SUMK 框架入门

快速搭建 web 应用

1. 创建 maven 工程，并将工程的 jdk 版本设为 8

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.1</version>
            <inherited>true</inherited>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
    </plugins>
</build>
```

2. 引入 sumk.jar、sumk-log.jar

```
    <dependency>
        <groupId>com.github.youtongluan</groupId>
        <artifactId>sumk-log</artifactId>
        <version>2.10.2</version>
    </dependency>
    <dependency>
        <groupId>com.github.youtongluan</groupId>
        <artifactId>sumk</artifactId>
        <version>2.10.2</version>
    </dependency>
```

3. 启动程序，关键代码是 SumkServer.start()或 SumkServer.main(args)，其它代码都是可选的

4. 配置信息，所有的配置文件都在 app.properties 文件中

```
#这个文件是UTF-8编码
sumk.ioc=org.test
sumk.http.port=8081

```

完成以上四步，就搭建了一个 web 应用。
参见 https://github.com/youtongluan/sumk-server-demo/tree/master/web-server

编写第一个 web 接口

```
5
6  @Bean
7  public class DemoAction {
8
9      @Web
10     public String info(String name, Integer age) {
11         return "名字:"+name+",年龄:"+age;
12     }
13 }
14
```

在浏览器输入后面的网址，就可以访问了。http://localhost:8081/rest/info?data={“age”:23}

因为 name 没有声明是必传参数，所以上面的请求参数里可以没有 name 字段。如果要将 name 设为必传，需要用上 @Param 注解，然后访问地址要改成这样：http://localhost:8081/rest/info?data={"name":"张三","age":23}

为了安全性，客户端推荐使用 POST 请求，为了演示方便，这里都是用 get 请求

使用数据库

1. 在 app.properties 中添加数据库配置

```
5 s.db.sumk.1.type=wr
6 s.db.sumk.1.url=jdbc:mysql://127.0.0.1:3306/sumk?characterEncoding=utf-8&useOldAliasMetadataBehavior
7 s.db.sumk.1.username=root
8 s.db.sumk.1.password=root
```

2. 创建表的 pojo 对象。
   a) @SoftDelete 表示软删除，如果使用物理删除，就可以去掉这个注解。
   b) lastUpdate 对应的数据库字段名为 last_update，如果不是这个名字，要通过@Column 来指定。

```
11 @Table
12 @SoftDelete(value = "enable", columnType = Byte.class)
13 public class Student {
14
15     @Column(columnType = ColumnType.ID_BOTH)
16     private Long id;
17     private String name;
18     private Integer age;
19     private Date lastUpdate;
20
```

3. 在 mysql 上创建名字为 sumk 的数据库，数据库编码 utf-8。建表可以通过语句  db.sql ，也可以通过 CodeToolTest. generateDBTable()来自动创建表格。
4. 使用数据库，比如往数据库写入一条记录

```
// http://localhost:8081/rest/insert?data={"name":"测试","age":23}
@Web
@Box
public Student insert(String name,Integer age) {
    Student user = new Student();
    user.setAge(age);
    user.setName(name);
    DB.insert(user).execute();//user没有指定id，系统会智能检测id并生成。
    return user;
}
```

在浏览器输入下面地址，就可以插入一条记录了，返回值是记录的 id。
http://localhost:8081/rest/insert?data={"name":"测试","age":23}

5. DB 操作的文档参见这里：https://my.oschina.net/u/819657/blog/3008795
6. DB 操作默认为 mysql，如果使用 progressSQL 等其它数据库，请引入他们的驱动包，并修改 s.db.sumk.1.driverClassName 指定驱动类

使用 SDB

Sumk 的 orm 虽然提供了缓存、事件等功能，但它只支持单表操作。如果需要做关联操作，推荐使用 SDB 进行操作。SDB 类似于 mybatis，相比于 mybatis，SDB 的功能要少很多，但是它更轻，占资源更少，并且无缝对接 sumk-log。此外 SDB 还支持 sql 文件的动态变更并且支持远程 sql 文件（需要实现它的适配器）。SDB 使用方法如下：

1. 在 resources 底下创建 sql 文件夹，然后在里面存放 xml 文件。

2. 编写 xml 文件

