目录

概述

简介

quick-frame是一款基于SpringBoot2 + Spring + Mybatis的敏捷开发系统；它是一款具有代码生成功能的智能快速开发平台；是以Spring Framework为核心容器，Spring MVC为模型视图控制器，Mybatis为数据访问层。

quick-frame的核心设计目标是开发迅速、代码量少、学习简单、功能强大、轻量级、易扩展、Restful。

quick-frame主要定位于企业快速开发平台建设，已内置很多优秀的基础功能和高效的工具，包括：系统权限组件、核心工具组件、代码生成、 数据推送、日志、请求限流、数据缓存、前后端交互协议、datasong大数据平台访问驱动、定时任务、xss过滤、防重复提交、统一跨域处理、统一异常处理、okhttp客户端、ssh客户端、redis工具、k8s工具、rpc工具、springboot-admin监控、https支持等。

quick-frame可以快速协助java开发人员解决60%的重复工作，让开发人员更多关注业务逻辑的实现，解放JAVA开发人员的开发压力，提高开发效率，为企业节省项目研发成本，减少开发周期。 quick-frame主要面向基于 Web 的中小规模的应用程序,新手能在较短时间内入门，核心具有良好的定制性且插件易于扩展。

quick-frame以gradle作为构建工具实现项目构建。

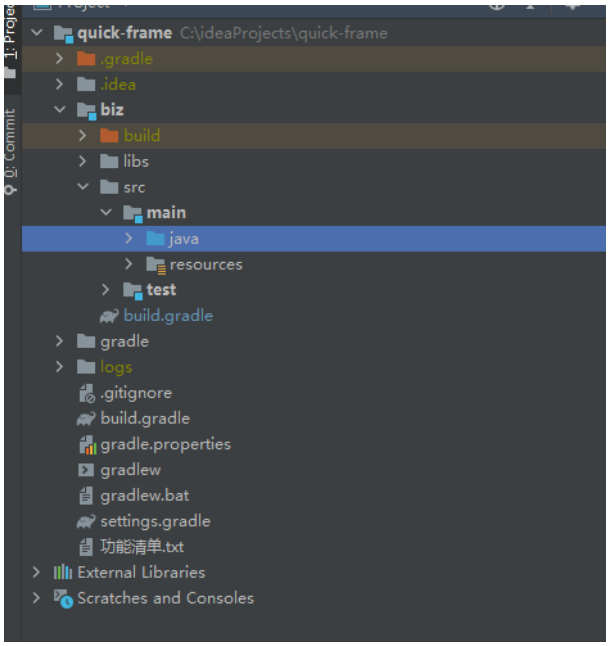
quick-frame的JDK版本兼容8与11

目前已开放的功能如下：

```
compile "com.iscas:templet:${gegeralVersion}" //前后台交互协议模板定义
compile "com.iscas:common-tools:${gegeralVersion}" //基础工具类
compile "com.iscas:common-web-tools:${gegeralVersion}" //web相关工具类
compile "com.iscas:common-rpc-tools:${gegeralVersion}" //rpc相关工具类
compile "com.iscas:common-redis-tools:${gegeralVersion}" //redis相关工具类
compile "com.iscas:biz-mp:${gegeralVersion}" //mybatis相关
compile "com.iscas:biz-base:${gegeralVersion}" //biz-base相关
```

项目搭建

项目以gradle搭建，目录结构如下



重点功能说明如下：

- gradle.properties

```
定义maven库的地址、引用依赖的版本等基础配置项
nexusUrl=http://172.16.10.190:8081/repository/maven-public/
#nexusUrl=http://maven.aliyun.com/nexus/content/groups/public
aliyunUrl=http://maven.aliyun.com/nexus/content/groups/public
springUrl=https://repo.spring.io/libs-milestone/


#java-version
javaVersion=1.8

## dependency versions.
springBootVersion=2.4.0
platformVersion=Cairo-SR8
junitVersion=4.12
lombokVersion=1.18.4

#自研模块相关依赖的版本
gegeralVersion=0.0.1-2021.01.01-3
```

- setting.gradle

```
rootProject.name = 'quick-frame'
include 'biz'
```

- 总体build.gradle

```
buildscript {
    repositories {
        maven { url "${nexusUrl}" }
        maven {url "${aliyunUrl}"}
        maven { url "${springUrl}" }
        jcenter()
```

```
        mavenCentral()

    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
    }
}

subprojects  {
    apply plugin: 'java'
    apply plugin: 'org.springframework.boot'
    apply plugin: 'idea'
    apply plugin: 'io.spring.dependency-management'

    group = 'com.iscas'
    version = '0.0.1'
    sourceCompatibility = "${javaVersion}".toFloat()
    targetCompatibility = "${javaVersion}".toFloat()


    configurations {
        compile.exclude group: 'org.springframework.boot', module: 'spring-boot-starter-tomcat'
        compile.exclude group: 'org.apache.tomcat'
        compile.exclude group: 'org.apache.tomcat.embed'
    }

    repositories {
        maven { url "${nexusUrl}" }
        maven {url "${aliyunUrl}"}
        maven { url "${springUrl}" }
        jcenter()
        mavenCentral()
    }
    tasks.withType(JavaCompile) {
        options.encoding = "UTF-8"
    }
    dependencies {
//        compile 'org.projectlombok:lombok'
        annotationProcessor "org.projectlombok:lombok:${lombokVersion}"
        compileOnly "org.projectlombok:lombok:${lombokVersion}"
        testAnnotationProcessor "org.projectlombok:lombok:${lombokVersion}"
        testCompileOnly "org.projectlombok:lombok:${lombokVersion}"
        //junit
        testCompile group: 'junit', name: 'junit', version:"${junitVersion}"
    }

    //这里一定得要。在多模块下，不然编译失败，因为不会把依赖模块给打包。
    jar {
        enabled = true
    }
}

def bootJarModules() {
    subprojects.findAll {it.name.equals('biz')/* || it.name.equals("rule-engine")*/}
}

task bootApp {
    dependsOn /*subprojects.clean,*/ bootJarModules().bootJar
    doLast {
        println "bootApp执行结束"
    }
}
```

- biz模块的build.gradle

```


dependencies {
```

```
    compile "com.iscas:templet:${gegeralVersion}" //前后台交互协议模板定义
    compile "com.iscas:common-tools:${gegeralVersion}" //基础工具类
    compile "com.iscas:common-web-tools:${gegeralVersion}" //web相关工具类
    compile "com.iscas:common-rpc-tools:${gegeralVersion}" //rpc相关工具类
    compile "com.iscas:common-redis-tools:${gegeralVersion}" //redis相关工具类
    compile "com.iscas:biz-mp:${gegeralVersion}" //mybatis相关
    compile "com.iscas:biz-base:${gegeralVersion}" //biz-base相关
    compile fileTree(dir:'libs',include:['*.jar']) //引入本地jar
}
```

