ThinkJS 2.0 Documentation

快速入门

介绍

ThinkJS 是一款使用 ES6/7 特性全新开发的 Node.js MVC 框架，使用 ES7 中 `async/await`，或者 ES6 中的 `*/yield` 特性彻底解决了 Node.js 中异步嵌套的问题。同时吸收了国内外众多框架的设计理念和思想，让开发 Node.js 项目更加简单、高效。

使用 ES6/7 特性来开发项目可以大大提高开发效率，是趋势所在。并且新版的 Node.js 对 ES6 特性也有了较好的支持，即使有些特性还没有支持，也可以借助 Babel 编译来支持。

特性

使用 ES6/7 特性来开发项目

借助 Babel 编译，可以在项目中大胆使用 ES6/7 所有的特性，无需担心哪些特性当前版本不支持。尤其是使用 `async/await` 或者 `*/yield` 来解决异步回调的问题。

```JavaScript
//user controller, home/controller/user.js
export default class extends think.controller.base {
  //login action
  async loginAction(self){
    //如果是get请求，直接显示登录页面
    if(this.isGet()){
      return this.display();
    }
    //这里可以通过post方法获取所有的数据，数据已经在logic里做了校验
    let data = this.post();
    let md5 = think.md5('think_' + data.pwd);
    //用户名和加密后的密码去匹配数据库中对于的条目
    let result = await this.model('user').where({name: data.name, pwd: md5}).find();
    //如果未匹配到任何数据，表示用户名或者密码错误
    if(think.isEmpty(result)){
      return this.fail('login fail');
    }
    //获取到用户信息后，将用户信息写入session
    await this.session('userInfo', result);
    return this.success();
  }
}
```

上面的代码我们使用了 ES6 里的 `class`，`export`，`let` 以及 ES7 里的 `async` 和 `await` 等特性，虽然查询数据库和写入 `Session` 都是异步操作，但借助 `async/await`，代码都是同步书写的。最后使用 `Babel` 进行编译，就可以稳定运行在 Node.js 的环境中了。

支持多种项目结构和多种项目环境

项目支持单模块模式、普通模式、分模块模式等多种项目结构，可以满足各种项目复杂度的开发。

默认支持 `development`，`testing` 和 `prodution` 3 种项目环境，可以在不同的项目环境下进行不同的配置，满足在不同环境下的配置需求，同时还可以基于项目需要进行扩展。

支持丰富的数据库

ThinkJS 支持 `mysql`，`mongodb`，`sqlite` 等常见的数据库，并且封装了很多操作数据库的接口，无需手动拼接 SQL 语句，还可以自动防止 SQL 注入等安全漏洞。同时支持事务、关联模型等高级功能。

代码自动更新

ThinkJS 内置了一套代码自动更新的机制，文件修改后立即生效，不用重启 Node.js 服务，也不用借助第三方模块。

自动创建 REST 接口

使用 `thinkjs` 命令可以自动创建 REST 接口，不用写任何的代码即可完成 REST API 的开发。如果想在 REST 接口中过滤字段或者进行权限校验，也很方便处理。

支持多种 WebSocket 库

ThinkJS 支持 `socket.io`， `sockjs` 等常见的 WebSocket 库，并且对这些库进行包装，抹平各个库之间接口调用上的差异，给开发者一致的体验。

丰富的测试用例

ThinkJS 含有 1500+ 的测试用例，代码覆盖率达到 95% ，每一次修改都有对应的测试用例来保障框架功能的稳定。

支持命令行调用执行定时任务

ThinkJS 里的 `Action` 除了可以响应用户的请求，同时支持在命令行下访问，借助这套机制就可以很方便的执行定时任务。

Hook 和 Middleware

ThinkJS 使用 Hook 和 Middleware 机制，可以灵活的对访问请求进行拦截处理。

详细的日志

ThinkJS 内置了详细的日志功能，可以很方便的查看各种日志，方便追查问题。

HTTP 请求日志

```
[2015-10-12 14:10:03] [HTTP] GET /favicon.ico 200 5ms
[2015-10-12 14:10:11] [HTTP] GET /zh-CN/doc.html 200 11ms
[2015-10-12 14:10:11] [HTTP] GET /static/css/reset.css 200 3ms
```

Socket 连接日志

```
[2015-10-12 14:13:54] [SOCKET] Connect mysql with mysql://root:root@127.0.0.1:3306
```

错误日志

```
[2015-10-12 14:15:32] [Error] Error: ER_ACCESS_DENIED_ERROR: Access denied for user 'root3'@'localhost' (using password: YES)

[2015-10-12 14:16:12] [Error] Error: Address already in use, port:8360. http://www.thinkjs.org/doc/error.html#EADDRINUSE
```

丰富的路由机制

ThinkJS 支持正则路由、规则路由、静态路由等多种路由机制，并且可以基于模块来设置。可以让 URL 更加简洁的同时又不丢失性能。

支持国际化和多主题

ThinkJS 使用很简单的方法就可以支持国际化和多主题等功能。

与其他框架的对比

与 express/koa 对比

express/koa 是 2 个比较简单的框架，框架本身提供的功能比较简单，项目中需要借助大量的第三方插件才能完成项目的开发，所以灵活度比较高。但使用很多第三方组件一方便提高了项目的复杂度，另一个方便第三方插件质量参差不齐，也会带来内存泄漏等风险。

koa 使用 ES6 里的 `*/yield` 解决了异步回调的问题，但 `*/yield` 只会是个过渡解决方案，会被 ES7 里的 `async/await` 所替代。