```
<sql id="select">
    select id,name,age,last_update as lastUpdate from student
    <items open="where" separator="AND">
        <if test="id">id = #{id}</if>
        <if test="name" falseby="empty">name= #{name}</if><!-- 空字符串会被过滤掉，不作为查询条件 -->
        <if test="age">age= #{age}</if>
        <if test="lastUpdate">last_update= #{lastUpdate}</if>
    </items>
</sql>
<sql id="selectByIds">
    select id,name,age,last_update as lastUpdate from student
    <if test="ids">
        where id in
        <foreach collection="ids" item="id" separator="," open="(" close=")">#{id}</foreach>
    </if>
</sql>
```

3. 通过 SDB 调用语句。SDB 只支持 map 参数，并且 map 中的 value 只能是 String、原始对象、集合。好在 sumk 提供了 S.beans 工具类做对象和 map 间的转换。

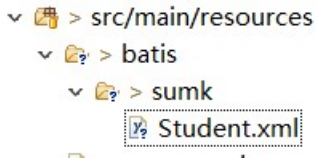
```
// http://localhost:8081/rest/selectByIds?data={"ids":[1,2]}
@Web
@Box
public List<Map<String, Object>> selectByIds(List<Long> ids){
    Map<String,Object> map=new HashMap<>();
    map.put("ids", ids);
    return SDB.list("student.selectByIds", map);
}
```

引入 mybatis

1、 在 pom.xml 中引入 mybatis.jar

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.1</version>
</dependency>
```

2、 在 resource 的 batis/sumk 底下放置 mybatis 的 sql 文件



3、 使用 SqlSessionHolder.session().selectOne(***)就可以使用原生的 mybatis 方法进行操作

配置

每个应用都需要配置，sumk 的配置默认放在 app.properties 文件中，属性支持**动态变更**，修改 app.properties 的值，一般 1 分钟内就能读到最新的值。这里除了存放框架所需要的配置，也可以存放开发者自定义的配置。框架配置以 sumk、s 这 2 个开头，开发者要避开这几个

前缀。获取配置方式如下图所示：

```
@Web
public String appInfo(@Param(required=true,cnName="名字") String name) {
    return AppInfo.get(name);
}
```

日志

详情访问 https://github.com/youtongluan/sumk-log，日志会额外打印用户 id 等，支持自定义日志，提供统一日志接口。
日志的几个有趣的应用：

1、 因为配置中心有定时更新的功能，所以日志的级别以及输出方式可以动态调整
2、 因为配置中心支持统一配置中心，所以可以通过统一配置统一修改各个应用的日志级别。如果统一配置中心有管理界面的话，就可以通过界面来调整日志属性
3、 可以通过自定义日志接口来自定义日志的记录行为，自定义方式：通过 LogAppenderFactory. registeAppender()将你的 LogAppender 注册进来。然后在配置里使用 s.log.名称=***来配置它
4、 通过 Plugin 和统一日志的结合来做性能采集和预警。在 Plugin 里编写指标采集，比如 JVM 指标等，然后通过上述的自定义日志方式，将该模块的日志传输到远程的分析程序里。

Web 权限校验

1、 用户登录，继承 AbstractLoginServlet 并且添加@Bean 注解，就会被当成用户登录接口（见 MyLoginServlet ），访问方式（返回值是密钥）：http://localhost:8081/login?username=admin&password=123456&code=9999
2、 需要登录后才能访问的接口，使用 @Web(requireLogin=**true**) 注解就行（见 UserAction.userInfo()接口），访问方式：http://localhost:8081/rest/userInfo
3、 如果需要更复杂的权限校验方式，只要继承 WebFilter 类，并用@Bean 注解就行了

单点登录限制

许多应用都有 pc 版、app 版、微信版等多种版本，这样用户可能在多个地方同时登录，默认情况下多个地方登录同时有效，如果想限制用户只能在一个地方登录，及登录了 app 之后，pc 上就要被强制退出。这时你只要加个配置就好了：sumk.http.session.single=1

Web 服务签名加密

Web 服务除了使用 https 外，sumk 框架内置了签名验证、参数加密、返回值加密等，签名、加密的算法都可以自定义，参见@Web 注解。以下地址可以找到例子 https://github.com/youtongluan/sumk/tree/master/src/test/java/org/test

Redis 以及分布式 session

随着业务量的增长，可能会引入 nginx 等负载均衡，这时候就需要分布式 session。对于 sumk 框架，只需要加入如下配置，就启用了分布式缓存。其中 session 表示启用分布式缓存，default 表示系统中引入了 redis 功能。

```
s.redis.default=127.0.0.1
#s.redis.default.db=2
#s.redis.default.password=XXX
s.redis.default.alise=session
```

redis 使用示例：

```
@Web
public long incrInRedis(String name) {
    return RedisPool.get("demo").incr(name);
}
```