项目打包方式

1. 打可运行jar包

   第一种方式是在IDEA-gradle窗口-biz(业务模块)-Tasks-build下运行bootJar
   第二种方式是进入biz(业务模块)模块目录，运行gradle bootJar。当然这个前提是本地配置gradle环境。
   打好的Jar包在biz(业务模块)/build/libs下

2. 打war包

   业务模块 build.gradle中和主函数内都已经作了对打War包的配置处理，打War包和打Jar包类似：

   第一种方式是在IDEA-gradle窗口-biz(业务模块)-Tasks-build下运行bootWar
   第二种方式是进入biz(业务模块)模块目录，运行gradle bootWar。当然这个前提是本地配置gradle环境。
   打好的War包在biz(业务模块)/build/libs下

业务模块使用说明

mybatis配置及使用

单数据源配置

1、在配置文件中指定active，检查包含参数值中包含dev
spring.profiles.active=dev 2、修改application-dev.properties配置文件中的参数
（1）、修改数据源：spring.datasource.names=mysql1
（2）、修改数据库连接参数
如：数据库连接 ：spring.datasource.druid.mysql1.url
用户名 ：spring.datasource.druid.mysql1.username
密码 ：spring.datasource.druid.mysql1.password
(3)、spring.datasource.druid.mysql1.pointcut可不配置
注：mysql1的名字可随意改，但要跟数据库连接参数中的key保持一致

多数据源配置

1、在配置文件中指定active，检查包含参数值中包含dev
spring.profiles.active=dev
2、修改application-dev.properties配置文件中的参数
（1）、修改数据源：spring.datasource.names=mysql1，mysql2
（2）、按照单数据源配置方式修改其他数据库连接参数
如：数据库连接 ：spring.datasource.druid.mysql2.url
用户名 ：spring.datasource.druid.mysql2.username
密码 ：spring.datasource.druid.mysql2.password
(3)、第一个数据源spring.datasource.druid.mysql1.pointcut可不配置
其他数据源都需要配置该参数
注：mysql1、mysql2的名字可随意改，注意要跟数据库连接参数中的key保持一致

mybatis-plus基本使用

参考samples包下的LogInfoControllerTest类
更高级的用法参考官方文档：https://baomidou.com/guide/

执行动态sql

有时候需要执行更加复杂的sql语句，如复杂的关联查询、流式查询等，其中一种方式就是使用动态sql。
动态sql示例参考DynamicSqlController

数据源扩展

自定义plugins、typeHandler、enumHandler等都可以通过扩展类添加到sqlSessionFactory中，参考SqlSessionFactoryCustomizerPluginsTest

代码生成器

用浏览器或postman访问接口，请求为get请求，接口为http://192.168.100.73:7901/demo/mp/generator
其中ip和port要替换成服务运行的地址和端口，第一级路径demo要替换成项目的server.servlet.context-path

logback日志配置

配置文件名称

1、application.properties文件中对logback进行配置
##########logback.xml配置############
logging.config=classpath:logback.xml

