JsSaaS开发文档

一：平台介绍

1.1 JsSaaS平台介绍

重要的事情说三遍，平台免费、平台免费、平台免费

JsSaaS平台是IT发展的时代产物，顺应市场需求萌发的一款IT快速开发框架：WEB项目快速开发平台，类似JsNode，但比JsNode简单，更容易开发。

JsSaaS平台依托系统级开发语言作为底层执行语言，将JavaScript快速高效转义成可执行的指令。实现了ECMAScript 6的大部分转义工作，实际开发中已经不存在问题，底层对JavaScript函数进行了原生扩展，以满足快速开发需求。

JsSaaS平台编译时已经集成了运行时需要的源码，在部署时无需依赖环境。Windows生产可执行对exe文件；Linux及Mac生成可执行对二进制文件；支持Microsoft Windows 7/ Server 2008 及以上系统；macOS支持 10.15及以上版本。运行JsSaaS平台无需第三方环境支撑，如Java需要java虚拟机运行环境，jssaas直接运行，运行效率接近C++。

由于采用系统级开发语言做为底层执行语言，支持大并发。每个HTTP请求都是独立运行线程，互不干扰。生产模式下每次HTTP执行的代码就是一个JavaScript文件，每个完整的业务是一个JavaScript文件，确保更新不会相互影响，使得系统升级变得超级简单，只需要将代码文件更新到服务器，无缝刷新代码缓存，即可实现业务

逻辑无缝更新，

最大化减小更新分险，实现无缝更新业务系统。JsSaas的最终将实现生成环境只需要部署一次，后面的业务更新通过刷新缓存更新业务代码，从而实现无缝更新。

站在微服务角度看JsSaaS平台，会发现它时多么优秀，解决了微服务跨系统访问产生对不可控性。一个JavaScript业务文件集成了全部的微服务代码，全部的业务在本机实现，执行跨微服务业务，不需要执行跨微服务请求，特别时碰到数据库事务回滚，更是简单,都在同一业务代码中实现。

JsSaaS平台自带定时器，直接执行业务脚本，最小执行单位：秒。

JsSaaS平台自带队列，有效解决秒杀等并发下单场景，避免超卖，被恶意大并发下单。

JsSaaS平台自带事件锚点，实现动态事件触发，由一个基础事件触发其他业务，可通过锚点配置实现业务扩展。如新用户注册，注册成功需要执行的业务可能随着时间会有不同的业务。通过时间锚点，可以实现功能模块自由配置，不启用的模块，在执行事件锚点会自动过滤掉，已达到自由配置功能模块需求，开发者可以通过不同客户需求配置不同模块。

JsSaaS平台支持WEB开发，也支持命令行软件。主要区别：WEB开发支持HTTP端口访问；命令行模式用于本地业务执行。

1.2 JsSaaS能做什么

一：WebService

1：网站

2：API接口

3：WEB项目开发

4：缓存服务，结合数据库

二：命令行模式应用软件

1：数据库数据整理

2：IO文件处理

3：图片处理

4：定时任务执行器

1.3 平台目标

让后台代码开发变得简单高效

项目只部署一次，后续无缝更新

让前端开发人员成为全栈大牛

只要你懂JavaScript+SQL，您就有成为全栈工程师。包揽前后端开发。

前端开发：WEB端（浏览器）+客户端（用uniapp一次开发，发布h5+微信小程序+An

droid+IOS）、后台管理系统

后端开发：JsSaaS平台

1.4 规划实现功能

第一期规划：

1:webService【已实现】

JSSaaS就是一个动态的WebService平台，配合nginx会更好

2:静态资源访问【已实现】

静态资源如图片css等资源，建议采用nginx访问

3:请求入口地址配置完全匹配【已实现】

请求方式配置GET/POST/PUT/DELETE/OPTION

4:数据库操作【已实现】

支持MYSQL数据库支持mysql/mssql/sqlite，数据库新增删除修改支持实现，数据库事务回滚支持实现

5：JavaScript脚本解析器【已实现】

6：系统支持事件触发功能触发的事件可以嵌套

7：系统支持全局缓存功能支持过期时间

8：系统支持异步执行功能无访问值，无需等待执行完成

9：系统支持异步组执行功能有返回值，需要等待全部执行完成并返回执行结果，

才能执行下一行代码

10：系统支持计划任务实现

第二期功能规划：

1：URL支持正则【已实现】

2：Excel/PDF导出【已实现】

3：UI管理平台URL等json配置管理【已实现】

4：支持ES6大部分功能【已实现】

第三期功能规划：

1：完整支持MSSQL、oracle数据库

2：实现生产环境无缝更新，全部的配置和脚本都实现缓存刷新

3：JS业务代码加密，防止被恶意修改，保护开发项目源码

第四期功能规划：

1：业务代码版本控制一个接口支持多个版本业务，支持默认

2：后端接口拖拽配置实现配置读取的数据库，表，字段等数据

3：前端界面拖拽布局

4：后端实现业务逻辑拖拽组合，自定义模式

二：开发说明

2.1 平台模式

开发模式：执行develop文件夹代码，每次执行需要重新生成业务代码。

预生产模式：执行develop文件夹代码，第一次执行是将生成的业务代码缓存起来，可通过刷新缓存重新生成业务代码。（暂未实现）

生产模式：执行runtime代码，执行单个JS业务代码文件。

2.2 数据库模式

单数据库singledb/多数据库multidb

单数据库模式：develop目录下直接是项目目录，可以认为整个项目就是一个微服务。

多数据库模式：develop目录下是各个微服务项目，微服务项目可以包含数据库，也可以不包含数据库（将数据库归为内部项目）。

2.3 项目目录说明

单数据库项目：根目录

├─/ 根目录，只放jssaas平台相关文件

├─JsSaaS平台运行包，

Windows为jssaas.exe，Mac和Linux为jssaas，jssaaas执行包可更改

├──config.json JsSaaS平台配置文件，修改后需要重新jssaas才能生效

├──jscode 业务代码文件夹

，项目文件夹，项目文件夹可以不与jssaas不在同一目录下

├──sysfun

内置类，不参与业务代码，为更好的了解内置类，便于开发（JS提示）

一：单数据库模式：

******** 以下为开发模式的业务代码（开发用） ********

├──jscode/app.json 项目配置文件，可通过刷新功能实现更改生效

├──jscode/develop/ 业务代码文件夹

├──jscode/develop/global.js 全局JS代码，通常用于全局变量

├──jscode/develop/database.json 项目数据库配置

├──jscode/develop/bll/ bll业务代码，初始文件通过生成实现

├──jscode/develop/bll_dal / bll业务代码基类，基础函数，文件通过生成实现

├──jscode/develop/crond/ 定时任务业务代码，文件名以crond_开头

├──jscode/develop/dbschema/

数据库对应实体层，底层生成模式为class模式该目录代码有涉及到开发，函数模式下不使用该目录

├──jscode/develop/event/ 锚点事件配置，文件名以event_开头的json文件

├──jscode/develop/event/event_fun/ 锚点业务代码，文件名以event_开头

├──jscode/develop/init/ 项目初始化业务代码，文件名以init_开头

├──jscode/develop/js/ jssaas内置函数和自定义函数文件夹

├──jscode/base/js/jsfun/ jssaas平台提供内置类对应的调用方法

，可以自定义处理已知的内部错误，不抛出异常或者记录错误信息

├──jscode/develop/js/jsplugin/ jssaas平台提供内置插件对应的调用方法

├──jscode/develop/main/ 项目入口业务代码，文件名以main_开头

********** 以下为生产模式的业务代码（生产用） **********

├──jscode/app.json 项目配置文件，可通过刷新功能实现更改生效

├──jscode/runtime/ 业务代码文件夹

├──jscode/runtime/global.js 全局JS代码，通常用于全局变量

├──jscode/runtime/database.json 项目配置文件

├──jscode/runtime/async/ 异步函数实现，文件名以async_开头

├──jscode/runtime/crond/ 定时任务业务代码，文件名以crond_开头

├──jscode/runtime/event/ 锚点事件业务代码，文件名以event_开头

├──jscode/runtime/init/ 项目初始化业务代码，文件名以init_开头

├──jscode/runtime/main/ 项目入口业务代码，文件名以main_开头

二：多数据库模式：

********** 以下为开发模式的业务代码（开发用） **********

├──jscode/app.json 项目配置文件，可通过刷新功能实现更改生效

├──jscode/develop/ 业务代码文件夹

├──jscode/develop/global.json 全局JS代码配置文件，通常用于全局变量

├──jscode/develop/js/ jssaas内置函数和自定义函数文件夹

├──jscode/develop/js/jsfun/ jssaas平台提供内置类对应的调用方法

，可以自定义处理已知的内部错误，不抛出异常或者记录错误信息

├──jscode/develop/js/jsplugin/ jssaas平台提供内置插件对应的调用方法

**********微服务以index、user为例**********

******微服务index：非数据库微服务，没用的文件皆可删除******

├──jscode/develop/index/crond/

定时任务业务代码，文件名以index_crond_开头

├──jscode/develop/index/event/

锚点事件配置，文件名以index_event_开头的json文件

├──jscode/develop/index/event/event_fun/

锚点业务代码，文件名以index_event_开头

├──jscode/develop/index/init/

项目初始化业务代码，文件名以index_init_开头

├──jscode/develop/index/main 项目入口业务代码，文件名以index_main_开头

** 微服务user：数据库微服务 *********

├──jscode/develop/user/database.json 微服务配置文件

├──jscode/develop/user/api/ 对外业务入口，文件名以user_开头

├──jscode/develop/user/bll/

bll业务代码，初始文件通过生成实现，文件名以user_表名 开头

├──jscode/develop/user/bll_dal/

bll业务代码基类，初始文件通过生成实现，文件名以user_表名开头

├──jscode/develop/user/crond/

定时任务业务代码，文件名以user_crond_开头

├──jscode/develop/user/dbschema/ 项目数据库对应表结构，

├──jscode/develop/user/event/

锚点事件业务代码，文件名以user_event_开头

├──jscode/develop/user/init/

项目初始化业务代码，文件名以user_init_开头

├──jscode/develop/user/main/ 项目入口业务代码，文件名以user_main_开头

******* 以下为生产模式的业务代码（生产用） **********

├──jscode/runtime/ 业务代码文件夹

├──jscode/runtime/global.js 全局JS代码配置文件，通常用于全局变量 ***

index 微服务 ***

├──jscode/runtime/index/database.json 项目配置文件

├──jscode/runtime/index/crond

定时任务业务代码，文件名以index_crond_开头

├──jscode/runtime/index/event

锚点事件业务代码，文件名以index_event_开头 ├──jscode/runtime/index/init

项目初始化业务代码，文件名以index_init_开头

├──jscode/runtime/index/main 项目入口业务代码，文件名以index_main_开头

*** user 微服务 ***

├──jscode/runtime/user/database.json 项目配置文件

├──jscode/runtime/user/crond 定时任务业务代码，文件名以user_crond_开头

├──jscode/runtime/user/event 锚点事件业务代码，文件名以user_event_开头

├──jscode/runtime/user/init 项目初始化业务代码，文件名以user_init_开头

├──jscode/runtime/user/main 项目入口业务代码，文件名以user_main_开头

jscode文件夹，除了develop和runtime文件夹外，其他都是自定义的，不参与代码编译，除了在代码中有调用到。

2.4 解析机制

开发模式（develop目录）：

项目启动时执行业务代码初始化，每次请求都会重新执行一次（除crond/init函数外）。

生产模式（runtime目录）：

项目启动时执行业务初始化；每次请求都是从缓存中读取

2.5 文件命名规则

函数名确保全局唯一，否则在执行会报错，或者执行的业务代码不是开发者期待执行的业务。

单数据库项目：

命名以功能开头，jscode/develop/crond、event、init、main等，如crond,event,init,main这些文件夹下等文件必须以对应文件夹名开头，在具体的文件函数也是必须以该文件夹名称开头，如main_index.js，入口主函数命名规则：main_xxx(params){}

多数据库项目（多微服务项目）：命名以该微服务名称开头，如jscode/develop/user/main_index.js的函数：user_main_xxx(params){}.

