级
  exp: number, // 用户经验值
  rank: number, // 排位赛段位
  bravepoints: number,
  winningstreak: number,
  wanttoplay: Hero[]
}

export interface Hero {
  id: number,
  name: string,
  icon: string,
  classify: string[]
}
```

模拟英雄数据，heros.ts

```
import { Hero } from "../src/api/types";

export const heros: Hero[] = [
    { id: 1, name: '百里玄策', icon: 'xc.jpg', classify: ['刺客'] },
    { id: 2, name: '孙悟空', icon: 'swc.jpg', classify: ['刺客'] },
    { id: 3, name: '鲁班七号', icon: 'lbqh.jpg', classify: ['射手'] },
    { id: 4, name: '后羿', icon: 'hy.jpg', classify: ['射手'] },
    { id: 5, name: '王昭君', icon: 'wzj.jpg', classify: ['法师'] },
    { id: 6, name: '貂蝉', icon: 'dc.jpg', classify: ['法师'] },
    { id: 7, name: '钟馗', icon: 'zk.jpg', classify: ['法师', '辅助'] },
    { id: 8, name: '牛魔', icon: 'nm.jpg', classify: ['坦克', '辅助'] },
    { id: 9, name: '亚瑟', icon: 'ys.jpg', classify: ['战士', '坦克'] },
    { id: 10, name: '吕布', icon: 'lb.jpg', classify: ['战士'] },
]
```

模拟玩家数据，player.ts

```
import faker from 'faker'
import { Player, Hero } from '../src/api/types'
import { heros } from "./heros";

// 使用中文名称
faker.locale = 'zh_CN'

// 模拟用户数据
const playerCount = 100
const playerList: Player[] = []

for (let i = 1; i < playerCount; i++) {
  playerList.push({
    id: i,
    acountname: faker.name.findName(),
    avatar: faker.image.avatar(),
    bravepoints: faker.random.number(1000),
    exp: faker.random.number(10000),
    level: faker.random.number(30),
    nickname: faker.name.findName(),
    rank: faker.random.number(200),
    wanttoplay: Array.from(genWantoplay()),
    winningstreak: faker.random.number(10)
  })
}

// 模拟想用英雄：只有三个不能重复
function genWantoplay() {
  let wanttoplay: Set<Hero> = new Set();
  while(wanttoplay.size < 3) {
    wanttoplay.add(heros[faker.random.number(9)])
  }
  return wanttoplay;
}
```

返回玩家列表数据，players.ts

```
export const getPlayers = (req: Request, res: Response) => {
  const { acountname, nickname, page = 1, limit = 20 } = req.query

  let mockList = playerList.filter(item => {
    if (acountname && item.acountname.indexOf(acountname) < 0) return false
    if (nickname && item.nickname.indexOf(nickname) < 0) return false
    return true
  })

  const pageList = mockList.filter((_, index) => index < limit * page && index
>= limit * (page - 1))

  return res.json({
    code: 20000,
    data: {
      total: mockList.length,
      items: pageList
    }
  })
}
```

编写getPlayer接口

根据玩家id获取该玩家详情。

定义接口，swagger.yml

```
/players/{id}: # 多了一个参数
    get:
      tags:
        - players
      summary: 获取指定玩家信息
      description: 获取id对应玩家信息
      operationId: getPlayer
      security: # 定义security可以保护该路由，要求用户授权
```

```
          - AccessTokenAuth: []
        parameters:
          - name: id
            in: path
            required: true
            description: Player id.
            schema:
              type: integer
              format: int64
        responses:
          200:
            description: OK
          401:
            $ref: '#/components/responses/Unauthorized'
          404:
            $ref: '#/components/responses/NotFound'
```

实现接口，players.ts

```
export const getPlayer = (req: Request, res: Response) => {
  const { id } = req.params
  for (const player of playerList) {
    if (player.id.toString() === id) {
      return res.json({
        code: 20000,
        data: {
          player
        }
      })
    }
  }
  return res.json({
    code: 70001,
    message: 'player not found'
  })
}
```

编写createPlayer接口

创建用户。

定义接口，swagger.yml

```
/players:
    get:
      ...
    post: # 在get后面增加一个post定义
      summary: 创建玩家
      tags:
        - players
      description: 创建玩家，返回新创建玩家数据
      operationId: createPlayer
      security:
        - AccessTokenAuth: []
      parameters:
        - name: article
          in: query
          required: true
          description: 新建玩家数据
          schema:
            $ref: '#/components/schemas/Player'
      responses:
        200:
          description: OK
        401:
          $ref: '#/components/responses/Unauthorized'
```

实现接口，player.ts

```
export const createPlayer = (req: Request, res: Response) => {
  const { player } = req.body
  return res.json({
    code: 20000,
    data: {
      player
    }
  })
}
```

编写updatePlayer接口

更新某个指定id的player

定义接口，swagger.yml

```
/players/{id}:
    get:
      ...
    put: # 在get后面定义put接口
      tags:
        - players
      summary: 更新玩家信息
      description: 更新指定id的玩家信息
      operationId: updatePlayer
      security:
        - AccessTokenAuth: []
      parameters:
        - name: id
          in: path
          required: true
          description: 玩家id
          schema:
            type: integer
            format: int64
        - name: player
          in: query
          required: true
          description: 要更新的玩家数据
          schema:
            $ref: '#/components/schemas/Player'
      responses:
        200:
          description: OK
        401:
          $ref: '#/components/responses/Unauthorized'
        404:
          $ref: '#/components/responses/NotFound'
```

实现接口，player.ts

```
export const updatePlayer = (req: Request, res: Response) => {
  const { id } = req.params
  const { player } = req.body
  for (const v of playerList) {
    if (v.id.toString() === id) {
      return res.json({
        code: 20000,
        data: {
          player
        }
      })
    }
  }
  return res.json({
    code: 70001,
    message: 'player not found'
  })
}
```

编写接口deletePlayer

删除指定id对应的玩家

接口定义，swagger.yml

```yaml
  /players/{id}:
    # 在put下面添加delete方法定义
    delete:
      tags:
        - players
      summary: 删除玩家信息
      description: 删除指定id玩家信息
      operationId: deletePlayer
      security:
        - AccessTokenAuth: []
      parameters:
        - name: id
          in: path
          required: true
          description: 玩家id.
          schema:
            type: integer
            format: int64
      responses:
        200:
          description: OK
          content:
            application/json:
              schema:
                type: object
                properties:
                  code:
                    type: integer
                    format: int32
        401:
          $ref: '#/components/responses/Unauthorized'
        404:
          $ref: '#/components/responses/NotFound'
```

实现接口，player.ts

```
export const deletePlayer = (req: Request, res: Response) => {
  return res.json({
    code: 20000,
  })
}
```