测试网址：http://localhost:8081/rest/incrInRedis?data={"name":"test"}

统一配置中心

一旦应用部署了多个，就会需要配置中心。Sumk 内置了 http 和 zookeeper 两种协议的配置中心。Spring boot 的配置中心是 http 协议，sumk 能够兼容它。如果这 2 种不满足需求，开发者还可以使用自定义的配置管理，比如本地文件+配置中心的方式

使用示例：

```
UrlSystemConfig config=new UrlSystemConfig(new URL("http://localhost:8080/app.properties"));
SumkServer.start(config);
```

缓存系统

ORM 默认使用了表级缓存，不需要开发者关心。有三种方式可以关闭缓存。

- 配置中的 sumk.sql.fromCache 和 sumk.sql.toCache 是系统级配置
- @Table 中的 cacheType 是表级配置
- Insert、Select 等类的 fromCache()、ToCache()是方法级开关

读写分离

分布式 session 只是 web 端的扩展，随着业务发展，数据库也要扩展。读写分离是很常见的读写分离，它的好处是不破坏数据库事务。在 sumk 中，只要多配置几个数据源，示例中的数据源只配置了 s.db.sumk.1，我们可以加上 s.db.sumk.2 等。并且可以通过 weight 配置它们的权重。

多数据源

示例中的数据源只配置了 s.db.sumk.1，如果我们增加 s.db.test.1 的配置，就增加了一个 test 数据源，它跟 sumk 没有任何关系。使用的时候，通过@Box(value="test")就可以使用这个数据源。

监听表的读写

```
@Bean
public class TableListener implements SumkListener {

    @Override
    public Collection<String> acceptType() {
        return Arrays.asList(Const.LISTENER_DB_MODIFY);
    }

    @Override
    public void listen(Object ev) {
        if(!InsertEvent.class.isInstance(ev)){
            return;
        }
        InsertEvent event=(InsertEvent)ev;
        try {
            PojoMeta pm = PojoMetaHolder.getTableMeta(event.getTable());
            List<Map<String, Object>> list = event.getPojos();
            if (pm == null || list == null) {
                return;
            }
            for (Map<String, Object> map : list) {
                Log.get(this.getClass()).error("{}表插入了一条数据：{}",event.getTable(),map);
            }
```

实现 SumkListener 接口的 bean，然后 acceptType 方法返回 listener_db_modify

接口文档

1. 随机使用一串字符，比如 helloworld，然后用 md5 工具生成它的哈希值，示例代码：S.hash.digest ("helloworld".getBytes())，它等到值 fc5e038d38a57032085441e7fe7010b0
2. 在配置文件中增加：sumk.acts.md5=fc5e038d38a57032085441e7fe7010b0
3. 使用 http://localhost:8081/_sumk_acts?sign=helloworld&mode=http&pretty=1 可以查看 web 的接口出入参，开发者可以在这个基础上，制作文档生成工具。使用 http://localhost:8081/_sumk_acts?sign=helloworld&mode=rpc&pretty=1，可以查看 rpc 接口出入参情况。

4. 如果想要更详细的文档信息，在上面的地址要加上 full=1 参数

接口访问统计

1. 随机使用一串字符，比如 helloworld，然后用 md5 工具生成它的哈希值，示例代码：S.MD5.encrypt("helloworld".getBytes())，它等到值 fc5e038d38a57032085441e7fe7010b0
2. 在配置文件中增加：sumk.http.monitor=fc5e038d38a57032085441e7fe7010b0
3. 访问一些 web 接口，输入后面的地址，就可以查看地址的有效访问情况：http://localhost:8080/_sumk_monitor?sign=helloworld&statis=1

微服务的搭建

1. 创建一个 maven 工程，引入 sumk.jar 及其依赖包（推荐通过 sumk-log 来引入 sumk）。
2. 编写启动类，这个步骤也与 web 服务的搭建一样（实际上一个工程既可以提供 web 服务、也可以提供微服务）
3. 在 app.properties 中做如下配置

```
#这个文件是UTF-8编码
sumk.ioc=org.test
sumk.rpc.port=1021
sumk.zkurl=127.0.0.1:2181