函数的推荐命名：fnXxxx，或者xx_xxx 这两种模式，

fnXxxx:fn开头，后面的第一个字符必须大写

xx_xxx 函数中必须包含_，

如果不采用这两种命名，在特殊情况下，可能无法识别到该函数，在编译时，无法将对应的函数代码带入业务代码中。

2.6 入口主函数配置

1:web模式

请求入口：main函数，

单数据库：jscode/develop/main/目录下js代码：如：main_xxx

```
/**
 * 首页显示
 * @param params
 * @jssaas url /,/index.html
 * @jssaas method get
 * @jssaas timeout 20000
 */
function main_index_home(params) {
    return fnRenderTemplate("/html/index.html")
```

```
}
```

多数据库（以user为例）：jscode/develop/user/main/目录下js代码：如：user_main_xxx

```
/**
 * 首页显示
 * @param params
 * @jssaas url /,/index.html
 * @jssaas method get
 * @jssaas timeout 20000
 */
function user_main_index_home(params) {
    return fnRenderTemplate("/html/index.html")
}
```

2：命令行模式

在jscode/develop/funlist.js, "entryFunction"= "main"

对应的函数为入口主函数

2.7 定时任务主函数配置

JS代码必须是在执行文件夹下才能生效，

单数据库：jscode/develop/crond/目录下js代码：如：crond_xxx.js，函数名必须是crond_xxx开头

多数据库（user为例）：jscode/develop/user/crond/目录下js代码：如：user_crond_xxx.js，函数名必须是user_crond_xxx开头

执行业务不能涉及Request的内容，COOKIE，SESSION，请求参数等

```
/**
 * 定时任务 每5秒执行一次
 * @jssaas  crond *\/5 * * * * *
 * @jssaas timeout 2000
 */
function crond_test_aa() {
    console.log("定时任务","crond_test_aa",fnTime_timestamp());
}

/**
```

```
 * 定时任务 每分钟执行一次,
 * @jssaas  crond 0 * * * * *
 * @jssaas timeout 2000
 */
function crond_test_bb() {
    console.log("定时任务","crond_test_bb",fnTime_timestamp());
}
```

@jssaas crond 后面为触发周期，有六个位置需要配置，分别是秒 分 时 天 月 周 ，配置规则与Linux的crond配置是一样的。

如果规则值存在*/ ，则需要加入转义为*\/*，否则会被默认为JS的注释结束符号。

定时任务：每秒执行一次："* * * * * *"

定时任务：每分钟的0秒执行一次："0 * * * * *"

定时任务：每小时的0分0秒执行一次："0 0 * * * *"

定时刷新用户列表：每天5点执行一次："0 0 5 * **"

2.8 初始化主函数配置

JS代码必须是在执行文件夹下才能生效，

单数据库：jscode/develop/init/目录下js代码：如：init_xxx.js，函数名必须是init_xxx开头

多数据库（user为例）：jscode/develop/user/init/目录下js代码：如：user_init_xxx.js，函数名必须是user_init_xxx开头

执行业务不能涉及Request的内容，COOKIE，SESSION，请求参数等

```
/**
 * 初始化用户列表
 * @jssaas  timeout 100000
 */
function init_test_aa() {
    console.log("项目初始化","init_test_aa");
}
```

2.9 锚点事件函数配置

事件锚点，通过调用

fnEvent_call(“event_xx_xx”,param),所有的事件参数只有一个，可以是任意类型。

event/event_test.json 配置信息：

```
{
  "id": "event_test_a",
  "title": "项目事件触发规则",
  "sync_error_continue": true,
  "sync": [
    {
      "title": "测试同步1",
      "function": "event_sync_1",
      "timeout": 2000000
    },
    {
      "title": "测试同步2",
```

```
      "function": "event_sync_2",
      "timeout": 2000000
    }

  ],
  "async": [
    {
      "title": "测试异步",
      "function": "event_async_1",
      "timeout": 2000000
    },
    {
      "title": "测试异步",
      "function": "event_async_2",
      "timeout": 2000000
    }
  ]
}
```

```
/*
说明：事件锚点调用，事件锚点可以随时更改有一个单一业务引发的其他业务逻辑关系，通过配置事件锚点即可快速业务配置，而不用去更改已有的代码业务；使用业务配置有利于业务梳理。
★function fnEvent_call(eventname,params):
触发事件锚点。触发事件名称必须是字符串，需要event.json文件配置有该触发方法对应执行同步函数列表和异步函数列表
 */


/**
 * 首页显示
 * @param params
 * @jssaas url /,/index.html
 * @jssaas method get
 * @jssaas timeout 20000
 */
function main_index_home(params) {
    fnEvent_call("event_test_a",params); //唤醒事件
```

```
    return fnRenderTemplate("/html/index.html")
}
```

2.10 入口函数的JS注释配置@jssaas说明

JS注释配置@jssaas说明，格式：@jssaas xxx yyyy 以空格为分隔符

```
 * @jssaas url /,/index.html      如果支持多个入口请求地址，用,分隔
 * @jssaas method get      请求方法：支持get,post,option,put,delete
 * @jssaas timeout 20000  超时毫秒
 * @jssaas  crond *\/5 * * * * *
```

定时器周期规则，如果包含*/，需要转义为*\/

2.11 函数入口超时配置

超时有默认超时配置值，app.json->timeout下各自配置默认值

```
 "timeout": {
    "async_desc": "异步超时",
    "async": 3000,
    "crond_desc": "定时任务",
    "crond": 2000,
    "event_desc": "事件",
```

```
    "event": 2000,
    "init_desc": "初始化",
    "init": 2000,
    "request_desc": "请求",
    "request": 20000
  }
```

JS文件：在主函数JS注释中配置：

```
/**
 * 定时任务 每5秒执行一次
 * @jssaas  crond *\/5 * * * * *
 * @jssaas timeout 2000
 */
function crond_test_aa() {
    console.log("定时任务","crond_test_aa",fnTime_timestamp());
}
```

json文件：在入口函数统计配置属性："timeout"。

三：配置文件说明

3.1 /config.json平台配置文件

config.json是平台配置文件；修改后如果要生效需要重启jsaas平台

===>> web模式

config.json内容如下：

```
 {
  "dbtype": "singledb",
  "dbtype_desc": "数据库模式，单数据库：singledb，多数据库：multidb",
  "hostport": ":8080", //访问端口号。Web模式特有
  "path_jscode": "./jscode",
  "path_jscode_bak": "$ROOTPATH$/../site_jssaas_v2/jscode",
  "projectname": "项目名称",
  "promode": "dev", //运行模式，Web模式特有
  "promode_desc": "运行模式：开发模式dev,生产模式pro; ",
  "team": "开发团队",
  "version": "1.0.0"
}
```

===>> 命令行模式

```
{
  "dbtype": "singledb",
  "path_jscode": "./jscode",
  "path_jscode_bak": "$ROOTPATH$/../site_jssaas_v2/jscode",
  "projectname": "项目名称",
  "team": "开发团队",
  "version": "1.0.0"
}
```

所有的JSON配置文件支持$ROOTPATH$和 $JSCODEPATH$目录变量

3.2 /jscode/app.json 项目配置文件（WEB模式）

```
{
  "cros_desc": "跨域配置",
  "cros": {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Credentials": "true",
    "Access-Control-Allow-Methods": "*",
```

```
    "Access-Control-Allow-Headers": "Content-Type,X-Requested-With,*",
    "Access-Control-Expose-Headers": "*"
  },
  "log": {
    "writesql_desc": "是否记录执行等SQL语句",
    "writesql": false,
    "writesqlfile": "sqllog",
    "logtype": "console",
    "logtype_desc": "日志显示类型,console,file",
    "loglevel": "debug",
    "loglevel_desc": "记录等级,error,log,debug,",
    "enable_consolelog": true
  },
  "static_desc": "静态本地目录",
  "static": [
    {
      "alias": "/res/",
      "path": "$JSCODEPATH$/static"
    },
```

```
    {
      "alias": "/admin/",
      "path": "$JSCODEPATH$/CCadmin"
    }
  ],
  "setting": {
    "debuginfo_desc": "如果返回是JSON结果，数据加入debugInfo信息",
    "debuginfo": false,
    "template_desc": "HTML解析模板，左边有右边标识， 默认为{{ 和 }}",
    "templatedelims_left": "",
    "templatedelims_right": "",
    "urlrouter_desc": "是否启用URL路由",
    "urlrouter": true
  },
  "session": {
    "cookiename_desc": "Session在客户端等COOKIE名称",
    "cookiename": "goSessionID",
    "timeoutsecond_desc": "session失效时间，单位秒",
    "timeoutsecond": 36000,
```

```
  "gchour_desc": "session资源回收周期：单位小时",
  "gchour": 10
},
"globalcache": {
  "globalcache": "应用内全局缓存",
  "gchour_": "session资源回收周期：单位小时",
  "gchour": 10
},
"cache_redis": {
  "cache_redis": "redis配置",
  "addr": "127.0.0.1:6379",
  "password": "",
  "db": 0
},
"timeout_desc": "超时配置，单位：毫秒",
"timeout": {
  "async_desc": "异步超时",
  "async": 3000,
  "crond_desc": "定时任务",
```

```
    "crond": 2000,
    "event_desc": "事件",
    "event": 2000,
    "init_desc": "初始化",
    "init": 2000,
    "request_desc": "请求",
    "request": 20000
  }
}
```

3.3 /jscode/app.json项目配置文件（命令行模式）

```
{
  "log": {
    "writesql_desc": "是否记录执行等SQL语句",
    "writesql": false,
    "writesqlfile": "sqllog",
    "logtype": "console",
    "logtype_desc": "日志显示类型,console,file",
    "loglevel": "debug",
```

```
  "loglevel_desc": "记录等级,error,log,debug,",
  "enable_consolelog": true
},
"setting": {
  "debuginfo_desc": "如果返回是JSON结果，数据加入debugInfo信息",
  "debuginfo": false,
  "template_desc": "HTML解析模板，左边有右边标识， 默认为{{ 和 }}",
  "templatedelims_left": "",
  "templatedelims_right": ""
},
"globalcache": {
  "globalcache": "应用内全局缓存",
  "gchour_": "session资源回收周期：单位小时",
  "gchour": 10
},
"cache_redis": {
  "cache_redis": "redis配置",
  "addr": "127.0.0.1:6379",
  "password": "",
```

```
    "db": 0
  },
  "timeout_desc": "超时配置，单位：毫秒",
  "timeout": {
    "async_desc": "异步超时",
    "async": 3000,
    "crond_desc": "定时任务",
    "crond": 2000,
    "event_desc": "事件",
    "event": 2000,
    "init_desc": "初始化",
    "init": 2000
  }
}
```