2、配置文件名称 logback.xml

配置文件内容

stc-pub日志配置

```
<!-- 日志记录器，日期滚动记录 -->
<appender name="FILEERROR" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <!-- 正在记录的日志文件的路径及文件名 -->
    <file>${LOG_PATH}/${APPDIR}/log_error.log</file>
    <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <!-- 归档的日志文件的路径，例如今天是2013-12-21日志，当前写的日志文件路径为file节点指定，可以将此文件与file指定文件路径设置为不同路径，从而将当
        而2013-12-21的日志文件在由fileNamePattern指定。%d{yyyy-MM-dd}指定日期格式，%i指定索引 -->
        <fileNamePattern>${LOG_PATH}/${APPDIR}/error/log-error-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
        <!-- 除按日志记录之外，还配置了日志文件不能超过2M，若超过2M，日志文件会以索引0开始，
        命名日志文件，例如log-error-2013-12-21.0.log -->
        <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
            <maxFileSize>2MB</maxFileSize>
        </timeBasedFileNamingAndTriggeringPolicy>
    </rollingPolicy>
    <!-- 追加方式记录日志 -->
    <append>true</append>
    <!-- 日志文件的格式 -->
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %logger Line:%-3L - %msg%n</pattern>
        <charset>utf-8</charset>
    </encoder>
    <!-- 此日志文件只记录info级别的 -->
    <filter class="ch.qos.logback.classic.filter.LevelFilter">
        <level>error</level>
        <onMatch>ACCEPT</onMatch>
        <onMismatch>DENY</onMismatch>
    </filter>
</appender>

<!-- 日志记录器，日期滚动记录 -->
<appender name="FILEWARN" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <!-- 正在记录的日志文件的路径及文件名 -->
    <file>${LOG_PATH}/${APPDIR}/log_warn.log</file>
    <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <!-- 归档的日志文件的路径，例如今天是2013-12-21日志，当前写的日志文件路径为file节点指定，可以将此文件与file指定文件路径设置为不同路径，从而将当
        而2013-12-21的日志文件在由fileNamePattern指定。%d{yyyy-MM-dd}指定日期格式，%i指定索引 -->
        <fileNamePattern>${LOG_PATH}/${APPDIR}/warn/log-warn-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
        <!-- 除按日志记录之外，还配置了日志文件不能超过2M，若超过2M，日志文件会以索引0开始，
        命名日志文件，例如log-error-2013-12-21.0.log -->
        <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
            <maxFileSize>2MB</maxFileSize>
        </timeBasedFileNamingAndTriggeringPolicy>
    </rollingPolicy>
    <!-- 追加方式记录日志 -->
    <append>true</append>
    <!-- 日志文件的格式 -->
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %logger Line:%-3L - %msg%n</pattern>
        <charset>utf-8</charset>
    </encoder>
    <!-- 此日志文件只记录info级别的 -->
    <filter class="ch.qos.logback.classic.filter.LevelFilter">
        <level>warn</level>
        <onMatch>ACCEPT</onMatch>
        <onMismatch>DENY</onMismatch>
    </filter>
</appender>

<!-- 日志记录器，日期滚动记录 -->
<appender name="FILEINFO" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <!-- 正在记录的日志文件的路径及文件名 -->
    <file>${LOG_PATH}/${APPDIR}/log_info.log</file>
    <!-- 日志记录器的滚动策略，按日期，按大小记录 -->
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <!-- 归档的日志文件的路径，例如今天是2013-12-21日志，当前写的日志文件路径为file节点指定，可以将此文件与file指定文件路径设置为不同路径，从而将当
```

```
        而2013-12-21的日志文件在由fileNamePattern指定。%d{yyyy-MM-dd}指定日期格式，%i指定索引 -->
        <fileNamePattern>${LOG_PATH}/${APPDIR}/info/log-info-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
        <!-- 除按日志记录之外，还配置了日志文件不能超过2M，若超过2M，日志文件会以索引0开始，
        命名日志文件，例如log-error-2013-12-21.0.log -->
        <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
            <maxFileSize>2MB</maxFileSize>
        </timeBasedFileNamingAndTriggeringPolicy>
    </rollingPolicy>
    <!-- 追加方式记录日志 -->
    <append>true</append>
    <!-- 日志文件的格式 -->
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %logger Line:%-3L - %msg%n</pattern>
        <charset>utf-8</charset>
    </encoder>
    <!-- 此日志文件只记录info级别的 -->
    <filter class="ch.qos.logback.classic.filter.LevelFilter">
        <level>info</level>
        <onMatch>ACCEPT</onMatch>
        <onMismatch>DENY</onMismatch>
    </filter>
</appender>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!--encoder 默认配置为PatternLayoutEncoder-->
    <encoder>
        <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %logger Line:%-3L - %msg%n</pattern>
        <charset>utf-8</charset>
    </encoder>
    <!--此日志appender是为开发使用，只配置最底级别，控制台输出的日志级别是大于或等于此级别的日志信息-->
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
        <level>debug</level>
    </filter>
</appender>

<logger name="org.springframework" level="INFO" />
<logger name="org.apache.ibatis" level="DEBUG" />
<logger name="com.iscas.biz" level="DEBUG" />
<logger name="sun.rmi.transport.tcp" level="INFO" />
<logger name="org.apache.http" level="INFO" />
<logger name="com.sun.mail.smtp" level="INFO" />
<logger name="javax.management" level="INFO" />
<logger name="sun.rmi" level="INFO" />
<logger name="de.codecentric" level="INFO" />
<logger name="druid.sql.Connection" level="INFO" />
<logger name="druid.sql.Statement" level="INFO" />
<logger name="druid.sql.ResultSet" level="INFO" />
<logger name="org.hibernate.validator" level="INFO" />
<logger name="org.mybatis.spring.mapper" level="INFO" />
<logger name="org.xnio.nio" level="INFO" />
<logger name="springfox.documentation" level="INFO" />
<logger name="springfox.bean" level="INFO" />
<logger name="com.baomidou.mybatisplus.core" level="INFO" />
<logger name="io.undertow" level="INFO" />
<logger name="io.micrometer.core" level="INFO" />
<logger name="com.baomidou.mybatisplus.extension.spring" level="INFO" />
<logger name="Validator" level="INFO" />
<logger name="io.lettuce" level="INFO" />
<logger name="io.netty" level="INFO" />
<logger name="org.springframework.boot.actuate.redis.RedisReactiveHealthIndicator" level="ERROR" />
<logger name="org.neo4j.driver" level="INFO" />
<logger name="root" level="DEBUG" />

<!-- 生产环境下，将此级别配置为适合的级别，以免日志文件太多或影响程序性能 -->
<root level="DEBUG">
    <appender-ref ref="FILEERROR" />
    <appender-ref ref="FILEWARN" />
    <appender-ref ref="FILEINFO" />
    <!-- 生产环境将请stdout去掉 -->
    <appender-ref ref="STDOUT" />
</root>
```

undertow配置

undertow配置

1、配置文件：application.properties
2、配置项：

```
##################undertow服务配置#################
#undertow服务设置
#因为是NIO设置为跟CPU核心数差不多就行
server.undertow.threads.io=16

#阻塞任务线程池, 当执行类似servlet请求阻塞IO操作, undertow会从这个线程池中取得线程
#它的值设置取决于系统线程执行任务的阻塞系数，默认值是IO线程数*8
server.undertow.threads.worker=256

#以下的配置会影响buffer,这些buffer会用于服务器连接的IO操作,有点类似netty的池化内存管理
#每块buffer的空间大小,越小的空间被利用越充分，不要设置太大，以免影响其他应用，合适即可
server.undertow.buffer-size=1024

#是否分配的直接内存(NIO直接分配的堆外内存)
server.undertow.direct-buffers=true
```