```

这样就完成了微服务工程的搭建。因为微服务依赖于 zookeeper，所以要先启动 zookeeper 才能启动微服务程序。示例用的是本地 zookeeper。工程示例：https://github.com/youtongluan/sumk-server-demo/tree/master/rpc-server

编写第一个微服务

```
@Bean
public class DemoAction {

    @Soa
    public String echo(@Param(required=true,cnName="名字") String name) {
        return "你好 "+name;
    }
```

只要在类上加@Bean 注解，在方法上加@Soa 注解就可以了。前提是该类的包名在 sumk.ioc 配置的目录下

调用微服务

Sumk 的微服务调用支持同步、异步、回调三种方式。其中回调可以同时与前 2 种存在。增加使用的灵活性。支持数组、map、json 三种传参方式，相比之下数组使用起来最方便，推荐使用数组方式。

1、 在 web-server 工程里，添加下面的配置

```
#试验微服务的时候开启这个
sumk.zkurl=127.0.0.1:2181
sumk.rpc.client.start=1
```

2、 调用 echo 服务

```
@Web
public String echoFromRpc(String name) {
    String ret = Rpc.call("echo",name);
    return GsonUtil.fromJson(ret,String.class);
}
```

在浏览器里输入 http://localhost:8081/rest/echoFromRpc?data={"name":"游夏"} ，就可以调用 echoFromRpc 方法，然后通过它来调用微服务。

通过接口调用的微服务

1、服务端需要在实现类上加@SoaClass 注解。实现类如果没有继承接口，或继承了多个接口，就需要在被调用方建一个同包名、通类名的接口（从 class 变成了 interface）

```
@Bean
@SoaClass
public class HelloActionImpl implements HelloAction {

    @Override
    public String reply(String msg) {
        return "reply for "+msg;
    }

}
```

2、调用方配置要扫描接口所在的包名，该包及子包下的所有接口都被认为是 rpc 接口。也可以用 sumk.rpc.intfclient.interface 来指定具体的接口。如果要排除掉里面的某些接口，用 sumk.rpc.intfclient.exclude 来指定，它支持在头尾地方出现通配符*。

```
#接口方式调用rpc时，客户端要配置接口所在的包
sumk.rpc.intfclient.package=org.test.action
```

3、 发起 rpc 调用

```
// http://localhost:8081/rest/echo/by/intf
@Web("echo/by/intf")
public String echoByIntf() {
    /*
     * 不需要对返回值做json处理，如果返回的是泛型，要做如下处理（只要全局做一次就可以）：
     * 比如返回List<SumkDate>，就要将这个类型进行注册
     * JsonTypes.registe(new TypeToken<List<SumkDate>>(){}.getType());
     */
    return this.helloAction.reply("接口方式的微服务");
}
```

代码自动生成

1、在 maven 上引入 sumk-codetool, 引入方式如下：

```
<dependency>
    <groupId>com.github.youtongluan</groupId>
    <artifactId>sumk-codetool</artifactId>
    <version>1.0.1</version>
    <scope>test</scope>
</dependency>
```

2、 在配置文件中配置 sumk.code.output=D:\\output\\ ，值是生成文件的存放目录。

3、 执行 CodeTool.generateDao(Student.class); 就能生成 dao 代码，其中 Student 是经过 @Table 注解的 pojo 类

更详细的例子参见：https://github.com/youtongluan/sumk-codetool/tree/master/sumk-codetool-test

监听器

定义监听器。关键是@Bean、SumkListener、acceptType()

```java
@Bean
public class InfoListener implements SumkListener {

    /*
     * 使用EventBus将listen_info事件发布出去，就可以在这里监听到
     */
    @Override
    public Collection<String> acceptType() {
        return Arrays.asList(Constants.LISTEN_TYPE);
    }

    @Override
    public void listen(Object ev) {
        Log.get("listen").info("监听到了事件: {}",ev);
    }
```

发布事件

```java

    @Inject(Constants.LISTEN_TYPE)
    private EventBus bus;

    // http://localhost:8081/rest/info?data={"name":"张三","age":23}
    @Web
    public String info(@Param("名字") String name, Integer age) {
        bus.publish(name);
        return "名字:"+name+",年龄:"+age;
    }
```