3.4 /jscode/develop/funlist.json（命令行模式）

默认funlist.json配置，

当id=main为默认执行业务，当id=main不存在时，则不执行业务，等待窗口输入

```
[
  {
    "id": "main",
    "title": "默认入口函数",
    "function": "{PROJECT}main_home",
    "params": "()"
  },
  {
    "id": "sys_getevndata",
    "title": "环境信息",
    "function": "{JSAPROJECT}main_sys_getevndata",
    "params": "()"
  },
  {
    "id": "sys_createtable",
    "title": "生成数据库映射代码",
    "function": "{JSAPROJECT}main_sys_createdbcode",
    "params": "(project,table,codetype)"
  },
```

```
{
  "id": "sys_projectinit",
  "title": "初始化项目",
  "function": "{JSAPROJECT}main_sys_projectinit",
  "params": "(project)"
},
{
  "id": "sys_getinit",
  "title": "取得初始化业务列表",
  "function": "{JSAPROJECT}main_sys_getinit",
  "params": "()"
},
{
  "id": "sys_getevent",
  "title": "取得事件业务列表",
  "function": "{JSAPROJECT}main_sys_getevent",
  "params": "()"
},
{
```

```
    "id": "sys_getcrond",
    "title": "取得定时任务列表",
    "function": "{JSAPROJECT}main_sys_getcrond",
    "params": "()"
  },
  {
    "id": "sys_getfunction",
    "title": "取得执行的函数代码",
    "function": "{JSAPROJECT}main_sys_getfunction",
    "params": "(functionname)"
  }
]
```

3.5 /jscode/develop/global.js 全局文件

全局变量，依据项目需要配置，每个执行的JS代码自动包含global.js脚本；

```
var DBkey_News = "news";  //多数据库时存在。单数据库不需要数据库Key。


//全局变量，在最终的执行JS代码前面会导入改文件的代码，推荐在该文件定义全局变量，不建议在该文件写入函数
```

```
/**
 * 全局变量
 * @type {string}
 */
var test="Test";
```

3.6 /jscode/develop/{PROJECT/}database.json

数据库配置文件

```
{
  "_desc": "数据库类型，支持sqlite/mysql/mssql/oracle",
  "default_db": "sqlite",// db_list配置的子项属性名称
  "db_list": {
    "com": {
      "db_type": "mysql",
      "mysql_loginname": "",
      "mysql_loginpwd": "",
      "mysql_server": "",
```

```
    "mysql_port": 3306,
    "mysql_dbname": "",
    "sqlite_driverName": "",
    "sqlite_dataSourceName": "$JSCODEPATH$/database/database.db",
    "mssql_server": "",
    "mssql_database": "",
    "mssql_userid": "",
    "mssql_password": "",
    "mssql_port": 3306,
    "mssql_encrypt": "",
    "oracle_user": "",
    "oracle_pwd": "",
    "oracle_server": "",
    "oracle_port": 3306,
    "oracle_database": ""
  },
  "mysql_win": {
    "db_type": "mysql",
    "mysql_loginname": "root",
```

```
        "mysql_loginpwd": "root",
        "mysql_server": "localhost",
        "mysql_port": 3306,
        "mysql_dbname": "erp"
      },
      "sqlite": {
        "db_type": "sqlite",
        "sqlite_driverName": "sqlite3",
        "sqlite_dataSourceName": "$JSCODEPATH$/database/database.db"
      }
    },
  "tablenameprefix": "t",
  "tablenameprefix_desc": "表名前缀"
}
```

四：项目开发

4.1 新建项目

下载执行包

从网盘下载jssaas平台 https://www.123pan.com/s/fbJZVv-zEkWA.html

下载对应运行平台的jssaas运行包。

在命令行模式下执行jssaas包，

需要按提示配置（只有jssaas包会先执行配置）

a：配置运行类型：1：webservice（网站接口）、2：software（命令行软件）

1.1：配置运行模式：1:开发模式、2:生产模式，回车默认值为1

1.2：配置端口地址，回车默认值为8888

b：配置运行类型：1：singledb（单数据库模式）、2：multidb（多数据库模式）

4.2 Hello World

WEB模式

新建项目后，默认启动jssaas平台，将进入端口为8888的web网站，网站地址：http://localhost:8888/ 进入helloworld首页

4.3 JS脚本开发说明

javaScript采用 严格模式（strict mode）即在严格的条件下运行。

不允许使用未声明的变量： 全局变量使用var定义，局部变量使用let定义。

系统函数的实现在/sysfun目录下的js文件，这些文件不参与具体的业务代码，只是提供给开发者查看。

jscode/develop/js/jsfunc/目录下的js文件是内置函数的引用。

五：内置函数说明

5.1 内置函数介绍

内置函数作为JS ES6外的补充，将常用的函数在底层实现，能提高执行效率

内置函数对应为类的静态函数，则映射为函数模式

内置函数对应为类的非静态函数（类需要实例化），则映射为类模式

命名规则

1：无需实例化，命名规则为：fnXxx_xxx()，Xxx为归类，xxx为具体函数名，如：fnIO_write()，fnCache_get()

2：类，需要实例化，命名规则为：类型:clsXxx，如clsExcel，clsPdf，函数名：exlClose()、pdfAddPage()

一定要确保函数名全局唯一。

5.2 /sysfunc目录

系统函数的实现在/sysfunc目录下的js文件，对应的调用在/jscode/develop/js/jsfun/目录下的js文件

/sysfun目录下的js文件不参与具体的javascript执行，只是为了开发者更直观的了解内置函数，提供空函数。底层实现的函数如果有返回值，必须是{“code”:0,”data”:xxx,”msg”:””} code小于0时，msg有值。

内置函数列表：

/sysfun/plugin/

插件文件夹，支持一些常用的，通用的插件，如阿里云、微信、支付宝、华为云等

jsa.js Jssaas平台函数

jsaAsync.js 异步执行

jsaBase64.js base64相关处理，

jsaCache.js 项目自身全局缓存

jsaCookie.js Cookie缓存读写操作

jsaCrypto.js 加密操作

jsaDb_multidb.js 多数据库操作

jsaDb_singledb.js 单数据库操作

jsaEvent.js 事件锚点触发函数

jsaExcel.js Excel 操作

jsaHttp.js 执行http请求，支持get/post/put/delete 及各个选项值。

jsaImage.js 图片处理

jsaImageUtil.js 图片静态函数

jsaIO.js IO操作

jsaJson.js JSON操作

jsaLock.js 锁操作

jsaPdf.js PDF 文件操作

jsaQueue.js 队列操作

jsaQueueSelftime.js 自带生命周期的队列操作

jsaRedis.js Reids操作

jsaRequest.js http请求信息读取

jsaResponse.js http响应操作

jsaSession.js session操作

jsaString.js 字符串处理

jsaTemplate.js 模板操作

jsaTime.js 时间函数

jsaZip.js zip操作

5.3 js/jsfun/clsExcel Excel操作

==========》》》 JS代码 《《《 ==========

```
    //新建Excel文件
```

```
let exl = new clsExcel();

//设置单元格值

exl.exlSetCellValue("A1", "AAAA")

//新建工作簿

let index = exl.exlNewSheet("新版本")

//设置活动工作簿

exl.exlSetActiveSheetIndex(index)

exl.exlSetCellValue("A1", "AAAA")

exl.exlSetCellValue("B1", "AAAABB")

exl.exlSetCellValue("A2", "BBBB")


//取得活动工作簿的索引

let index2 = exl.exlGetActiveSheetIndex()

fnResponse_write("当前工作簿索引: " + index2)


//取得工作簿列表

let sheets = exl.exlGetSheetList()

console.log(sheets)

fnResponse_write(sheets)
```

```
    //取得工作簿名称
    let sheetName = exl.exlGetSheetName(index2)
    //合并单元格
    exl.exlMergeCell(sheetName, "A1", "B2")
    //变更工作簿名称
    exl.exlSetSheetName("新版本", "新版本B")
    //设置活动工作簿单元格样式
    let aa = {"color": "00FFFF", "size": 26, "bold": true, "italic":
true, "horizontal": "right","vertical": "top", "bgColor": "112233"};
    exl.exlSetCellStyle("A1", "A2", aa);
    //设置工作簿单元格样式
    let bb = {"color": "FF0000", "size": 36, "bold": false, "italic":
true, "horizontal": "left", "vertical": "bottom", "bgColor": "C0C0C0"};
    exl.exlSetSheetCellStyle("Sheet1", "A1", "A2", bb);


    //批量设置工作簿数据
    let data2 = [{"title": "AAAAA", "price": 123.12}, {"title":
"BBBBBB", "price": 1299}]
    let columns = [{"cn": "title", "title": "商品名称"}, {"cn": "price",
```

```
"title": "价格"}]

    exl.exlSetData(data2, columns)



    //保存EXCEL到本地EXCEL文件

    let file = fnJsa_rootPath() + "/aa.xlsx";

    exl.exlSaveAs("aa.xlsx")

    if (fnIO_exist(file)) {

        fnResponse_write("输出EXCEL：aa.xlsx成功")

    }



    //实例化EXCEL,打开EXCEL 文件

    let exl2=new clsExcel(file);

    // //取得EXCEL活动工作簿数据

    let data3=exl2.exlGetData();

    console.log("readData:",data3)
```

5.4 js/jsfun/clsPdf PDF操作

```
==========》》》    JS代码    《《《 ==========

    //实例化pdf
```

```
let pdf=new clsPdf();

//添加字体文件

pdf.pdfAddFontFile("font/simfang.ttf","宋体")

pdf.pdfAddFontFile("font/STHUPO.TTF","ST")



//添加图片

pdf.pdfAddImage(fnJsa_rootPath()+"/images/a.jpg",0,0,100,100)

//换行，高度200，确保文字能看得到

pdf.pdfLn(100)

//输出文本，英文

pdf.pdfWrite(16,"test::ASAAA")

//输出文本，默认不支持中文，乱码，需要制定字体

pdf.pdfWrite(16,"中文能否正常现实文字信息")

pdf.pdfText(110,10,"TEXT")



pdf.pdfSetFont("宋体",16)

let linkID= pdf.pdfWriteLink(25,"链接，点我进入第三页");
```

```
//新增一页

pdf.pdfAddPage()



//设置使用字体及大小

pdf.pdfSetFont("宋体",16)

pdf.pdfWrite(26,"中文能否正常现实文字信息")

pdf.pdfLn(10)

pdf.pdfSetTextColor("#FF00FF")

pdf.pdfSetMargin(50,10,50)

pdf.pdfWrite(26,"中文能否正常现实文字信息")



pdf.pdfSetFont("ST",16)

pdf.pdfWrite(56,"中文能否正常现实文字信息2")

//取得当前字体大小

fnResponse_write("fontsize:"+ pdf.pdfGetFontSize())

pdf.pdfSetMargin(0,0,0)

//换行

pdf.pdfLn(30)

let pagecount= pdf.pdfPageCount()
```

```
    pdf.pdfWrite(26,"总页数: "+pagecount)


    //新增一页
    pdf.pdfAddPage()
     pdf.pdfSetFont("ST",16)
    pdf.pdfWrite(16,"中文能否正常现实文字信息2")
    pdf.pdfSetLinkPageLocation(linkID,20,-1)
    //换行
    pdf.pdfLn(20)
    let html="  我是中国人,    我爱中国";
    pdf.pdfWriteHtml(20,html)
    pdf.pdfLn(20)
    pdf.pdfAddFontFile("font/STKAITI.TTF","ST","U")
    pdf.pdfAddFontFile("font/STLITI.TTF","ST","B")
    pdf.pdfAddFontFile("font/STLITI.TTF","ST","BI")
    pdf.pdfAddFontFile("font/STXINGKA.TTF","ST","I")
    let html2="You can now easily print text mixing different styles:
<b>bold</b>, " +
        "<i>italic</i>, <u>underlined</u>, or <b><i><u>all at
```

```
once</u></i></b>!<br><br>" +
        "<center>You can also center text.</center>" +
        "<right>Or align it to the right.</right>" +
        "You can also insert links on text, such as " +
        "<a href='http://www.fpdf.org'>www.fpdf.org</a>, or on an image:
click on the logo.";
    pdf.pdfWriteHtml(20,html2)

    let file=fnJsa_rootPath()+"/aa.pdf"
    pdf.pdfSave(file)
```

