金融中间件spider概要设计

本文档的目的是定义远程过程调用中间件的所有设计规约，应实现的特性，相关的术语及其含义，软件的运行环境和兼容性要求。

## 最新版本

最新版本已更新至2.0.0-RELEASE，具体版本release-note参见io-spider\com.ld.net.spider\src\main\java\version.txt。

## 术语

spider：远程过程调用中间件，简称spider。下文均以spider代表远程过程调用中间件。

服务号：也称为功能号，可互换。长度为固定的八位ASCII字符，建议由八位数字组成。

## 目标

金融中间件需要满足以下要求：

* 能够根据请求包里面的功能号、子系统号进行静态路由；
* 支持多租户。能够根据请求包里面的机构号、产品（系统）号、版本号进行动态路由；  
  saas架构的理想模式就是所有的业务实现均原生支持多租户。  
  如果做不到原生支持多租户，按照一个租户一个runtime，也需要动态能够设置某个runtime提供对哪个客户的支持，这肯定不能让运维动态更改配置文件，他只需要中心化指定即可。即使能做到，也需要支持动态能够设置某个app提供对哪个客户的支持，因为一个app仅能够提供对有限个客户的支持，且mysql也不可能做到一个库支持所有。反之，如果做不到原生支持多租户，则app必须支持多数据源切换以及动态管理，因为通常一个客户通常难以大到需要一个app去支持的规模。
* 支持服务器集群中的各节点负载均衡（根据模式）；
* 支持基于中心化管理的大型部署和独立管理的中小型部署；
* 支持自动重连到断开的服务器节点；
* 支持中心化剔除已删除的服务器节点；
* 支持中心化增加服务器节点；
* 支持中心化删除服务器集群；
* 支持中心化增加服务器集群；
* 一个可执行组件，支持任何时候节点（Work Node，WN）作为代理服务器（Node Broker，NB）(ANB:access node broker；CNB:core node broker）或处理服务器（Node Processor，NP）（PNP（产品NP），CNP（机构NP）），或从开发角度分为业务服务处理器（Business Node Processor，BSNP）、微服务处理器（MSNP,Micro Servicce Node Processor）；
* 基于TCP/IP协议，支持客户端语言无关；
* 支持连接亲和性（没有亲和性，就不是一个好的负载均衡实现）；
* 节点间技术上对等，即允许同时满足A是B的客户端，B是A的客户端，只要保证路由不形成死循环即可；
* 支持报文加密；
* 能够自动校验报文是否被篡改；
* 支持客户端认证功能；
* 支持报文压缩；
* 支持是否启用服务端功能；
* 能够中心化监控整个私募服务系统中所有中间件的运行状况；
* 支持基于Spring IOC注解方式的Java原生式远程服务调用；
* 最小化节点本地（集中化）配置；
* 支持多种序列化机制；
* 支持List泛型；
* 支持客户端异步请求与回调（主要用于定时任务和跑批场景）；
* 支持服务可靠执行（可保证请求收到后不会丢失且肯定会按照提交的顺序（这是个复杂的策略，决定了并发性）执行，后续版本会考虑支持回调；
* 灵活的路由，任何服务都可以发送到任何集群（暂不支持到具体的节点）。

## 协议与兼容性

spider使用java语言开发，使用Spring作为IoC容器，采用TCP/IP协议，在此基础上，结合金融交易系统的特性进行针对性和重点设计，以更加灵活和高效的满足金融交易系统多租户、高可用、分布式部署的要求。

spider默认采用JSON作为序列化机制，后续版本可能会考虑支持protobuf（java/c++/c#均有类库支持）。

为了最大化性能以及稳定性，spider基于Oracle JDK1.8进行编译并应避免使用deprecated特性。

为了尽可能的适应各环境以及互联网应用，spider应能至少运行于tomcat/jboss应用服务器或原生java下。

## 部署模式

任何时候，Spider可运行于中心化管理模式或独立管理模式之一。

中心化管理模式：中心化模式要求必须启用服务中心。对于有几十个运行节点的大规模部署而言，通常增加或者减少一个节点/拆分服务需要进行的配置文件数量会很多，通常位于新增节点上游的各节点都需要修改相应的路由和对应的服务器参数。采用中心化管理模式，只需要登录服务中心修改相关配置即可，节点的变化会自动推送给相应的上游节点。运行于中心化管理模式时，可在服务中心查看整个平台中所有节点的健康状态、各服务的TPS、响应时间等。