常用配置

除数据库、redis、系统启动需要的参数外，其它参数一般不需要重启。修改后一般 30 秒内生效（2.8.2 以前是 1 分钟）。如果是 boolean 型配置，1 和 true 是等价的。部分常用配置，全角半角也是通用的，比如最常用的 sumk.log.level。有一些配置是 wild 表达式，它表示可以在表达式的前面或后面出现*，并且支持逗号隔开，比如 a*,*b*。

常用配置默认在 app.properties 文件里，key 以及 value 前后的空格都会被自动去除。

配置名	作用	默认自	是否需要重启
sumk.appId	应用的 AppId		是
sumk.ioc	IOC 扫描的包名，wild 表达式		是
sumk.test	如果设置了这个值，并且 web 接口通过 thisIsTest 传入了相同的值，该表示该请求处于测试模式。它不会真正修改数据库，并且 orm 也不会真正修改缓存。它所调用的微服务，以及微服务调用的微服务，都遵循这个约定	null（关闭测试模式）	否
sumk.ioc.exclude	wild 表达式，过滤掉代码中不用的 bean。如果多的话，还可以用 sumk.ioc.exclude.1，sumk.ioc.exclude.2 来分开指定		是
sumk.ioc.optional	配置哪些类如果加载失败可以忽略掉。比如 org.apache.*就表示 apache 的类如果加载失败就忽略掉，而不是系统启动失败	空。类加载失败就导致系统停止	是
sumk.log.level	日志级别	Info	否
s.log.day	格式为 path:/logs/app-#.log; module:*。也可简略为/logs/app-#.log		否
sumk.log.body.maxlength	日志体的最大长度	1500	否
sumk.log.maxLogNameLength	日志名称的最大长度	32	否
sumk.log.console	1 表示无论如何都打印控制台日志	0	否
sumk.valid.name.cn	@Param 参数校验不通过的时候，错误信息里，字段名称是否使用参数的中文名称	1	否
sumk.valid.name.raw	当 sumk.valid.name.cn=0 时才生效。@Param 参数校验不通过的时候，错误信息里，参数名称使用英文名称。如果禁用。会直接使用“参数”来替代具体的参数名字	1	否
sumk.bizexception.fullstack	是否打印 BizException 的堆栈信息	3.2 开始默认 true	否
http 服务配置	含有 jetty 的表示使用 jetty 作为服务器的时候才		

	支持		
sumk.http.port	http 服务的端口，大于 0 才有意义	-1	是
sumk.http.host	本机监听 http 服务所用的 ip。支持精确指定和 wild 表达式两种	0.0.0.0	是
sumk.webserver.root	Web 项目的项目名，旧版 key 为 sumk.jetty.web.root	/	是
sumk.http.servlet.rest	@Web 接口所属的 servlet	/rest/*	是
sumk.http.act.ingorecase	1 表示接口名不区分大小写	0	是
sumk.webserver.resource	Web 项目静态资源的目录。可以是文件夹的绝对路径，也可以是相对于 user.dir 的相对路径，也可以是 jar:file:XX.zip!/这种 jar 或 zip 文件		是
sumk.http.log.reqsize	http 日志中请求的最大长度	1000	否
sumk.http.log.respsize	http 日志中响应的最大长度	1000	否
sumk.http.log.warn.time	http 请求的处理超过这个时间，http 的日志级别变为 warn	3000	否
sumk.http.log.info.time	http 请求的处理超过这个时间，http 的日志级别变为 info	1000	否
org.eclipse.jetty.server.Request.maxFormContentSize	表单请求 body 的最大长度，这个是 jetty 自身的参数	200000	是
sumk.webserver.ssl.keyStore sumk.webserver.ssl.storePassword sumk.webserver.ssl.managerPassword sumk.webserver.ssl.alias	https 的相关配置		是
sumk.http.session.single	1 启用同账号的互踢功能	0	否
sumk.http.session.timeout	http 的 Session 过期时间，单位秒	1800	否
sumk.http.error.XXX	定制某个异常码的出错信息		否
sumk.webserver.disable	1 表示禁用内部的 jetty，需要外部的 tomcat 或其它 web 容器。这时需要在 web.xml 中添加: <listener> <listener-class>org.yx.main.SumkLoaderListener</listener-class> </listener>	0	是
sumk.http.multipart.location	文件上传时临时文件目录。默认在系统的临时文件里		是
sumk.http.multipart.maxFileSize	上传时单文件的最大大小	10M	是
sumk.http.multipart.maxRequestSize	上传时整个请求的最大大小，这个要大于单文件	50M	是
sumk.http.multipart.fileSizeThreshold	如果文件大小小于这个阀值，就使用内存代替临时文件	10K	是
sumk.http.name.sessionId	sessionId 所使用的 header 或 cookie 字段的名称	sid	是