5.5 js/jsfun/clsImg 图片相关操作

```
    //图片工具
    //载入图片资源
    let imgFile = fnJsa_jsCodePath() + "/demoresource/images/a.png"
    let image = fnImage_imageFromFile(imgFile)
    //图片尺寸
    console.log("图片尺寸:", fnImage_imageSize(imgFile))
```

```
//初始化实例，从图片载入

let imgFile0 = fnJsa_jsCodePath() + "/demoresource/images/a.png"

let img0 = new clsImage(imgFile0)

//保存PNG图片

img0.imgSavePNG(fnJsa_jsCodePath() + "/demoresource/images/a_2.png")



//初始化实例，新建画板

let img = new clsImage(700, 1200, "#FF00000")

//重置画板底色

img.imgClear("#cccbcb")

//画图片到画板

img.imgDrawImage(image, 20, 20)

//画实心矩形

img.imgDrawRectangle(50, 50, 100, 50, "#C0C0C0")

//画空心矩形

img.imgDrawRectangle(200, 50, 100, 50, "#C0C0C0", 2)

//画圆角实心矩形

img.imgDrawRoundedRectangle(350, 50, 100, 50, 5, "#C0C0C0")
```

```
//画圆角空心矩形

img.imgDrawRoundedRectangle(500, 50, 100, 50, 5, "#C0C0C0", 2)



//画实心圆

img.imgDrawCircle(100, 180, 50, "#C0C0C0")

//画空心圆

img.imgDrawCircle(220, 180, 50, "#C0C0C0", 2)

//画实心椭圆

img.imgDrawEllipse(370, 180, 80, 50, "#c0c0c0")

//画空心椭圆

img.imgDrawEllipse(550, 180, 80, 50, "#c0c0c0", 3)

//画直线

img.imgDrawLine(50, 50, 500, 200, "#00FF00", 3)

//画实心多边形

img.imgDrawRegularPolygon(3, 100, 300, 50, 0, "#C0C0C0", 0)

img.imgDrawRegularPolygon(6, 220, 300, 50, 0, "#C0C0C0", 0)

//画空心多边形

img.imgDrawRegularPolygon(3, 350, 300, 50, 0, "#C0C0C0", 4)

img.imgDrawRegularPolygon(6, 480, 300, 50, 0, "#C0C0C0", 4)
```

```
    //写文字

img.imgDrawString("ABC，我是中国人,ABC，我是中国人,ABC，我是中国人,ABC，我是中国人,ABC，我是中国人,ABC，我是中国人,"
        , 50, 400, 0, 0, "#FFFF00", fnJsa_jsCodePath() + "/demoresource/font/simfang.ttf", 30)
    //自动换行
    img.imgDrawStringWrapped("ABC，我是中国人 ,ABC，我是中 国人,ABC，我是中国 人,,"
        , 50, 500, 0, 0, 500, 1.5, "#FFFF00", 25, fnJsa_jsCodePath() + "/demoresource/font/simfang.ttf")

    //旋转、缩放、倾斜等需要创建两个画板。
    //创建临时画板，得到image资源
    let tmpImgRotate0 = new clsImage(200, 200)
    let tmpImgRotate1 = new clsImage(200, 200)
    // tmpImg.imgDrawRectangle(0,0,80,30,"#FF00FF")
    tmpImgRotate0.imgDrawString("123456789", 30, 30, 0, 0, "#0033ff",
```

```
fnJsa_jsCodePath() + "/demoresource/font/simfang.ttf", 30)
    let tmpImageRotate = tmpImgRotate0.imgImage()

    //旋转
    tmpImgRotate1.imgRotate(0, tmpImageRotate, 100, 100)
    tmpImgRotate1.imgRotate(0.1, tmpImageRotate, 100, 100)
    tmpImgRotate1.imgRotate(0.2, tmpImageRotate, 100, 100)
    tmpImgRotate1.imgRotate(0.3, tmpImageRotate, 100, 100)
    tmpImgRotate1.imgRotate(0.4, tmpImageRotate, 100, 100)
    tmpImgRotate1.imgRotate(0.5, tmpImageRotate, 100, 100)
    tmpImgRotate1.imgRotate(0.6, tmpImageRotate, 100, 100)
    tmpImgRotate1.imgRotate(0.7, tmpImageRotate, 100, 100)
    tmpImgRotate1.imgRotate(0.8, tmpImageRotate, 100, 100)
    tmpImgRotate1.imgRotate(0.9, tmpImageRotate, 100, 100)
    tmpImgRotate1.imgRotate(1, tmpImageRotate, 100, 100)
    //将旋转的结果画到画板
    img.imgDrawImage(tmpImgRotate1.imgImage(), 200, 600)

    //缩放
```

```
    let tmpImgScale0 = new clsImage(400, 50)
    let tmpImgScale1 = new clsImage(400, 200)
    // tmpImg.imgDrawRectangle(0,0,80,30,"#FF00FF")
    tmpImgScale0.imgDrawString("123456789", 30, 30, 0, 0, "#FFFF00",
fnJsa_jsCodePath() + "/demoresource/font/simfang.ttf", 30)
    let _scale = tmpImgScale0.imgImage()
    //执行缩放
    tmpImgScale1.imgScale(0.8, 0.8, _scale, 0, 0)
    tmpImgScale1.imgScale(2, 2, _scale, 50, 0)
    img.imgDrawImage(tmpImgScale1.imgImage(), 20, 800)


    //倾斜
    // img.imgShear(20,40,tmpImage,300,800)
    let tmpImgShear0 = new clsImage(50, 20)
    let tmpImgShear1 = new clsImage(800, 600, "#000000")
    // tmpImg.imgDrawRectangle(0,0,80,30,"#FF00FF")
    tmpImgShear0.imgDrawRectangle(0, 0, 50, 20, "#FFFF00")
    let _shear = tmpImgShear0.imgImage()
    //执行倾斜,当倾斜倍数都为1时, 会为空
```

```
tmpImgShear1.imgShear(0, 0, _shear, 0, 0)

tmpImgShear1.imgShear(0, 1, _shear, 100, 0)

tmpImgShear1.imgShear(1, 0, _shear, 200, 0)

tmpImgShear1.imgShear(0.8, 0.8, _shear, 300, 0)

img.imgDrawImage(tmpImgShear1.imgImage(), 20, 900)


//保存成PNG图片

img.imgSavePNG(fnJsa_jsCodePath() + "/demoresource/images/b.png")
```

5.6 js/jsfun/fnAsync.js □异步执行

fnAsync.js 异步执行

```
 * 异步执行，无返回值;
```

函数名称必须是字符串，不能为变量，如: fnAsync_run("fnLog", {"title": "AAAAA"})

异步组执行，有返回值，最后一个变量为函数的执行参数，且个数需要要执行的函数个数一致。

```
 *
```

同步执行，但同时执行多个异步进程，最终返回json数据；需要确保执行函数存在；执行时间为最长函数执行时间；返回时JSON属性为tag的值

```
    let data = [{"title": "AAAAA"}, {"title": "BBBBB"}]
    fnAsync_runGroup("fnLogGroup1", "fnLogGroup2", data)
```

```
/**
 * 异步
 * @param params
 * @jssaas url /funAsync.html
 * @jssaas method get
 * @jssaas timeout 200000
 */
function main_fun_async(params) {
    let t1 = fnTime_timestampMilli();
    fnAsync_run("fnLog", {"title": "AAAAA"})
    let t2 = fnTime_timestampMilli();
    return fnRenderTemplate("/html/async.html", {"msg":
```

```
"已经执行异步，异步10次，时间：" + (t2 - t1)})
}



/**
 * 异步日志
 * @param params
 * @jssaas url /funAsync_log.html
 * @jssaas method get
 * @jssaas timeout 200000
 */
function main_fun_async_log(params) {
    let file = fnJsa_rootPath() + "/aa.txt";
    let exist = fnIO_exist(file);
    if (exist) {
        let content = fnIO_getFileContent(file)
        return fnRenderText(content);
    }
    return fnRenderText("日志还未写入");
```

```
}



/**
 * 异步
 * @param params
 * @jssaas url /funAsyncGroup.html
 * @jssaas method get
 * @jssaas timeout 200000
 */
function main_fun_asyncGroup(params) {
    let t1 = fnTime_timestampMilli();
    let data = [{"title": "AAAAA"}, {"title": "BBBBB"}]
    fnAsync_runGroup("fnLogGroup1", "fnLogGroup2", data)
    let t2 = fnTime_timestampMilli();
    return fnRenderTemplate("/html/async.html", {"msg":
"已经执行异步组, 时间: " + (t2 - t1)})
}
```

```
function fnLog(params) {
    let file = fnJsa_rootPath() + "/aa.txt";
    for (let i = 0; i < 10; i++) {
        params["time"] = fnTime_timestamp();
        let content = fnIO_getFileContent(file)
        content = JSON.stringify(params) + "\n<br>" + content
        fnIO_saveFile(file, content, true)
        fnTime_sleep(2000);
    }

}

function fnLogGroup1(params) {
    let file = fnJsa_rootPath() + "/aaa.txt";

    params["time"] = fnTime_timestamp();
    let content = ""
    if(fnIO_exist(file)){
```

```
        content=fnIO_getFileContent(file)
    }
    content = JSON.stringify(params) + "\n<br>" + content
    fnIO_saveFile(file, content, true)
    fnTime_sleep(2000);

    return "fnLogGroup1"
}

function fnLogGroup2(params) {
    let file = fnJsa_rootPath() + "/aaa.txt";

    params["time"] = fnTime_timestamp();
    let content = ""
    if(fnIO_exist(file)){
        content=fnIO_getFileContent(file)
    }
    content = JSON.stringify(params) + "\n<br>" + content
    fnIO_saveFile(file, content, true)
```

```
    fnTime_sleep(5000);


    return "fnLogGroup2"
}
```