独立管理模式：独立管理模式无需启用服务中心。当节点数量较少，比如整个平台不超过十个节点时，采用独立模式通常会比中心化管理模式更简单。运行于独立管理模式时，可通过spider提供的restful api查看当前节点的运行状态。

## 服务标识

spider支持两种服务发布注解。

spider定义了两个自定义注解用于标识spider服务。

### 服务模块

@ServiceModule

public @interface ServiceModule {

  String subSystemId() **default** "0";

**int** broadcast() **default** 0; --设置该模块是否启用广播，0：不广播；1：广播但无需相应；2：广播并响应（1.0.5版本开始支持），只有具体的服务同时启用了，该服务才有效，用于NP需要主动发起广播给其他NP的时候

}

@ServiceModule只是一个类注解，具有该注解的接口将被认为是spider服务接口类，定义在该接口中是被标识为spider服务的必要条件。

如果希望某服务以广播模式执行，则必须在其所属的ServiceModule接口中设置broadcast = 1或2。

### 服务接口

@Service

@Retention(RetentionPolicy.RUNTIME)

@Target({ElementType.METHOD})

public @interface Service {

String serviceId(); --服务编号，12位ASCII字符，其中00000000-00000099为spider内部保留，00000100-00000199为服务中心保留

String desc(); --服务描述

int timeout() default 0; --超时时间，单位毫秒

**boolean** needLog() **default** **false**; --设置是否记录日志，1.0.3+

**int** broadcast() **default** **0**; --设置该请求是否广播，0：不广播；1：广播但无需相应；2：广播并响应（1.0.4版本开始支持），1.0.4+（广播的服务不支持抓包和拦截，不支持可信模式）

**boolean** his() **default** **false**; //是否历史查询功能

**boolean** batch() **default** **false**; //是否批处理功能

**short** bizUserType() **default** 1; //使用该服务的目标用户, 1：C端用户; 2: B端用户; 用于潜在的根据业务模式优化，暂未使用

}

@Service是一个方法注解，其包含三个属性，分别用于设置spider服务号，spider服务描述，spider服务超时时间。其中超时时间可选，默认为spider.xml中设置的超时时间，如果spider.xml中未定义，则默认为300秒。

只有定义在由@ServiceModule注解的接口中由@Service注解的方法才会被标识为spider服务，正确发布后，该接口可用于提供服务或者代理调用远程服务。

对于broadcast=2的服务，其返回值必须包装在io.spider.pojo.BroadcastResult的data属性中，否则会出现运行时类型转换失败，具体见该类的javadoc说明。

## 环境变量

spider有一些环境变量用于控制相关选项，目前有下列可设置的环境变量，如下所示：

* SPIDER\_LOG，默认/tmp/spider/stat/${nodeName}
* SPIDER\_HOME，默认/usr/local/spider/${nodeName}
* SPIDER\_CONFIG，指定spider.xml启动文件，默认classpath:spider.xml，见配置文件一节。

## 配置文件

spider有一个配置文件spider.xml，为xml格式，spider.xml采用DTD进行管理，用于管理spider的所有特性、路由、高可用等。

配置文件支持三种不同的方式进行指定：

1. 通过环境变量指定。SPIDER\_CONFIG环境变量指定spider启动文件所在的位置。
2. 通过java系统属性执行。java系统属性spider.config指定spider启动文件所在的位置。
3. 从classpath获取。该配置文件需存放在classpath\*:目录下，spider中间件启动时会自动从classpath\*下搜索第一个找到的spider.xml文件并将其作为spider的配置文件进行初始化，由于该配置文件通常需要被修改，所以一般不放置在jar中。