各配置项说明

各配置项说明

```
###########项目名称及端口配置####################
#应用上下文路径及项目路径
server.servlet.context-path=/quick-frame
#开启http2协议支持
server.http2.enabled=true
#https/http端口,如果未启动https，就是http端口，
server.port=17901

##########HTTPS配置#############################
#http转https端口,启用时改为true
server.ssl.enabled=false
server.http.port=17902
#证书生成方法参照项目目录下https证书生成.txt
server.ssl.key-store=classpath:httpsKey.p12
server.ssl.key-alias=undertowhttps
server.ssl.key-store-password=123456

##########指定active 激活一个或者多个配置文件####
spring.profiles.active=dev

##########jackson配置############################
spring.jackson.serialization.indent_output=true
#HTTP消息转换的首选JSON映射器
#spring.http.converters.preferred-json-mapper=jackson
spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
#spring.jackson.joda-date-time-format=yyyy-MM-dd HH:mm:ss
spring.jackson.time-zone=GMT+8

##########静态资源配置############################
spring.mvc.static-path-pattern=/**
spring.web.resources.static-locations=classpath:/META-INF/resources/,classpath:/resources/,\
classpath:/static/,classpath:/public/,file:${upload-path}

##########文件大小限制############################
spring.servlet.multipart.enabled=true
spring.servlet.multipart.max-file-size=10485760000
spring.servlet.multipart.max-request-size=52428800000

##########mybatis-plus配置########################
#数据库ID自增 AUTO(0),该类型为未设置主键类型 NONE(1),用户输入ID,该类型可以通过自己注册自动填充插件进行填充 INPUT(2),
#全局唯一ID (idWorker)ID_WORKER(3),全局唯一ID (UUID)UUID(4),字符串全局唯一ID (idWorker 的字符串表示)ID_WORKER_STR(5);
mybatis-plus.global-config.db-config.id-type=AUTO
#逻辑已删除值(默认为 1)
mybatis-plus.global-config.db-config.logic-delete-value=1
#逻辑未删除值(默认为 0)
mybatis-plus.global-config.db-config.logic-not-delete-value=0
#配置枚举 支持通配符 * 或者；分割
mybatis-plus.type-enums-package=com.iscas.biz.mp.test.model.enums
mybatis-plus.configuration.default-enum-type-handler=org.apache.ibatis.type.EnumOrdinalTypeHandler
####mybatis-plus 代码生成器配置####
mp.parent.path=biz\src\main\java
#这里最好用一个临时包作为生成目录，防止覆盖已有内容
mp.parent.package.name=com.iscas.biz.tmp
####mybatis pagehelper配置(使用mybatis-plus后可以废弃掉了，留着它是为了兼容以前的东西)####
pagehelper.helperDialect=mysql
pagehelper.reasonable=true
pagehelper.supportMethodsArguments=true
pagehelper.params=count=countSql
pagehelper.returnPageInfo=check
#配置驼峰属性自动映射
#mybatis.configuration.map-underscore-to-camel-case=true
```

```
##########跨域过滤器相关配置，其他配置参见README###
#不通过跨域过滤器的URL配置
cros.ignoreUrls[0]=/webSocketServer/*
cros.ignoreUrls[1]=/webSsh/*
#跨域允许的前端域名
cros.origin=*

##########限流配置#################################
#每秒产生令牌数目，不配置默认20个
rate.limiter.permitsPerSecond=80
#获取令牌最大等待时间毫秒，不配置默认500ms
rate.limiter.maxWait=500
#配置静态资源路径，防止被过滤器过滤
rate.limiter.staticUrl[0]=/api/**
rate.limiter.staticUrl[1]=/loginTest/**
rate.limiter.staticUrl[2]=.js
rate.limiter.staticUrl[3]=/webjars/
rate.limiter.staticUrl[4]=/swagger-resources/**
rate.limiter.staticUrl[5]=/webSocketServer/**
rate.limiter.staticUrl[6]=/webSocketTest/**
rate.limiter.staticUrl[7]=/online_setting.html
rate.limiter.staticUrl[8]=/druid2/**

##########logback.xml配置#########################
logging.config=classpath:logback.xml

##########redis缓存配置与caffeine有冲突，只能选其一##########

##########登录缓存时间配置##########
login.random.data.cache.time-to-live=600

##########caffeine缓存配置与redis缓存冲突####################
spring.cache.type=caffeine
spring.cache.caffeine.spec= maximumSize=1000,expireAfterWrite=600s
spring.cache.cache-names[0]=test
spring.cache.cache-names[1]=auth

##########redis缓存配置与caffeine缓存冲突####################
#spring.cache.type=redis
#spring.cache.redis.time-to-live=2000000
#spring.cache.cache-names[0]=test
#spring.cache.cache-names[1]=auth
#spring.redis.host=127.0.0.1
#spring.redis.port=6379
#spring.redis.timeout=1000000
#spring.redis.lettuce.pool.max-active=8
#spring.redis.lettuce.pool.max-idle=8
#spring.redis.lettuce.pool.min-idle=0
#spring.redis.lettuce.pool.max-wait=-1

##########token配置#############################################
#token过期时间(分钟)
token.expire=14440
#token保存在cookie的时间(默认与浏览器生命周期一致)
token.cookie.expire=-1

##########动态代理配置##########################################
spring.aop.auto=true
spring.aop.proxy-target-class=true

##########undertow服务配置#####################################
#undertow服务设置
#因为是NIO设置为跟CPU核心数差不多就行
server.undertow.threads.io=16

#阻塞任务线程池, 当执行类似servlet请求阻塞IO操作, undertow会从这个线程池中取得线程
#它的值设置取决于系统线程执行任务的阻塞系数，默认值是IO线程数*8
```

```
server.undertow.threads.worker=256

#以下的配置会影响buffer,这些buffer会用于服务器连接的IO操作,有点类似netty的池化内存管理
#每块buffer的空间大小,越小的空间被利用越充分，不要设置太大，以免影响其他应用，合适即可
server.undertow.buffer-size=1024

#是否分配的直接内存(NIO直接分配的堆外内存)
server.undertow.direct-buffers=true

##########endpoint端点配置###################################
#开放所有端点
#management.endpoints.base.biz.exposure.include=*
#开放某个端点
#management.endpoints.base.biz.exposure.include[0]=refresh
#management.endpoints.base.biz.exposure.include[1]=loggers

##########自定义表格相关#####################################
#表格定义对应表格
iscas.table.table-definition-table=xxtable_definition
#表格表头定义对应表格
iscas.table.header-definition-table=xxcolumn_definition
#表格统一主键
iscas.table.primary-key=id

##########虚拟文件上传路径###################################
upload-path=F:/tempx/

##########文件服务存储路径###################################
file.server.path=F:/fileserver/

##########请求和返回值的加解密,非HTTPS下可以使用，需要前端配合###
#true表示开启调试，不加密。（方便开发时测试）
rsa.encrypt.debug=false
#更换为自己的公钥
rsa.encrypt.publicKey=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCu13zNU/bODBZOM7veQxDde9kn\
wDljH7D6Wp2Ab5uJdeCwt2/9DDzmyk8CyLsMrNSTSvN7VdpGZMJWOHwu06hVv9uI\
KvCB9x5EyYpDx+2z4cqg2CNiAJIEcvx4VV9AT+/6TGQvujMX5wgPqlYnO0as6Wch\
MVUbwmKtyLcSoFKhGQIDAQAB

#更换为自己的秘钥
rsa.encrypt.privateKey=MIICeAIBADANBgkqhkiG9w0BAQEFAASCAmIwggJeAgEAAoGBAK7XfM1T9s4MFk4z\
u95DEN172SfAOWMfsPpanYBvm4l14LC3b/0MPObKTwLIuwys1JNK83tV2kZkwlY4\
fC7TqFW/24gq8IH3HkTJikPH7bPhyqDYI2IAkgRy/HhVX0BP7/pMZC+6MxfnCA+q\
Vic7RqzpZyExVRvCYq3ItxKgUqEZAgMBAAECgYEAmOztwglQuwQjdl8VkUADvlWX\
CIe/Kg4ME82yOSZIWoP59Vdc4m4TzqFhHkRIJrv8aqCB+nyLIEr9F8DsDWEft/4/\
BgeTGIC2H6rCTEUOq3kI59Zh9U2adW6d778QPmH8OEfa+/UF4uhIau5oHCPqE1sW\
xRZqGzk3J2xm2SiNp4kCQQDeNKC2SjaRA2ek9NXoNYdsqXj7L+67FO3pbvFI36Zo\
n+NSY0c9qUqTKT6S9ABaclNvDSr4gByfkfGpmKPc24hnAkEAyW7KxbhmOz8U1sJI\
r6bzTZFov5DMOies9xkCDAHZnU79a/qoFjSAdUUneBFQmXiF3WFCKI2j08P32hB7\
nESafwJBAML55vNNV2gTVsSV2YSyQ6yDYJb4TkB2cvRb1vic1oYDlhJa7s4aWEec\
7z0/QXgbzT4qjcQTlgomo6ivOUm4kIOCQQDEIr2NdGdJ4UncQ0Nsx9pi1MxPUekP\
xNMZIa4Ou+t/jKzmKm7LpRfN290mYHeyIJ89LQOfR15xeau2himtnILPAkBUyZPv\
SglOUk8WK/ozTy0RsQyqvEt2TJ1CnqONp20r26RhyOxiTNYl4dVlOMt6i5szzwsu\
uEcvVHYzAesj7OdQ

##########datasong相关配置###################################
Global.DbServer=http://172.16.10.180:15680
Global.DbName=portrait2

#datasong-client-plus 模式对应service包名，多个用数组形式一行一行配置
datasong.client.plus.packages[0]=com.iscas.base.biz.test.datasongplus.service
datasong.client.plus.packages[1]=com.iscas.base.biz.test.datasongplus.service2

##########spring-boot-admin配置###############################
spring.boot.admin.client.url=http://192.168.100.88:8769
spring.application.name=demo-app
spring.boot.admin.client.instance.service-base-url=http://192.168.100.88:7901
```

```
#management.endpoints.web.base-path=/demo
management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=ALWAYS

##########邮件配置##################################################
#字符集
spring.mail.default-encoding=UTF-8
#电子邮件地址
spring.mail.host=smtp.qq.com
#端口
spring.mail.port=25
#授权密码(不是登陆密码)
spring.mail.password=lwepzfpllcnvbjbc
#邮箱账号名
spring.mail.username=461402005@qq.com
#SSL 加密工厂
spring.mail.properties.mail.smtp.socketFactoryClass=javax.net.ssl.SSLSocketFactory
#开启debug模式
spring.mail.properties.mail.debug=false

##########kaptcha验证码相关配置###################################
kaptcha.enabled=true

##########websocket-使用rabbitmq相关配置#######################
#是否使用spring的websocket stomp注册
ws.stomp.register = true
#对应自己rabbitmq里的虚拟host
rabbitmq.virtual-host=/
rabbitmq.relay-host=192.168.100.88
rabbitmq.user=guest
rabbitmq.password=guest
rabbitmq.heartbeatSendInterval=5000
rabbitmq.heartbeatReceiveInterval=5000
#stomp协议的端口
rabbitmq.stomp.port=61613
#amqp协议的端口
rabbitmq.amqp.port=5672

##########NEO4J#####################################################
#spring.data.neo4j.uri=bolt://localhost:7687
#spring.data.neo4j.username=neo4j
#spring.data.neo4j.password=neo4j
spring.neo4j.uri=bolt://localhost:7687
spring.neo4j.authentication.username=neo4j
spring.neo4j.authentication.password=123456
spring.neo4j.connection-timeout=10s
spring.neo4j.pool.max-connection-pool-size=100
spring.neo4j.pool.max-connection-lifetime=1h
```