5.7 js/jsfun/fnBase64.js □base64相关操作

```
    let t1 = fnTime_timestampMilli();
    //二维码
    let base64Img = fnBase64_imageQrcode("TEST", 400);
    //验证码，可以将验证码写入session中，或全局缓存。
    let validcode = fnBase64_imageValidCode("123465", 200, 30, 30, true,
true, fnJsa_rootPath() + "/font/simfang.ttf")
    let validcode2 = fnBase64_imageValidCode("中国人水电费", 200, 30,
30, true, true)
    let validcode3 = fnBase64_imageValidCode("中国人水电费", 200, 30,
30, true, true, fnJsa_rootPath() + "/font/simfang.ttf")


    //保存base64图片
    fnBase64_Save(base64Img, fnJsa_rootPath() + "/aa.png", true)
```

```
    //保存base64数据
    let content =
"iVBORw0KGgoAAAANSUhEUgAAAZAAAAGQAQMAAAC6caSPAAAABlBMVEX///8AAABVwtN+AAA
BZklEQVR42uzaTW6DMBCG4UFdZMkROApHI0fzUXoEllkgpvIf2G5QqNQkdvR+m1K3T1aONR5
GCCGEEEIIIYfRJPKls4z26VumZP0KgbyY3OLTqCLSq67ut2mN6z0E0gYZ7Aa/ebJlWi1XXSC
Q9og9xiGQjyEXW5BAIK2RvSBxpHNLZ2sYCOQZ5NeNb/U7+S+XRAjk7SRL0rs4GwikDuKP8c5
VF6rxGB+278PhyQ+B1EXCTr+oEV08d0vdfvsTCKR+4uropPowXbE0QyCvJj5295pQM9ur3mB
/9AdnMgRSIUnfM7u/G3s+BzkeXRIhkNpIceNz/7sRkXvNYQikQlJ04TRu/qvEzT8/mFKDQKo
g+TiQbF24znXkTkwQQSDPIOl75PzGF0pnCKQxIkVLLXwKBNIgGTXrwi13m8MQSIUkn4E3snf
hjDycUoNAKiHlOJBI1rs4M0EEgfwzIYQQQggh5GPzEwAA//9+9kekl/2/6QAAAABJRU5ErkJ
ggg=="
    fnBase64_Save(content, fnJsa_rootPath() + "/bb.png")

    //处理base64内容
    let base64content = "5oiR5piv5Lit5Zu95Lq6dGVzdA==" //test 的base64值
    let result = fnUtil_base64ToString(base64content);
    console.log(result) //result:test

    let byteCotent = fnUtil_base64ToBytes( base64content)
```

```
    let txt = fnUtil_bytesToString(byteCotent)
    console.log("aaa:",  byteCotent, txt );

    let t2 = fnTime_timestampMilli();
    return fnRenderTemplate("/html/base64.html", {"img":
base64Img,"validcode":validcode,"validcode2":validcode2,"validcode3":val
idcode3,"t":t2-t1})
```

前端代码：

```
<div><img src="{{.img|url}}"  /></div>
<div><img src="{{.validcode|url}}"  /></div>
```

5.8 js/jsfun/fnCache.js Cache缓存

```
//应用级全部缓存，可以存储任何数据，但不建议存储大量数据（会占用大量的内
存），比Redis快（无需请求）
    let key = "cache_key"
```

```
//删除缓存

fnCache_remove(key)

//读取缓存

let value = fnCache_get(key)

console.log("缓存不存在: ", value)

//设置缓存

//字符串

fnCache_set(key, fnTime_now("Ymd"))

console.log("字符串: ", fnCache_get(key), typeof fnCache_get(key))

//数字

fnCache_set(key, fnTime_timestamp())

console.log("数字: ", fnCache_get(key), typeof fnCache_get(key))

//bool

fnCache_set(key, true)

console.log("bool: ", fnCache_get(key), typeof fnCache_get(key))

//数组

fnCache_set(key, ["1", "2", "3"])

console.log("数组字符串", fnCache_get(key), typeof fnCache_get(key))

//数组
```

```
    fnCache_set(key, [1, 2, 3])
    console.log("数组数字: ", fnCache_get(key), typeof fnCache_get(key))
    //对象,设置超时10秒
    let obj = {"time": fnTime_timestamp(), "title": "张三"}
    fnCache_set(key, obj, 10)
    console.log("对象: ", fnCache_get(key), typeof fnCache_get(key))
    console.log("缓存信息: ", fnCache_getInfo(key))

    //fnTime_sleep(1100)
    console.log("对象: ", fnCache_get(key), typeof fnCache_get(key))

    //设置超时时间戳
    let timeout = fnTime_timestamp() + 10
    fnCache_set(key, obj, timeout)
    console.log("缓存信息: ", fnCache_getInfo(key))
    //fnTime_sleep(1100)
    console.log("对象2:", fnCache_get(key), "timeout:", timeout, "now:",
fnTime_timestamp())
```

```
    //缓存列表
    fnCache_listRemove(key)
    console.log("列表", fnCache_listGet(key))
    //在列表最后一个元素，新增一个元素
    fnCache_listPush(key, 1)
    console.log("列表", fnCache_listGet(key), typeof
fnCache_listGet(key))
    fnCache_listPush(key, 3)
    fnCache_listPush(key, 5)
    console.log("列表", fnCache_listGet(key))
    //插入元素
    fnCache_listInsert(key, 4, 1)
    console.log("列表插入：", fnCache_listGet(key))


    //取得列表内容
    let content = fnCache_listGet(key)
    let listinfo = fnCache_listGetInfo(key)
    console.log("列表", content, listinfo)
    //弹出最后一个元素
```

```
    console.log("列表 pop:",
fnCache_listPop(key),"info:",fnCache_listGetInfo(key))
    //删除一个元素
    console.log("列表 delete:",
fnCache_listDelete(key,2),"info:",fnCache_listGetInfo(key))
    //弹出第一个元素
    console.log("列表 shift:", fnCache_listShift(key))


    console.log("列表", fnCache_listGet(key))
```

5.9 js/jsfun/fnCookie.js COOKIE操作

```
    //COOKIE, 设置后, 只能在下次请求生效, 当前请求不会改变值
    let key = "cookie_key"
    console.log("COOKIE_0:", fnCookie_get(key))
    //设置cookie, 5秒后失效
    fnCookie_set(key,fnTime_timestamp(),5)
    console.log("COOKIE_1:", fnCookie_get(key))


    //设置cookie, 50秒后失效
```

```
fnCookie_set(key,fnTime_timestamp(),fnTime_timestamp()+50)

console.log("COOKIE:", fnCookie_get(key))



//取得全部的cookie

console.log("all_COOKIE:", fnCookie_getAll())

//删除cookie

fnCookie_remove(key)
```

5.10 js/jsfun/fnCrypto.js 加密操作类

```
let key = "crypto_key"



//MD5 加密

console.log("MD5(123456):", fnCrypto_md5("123456"));

let salt = "12345678"

let aes = fnCrypto_aesEncrypt("123456", salt);

console.log("aes(123456):", aes);

console.log("aes(123456):", fnCrypto_aesDecrypt(aes, salt));
```

```
    let publicKey = `-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDcEUAg8j55PvsQqEEcFAaGyrWI
L/CroWdpoE6A8JK/E3pXZbPwJ8s6rfbPQLpkBKXVOP2/uzOHDL4ty+NFeWIbxlY2
lab80NfuRpIOEUnk4dB76BTwbwu4OtkflQ9kBr79c8wSussAR/EkOKvvBsZULeMF
27E+J+lG6gRaoBSALQIDAQAB
-----END PUBLIC KEY-----`;
    let privateKey = `-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDcEUAg8j55PvsQqEEcFAaGyrWIL/CroWdpoE6A8JK/E3pXZbPw
J8s6rfbPQLpkBKXVOP2/uzOHDL4ty+NFeWIbxlY2lab80NfuRpIOEUnk4dB76BTw
bwu4OtkflQ9kBr79c8wSussAR/EkOKvvBsZULeMF27E+J+lG6gRaoBSALQIDAQAB
AoGBAJVUY1a36qqtkQIpmhzqfc9DiFFEbqSYHqsvK3YVSQ69sdvSWHqTFjXYTE0w
mAo8mScJyC4tYna2r+o1mx/OkUITV6LxVrB/IcW44vTirL4uG0ubqOI6mCVLt9dG
V1n9lifo3F5JuvlfgTVHwb6YG8U5+dTqp9810bPWcCGzBs9FAkEA9PuiJqFXxmgz
KU/Nu4TAUjoRxmOZ+JtH36pPqL9FEH0wkH8pHMaPj2P+KqB8l4jYIhHCaSW7L0+R
0uRcvlGocwJBAOX2xtUfXa74CpXoGWBzXa8hejoqsrzRF5i4Po6wv6gcYra7Rsfm
i8A+mgPlUE0yCSQTQ1dM5fT5av23Nz1FLN8CQQCdo4oMp4dusUAylhqBToPepWUo
rpC2K11NbC8EG8faO2RTpq+Sx6Y7E5HKZiil7bV9/sWFmXTglaf092NCSBNHAkBh
w/ZoKPuVJVKqxI/1V1ae2+awz/JJH8OeY8YEt9PjTp4Q/bFrzvHe+z5TJaxn+0qz
kZTxinnBGkCIONo9LCmVAkEAiHvTCKEtUTplzmZcd6guZ538e2czBHhtKJFtcHT4
```

```
ydDZMXaTzbO65q3N5T6Vwl0gorKzRYo2OE2DHosoWhBd8Q==

-----END RSA PRIVATE KEY-----`


    let rsa = fnCrypto_rsaEncrypt("123456", publicKey)

    console.log("RSA_en:", rsa)

    console.log("RSA_de:", fnCrypto_rsaDecrypt(rsa, privateKey))


    let desCbc = fnCrypto_desCbcEncrypt("123456", salt)

    console.log("desCbc_en:", desCbc)

    console.log("desCbc_de:", fnCrypto_desCbcDecrypt(desCbc, salt))


    let desEcb = fnCrypto_desEcbEncrypt("123456", salt)

    console.log("desEcb_en:", desEcb)

    console.log("desEcb_de:", fnCrypto_desEcbDecrypt(desEcb, salt))
```

5.11 js/jsfun/fnDb_multidb.js

多数据库模式数据库操作类

```
    //查询
```

```
    let list = IndexBllDoctype.indexDoctypeQuery();
    console.log("查询列表:", list);

    let list2 = IndexBllDoctype.indexDoctypeQuery("pkid,ctitle",
{"cstatus": 1, "ctitle|like": "%开%"}, "order by pkid desc")
    console.log("查询列表2:", list2);
    let list3 = IndexBllDoctype.indexDoctypeQuery("cstatus",
{"ctitle|like": "%开%"}, "group by cstatus")
    console.log("查询列表3:", list3);
    console.log("查询行数据:", IndexBllDoctype.indexDoctypeInfo(14))
    let row = IndexBllDoctype.indexDoctypeQueryRow("pkid,ctitle",
{"ctitle|like": "%开%"}, "order by pkid desc")
    let _info = new IndexDoctype(row);
    console.log("info==>", _info);

    let tInfo = {"ctitle": "AAAA", "pkid": 0};
    console.log("===>", new IndexDoctype(tInfo))
    tInfo["ctitle"] = undefined;
    console.log("===>", new IndexDoctype(tInfo))
```

```
    console.log("查询行:", row);
    console.log("总条数:", IndexBllDoctype.indexDoctypeCount({"cstatus":
1}))


    //分页
    //无统计数据
    let page= IndexBllDoctype.indexDoctypePage("*",{"cstatus":1},"order
by pkid desc")
    console.log("分页: ", page)
    //有统计数据
    let page2=
IndexBllDoctype.indexDoctypePageCount("pkid,ctitle",{"cstatus":1},"order
by pkid desc",3,1)
    console.log("分页2: ", page2)


    //新增数据
    let newID=
IndexBllDoctype.indexDoctypeInsert({"ctitle":"AAAAAAAAA"})
    console.log("新增数据:", newID);
```

```
    console.log("新增数据1:",IndexBllDoctype.indexDoctypeInfo(newID))
    //更新数据
    IndexBllDoctype.indexDoctypeUpdateByID(newID,{"ctitle":null})
    console.log("新增数据2:",IndexBllDoctype.indexDoctypeInfo(newID))

    //删除数据
    console.log("删除条数:",
IndexBllDoctype.indexDoctypeDelete({"cstatus": -2}));
    console.log("删除指定ID的条数: ",
IndexBllDoctype.indexDoctypeDeleteByID(200))
```