三者的优先级为先从环境变量获取，如果能获取到则使用SPIDER\_CONFIG环境变量指定的启动文件。如果为空，则从java系统属性获取。如果还是获取不到，则从默认的classpath\*目录下获取。如果三个地方都获取不到，则启动失败。

spider的配置文件结构如下：

<?xml version="1.0" encoding="UTF-8"?>

<spider>

<nodeName value="service" cloud=”true” role=”prod” serviceCenter=”0.0.0.0:7070” appVersion=”” serviceDefineType=”ld” charset=”UTF-8” dev=”true” slowLongTime=”3000” dumpStat=”true” tcpdump=”false” detectInterval=”15000” needSFtp=”true” SFtpPort=”” SFtpUsername=”” SFtpPassword=”base64加密密码”

*supportPlainParams*=”true” noLoggingList=”sysL.1.1,sysL.1.2” loggingList=”” enableEpoll=”yes ” suppressErrorNoList=”” threadAffinity=”false”/>

<plugins [pipeline=”customPluginName1;customPluginName2;spider.beforeFilter; spider.localService;spider.channel;spider.afterFilter”]>（注：尚未实现，需框架调整，较为麻烦）

<plugin pluginId="spider.localService" serviceTimeout="10000" zlibCompress="false" encrypt="false" anonymous=”false” serviceProxyPackage=”com.medsoft.spider.client” >

<server enable=”false” port=”19181” reliable=”false” threadCount=”200” serviceExportPackage=”” ha=”false” forceRecovery=”0” haRemoteServerAddress =”ip:port” ipPrefix=”172.18” remoteServerPassword =”” maxQueueCount=”100000” ssl=”false” />

</plugin>

<plugin pluginId=”spider.channel” ssl=”false”>

<cluster clusterName=”spider-server” connectionSize=”1” lbType=”1” fields=””>

<workNode address=”127.0.0.1” port=”7070” />

<!-- < workNode address=”127.0.0.1” port=”7071” /> 🡪

</ cluster>

<!—

< cluster clusterName=”sys-service” connectionSize=”1”>

< workNode address=”127.0.0.1” port=”7777” />

</ cluster>

🡪

</plugin>

<plugin pluginId=”spider.filter”>

<filter>io.spider.demo.filter.BeforeFilterImpl1 </filter>

<filter>io.spider.demo.filter.BeforeFilterImpl2</filter>

<filter> io.spider.demo.filter.AfterFilterImpl1 </filter>

<filter> io.spider.demo.filter.AfterFilterImpl2 </filter>

</plugin>

<plugin pluginId=”spider.customPlugin”>

<customPlugin name=”” class=””>

<args>

<arg name=””></arg>

<arg name=””></arg>

<arg name=””></arg>

</args>

</customPlugin>

</plugin>

</plugins>

<routeItems>

<!-- <routeItem serviceId=”99\*” clusterName=”spider.localService” /> 🡪

<routeItem serviceId=”rskL.6.30; rskL.6.31;” appVersion=”” SUBSYS=”” systemId=”” companyId=”” clusterName=”spider-server” />