前后台交互协议

前后台交互协议简介

前后端定义交互协议，是约定大于配置的一种模式体现，前后端根据固定的交互协议，可封装通用的组件，实现更高效的开发

目前已实现的交互协议包括：返回协议、表格协议、树协议、下拉列表协议、图标协议

所有交互协议的实体定义都在依赖`com.iscas:templet`中

返回协议

所有请求统一返回格式实体：ResponseEntity

```
@Data
@ToString(callSuper = true)
public class ResponseEntity<T> implements Serializable {

    /**
     * 状态信息
     */
    protected String message;

    /**
     * 服务器内部错误描述
     */
    protected String desc;

    /**
     * 返回值
     */
    protected T value;

    /**
     * 访问URL
     */
    protected String requestURL;

    protected long tookInMillis;

    protected int total;

    public ResponseEntity(Integer status, String message) {
        super();
        this.message = message;
    }

    public ResponseEntity() {
        super();
        this.message = "操作成功";
    }
    public ResponseEntity(String message){
        super();
        this.message = message;
    }
}
```

表格协议

表格协议后端封装组件使用demo在 `com.iscas.biz.samples.interractionProtocol.controller.TableDefinitionControllerTest` 类中

需要导入的元数据表在 `com.iscas.biz.samples.interractionProtocol` 包中

表格协议说明如下：

表格协议包括返回协议、表头定义、查询定义等

- 表格返回协议实体在返回时是统一返回格式实体的value属性，其定义如下：

```
@Data
@ToString(callSuper = true)
public class TableResponseData<List> implements Serializable{
    /*返回总条目*/
    protected Long rows;
    /*返回的具体数据，是个集合*/
    private List data;

}
```

- 表头协议在返回时是统一返回格式实体的value属性

1. 表头总体定义TableHeaderResponseData

```
public class TableHeaderResponseData implements Serializable {
    /*表头列信息*/
    protected List<TableField> cols;
    /*表的一些设置信息*/
    protected TableSetting setting;
}
```

2. 表头属性TableField

```
@Data
@ToString(callSuper = true)
@Accessors(chain = true)
public class TableField implements Serializable{
    /*表字段名称*/
    protected String field;
    /*表字段显示名称*/
    protected String header;
    /*是否可编辑，默认不可编辑*/
    protected boolean editable = true;
    /*此列是否支持排序，默认不支持*/
    protected boolean sortable = false;
    /*此列的类型*/
    protected TableFieldType type = TableFieldType.text;
    /*如果是下拉列表,返回的下拉列表信息*/
    protected List<ComboboxData> option ;
    /**是否查询*/
    protected boolean search = false;
    /*查询方式*/
    protected TableSearchType searchType = TableSearchType.exact;
    /*是不是link*/
    protected boolean link = false;

    /**是不是可以新增*/
    protected boolean addable = true;

    /**是不是隐藏*/
    protected boolean hidden = false;

    /**嵌套类型，super 名*/
    protected String parent;

    /*校验规则*/
```

```
    protected Rule rule;

    /**搜索方式*/
    protected String searchWay = null;

    /**
     * 如果是下拉列表,下拉列表的URL
     * */
    protected String selectUrl = null;
}
```