5.12 js/jsfun/fnDb_singledb.js

单数据库模式数据库操作类

```
//查询
    let list = BllDoctype.doctypeQuery();
    console.log("查询列表:", list);

    let list2 = BllDoctype.doctypeQuery("pkid,ctitle", {"cstatus": 1,
```

```
"ctitle|like": "%开%"}, "order by pkid desc")

    console.log("查询列表2:", list2);

    let list3 = BllDoctype.doctypeQuery("cstatus", {"ctitle|like":
"%开%"}, "group by cstatus")

    console.log("查询列表3:", list3);

    console.log("查询行数据:", BllDoctype.doctypeInfo(14))

    let row = BllDoctype.doctypeQueryRow("pkid,ctitle", {"ctitle|like":
"%开%"}, "order by pkid desc")

    let _info = new ModDoctype(row);

    console.log("info==>", _info);



    let tInfo = {"ctitle": "AAAA", "pkid": 0};

    console.log("===>", new ModDoctype(tInfo))

    tInfo["ctitle"] = undefined;

    console.log("===>", new ModDoctype(tInfo))

    console.log("查询行:", row);

    console.log("总条数:", BllDoctype.doctypeCount({"cstatus": 1}))



    //分页
```

```
    //无统计数据
    let page= BllDoctype.doctypePage("*",{"cstatus":1},"order by pkid
desc")
    console.log("分页: ", page)
    //有统计数据
    let page2=
BllDoctype.doctypePageCount("pkid,ctitle",{"cstatus":1},"order by pkid
desc",3,1)
    console.log("分页2: ", page2)

    //新增数据
    let newID= BllDoctype.doctypeInsert({"ctitle":"AAAAAAAAA"})
    console.log("新增数据:", newID);
    console.log("新增数据1:",BllDoctype.doctypeInfo(newID))
    //更新数据
    BllDoctype.doctypeUpdateByID(newID,{"ctitle":null})
    console.log("新增数据2:",BllDoctype.doctypeInfo(newID))

    //删除数据
```

```
console.log("删除条数:", BllDoctype.doctypeDelete({"cstatus": -2}));

console.log("删除指定ID的条数: ", BllDoctype.doctypeDeleteByID(200))
```

5.13 js/jsfun/fnEvent.js 锚点事件

```
//触发时间，异步优先执行（但不会等待），异步内部报错也不会响应执行，同步报错会根据配置sync_error_continue 值处理是否继续下一个同步函数。

fnEvent_call("event_test_a", {"aa": "aaa", "fun": "event_test_a"})

fnEvent_call("event_test_b", {"bb": "bbbb", "fun": "event_test_b"})
```

5.14 js/jsfun/fnHttp.js HTTP请求

```
//如果请求的是https，内部会自行加载证书

let url = "https://www.baidu.com"

// let result= fnHttp_get(url)

// console.log(result);


url = "http://localhost:8888/funHttpresponse.html"

let data = {"aaa": "AAA", "id": "123", "price":
12.12,"online":true,"onsale":false}

let resultGet = fnHttp_get(url, data)
```

```
    console.log("get:",resultGet);


    let resultForm = fnHttp_postForm(url, data)
    console.log("postForm:", resultForm)


    data["obj"]={"title":"bbb","time":fnTime_timestamp()};
    let resultPayload=fnHttp_postJSON(url,data)
    console.log("postJSON:",resultPayload)


    let resultPostTxt=fnHttp_postText(url,JSON.stringify(data))
    console.log("postTxt:",resultPostTxt)


    let
optionGet={"method":"get","timeout":2000,"body":null,"header":{"aa":"aa"
},"cookie":{"bb":"bb"}}
    console.log("OPTION_GET:", fnHttp_option(url+"?aa=aaa",optionGet))


    let
optionPostForm={"method":"post","timeout":2000,"body":"aa=aaa&bbb=234&cc
```

```
=123.123","header":{"Content-Type":
"application/x-www-form-urlencoded"},"cookie":{"bb":"bb"}}
    console.log("OPTION_POSTFORM:",
fnHttp_option(url+"?aa=aaa",optionPostForm))

    let
optionPostJSON={"method":"post","timeout":2000,"body":JSON.stringify(dat
a),"header":{"Content-Type": "application/json"},"cookie":{"bb":"bb"}}
    console.log("OPTION_POSTJSON:",
fnHttp_option(url+"?aa=aaa",optionPostJSON))

    let
optionPostTxt={"method":"post","timeout":2000,"body":"aa=aaa&bbb=234&cc=
123.123","header":{"Content-Type": "text/plain"},"cookie":{"bb":"bb"}}
    console.log("OPTION_POSTTxt:",
fnHttp_option(url+"?aa=aaa",optionPostTxt))

    let
optionPostBytes={"method":"post","timeout":2000,"body":"aa=aaa&bbb=234&c
```

```
c=123.123","header":{"Content-Type": "text/plain"},"cookie":{"bb":"bb"}}

    console.log("OPTION_POSTBytes:",

fnHttp_optionBytes(url+"?aa=aaa",optionPostBytes))




/**

 * HTTP操作响应

 * @param params

 * @jssaas url /funHttpresponse.html

 * @jssaas method get,post

 * @jssaas timeout 200000

 */

function main_fun_httpresponse(params) {

  return fnRenderText(JSON.stringify(params))

}
```

5.15 js/jsfun/fnIO.js IO操作

```
    //创建文件夹

    fnIO_createFolder(fnJsa_jsCodePath()+"/demoresource/io")

    let file=fnJsa_jsCodePath()+"/demoresource/io/io1.txt"

    let fileNew=fnJsa_jsCodePath()+"/demoresource/io/io12.txt"

    //保存文件

    fnIO_saveFile(file, "当前时间: "+fnTime_now("Ymd H:i:s"),true)

    //复制文件

    fnIO_copy(file,fileNew)

    //复制文件夹


fnIO_copyFolder(fnJsa_jsCodePath()+"/demoresource/io",fnJsa_jsCodePath()
+"/demoresource/io_copy")


    //IO存在, 可具体到文件、文件夹

    console.log("IO存在", fnIO_exist(file))

    console.log("文件存在", fnIO_exist(file,1))

    console.log("文件夹存在", fnIO_exist(file,2))
```

```
    console.log("文件夹存在",
fnIO_exist(fnJsa_jsCodePath()+"/demoresource/io",2))


    console.log("文件字节：", fnIO_getFileByte(file))
    console.log("文件文本：", fnIO_getFileContent(file))
    console.log("文件夹列表1：",
fnIO_getFiles(fnJsa_jsCodePath()+"/demoresource/io",".txt"))
    console.log("文件夹列表2：",
fnIO_getFiles(fnJsa_jsCodePath()+"/demoresource/io",".txt",false,true))
    console.log("文件夹列表3：",
fnIO_getFiles(fnJsa_jsCodePath()+"/demoresource/io",".txt",true))
    console.log("文件夹列表4：",
fnIO_getFiles(fnJsa_jsCodePath()+"/demoresource/io",".txt",true,true))
    console.log("文件夹列表5：",
fnIO_getFiles(fnJsa_jsCodePath()+"/demoresource/io","",true,true))


    //删除文件或目录
    fnIO_remove(fileNew)
```

```
fnIO_copyFolder(fnJsa_jsCodePath()+"/demoresource/io_copy",fnJsa_jsCodeP
ath()+"/demoresource/io_copy2")
    fnIO_remove(fnJsa_jsCodePath()+"/demoresource/io_copy2",true)

    //复制文件
    fnIO_copy(file,fileNew)
    //重命名
    fnIO_rename(fileNew,fnJsa_jsCodePath()+"/demoresource/io/io13.txt")
    fnIO_saveFile(file,"当前时间: "+ fnTime_now("Y-m-d H:i:s"))
    let bytes= fnIO_getFileByte(file)

fnIO_saveFileByte(fnJsa_jsCodePath()+"/demoresource/io/io14.txt",bytes,t
rue)
```

5.16 js/jsfun/fnJsa.js 系统运行环境

```
    //Jsa平台定时任务
    console.log("Jsa平台定时任务:", fnJsa_dataCrond())
    //Jsa平台事件配置
```

```
    console.log("Jsa平台事件配置:",  fnJsa_dataEventConfig())
    //Jsa平台事件配置需要执行的函数
    console.log("Jsa平台事件配置需要执行的函数:",  fnJsa_dataEvent())
    //Jsa平台函数列表
    console.log("Jsa平台函数列表:",  fnJsa_dataFun())
    //Jsa平台初始化函数
    console.log("Jsa平台初始化函数:",  fnJsa_dataInit())
    //Jsa平台入口主函数
    console.log("Jsa平台入口主函数:",  fnJsa_dataMain())
    //Jsa平台HTTP请求地址
    console.log("Jsa平台HTTP请求地址:",  fnJsa_dataUrl())
    //JSA平台执行环境
    console.log("JSA平台执行环境: ",JSON.stringify( fnJsa_env()))
    //取得全局文件的内容：global.js
    console.log("", fnJsa_globalJs());
    //当前执行的是否是异步调用

console.log("当前是否是异步调用，异步调用无法取得http的request信息:",
fnJsa_isAsync())
```

```
    fnAsync_run("fun_jsa_async",null)

    //jscode路径

    console.log("jscode路径: "+    fnJsa_jsCodePath());


    console.log("微服务列表: ",    fnJsa_microservices())

    console.log("Jssaas平台名称: ", fnJsa_name());

    console.log("jssaas平台根目录:", fnJsa_rootPath());

    console.log("jsaaas平台运行模式: ", fnJsa_runMode());

    console.log("jssaas平台版本: ", fnJsa_version());

    console.log("jssaas平台版本: ",     fnJsa_versionCode())


    //强制退出  防止数据库事务执行到一半被强迫终止而无法回滚。

    // fnJsa_exit("JSA平台强制退出")

    // fnJsa_exitHtml("强制输出html:<b style='color:
darkred;'>JSA平台强制退出</b>")

    // let
json={"title":"JSA平台强制退出","obj":{"id":123,"price":123.45}}

    // fnJsa_exitJson(JSON.stringify(json))
```