</routeItems>

</spider>

配置文件中的所有节点元素（element）名和属性名均为大小写敏感，且采用驼峰式命名，并尽可能使用英文全称。

配置文件的使用元素或者属性的规范为：如果对象是特性，则作为属性；如果对象本身是主体，则作为元素。

各节点元素（element）以及属性的含义如下（绿色标注为已实现特性）：

|  |  |  |  |
| --- | --- | --- | --- |
| 元素 | 属性（--代表元素本身） | 可选 | 默认值，含义和取值范围 |
| spider | -- | 否 | spider配置文件根 |
| nodeName | -- | 否 | spider节点基本信息 |
| value | 否 | Spider节点名称，相同名称的spider将自动组成集群，任意字符串，用于cloud模式 |
| checkPidFile | 是 | 是否检查pid文件。true:检查，false:不检查。默认false，不检查。 |
| dev | 是 | 运行模式，用于控制日志输出级别。true:开发模式，将输出所有日志信息；false:生产模式，将自动禁止输出debug级别的日志。默认false。 |
| cloud | 否 | spider节点运行环境，true：运行于服务中心模式，将自动接收来自服务中心推送的下游节点变化，适合于大规模部署；false：运行于独立管理模式，可通过restful api管理相关节点变化。 |
| role | 是 | spider角色，client/sc/prod/nb/np：作为生产服务器运行；sc：作为服务中心运行。默认生产服务器。除非配置为sc，否则均表示生产服务器。只不过只有该节点配置为np时:一、并行处理插件才会生效，具体可见并行执行插件一节；业务广播才会派发给本节点处理（否则会出现服务无法找到的异常）。 |
| serviceCenter | 否 | 服务中心地址，ip:port格式。 |
| appVersion | 是 | 长度为最长为12位，建议xx.xx.xx格式，本spider节点提供的应用服务版本，用于灰度升级，任意字符串，具体参见灰度升级一节。默认””，表示非特定版本。 |
|  | *logMsgIdPrefix* | 是 | 记录spider请求应答日志时，是否rpcMsgId作为前缀。true:启用; false:不启用。集中化存储时建议启用，否则不建议启用。默认false:不启用。 |
|  | serialMethod | 是 | 序列化机制，计划支持： json/protobuf/protostuff。默认json（当前仅支持json）。  在java版本中，我们在spider 1.0.8版本中实现的是基于完整参数列表构造器的反序列化，其性能相对于标准化的json反序列化，性能平均提升超过10x倍，视属性的数量而定。 |
| serviceDefineType | 是 | 注解方式，ld:ld原注解方式，spider:spider注解方式，默认spider注解方式。为了最大程度兼容标准方式和ld专有方式，当选择ld时，spider标准注解的服务也将自动被发布或代理。两者之间语义相互兼容。 |
| charset | 否 | 全局编码格式，UTF-8或GBK，主要用于适配其他语言比如C/C++编写的程序，目前主要用于对接LDPack打包的客户端。建议整个环境要么UTF-8，要么GBK，尽量避免有些UTF-8，有些GBK，不然容易出错。 |
| slowLongTime | 是 | 慢请求执行时间，执行时间超过该长度的请求会被自动写到本地慢日志。默认3000毫秒，1.0.8版本调整为100毫秒。实时生效。 |
| dumpStat | 是 | 是否启用定期dump服务性能指标到本地，true：是，每隔5分钟会自动dump一次，该参数独立于cloud参数；false：否。默认:true。 |
| logOutput | 是 | 慢日志以及性能统计的输出目的地。file:保存到本地，默认为/tmp/spider/stat/${nodeName}。mongodb:保存到mongodb服务器。默认:file。 |
| mongo.db | 是 | Mongo数据库名，默认spider |
| mongo.statCol | 是 | Mongo集合名，stat |
| mongo.slowCol | 是 | Mongo集合名，slow\_request |
| mongo.uri | 是 | Mongo服务器地址，默认localhost:27017 |
| tcpDump | 是 | 是否启用动态抓包和拦截。true：是；false:否。默认false。如果多个客户端同时设置了抓相同功能号的包，则所有都会收到结果。该特性会严重影响性能，同时存在严重安全隐患，生产环境谨慎开启。 |
| tcpDumpMode | 是 | 启用动态抓包的情况下，请求发送给客户端的模式。push：主动推送给客户端（要求客户端支持接收）；pull：客户端主动来查询。 |
| detectInterval | 是 | 心跳检测间隔，默认60000毫秒。 |