3. 表格全局属性定义TableSetting

```
@Data
@ToString(callSuper = true)
@Accessors(chain = true)
public class TableSetting implements Serializable{

    /**否显示复选框*/
    protected Boolean checkbox = false;

    /**
     * 表前说明
     * 用不到，会在未来版本删除
     * */
    @Deprecated
    protected String frontInfo;

    /**
     *表后说明
     * 用不到，不i在未来本本删除
     * */
    @Deprecated
    protected String backInfo;

    /**
     * 表的标题*/
    protected String title;

    /**表的显示形式*/
    protected TableViewType viewType = TableViewType.multi;

    /**单元格内可不可编辑*/
    protected boolean cellEditable = false;

    /**
     * 数据列自生成列说明
     * */
    protected List<ButtonSetting> buttonSetting;
}
```

- 表格查询是前端请求后端表格数据时的结构

TableSearchRequest:

```
@Data
@ToString(callSuper = true)
public class TableSearchRequest<T> implements Serializable{
    /*当前页码，默认为1*/
    protected Integer pageNumber = 1;
    /*每页显示条目，默认为10*/
    protected Integer pageSize = Integer.MAX_VALUE;
    /*排序的列*/
    protected String sortField;
    /*升序或者降序*/
    protected TableSortType sortOrder = TableSortType.asc;
    /*查询条件(扩展用)*/
    protected T filter;
```

```
    /*查询方式，为了与原有方式兼容，扩展一个查询方式*/
    protected Map<String,TableSearchType> searchType;

    /**下拉列表查询条件*/
    protected Map<String, List> optionsFilter;
}
```

TableGeneralSearchFilter:

```
@Data
@ToString(callSuper = true)
public class TableGeneralSearchFilter implements Serializable{
    protected Map<String,List> searchFilter;

}
```

树协议

树协议的demo在 `com.iscas.biz.samples.interractionProtocol.controller.TreeControllerTest` 类中

树协议说明如下：

```
@Data
@ToString(callSuper = true)
public class TreeResponseData<T> implements Serializable ,Cloneable{
    /*显示名称*/
    protected String label;
    /*对应ID*/
    protected Object id;
    /*是否展开*/
    protected Boolean expanded = false;
    /*是否可选*/
    protected Boolean selectable = true;
    /*是否选中*/
    protected Boolean selected = false;
    /*子节点*/
    protected List<TreeResponseData<T>> children;
    /**前台Path,不是必用*/
    protected String path;

    /*当前节点对应的值(实体对象值)*/
    protected T data;
}
```

下拉列表协议

下拉列表协议后端封装组件使用demo在 `com.iscas.biz.samples.interractionProtocol.controller.ComboboxControllerTest` 类中

协议说明如下：

```
@Data
@EqualsAndHashCode(callSuper = false)
@ToString(callSuper = true)
public class ComboboxData<T> implements Serializable {
    /*label*/
    protected String label;
    /*value*/
    protected Object value;

    /** data*/
    protected T data;

    /**如果是树形下拉列表使用此属性*/
    protected Object children;
}
```

datasong-client-plus驱动

配置

- 修改application.properties

```
####################################datasong相关配置############################################
Global.DbServer=http://172.16.10.180:15680
Global.DbName=portrait2

#datasong-client-plus 模式对应service包名，多个用数组形式一行一行配置
datasong.client.plus.packages[0]=com.iscas.base.biz.test.datasongplus.service
datasong.client.plus.packages[1]=com.iscas.base.biz.test.datasongplus.service2
datasong.client.plus.packages[2]=com.iscas.biz.samples.datasongPlus.service
```

- 在启动类添加 `@EnableDatasongClientPlus` 注解

例子

datasong-client-plus的实现模仿spring-data-jpa,基本访问方式可参阅相关文档

demo在`com.iscas.biz.samples.datasongPlus`包内

swagger的使用

swagger

> Swagger 是一个规范和完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。 总体目标是使客户端和文件系统作为服务器以同样的速度来更新。文件的方法、参数和模型紧密集成到服务器端的代码，允许 API 来始终保持同步。Swagger 让部署管理和使用功能强大的 API 从未如此简单。

1. 后端编写接口

参考demo `com.iscas.biz.samples.swagger.SwaggerControllerTest`

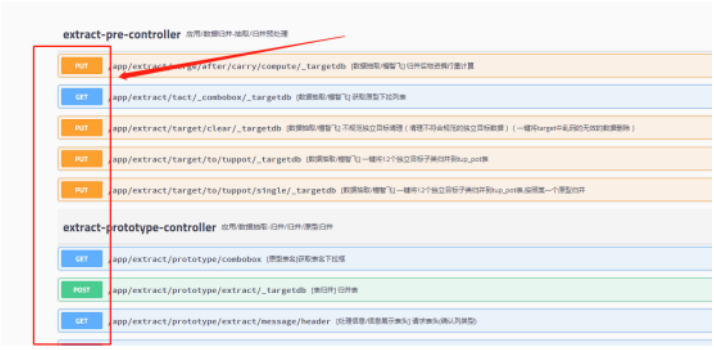
2. 前端读取swagger方式

- swagger文档的路径

Swagger文档的路径统一为http://<项目IP地址>:<项目端口>/<项目前缀>/swagger-ui/

- 请求方式

请求方式很直观，每个接口的最左边就是请求方式



- 请求参数-是否必填项

如果参数后面标着required，此参数必须传，否则可以不传

- 4、请求参数-参数类型

如果是简单数据类型，参数名字下面就标注了它的类型，如果是复杂数据类型，点击model会显示注释信息(前提是后端人员对实体编写了@ApiModel、@APiModelProperty注解)

防重复提交

防重复提交

在启动类配置 `@EnableNoRepeatSubmit(lockType = NoRepeatSubmitLockType.JVM)`

lockType可选有NONE、JVM、REDIS

NONE为不启用，JVM为使用JVM锁（默认实现）、REDIS使用REDIS锁，如果使用REDIS需实现 `INoRepeatSubmitRedisHandler` 接口

使用demo在 `com.iscas.biz.samples.norepeatSubmit` 包内

请求限流

全局限流

在biz模块启动类BizApp上添加注解即可开启全局限流：@EnableRateLimiter

接口限流

在controller层接口方法上添加注解@MethodRateLimit
参数说明：
1、permitsPerSecond 每秒支持的并发数
2、maxWait 最大等待时间，超时时反馈服务繁忙
例：@MethodRateLimit(permitsPerSecond = 500,maxWait = 500)
表示每秒支持500的并发，若请求过多时，请求等待时间超过500毫秒，就不再处理该请求
参考示例：RateLimiterControllerTest