5.17 js/jsfun/fnJson.js json文件操作

```
    let jsonFile = fnJsa_jsCodePath() + "/demoresource/test.json"

    let
data={"title":"标题","info":{"title":"商品","price":123.42,"tag":"aa"},"
ts":fnTime_timestamp()};

    fnIO_saveFile(jsonFile,JSON.stringify(data),true)

      data = fnJson_config(jsonFile)

    console.log("json数据: ",data)



    fnJson_configUpdate(jsonFile,"info.price",100)

    fnJson_configUpdate(jsonFile,"info.addtime",fnTime_timestamp())

    fnJson_configDelete(jsonFile,"info.tag")

    data = fnJson_config(jsonFile)

    console.log("json数据: ",data)



    //JSON 列表操作

    let jsonListFile=fnJsa_jsCodePath()+"/demoresource/json/list.json"

    fnIO_createFolder(jsonListFile)
```

```
    let
listData=[{"id":1,"title":"标题1"},{"id":2,"title":"标题2"},{"id":3,"tit
le":"标题3"}]
    fnIO_saveFile(jsonListFile,JSON.stringify(listData),true)
    console.log("info(id=2):", fnJson_listInfo(jsonListFile,{"id":2}));
    for (let i = 4; i < 20; i++) {
        fnJson_listInsert(jsonListFile,{"id":i,"title":"标题"+i})
    }

    console.log("list数据,新增:", fnJson_list(jsonListFile));
    fnJson_listUpdate(jsonListFile,{"title":"标题3333"},{"id":3})
    console.log("list数据, 更新:", fnJson_list(jsonListFile));
    fnJson_listDelete(jsonListFile,{"id":1})
    console.log("list数据, 删除:", fnJson_list(jsonListFile));

    console.log("list分页1: ", fnJson_listPage(jsonListFile,"id"))
    console.log("list分页2: ",
fnJson_listPage(jsonListFile,"id",{},"",2))
    console.log("list分页3: ",
```

```
fnJson_listPage(jsonListFile,"id",{},"",3))
```

5.18 js/jsfun/fnLock.js 锁操作

```
console.log("最多能支持的锁数量:", fnLock_maxLockNum());


// console.log("时间戳",fnTime_timestamp())
//加入锁，10秒后自动解锁，强烈建议加入自动解锁时间，防止一直被占用。
let index=params["get"]["index"]
let lockIndex=0;
if (index!=""){
    lockIndex=parseInt(index)
}
fnLock_lock(lockIndex,10000)
console.log("当前启用的LOCK列表:", fnLock_list());
//解锁
fnLock_unlock(Lock_main)
```

5.19 js/jsfun/fnObject.js 对象操作

```
let obj = {"zzz": "ZZZZ", "aa": "AAAA", "title": "TITLE"}
```

```
if (obj["bb"] == undefined) {
    console.log("bb is undefinded")
} else {
    console.log("bb is not undefinded")
}
//这边bb已经声明，所以可以直接与undefined对比判断
let bb = obj["bb"]
if (bb == undefined) {
    console.log("bb2 is undefinded")
} else {
    console.log("bb2 is not undefinded")
}
```

//这边的cc未声明，无法确定cc是否之前已经声明，无法直接用==undefined对比判断，需要通过判断类型才能确定cc是否已经声明。

```
// cc的类型值
console.log("cc is:", typeof cc)
// cc的类型值的类型
```

```
console.log("cc is:", typeof typeof cc)


if (typeof cc == "undefined") {
    console.log("cc is undefinded")
} else {
    console.log("cc is not undefinded")
}


//取得对象的第一个属性值
console.log("对象的第一个属性值：", fnObject_current(obj))


console.log("对象是否为空：", fnObject_isNotNull(obj));
console.log("对象是否为空：", fnObject_isNotNull(obj["dd"]));
```

5.20 js/jsfun/fnQueue.js 队列操作

```
//队列遵循的是先进先出（FIFO）的原则，即最先进入队列的元素将是最先被移除
```

的。队列的操作主要限定在队头和队尾进行，包括入队（enqueue）操作在队尾添加元素，出队（dequeue）操作在队头移除元素。队列中没有元素时，称为空队列。

```
//构造队列
let qKey = "q_1"
let qKey2 = "q_2"
//初始化队列,默认元素ID为当前http请求ID值，也可以自定义
fnQueue_init(qKey, 10, 20000)
let elementID0 = "elementID_0"
let elementID1 = "elementID_1"
let elementID2 = "elementID_2"
let elementID3 = "elementID_3"
let elementID4 = "elementID_4"
let elementID5 = "elementID_5"
for (let i = 0; i < 6; i++) {
    fnQueue_push(qKey, "elementID_" + i)
}
for (let i = 0; i < 6; i++) {
    fnQueue_push(qKey2, "elementID_" + i)
```

```
}


console.log("队列全部元素:",    fnQueue_list(qKey))

console.log("队列长度:", fnQueue_len(qKey))

console.log("是否首个元素:0:", fnQueue_isFirst(qKey, elementID0))

console.log("是否首个元素:1:", fnQueue_isFirst(qKey, elementID1))

console.log("弹出首个元素:", fnQueue_pop(qKey))

console.log("是否首个元素:0:", fnQueue_isFirst(qKey, elementID0))

console.log("是否首个元素:1:", fnQueue_isFirst(qKey, elementID1))

console.log("队列长度:", fnQueue_len(qKey))

console.log("队列全部元素:",    fnQueue_list(qKey))


console.log("队列2全部元素:",    fnQueue_list(qKey2))

console.log("全部队列", fnQueue_allList())


console.log("删除队列元素", fnQueue_delete(qKey,elementID4))

console.log("队列长度:", fnQueue_len(qKey))

console.log("队列全部元素:",    fnQueue_list(qKey))
```

```
    console.log("队列首元素: ", fnQueue_first(qKey))

    console.log("队列当前元素默认ID", fnQueue_defaultID())

    console.log("队列当前元素索引位置", fnQueue_index(qKey,elementID4))

    console.log("队列信息: ", fnQueue_info(qKey))

    //判断队列元素是否是首元素，如果是，则进入业务
    if (fnQueue_isFirst(qKey, elementID1)) {
        console.log("queue_run_business 执行业务:", qKey, elementID1)
    } else {
        console.log("queue_run_business 非法进入业务:",qKey, elementID1)
    }
```

//判断队列元素是否是首元素，如果还不是首元素，可以for也可以一直选项等待队列的首元素弹出

```
    for (let i = 0; i < 100; i++) {
        if (fnQueue_isFirst(qKey, elementID1)) {
            console.log("我已经是队列首元素了，开始执行业务");

console.log("我的业务执行完了，记得将首元素移出，可以弹出首元素，也可以删除指定元素ID")

//如果已经知道首元素ID，建议用fnQueue_delete，删除指定元素比较靠谱，防止当前使用首元素已经被弹出
            //弹出首元素
            fnQueue_pop(qKey)
            //删除执行元素
            //fnQueue_delete(qKey,elementID1)
        } else {
            console.log("我还没进入队列首元素，只能耐心等待");

//休眠时间最好是执行业务的时间，或者1/2，太长了就会浪费大家等待时间，太短会造成CPU的资源浪费，适合最好
            fnTime_sleep(20)
```

```
        }
    }


    let bln= fnQueue_waitToFirst(qKey,elementID1,5000)
    if(bln){
        console.log("我已经是队列首元素了，开始执行业务");

console.log("我的业务执行完了，记得将首元素移出，可以弹出首元素，也可以删除指定元素ID")

//如果已经知道首元素ID，建议用fnQueue_delete，删除指定元素比较靠谱，防止当前使用首元素已经被弹出
        //弹出首元素
        fnQueue_pop(qKey)
        //删除执行元素
        //fnQueue_delete(qKey,elementID1)
    }else {
        console.log("我已经被删除了");
```

```
}



console.log("清除队列元素", fnQueue_clear(qKey));

console.log("清除全部队列元素", fnQueue_clearAll());
```

5.21 js/jsfun/fnRedis.js Redis操作

```
let key = "redis_key"


console.log("初始化redis连接配置", fnRedis_init())
for (let i = 0; i < 9; i++) {
    console.log("================>>>" + i)
    let value = null
    if (i == 0) {
        value = "test测试";
    } else if (i == 1) {
        value = 1;
    } else if (i == 2) {
        value = 11234567891;
```

```
        } else if (i == 3) {
            value = 11267.132;
        } else if (i == 4) {
            value = {"title": "AAAAA", "price": 123.23};
        } else if (i == 5) {
            value = [12, 34, 231, 54234];
        } else if (i == 6) {
            value = [{"title": "AAAAA", "price": 123.23}, {"title":
"BBB", "price": 552.23}];
        } else if (i == 7) {
            value = true;
        } else if (i == 8) {
            value = false;
        }
        console.log("设置缓存", fnRedis_set(key, value, 100))
        console.log("取得缓存数据", fnRedis_get(key));

        console.log("取得缓存数据字符串：", fnRedis_getString(key))
        console.log("取得缓存数据数字：", fnRedis_getFloat(key))
```

```
        console.log("取得缓存数据布尔：", fnRedis_getBool(key))
        console.log("取得缓存数据数组：", fnRedis_getArray(key))
        console.log("取得缓存数据对象：", fnRedis_getMap(key))

    }

    // console.log("设置缓存数据：", fnRedis_set(key,
fnTime_timestamp()))
    console.log("移出缓存数据：", fnRedis_remove(key));
```

5.22 js/jsfun/fnRender.js 渲染操作

```
    //测试时，将前面return行的代码注释掉，

//render函数需要和return配合使用，将当前请求的最终结果返回给jssaas平台，
jssaas平台按配置信息解析处理。
    //====返回JSON结构
    return fnRender_error("错误信息")
```

```
    return fnRender_json({"title": "AAAA", "price": 123})


    //=====导出流文件
    let bytes = fnIO_getFileByte(fnJsa_jsCodePath() +
"/demoresource/aa.txt")
    //无需配置头部信息
    return fnRender_exportFile("aaa.txt", bytes)
    //自定义导出配置，未配置，则返回文本内容
    return fnRender_fileByte({}, bytes)


    //=====页面输出
    //模板输出
    return fnRender_template("html/index.html", {"t":
fnTime_timestamp()})
    //模板内容输出
    let html = `
<body>
    <h1>标题</h1>
    <h2>当前时间戳：{{.t}}</h2>
```

```
</body>`;

    return fnRender_templateContent(html, {"t": fnTime_timestamp()})

    //html内容输出

    return fnRender_html(html)

    //没数据页面

    return fnRender_nodata("内部异常导致无可用数据")



    //自定义输出

    return fnRender_option(html, {})

    //页面转跳

    return fnRender_redirect("http://www.baidu.com")




    //====返回文本

    return fnRender_text("我是测试文本")
```

5.23 js/jsfun/fnRequest.js HTTP请求操作

```
    //请求地址:
```

```
    console.log("取得请求地址信息：", fnRequest_url())
    console.log("取得请求完整地址信息：", fnRequest_urlFull())
    //Nginx配置HTTPS站点时，需要增加设置头部信息：proxy_set_header
Jssaas-Scheme $scheme; #将请求协议http、https信息放入头部传递给jsaaas
    console.log("取得请求完整地址信息(http)：", fnRequest_urlFull(true))
    //头部信息
    console.log("取得头部信息：", fnRequest_headerGet("User-Agent"))
    console.log("取得头部信息：", fnRequest_headerAll())
    // 请求参数数据：
    //get
    console.log("取得GET请求参数：", fnRequest_get())
    console.log("取得GET请求数组参数：", fnRequest_getArray())
    console.log("取得GET请求正则地址参数数组：",
fnRequest_urlRegParams())
    //post
    console.log("取得请求Post Form信息：", fnRequest_post())
    console.log("取得请求Post Form数组信息：", fnRequest_postArray())
    console.log("取得请求JSON（post payload请求）信息：",
fnRequest_json())
```

```
    //file
    console.log("取得请求文件信息: ", fnRequest_file())
    console.log("取得请求文件信息(包含文件内容): ",
fnRequest_file(true))
    console.log("取得请求文件保存: ",
fnRequest_fileSave(fnJsa_jsCodePath()+"/aa.xlsx"))
    //body
    console.log("取得请求BODY信息: ", fnRequest_body())


    //IP:
    console.log("取得客户端IP信息: ", fnRequest_clientIP())
    //连接ID
    console.log("取得请求连接ID: ", fnRequest_id())


/**
 * 动态地址
 * @param params
 * @jssaas url /funRequest-(.*)-(.?)(.+)(.?)\.html
 * @jssaas urlisreg true
```

```
 * @jssaas urlregindex 5
 * @jssaas method get,post,get,put,delete
 * @jssaas timeout 10000
 */
function main_fun_request2(params) {
    let t1 = fnTime_timestampMilli();

    // 请求参数数据:
    //get
    console.log("取得GET请求参数: ", fnRequest_get())
    console.log("取得GET请求数组参数: ", fnRequest_getArray())
    console.log("取得GET请求正则地址参数数组: ", fnRequest_urlParams())
    let t2 = fnTime_timestampMilli();
    return fnRender_template("/html/index.html", {"t": t2 - t1})
}
```