| needSFtp | 是 | 是否启用SFTP功能，true：启用；false：不启用 |
| SFtpPort | 是 | SFTP端口 |
| SFtpUsername | 是 | SFTP用户名 |
| SFtpPassword | 是 | SFTP密码，base64加密后字符串 |
| baseUpStream | 是 | 是否基于下游客户端的配置。true:基于下游服务器的配置自动调整timeout/anonymous/compress/encrypt，而不是使用配置文件中的全局值。这要求先支持通道级别的关键特性。false：基于配置文件中的值。（暂不考虑实现，20170316，相对来说比较可控，不做跟HTTP服务器一样兼容各种浏览器的模式） |
| *supportPlainParams* | 是 | 是否支持参数枚举方式。true:支持；false:否。默认false。该参数技术实现上与注解方式是否为ld无关。 |
| noLoggingList | 是 | 开发模式不记录发送、接收请求体的服务列表，不支持通配符，逗号或者分号分隔。更改配置文件后60秒内自动生效。 |
|  | loggingList | 是 | 生产模式记录发送、接收请求体的服务列表，不支持通配符，逗号或者分号分隔。在不少的情况下，我们通常希望仅仅记录某些功能号调用的日志，不记录其他以便快速排查问题。此时通过强行记录日志的列表通常能够达到更好的效果。更改配置文件后60秒内自动生效。 |
|  | enableEpoll | 是 | 是否启用epoll，true:启用，false:不启用，默认不启用。 |
|  | suppressErrorNoList | 是 | 不打印信息的错误编号列表，主要是防止某些业务全部通过Exception返回，哪怕技术上本身是正确的，只是逻辑结果不正确。不支持通配符，逗号或者分号分隔。更改配置文件后60秒内自动生效。 |
|  | threadAffinity | 是 | 使用启用线程CPU亲和性。true：启用，false：不启用，默认不启用。 |
|  | enableFluidControl | 是 | 本服务器是否主动启用流控。true:启用; false:不启用。默认不启用。1.0.10 |
|  | fluidControlMode | 是 | 流控模式。all:全部服务统一流控。whitelist:[功能号列表，逗号或者分号分割]:白名单流控（通过功能号清单指定不受流控的功能列表）。1.0.10 |
| plugins | -- | 否 | spider插件列表，当前版本一共有3个插件，插件标识符不可修改，否则会导致spider启动异常 |
| plugin  (pluginId= spider.localService) | -- | 否 | spdier插件信息，不同的插件具有不同的属性，spider核心引擎插件，用于设置spider核心的基本特性  在netty的实现上，tcp队列长度直接取/proc/sys/net/core/somaxconn的值，没有提供API进行修改，故若需修改，需在OS层面修改，spider原计划支持，后来取消 |
| serviceTimeout | 是 | 服务超时时间，可在service级别覆盖，默认300秒，单位毫秒，正整数 |
| zlibCompress | 是 | 是否启用全局zlib压缩请求包，true或者false，默认false。建议局域网内不启用，非局域网内启用。 |
| encrypt | 是 | 是否启用AES256加密请求包，true或者false，默认false |
| serviceProxyPackage | 是 | 作为spider客户端时要调用的远程服务的包路径，以;或,分隔。只要在该参数上设置了相关路径，服务端如果在serviceExportPackage参数上设置了对应路径，本客户端就可以通过@Autowired注入方式调用远程服务端对应包含提供的服务。  为了确保远程调用的正确性，对于进行RPC调用的情况，需要确保在本节点中不包含代理类的实现，否则启动时默认情况下Spring使用类型注入时会出现多个实现的异常，否则就需要使用Qualifier或者Resource注解。  支持路径表达式为模式匹配。  Spider 20161228 版本开始，支持功能号二次过滤。格式为在最后加上“:”，如com.ld.net:pL.12.32,pL.21.32,pL.22.\*,pL.31.3?。其中\*代表任意多个字符，?代表任意一个字符。 |
|  | excludeProxyPackage | 是 | 要从自动代理路径中排除的路径。 |
| anonymous | 是 | 服务器是否允许无认证连接。true:允许；false:不允许。默认true。具体见安全一节。该参数在服务端设置，客户端根据服务端的响应报文被动执行。当节点作为客户端角色时，该参数没有作用，也即无需设置。 |
| server | -- | 否 | spider运行于服务器模式时的相关信息 |