而 ThinkJS 提供了整套解决方案，每个功能都经过了严格的性能和内存泄漏等方面的测试，并且在项目中可以直接使用 ES6/7 所有的特性。

与 sails 对比

sails 也是一个提供整套解决方案的 Node.js 框架，对数据库、REST API、安全方面也很多封装，使用起来比较方便。

但 sails 对异步回调的问题还没有优化，还是使用 callback 的方式，给开发带来很大的不便，导致项目中无法较好的使用 ES6/7 特性。

ThinkJS 的不足

上面说了很多 ThinkJS 的优点，当然 ThinkJS 也有很多的不足。如：

- 框架还比较新，缺少社区等方面的支持
- 还没有经过超大型项目的检验

ES6/7 参考文档

关于 ES6/7 特性可以参考下面的文档：

- learn-es2015
- ECMAScript 6 入门
- ECMAScript 6 Features
- ECMAScript 6 compatibility table
- ECMAScript 7 Features
- ECMAScript 7 compatibility table

创建项目

安装 Node.js

ThinkJS 是一款 Node.js 的 MVC 框架，所以安装 ThinkJS 之前，需要先安装 Node.js 环境，可以去 官方 下载最新的安装包进行安装，也可以通过其他一些渠道安装。

安装完成后，在命令行执行 `node -v`，如果能看到对应的版本号输出，则表示安装成功。

ThinkJS 需要 Node.js 的版本 `>=0.12.0`，如果版本小于这个版本，需要升级 Node.js，否则无法启动服务。建议将 Node.js 版本升级到 `4.2.1`。

安装 ThinkJS

通过下面的命令即可安装 ThinkJS：

```bash
npm install thinkjs -g --verbose
```

如果安装很慢的话，可以尝试使用 taobao 的源进行安装。具体如下：

```bash
npm install thinkjs -g --registry=https://registry.npm.taobao.org --verbose
```

注：如果之前安装过 ThinkJS 1.x 的版本，可能需要将之前的版本删除掉，可以通过 `npm uninstall -g thinkjs-cmd` 命令删除。

创建项目

ThinkJS 安装完成后，就可以通过下面的命令创建项目：

```bash
thinkjs new project_path; #project_path为项目存放的目录
```

如果想用 `ES6` 特性来开发项目的话，可以创建一个 `ES6` 模式的项目，具体如下：

```bash
thinkjs new project_path --es6; #project_path为项目存放的目录
```

如果能看见类似下面的输出，表示项目创建成功了：

```
  create : demo/
  create : demo/package.json
  create : demo/.thinkjsrc
  create : demo/nginx.conf
  create : demo/README.md
  create : demo/www/
  create : demo/www/index.js
  create : demo/app
  create : demo/app/common/runtime
  create : demo/app/common/config
  create : demo/app/common/config/config.js
  create : demo/app/common/config/view.js
  create : demo/app/common/config/db.js
  ...
  create : demo/app/home/logic
  create : demo/app/home/logic/index.js
  create : demo/app/home/view
  create : demo/app/home/view/index_index.html

  enter path:
  $ cd demo/

  install dependencies:
  $ npm install

  run the app:
  $ npm start
```

关于创建项目命令的更多信息，请见 扩展功能 -> ThinkJS 命令。

安装依赖

项目安装后，进入项目目录，执行 `npm install` 安装依赖，可以使用 `taobao` 源进行安装。

```bash
npm install --registry=https://registry.npm.taobao.org --verbose
```

编译项目

如果创建项目时加上了 `--es6` 参数，代码需要编译后才能运行。那么需要先在项目下执行命令 `npm run watch-compile` ，这样文件有修改后就会自动编译了。

执行命令后会挂起一个进程，注意不要结束这个进程，其他命令可以再新开一个标签页里执行。

启动项目

在项目目录下执行命令 `npm start` ，如果能看到类似下面的内容，表示服务启动成功。

```
[2015-09-21 20:21:09] [THINK] Server running at http://127.0.0.1:8360/
[2015-09-21 20:21:09] [THINK] ThinkJS Version: 2.0.0
[2015-09-21 20:21:09] [THINK] Cluster Status: closed
[2015-09-21 20:21:09] [THINK] WebSocket Status: closed
[2015-09-21 20:21:09] [THINK] File Auto Reload: true
[2015-09-21 20:21:09] [THINK] App Enviroment: development
```

访问项目

打开浏览器，访问 `http://127.0.0.1:8360/` 即可。

如果是在远程机器，需要通过远程机器的 IP 访问，同时要保证 8360 端口可访问。

项目结构

通过 thinkjs 命令创建完项目后，项目目录结构类似如下：

```
|-- nginx.conf
|-- package.json
|-- src
|   |-- common
|   |   |-- bootstrap
|   |   |   |-- generate_icon.js
|   |   |   `-- middleware.js
|   |   |-- config
|   |   |   |-- config.js
|   |   |   |-- env
|   |   |   |   |-- development.js
|   |   |   |   `-- production.js
|   |   |   |-- hook.js
|   |   |   |-- locale
|   |   |   |   |-- en.js
|   |   |   |   `-- zh-CN.js
|   |   |   `-- route.js
|   |   |-- controller
|   |   |    `-- error.js
|   |   `-- runtime
|   `-- home
|       |-- config
|       |-- controller
|       |   |-- base.js
|       |   `-- index.js
```