5.24 js/jsfun/fnResponse.js HTTP响应处理

```
  fnResponse_headerSet({"choice": "ddddd"})
    //测试转跳时取消下面注释
```

```
    //fnResponse_redirect("http://www.baidu.com")


    //HTTP响应追加内容
    fnResponse_write("test测试 write")
    fnResponse_append("test测试 append")
    //关闭连接后就不能response输出
    fnResponse_closeConnect()
```

5.25 js/jsfun/fnSession.js Session对象操作

```
    let key = "sesseion_key"
    console.log("session值: ", fnSession_get(key))

    console.log("设置Session值, 字符串: ", fnSession_set(key,
fnTime_timestamp()))
    console.log("session值: ", fnSession_get(key))
    console.log("设置Session值, 数字: ", fnSession_set(key, 123))
    console.log("session值: ", fnSession_get(key))
    console.log("设置Session值, 字符串: ", fnSession_set(key, 132.456))
```

```
    console.log("session值: ", fnSession_get(key))

    console.log("设置Session值, 布尔: ", fnSession_set(key, true))

    console.log("session值: ", fnSession_get(key))

    console.log("设置Session值, 字符串数组: ", fnSession_set(key,
["aaaa", "测试"]))

    console.log("session值: ", fnSession_get(key))

    console.log("设置Session值, 对象: ", fnSession_set(key, {"title":
"标题AA", "time": fnTime_timestamp(), "list": ["aaa", "bbb"]}))

    console.log("session值: ", fnSession_get(key))

    console.log("cookie ID 值: ", fnSession_getID())

    console.log("删除当前用户指定Session值: ", fnSession_remove(key))

    console.log("删除当前用户全部Session值: ", fnSession_removeAll())

    console.log("清空服务端全部的Session: ", fnSession_flush())
```

5.26 js/jsfun/fnStack.js 栈操作

```
    //栈遵循的是后进先出的原则


    //构造栈
```

```
let qKey = "q_1"

let qKey2 = "q_2"

//初始化栈,默认元素ID为当前http请求ID值，也可以自定义

fnStack_init(qKey, 10, 20000)

let elementID0 = "elementID_0"

let elementID1 = "elementID_1"

let elementID2 = "elementID_2"

let elementID3 = "elementID_3"

let elementID4 = "elementID_4"

let elementID5 = "elementID_5"

for (let i = 0; i < 6; i++) {

    fnStack_push(qKey, "elementID_" + i)

}

for (let i = 0; i < 6; i++) {

    fnStack_push(qKey2, "elementID_" + i)

}



console.log("栈全部元素:",    fnStack_list(qKey))

console.log("栈长度:", fnStack_len(qKey))
```

```
console.log("是否首个元素:0:", fnStack_isFirst(qKey, elementID5))

console.log("是否首个元素:1:", fnStack_isFirst(qKey, elementID4))

console.log("弹出首个元素:", fnStack_pop(qKey))

console.log("是否首个元素:0:", fnStack_isFirst(qKey, elementID5))

console.log("是否首个元素:1:", fnStack_isFirst(qKey, elementID4))

console.log("栈长度:", fnStack_len(qKey))

console.log("栈全部元素:",    fnStack_list(qKey))


console.log("栈2全部元素:",    fnStack_list(qKey2))

console.log("全部栈", fnStack_allList())


console.log("删除栈元素", fnStack_delete(qKey,elementID3))

console.log("栈长度:", fnStack_len(qKey))

console.log("栈全部元素:",    fnStack_list(qKey))



console.log("栈首元素: ", fnStack_first(qKey))


console.log("栈当前元素默认ID", fnStack_defaultID())
```

```
    console.log("栈当前元素索引位置", fnStack_index(qKey,elementID2))


    console.log("栈信息: ", fnStack_info(qKey))


    //判断栈元素是否是首元素，如果是，则进入业务
    if (fnStack_isFirst(qKey, elementID4)) {
        console.log("stack_run_business 执行业务:", qKey, elementID4)
    } else {
        console.log("stack_run_business 非法进入业务:",qKey, elementID4)
    }



//判断栈元素是否是首元素，如果还不是首元素，可以for也可以一直选项等待栈的首元素弹出
    for (let i = 0; i < 100; i++) {
        if (fnStack_isFirst(qKey, elementID4)) {
            console.log("我已经是栈首元素了，开始执行业务");
```

```
console.log("我的业务执行完了，记得将首元素移出，可以弹出首元素，也可以删除指定元素ID")

//如果已经知道首元素ID，建议用fnStack_delete，删除指定元素比较靠谱，防止当前使用首元素已经被弹出
            //弹出首元素
            fnStack_pop(qKey)
            //删除执行元素
            //fnStack_delete(qKey,elementID1)
        } else {
          //  console.log("我还没进入栈首元素，只能耐心等待");

//休眠时间最好是执行业务的时间，或者1/2，太长了就会浪费大家等待时间，太短会造成CPU的资源浪费，适合最好
            fnTime_sleep(20)
        }
    }
```

```
let bln= fnStack_waitToFirst(qKey,elementID4,5000)
if(bln){
    console.log("我已经是栈首元素了，开始执行业务");

console.log("我的业务执行完了，记得将首元素移出，可以弹出首元素，也可以删除指定元素ID")

//如果已经知道首元素ID，建议用fnStack_delete，删除指定元素比较靠谱，防止当前使用首元素已经被弹出
    //弹出首元素
    fnStack_pop(qKey)
    //删除执行元素
    //fnStack_delete(qKey,elementID1)
}else {
    console.log("我已经被删除了") ;
}

console.log("清除栈元素", fnStack_clear(qKey));
```

```
    console.log("清除全部栈元素", fnStack_clearAll());
```

5.27 js/jsfun/fnString.js 字符串处理

```
   console.log("GUID:", fnString_guid());
    let txt="123456.12";
    console.log("数字转中文：", fnString_numberToChinese(txt));
    console.log("数字转大写中文数字：",
fnString_numberToChineseUppercase(txt));
```

5.28 js/jsfun/fnTime.js 时间操作

```
    let ts=fnTime_timestamp()
    console.log("格式化时间", fnTime_format(ts,"Y-m-d H:i:s"))
    console.log("当前时间：",fnTime_now("Y-m-d H:i:s"))
    console.log("取得时间戳 毫秒：", fnTime_timestampMilli())
    console.log("取得时间戳 秒：", fnTime_timestamp())


    console.log("休眠2秒：", fnTime_sleep(2000))
    console.log("取得时间戳 秒：", fnTime_timestamp())
```

5.29 js/jsfun/fnUtil.js 工具集合

```
let txt = "";

let bytes = fnUtil_stringToBytes(txt)

let txt2 = fnUtil_bytesToString(bytes)

let base64 = fnUtil_bytesToBase64(bytes)



let byts2=fnUtil_base64ToBytes(base64)

let txt3= fnUtil_base64ToString(base64)

console.log("生成随机数: ", fnUtil_randInt())
```

5.30 js/jsfun/fnZip.js zip操作

```
let zipFile=fnJsa_jsCodePath()+"/aa.zip"

// //压缩不存在的文件或目录

// fnZip_zip(fnJsa_jsCodePath()+"/demoresource2/",zipFile)



// // //压缩目录

// fnZip_zip(fnJsa_jsCodePath()+"/demoresource/",zipFile)



 let tmpPath=fnJsa_jsCodePath()+"/demoresource/";
```

```
    //压缩文件列表，添加套根目录
    // zipFile=fnJsa_jsCodePath()+"/aa_2.zip"
    // let files=[tmpPath+"aa.txt",tmpPath+"test.json",tmpPath+"io/"]
    // fnZip_zip(files,zipFile)

    //压缩文件列表，添加套根目录
    zipFile=fnJsa_jsCodePath()+"/aa_3.zip"
    let files2={"aa/aa.txt": tmpPath+"aa.txt","bb/cc.json":
tmpPath+"test.json","dd/": tmpPath+"io/"}
    fnZip_zip(files2,zipFile)

    //追加文件
    // fnZip_zip(fnJsa_jsCodePath()+"/runtime/",zipFile)


    // //解压缩目录
    // fnZip_unzip(zipFile,fnJsa_jsCodePath()+"/demoresource2/")
```

```
//将内置资源包释放出来

let bytes = fnInnerResouce_getBytes("jsaresource/jssaas_pkg.zip")

if (bytes == null || bytes.length == 0) {

    return "内置资源不存在";

}

let jscodePath = fnJsa_jsCodePath()

let tmpZip = jscodePath + "/tmp.zip"

fnIO_saveFileByte(tmpZip, bytes);

fnZip_unzipFile(tmpZip, jscodePath + "/tmp" , ["emptyproject/"])
```

六：开发注意

6.1 数据库设计说明

表主键为pkid，字段第一个字母为c，字段最好包含caddtime：新增时间,cmodifytime：修改时间。MySQL数据库务必包含最后一次修改字段,mysql更新时如果更新内容和原来内容一样，回返回更新失败。

6.2 开发注意

单微服务项目，入口函数（main文件夹）下的函数需要以main_开头

多微服务项目，入口函数（main文件夹）下的函数需要以xxx_main_开头,xxx为微服务目录名称。

判断一个变量是否存在 undefined

```
if(typeof global_database_excludeColumns!="undefined" ){

}
```

这儿需要在使用global_database_excludeColumns变量时，通过typeof来判断

判断一个变量是否合法 fn_objet_Valid(global_database_excludeColumns)

如果不能确实变量是否存在，可以先判断undefined，再判断fnObjet_Valid

在定时任务/初始化等无法取得请求句柄等情况下，时无法取得与cookie相关等数据，如cookie/session数据。

不要将变量的值定义为：undefined，确记确记确记

6.3 开发工具

推荐：WebStorm（请支持正版软件）

下载地址：https://www.jetbrains.com/zh-cn/webstorm/

网络上有试用期延长方法：

https://blog.csdn.net/qq_37699336/article/details/116528062

JavaScript代码格式需要与webstorm的默认定义js格式一样，否则在解析JavaScript会失败。

正确标准格式：

```
function main_index_home(params) {
    //这是是JavaScript代码

}
```

两个{}位置要确保

开发环境运行的平台与生存环境运行的平台无法互用。开发环境是有限制CPU使用个数和每分钟支持请求次数限制。生产环境需要从jssaas官网下载生产证书，才能解除CPU和请求次数限制。

七：开发群

7.1 ssaas开发群

前端技术人员开发群：361137248