| enable | 否 | 是否启用服务端，false代表不启用，仅作为客户端，true代表启用。如果为true，则port不能为空。 |
| port | 是 | 作为服务器时的端口号，1025-63335 |
| reliable | 是 | 服务器是否运行于可信模式。true：可信模式；false：非可信模式。具体参考可信模式一节，默认false。reliable采用redis作为持久化机制，reliable=true时，本地必须已经安装redis，redis安装配置手册见《redis安装配置指南》。 |
| ha | 是 | 可信模式下，是否启用高可用。true：需要配置一个本机之外的redis服务器，请求队列除保存在本地之外，还会在服务器上保存一份，本地的队列负责ACID中的D，远程建议为每秒持久化一次。false：仅在本地保存队列，考虑到单点性，建议不要关闭该参数。服务器模式下默认true。 |
| *haRemoteServerAddress* | 是 | 高可用模式下，远程服务器连接地址，ip:port格式。 |
| *remoteServerPassword* | 是 | 远程服务器密码，默认情况下，远程redis服务器不允许未认证访问，本机要求不设置认证 |
| *ipPrefix* | 是 | Spider所在服务器IP或前缀xxx.xxx[.xxx][.xxx]格式，高可用模式下不得为空，spider nodeName+IP唯一标识一个spider实例。使用ip前缀时，系统会自动查找本机在该子网下ip地址，适用于避免大规模部署和启动服务器时逐台更改IP地址繁琐的问题 |
| maxQueueCount | 是 | 服务器运行于可信模式时，可信队列的最大长度，防止异常时JVM内存溢出。默认为100000，最大1000000。 |
| forceRecovery | 是 | 服务器强制恢复级别。主要用于当本机器由于硬盘或者其他问题其他恢复时，将从HA模式下配置的远程请求存储服务器加载待处理的请求。取值范围从0-1（以后可能会变化）。0：代表仅从本地加载，为正常模式。1：代表从远程加载，用于异常恢复（为了避免加载冲突，从远程加载时，会在远程设置一个特殊标记，标记当前处于恢复模式，节点从远程加载时会先判断该标记是否为0，如果不为0，则不允许加载）。处于恢复模式时，spider只会恢复待处理的请求，并在处理完成后自动退出，而不会开启服务端模式。 |
| threadCount | 是 | 作为服务器时业务处理线程的数量，建议为cpu核心数的20-50倍之间，默认cpu数量的20倍，一般建议不要设置 |
| serviceExportPackage | 是 | 作为服务器时自动发布的spider服务的包路径，以;或,分隔。只要服务端在该参数上设置了相关路径，客户端只要在service-proxy-package参数上设置对应路径就可以直接通过@Autowired注入方式调用本服务端包下各类提供的服务。  支持路径表达式为模式匹配。  Spider 20161228 版本开始，支持功能号二次过滤。格式为在最后加上:，如com.ld.net:pL.12.32,pL.21.32,pL.22.\*，pL.31.3?。其中\*代表任意多个字符，?代表任意一个字符。 |
|  | excludeExportPackage | 是 | 要从自动发布路径中排除的路径。 |
| ssl | 是 | 是否启用服务端ssl。True:启用, false: 不启用。默认false，不启用。 |
|  | sslServerCert | 是 | 服务端证书的路径，可以classpath或者绝对路径 |
|  | sslServerKey | 是 | 服务端证书加密键 |
| plugin  (pluginId= spider. channel) | -- | 否 | spdier插件信息，不同的插件具有不同的属性，通道插件，通道下的每个cluster代表一个服务器集群，由旗下的workNode组成 |
| ssl | 是 | 是否启用服务端ssl。True:启用, false: 不启用。默认false，不启用（1.0.9开始支持）。 |
|  | sslClientCert | 是 | 客户端证书仓库的路径，可以classpath或者绝对路径 |
|  | sslClientKey | 是 | 客户端证书仓库加密键 |
|  |  |  |  |
| cluster | -- | 是 | 定义下游服务器集群 |
| clusterName | 否 | 定义集群的名称，需要和下游的服务器定义的nodeName相同，一个配置文件中的各clusterName必须不同 |
| lbType | 是 | 负载均衡模式。1:轮训；2:基于报文头（可选报文头）字段的request stick模式。默认1。 |
| fields | 是 | ldType=2时，用于计算目标节点的报文头字段列表，逗号或者分号分隔，建议取值为uid，API自带帮助器方法。 |
|  |  |  |
| balanceMode | 是 | 负载均衡模式。1:轮询（默认）；2:hash。默认轮询。 |