sumk.http.plain.key	加密接口的免密 key，不设置表示不允许。假设免密 key 设置为 hello，调试的时候 url 传递_plainKey=hello，(2.x 的用 plainKey)就可以明文调用。因为这个值可以动态变更，所以不存在安全漏洞		否
sumk.http.fusing	接口熔断，多个接口用逗号隔开。2.8.3 以上支持前后出现*通配符。比如 student/*,*write		否
sumk.http.login.enable	设置为 1 后@Web 的 requireLogin 属性才有作用	0	是
sumk.http.method.default	@Web 注解和登陆接口默认支持的 http 请求类型	POST,GET	是
sumk.http.method.all	系统支持的所有 http 请求类型	POST,GET,DELETE,PUT,PATCH	否
sumk.http.interrorcode	异常时，是否将 http 的异常码改为 int 类型	False	否
微服务的配置			
sumk.rpc.port	微服务使用的端口，0 表示随机端口。小于 0 不启用微服务	-1	是
sumk.zkurl	微服务用到的 zk 的地址（全局设置）		是
sumk.rpc.zk.client	客户端使用的 zk 地址，不设置的话就用全局的		是
sumk.rpc.zk.server	服务端使用的 zk 地址，不设置的话就用全局的		是
sumk.rpc.client.start	是否初始化 rpc 客户端	0	是
sumk.rpc.server.register	如果为 0，就不注册 zookeeper。修改这个参数，可以使注册 zookeeper 用到的那些参数动态生效	1	否
sumk.rpc.appId.enable	如果设置为 0，就不会在微服务路径前面加上 appId	1	是
sumk.rpc.host	服务端注册到 zookeeper 所使用的本机 ip，双网卡情况下有可能需要手工指定		依赖 register
sumk.rpc.weight	当前微服务服务器的权重	100	依赖 register
sumk.rpc.zk.port	注册到 zookeeper 上使用到的端口号，一般不需要配置。在某些极端情况下，或者需要配置		依赖 register
sumk.rpc.log.warn.time	耗时大于这个时间的日志会变成 warn 级别	3000	否
sumk.rpc.log.info.time	耗时大于这个时间的日志会变成 info 级别	1000	否
sumk.rpc.call.timeout	Rpc 调用的超时时间，也可以在代码里为每个请求单独指定	30000	否
sumk.rpc.server.include	Wild 表达式，只有符合当前路径名条件的微服务实例才能被监听和调用。默认名格式为 appId@IP		
sumk.rpc.server.exclude	Wild 表达式，符合当前路径名条件的微服务实例不被监听和调用		
sumk.rpc.publish.[api 名称]	显式控制 api 接口是否发布到 zk 上，本配置的		依赖

	优先级比@Soa 中的 publish 属性高		register
sumk.rpc.detailError	Rpc 服务端日志展示详细堆栈信息	0	否
sumk.rpc.log.server.exception	记录 Rpc 服务端日志的时候，是否要另外再打印一条异常信息	0	否
sumk.rpc.printRawStackTrace	SoaException 的异常打印是否使用原始的 Exception 打印方式。该参数意义不太大	0	否
Redis 配置	*与 default 是等价的（但不能混合使用），表示默认 redis。如果是某个名称的实例，将*改为 redis 名支持所有 commons-pool2 的属性		
s.redis.*	默认 Redis 实例的地址，多个以逗号隔开，并且要设置 type		是
s.redis.*.maxAttempts	默认 redis 实例的最大尝试次数	3	是
s.redis.*.db	默认 redis 实例使用的 db 库	0	是
s.redis.*.alias	默认 redis 实例的别名，别名可以在 RedisPool.get()中使用		是
s.redis.*.password	默认 redis 实例的密码		是
s.redis.*.timeout	默认 redis 实例的超时时间		是
s.redis.*.type	Redis 的类型：0 普通、1 cluster、 2 sentinel	0	是
数据库配置	配置中的 sumk 是数据源名称，就是@Box 中的 value,1 是该数据源的第几个数据库实例。无论读库还是写库，都可以配置多个。支持所有 commons-pool2 的属性		
s.db.sumk.1.driverClassName	驱动类型	com.mysql.jdbc.Driver	是
s.db. sumk.1.url			是
s.db. sumk.1.username			是
s.db. sumk.1.password			是
s.db. sumk.1.weight	数据库的权重	1	是
s.db. sumk.1.maxTotal	数据库最大连接数	50	是
s.db. sumk.1.maxIdle	数据库最大空闲数	10	是
s.db. sumk.1.minIdle	数据库最小空闲数	5	是
s.db. sumk.1.type	数据库的读写类型。wr 是读写,read 是只读,write 是只写	wr	
sumk.db.password.encry	1 表示启用数据库密码的加密模式。使用项目中的“数据库密码加密.exe”对密码进行加密	0	是
sumk.db.fromCache	Select 是否使用缓存	1	否
sumk.db.toCache	DB 修改的数据，是否同步到缓存	1	否
sumk.db.select.toCache	查询出来的数据是否更新到缓存。在 sumk.db.toCache 为 true 时本参数才有意义	1	否
sumk.unionlog.sql.time	执行时间大于或等于这个时间的 sql 语句才会被记录到统一日志	0	否
s.alias.db.{name}	给数据源{name}取别名	空	是