缓存使用

caffeine

配置

1、打开配置文件application.properties中的配置项:
spring.cache.type=caffeine
spring.cache.caffeine.spec= maximumSize=1000,expireAfterWrite=600s
spring.cache.cache-names[0]=test
spring.cache.cache-names[1]=auth
2、注释掉redis缓存相关

使用

1、写入缓存
CaffCacheUtils.set(key, value)
其中key为String类型，value为Object，value支持基本类型、对象类型、Map、Collection等
2、读取缓存
T value = (T)CaffCacheUtils.get(key)
其中key为String类型，返回值为Object类型，需要强转为T，T为写入缓存值的类型
参考：CaffeineCacheService

redis

配置

1、打开配置文件application.properties中的配置项:
spring.cache.type=redis
spring.cache.redis.time-to-live=2000000
spring.cache.cache-names[0]=test
spring.cache.cache-names[1]=auth
spring.redis.host=127.0.0.1
spring.redis.port=6379
spring.redis.timeout=1000000
spring.redis.lettuce.pool.max-active=8
spring.redis.lettuce.pool.max-idle=8
spring.redis.lettuce.pool.min-idle=0
spring.redis.lettuce.pool.max-wait=-1
2、注释掉caffeine缓存相关

使用

参考：RedisCacheService

定时任务

定时任务使用

1、获取CronTaskRegister
在配置类中可通过@Autowired注入：
@Autowired
private CronTaskRegister cronTaskRegister;
在普通类中可通过SpringService获取:
CronTaskRegister cronTaskRegister = SpringService.getBean("cronTaskRegister");
2、构建要执行的任务信息
SchedulingRunnable task = new SchedulingRunnable("beanName", "methodName", "params");
表示要执行beanname类中的methodName方法,给方法传的参数为params
3、执行定时任务
cronTaskRegister.addCronTask(task, "cron");
其中cron为cron表达式，如：0/10 * * * * ?表示每隔10秒执行一次任务
参考示例：ScheduleControllerTest

XSS过滤器

XSS配置

在biz模块启动类BizApp上添加注解即可开启XSS过滤：@EnableXssConfig

统一日志记录

使用

1、在biz模块启动类BizApp上添加注解：@EnableLog
2、在方法上使用@LogRecord(type = LogType.TEST, desc = "这是测试")
LogType参考com.iscas.biz.config.log.LogType
3、处理日志的地方位于com.iscas.biz.config.log.SotreLogService，
在该类的store方法内些处理日志的逻辑
参考示例：LogInfoRecordControllerTest、SotreLogService等

统一异常处理

异常说明

1、MethodArgumentNotValidException 请求参数非法异常，状态码 400
2、ConstraintViolationException 请求参数校验异常，状态码 400
3、LoginException 登录异常，状态码 401
4、AuthorizationRuntimeException 鉴权异常，状态码：401
5、AuthorizationException 鉴权异常，状态码：403
6、BaseException 服务器内部Exception异常，状态码 500
7、BaseRuntimeException 服务器内部RuntimeException异常，状态码 500
8、Throwable 服务器内部Throwable异常，状态码 500
9、AssertRuntimeException 断言异常，状态码 500
10、MaxUploadSizeExceededException 文件上传大小超过限制异常，状态码 500

验证码

验证码使用

1、在配置文件中配置 kaptcha.enabled=true
2、获取验证码
参考VerificationCodeControllerTest，kaptcha方法
3、校验验证码
参考VerificationCodeControllerTest，verify方法

跨域处理

跨域处理

系统已配置全局跨域处理，如需自定义参数配置，可在配置文件修改：

```
#不通过跨域过滤器的URL配置
cros.ignoreUrls[0]=/webSocketServer/*
cros.ignoreUrls[1]=/webSsh/*
#跨域允许的前端域名
cros.origin=*
cros.credentials=true
cros.methods=POST,GET,PUT,DELETE,OPTIONS
cros.headers=Content-Type,Data-Type,Access-Control-Allow-Headers,Authorization,X-Requested-With,Accept,DataType,responseType
cros.maxage=3600
```

http客户端

http客户端(okhttp)

okhttp使用demo在`com.iscas.biz.samples.okhttp`包内

websocket相关

配置

在启动类上添加注解：@EnableWebsocketStomp(pushType = WsPushType.SIMPLE) 。其中pushType 请参考 WsPushType类，SIMPLE表示数据在内存处理的普通用法；RABBITMQ表示消息会推入到rabbitmq，需要安装并开启rabbitmq服务；SERVER_CLUSTER_USE_RABBIT表示消息会推入到rabbitmp集群，需要安装并开启rabbitmp集群 。

使用

参考：WebsocketControllerTest

连接用户设置

1、提供了默认方式
2、可以通过实现UserAccessor接口扩展，参考：UserAccessorTest

https配置

https配置

在配置文件内添加https配置

```
###################################HTTPS配置#################################################
#http转https端口,启用时改为true
server.ssl.enabled=true
server.http.port=7902
#证书生成方法参照项目目录下https证书生成.txt
server.ssl.key-store=classpath:httpsKey.p12
server.ssl.key-alias=undertowhttps
server.ssl.key-store-password=123456
```

如果需自定义证书，生成方式如下：

```
1.进入jdk的bin目录下
2.keytool -genkey -alias undertowhttps -keyalg RSA -keysize 2048  -keystore E:/httpsKey.p12 -validity 365
3.将证书从E盘拷贝到resources下
```

文件上传下载

文件上传下载

1、文件上传
示例类：com.iscas.biz.samples.FileLoadControllerTest
方法：public ResponseEntity uploadTest(@RequestParam("file") MultipartFile[] files)
2、文件下载
示例类：com.iscas.biz.samples.FileLoadControllerTest
方法：public void downloadTest(@RequestParam("path") String path)
3、文件上传大小限制配置
配置文件：application.properties
配置项：spring.servlet.multipart.max-file-size=10485760000 //单个文件大小限制
spring.servlet.multipart.max-request-size=52428800000 //单次请求的文件大小限制
4、文件批量上传
示例类：com.iscas.biz.demo.updownload.FileServerController
方法：public ResponseEntity uploadTest(@RequestParam("file") MultipartFile[] files)
配置文件：application.properties
配置项：spring.servlet.multipart.enabled=true //设置为true则支持多文件上传，否则不支持
5、多文件打压缩包下载
示例类：com.iscas.biz.demo.updownload.FileServerController
方法：public void downloadZipTest(@RequestParam("path") String path)
6、文件上传至静态资源目录配置
配置文件：application.properties
配置项：spring.web.resources.static-locations=classpath:/META-INF/resources/,classpath:/resources/,\
classpath:/static/,classpath:/public/,file:${upload-path}
upload-path=F:/tempx/