sumk.db.select.max.limit	Select 对象里的 limit 默认值，也是最大值。ignoreMaxLimit(true)可以取消这个限制	10000	
sumk.db.select.max.offset	Select 对象里允许的 offset 最大值。ignoreMaxOffset(true)可以取消这个限制	10000	
sumk.db.readtype	@Box 里的默认读策略	ANY	

从 2.10.3 开始，sumk.jetty 开头的配置改名为 sumk.webserver

常用注解

数据库相关的注解，见文档《sumk-data 使用介绍.md》

注解	作用	说明
@Web	它注解的方法自动成为 web 接口	依赖于@Bean
@Upload	表示该 web 接口是上传接口 u，这时它的访问 url 格式为 upload/XX，而不是 rest/XX	依赖于@Web
@SumkServlet	用于定义原生的 servlet	依赖于@Bean
@SumkFilter	用于定义原生的 filter	依赖于@Bean
@Soa	它注解的方法自动成为 rpc 接口	依赖于@Bean
@SoaClass	它注解的类成为可以 rpc 调用的类。并且它的方法还可以用@Soa 注解，这时候@Soa 的 value 属性会被过滤掉	依赖于@Bean。sumk.rpc.intfserver.automatch 也能起到类似于@SoaClass 的效果，比如它的值为 io.abc.*，那么 io.abc.开头的所有类，都会被注册为 rpc 服务
@SoaClientConfig	仅对使用接口调用的 rpc 客户端有作用,注解作用在接口上面。 它用于定义额外的 soa 参数	依赖于 sumk.rpc.intfclient.package 或 sumk.rpc.intfclient.interface 配置，跟 sumk.rpc.intfclient.exclude 有关联
@Bean	用来定义 Bean，要注意的是，如果自定义了 Bean 的名称，那么 IOC.get(XX.class)未必能获取到它，要用 IOC.get(name,XX.class)才能获取到	类要有公有的无惨构造函数，并且该类要是 public 类型，并且不能是内部类、final 类、abstract 类
@ ConditionOnProperty	只有这个注解判定为 true 的类，它的@Bean 才有效。可以用逗号（并且）或者||（或者）分隔	
@Inject	注入	
@ Exclude	用来做排除	支持该注解的有：@SoaClient、pojo 字段
@ExcludeFromParams	用在 pojo 中，被注解的字段，不会接收客户端的数据。如果是 rpc 调用，该数据也不会发送给客户端	
@ExcludeFromResponse	注解的字段在 http 和 rpc 中不会返回	
@Param	用于参数校验或者文档注释	
@Priority	控制 IOC 初始化 Bean 对象的次序	

以上大部分注解都可以支持自定义，比如在 sumk.ioc 扫描的包里添加下面的类，就可以支持让 spring 的@Component 起到跟 sumk 的@Bean 一样的效果

```java
public class SpringComponent implements BootWatcher{

    @Override
    public List<Class<?>> publish(List<Class<?>> scanedClasses, Predicate<String> optional) throws Exception {
        SpecParsers.setBeanParser(clz -> {
            Component c= clz.getAnnotation(Component.class);
            if (c != null) {
                return new BeanSpec(c.value(), "", true);
            }
            return BuiltIn.BEAN_PARSER.apply(clz);
        });
        return null;
    }

}
```