其他組件

过滤器

1、继承 OncePerRequestFilter
2、 重写doFilterInternal方法
3、注册过滤器，参考FilterConfiguration和FilterTest

拦截器

1、实现HandlerInterceptor
2、 重写preHandle、postHandle方法
3、注册过滤器，参考InterceptorConfiguration和InterceptorTest

监听器

1、实现ServletRequestListener
2、 重写requestDestroyed、requestInitialized方法
3、注册过滤器，参考ListenerConfiguration和ListenerTest

工具类说明

概述

工具类的说明已生成java doc规范的API文档

下面是对这个工具类的简要说明

common-tools

assertion(断言工具包)

assertion工具类说明

```
AssertArrayUtils ----------------------------------包括数组断言相关工具函数(36个)
AssertCollectionUtils-----------------------------包括集合断言相关工具函数(4个)
AssertMapUtils------------------------------------包括Map断言相关工具函数(4个)
AssertObjUtils------------------------------------包括对象断言相关工具函数(4个)
AssertStrUtils------------------------------------包括字符串断言相关工具函数(4个)
```

captcha(验证码工具包)

验证码工具类说明

```
CaptchaUtils----------------------------------------验证码工具类(2个)
```

code(二维码工具包)

二维码工具类说明

```
ZXingCode-------------------------------二维码相关工具函数(5个)
```

core(核心工具包)

arithmetic(计算工具包)

计算工具说明

```
FloatExactArithUtils------------------------------浮点数精确计算工具函数(5个)
MathUtils-----------------------------------------数学计算扩展函数(1个)
```

array(数组工具包)

数组工具说明

```
ArrayRaiseUtils----------数组增强操作工具函数(2个)
```

classloader(类加载器工具包)

类加载器工具包说明

```
ClassLoaderSwapper----------------------类加载器切换相关工具
JarLoader--------------------------------破坏双亲委派的类加载工具
```

collection(集合工具包)

集合工具说明

```
CollectionRaiseUtils----------集合扩展工具相关函数(2个)
MapRaiseUtils----------Map扩展工具相关函数()
```

conver(类型转换工具包)

类型转换说明

```
BytesConvertUtils----------------------------字节转换相关工具函数(6个)
```

date(时间扩展工具包)

时间扩展工具说明

```
DateRaiseUtils----------时间扩展相关工具函数(10个)
DateSafeUtils----------线程安全的时间转换工具函数(2个)
```

io(io相关工具包)

file(文件操作工具包)

文件操作工具说明

```
FileTypeUtils-------------------------文件类型相关工具函数(3个)
FileUtils-----------------------------文件相关工具函数(34个)
JarPathUtils--------------------------jar包路径工具函数(1个)
PropsUtils----------------------------properties文件工具函数(2个)
ThumbnailPicUtils---------------------缩略图工具函数(1个)
```

rar(rar工具包)

rar工具说明

```
RarUtils------------------------------rar工具函数(1个)
```

serial(序列化工具包)

序列化工具

```
SerializableUtils----------序列化相关工具函数(2个)
```

zip(zip工具包)

zip工具说明

```
GzipUtils----------------------------------gzip压缩相关工具函数(4个)
ZipUtils-----------------------------------zip压缩相关工具函数(4个)
```

object(对象工具包)

对象工具说明

```
ObjectUtils----------------------------------------对象工具函数(3个)
```

random(随机工具包)

随机工具说明

```
RandomBaseInfoUtils----------------------------------------------------------随机身份信息相关工具函数(6个)
RandomStringUtils------------------------------------------------------------随机字符串相关工具函数(1个)
```

reflect(反射工具包)

反射工具说明

```
ClassUtils----------class相关操作工具类(3个)
ReflectUtils----------反射相关操作工具类(17个)
```

security(安全工具包)

安全工具说明

```
AesUtils----------------------------AES加解密相关工具函数(9个)
Base64Utils-------------------------Base64编解码相关工具函数(46个)
EscapeUtils-------------------------字符编解码工具函数(2个)
MD5Utils----------------------------MD5加解密工具函数(5个)
RsaUtils----------------------------RSA加解密相关工具函数(5个)
URLCoderUtils-----------------------URL编解码相关工具函数(5个)
```

string(字符串工具包)

字符串工具说明

```
StringRaiseUtils----------------------------字符串扩展相关工具函数(3个)
StringRegExUtils----------------------------正则表达式工具函数(8个)
```

valid(校验工具包)

校验工具说明

```
IpValidUtils-------------------------IP校验相关工具函数(1个)
```

exception(异常工具包)

异常处理说明

```
ExceptionUtils----------------------------------异常处理相关工具函数(1个)
```

gis(地理信息工具包)

地理信息工具说明

```
LatLonUtils----------------------------经纬度计算相关工具函数(1个)
```

ip2region(ip地域工具包)

IP地域工具说明

```
Ip2RegionUtils-------------------------IP地域相关工具函数(1个)
```

jscss(jscss工具包)

jscss工具说明

```
JsCssCompressUtils----------JSCSS压缩相关工具函数(4个)
```

office(office工具包)

office工具说明

```
ExcelUtils--------------------------------excel相关工具函数(15个)
Template2DocUtils------------------------使用freemarker将模板文件转为Word(4个)
Template2DocxUtils-----------------------使用freemarker将模板文件转为docx格式的Word(4个)
```

pagination(分页工具包)

分页工具说明

```
MemoryPageUtils----------------------------------内存分页相关工具函数()
```

pinyin(拼音工具包)

拼音工具说明

```
PinyinUtils----------------------------------拼音相关工具函数(3个)
```

url(url工具包)

url工具说明

```
URLUtils--------------------------URL解析工具函数(8个)
```

common-web-tools

cookie(cookie工具包)

cookie工具包说明

```
CookieUtils----------cookie相关工具函数(2个)
```

file(文件工具包)

文件工具说明

```
FileDownloadUtils------------------------------文件下载相关工具函数(15个)
```

json(json工具包)

json工具说明

```
JsonUtils----------JSON转换相关工具函数(12个)
```