| enableFluidControl | 是 | 是否启用流控。true:启用（启用流控时、流控法治基于心跳检测时发送给目标服务端的连续三个请求，根据三个的平均值是否>=60决定，目标服务器单节点时流控不生效）; false:不启用。默认不启用（1.0.11）。 |
| workNode | -- | 是 | 定义下游服务器集群中的成员节点。一个cluster范围内的workNode必须address+port唯一。 |
| address | 否 | 成员节点的ip地址 |
| port | 否 | 成员节点的端口号，对应于远程节点spider.xml中plugin pluginId="spider.localService"->server定义的端口。 |
|  | connectionSize | 是 | 跟下游服务器建立的tcp连接数量，1.0.0版本中该参数自动控制,该参数的合理值依赖于spider服务器具体的网络机制和网络是否支持多队列（1.0.10）。 |
| plugin  (pluginId= spider. filter) | filter | 是 | 过滤器插件，每个filter代表一个过滤器实例。具体参见《1.3 流水线插件》。 |
| routeItems | -- | 否 | 定义路由表的信息。路由用来配置将不同的服务请求转发到相应的spider服务器。  路由条目从上往下解析，当上面和下面的路由配置冲突时，使用上面的路由条目。 |
| consistent | 是 | 是否启用一致性路由策略。true:启用，此时spider运行时会根据功能号>版本号>机构号>子系统号>系统号(产品系统号)的规则，对路由表进行排序，此时无论路由条目的顺序如何，最终的路由目标是一致的；false:不启用，此时spider根据定义的路由条目顺序进行匹配，不同的路由条目顺序可能导致不同的路由结果。默认：false。建议启用。1.0.6 RC版本开始，默认:true。 |
| routeItem | -- | 否 | 定义路由条目，路由条目可以使用多个维度进行匹配以便灵活满足各种企业系统业务场景，目前支持根据功能号、版本号、子系统号、系统号、机构号五个维度进行组合，其中优先级从高到底为：功能号>版本号>机构号>子系统号>系统号(产品系统号)>历史标志>批处理标志。  五个维度中，功能号、子系统号、历史标志、批处理标志为静态属性，必须编译期确定（一般来说，这两者选一即可）。版本号、机构号、系统号（产品系统号）为运行时属性，可运行时设置，主要适用于多租户和灰度升级的场景。  其中功能号必须定义，如果匹配全部，则使用\*即可，功能号与各维度均为and关系；其他均可选。没有定义代表匹配所有，即\*。  至少需要定义一个指向本地处理插件的条目，最简单为<routeItem serviceId=”\*” clusterName=”spider.localService” />  多个路由条目的顺序会影响最后的路由结果，如下所示：  <routeItem serviceId=”11\*” appVersion=”1.0.2” clusterName=”BSNP-C00001v2” />  <routeItem serviceId=”11\*;21\*” companyId=”C00001” clusterName=”BSNP-C00001” />  在上述的条目中，如果某1.0.2版本C00001机构的11xxxxxx功能到本节点，会被转发给BSNP-C00001v2；如果上述路由条目顺序反一下，则会转发给BSNP-C00001。 |
| serviceId | 否 | 12位ASCII字符，定义本路由条目适配的功能号，功能号支持通配符格式，\*代表匹配全部功能，?代表匹配一个可见字符。功能号之间可以用;或,分隔。 |
| appVersion | 是 | 定义本路由条目匹配的应用版本号，不支持通配，多个应用版本号之间可通过;或,分隔。 |
| subSystemId | 是 | 定义本路由条目匹配的子系统号，不支持通配，多个子系统号之间可通过;或,分隔。 |
| systemId | 是 | 定义本路由条目匹配的系统号，不支持通配，多个系统号之间可通过;或,分隔。 |
| companyId | 是 | 定义本路由条目匹配的机构号，不支持通配，多个机构号之间可通过;或,分隔。 |
| his | 是 | 定义本路由条目匹配的历史标志，在当前库和历史库分开的情况下，直接在spider层面进行路由保证。 |
| batch | 是 | 定义本路由条目匹配的批处理请求标志，在当前库和批处理库分开的情况下，直接在spider层面进行路由保证。 |
| clusterName | 否 | 定义本条目中的功能将转发到的spider远程服务器。应该确保每条独立的路由均有不同的clusterName，如果多条路由具有相同的目标节点，应该在相应属性上使用;分隔的方式进行合并。 |

## 服务发布与代理

服务发布：spider.localService插件下的serviceExportPackage元素定义了spider节点作为服务器时自动对外发布的spider服务的包路径，以;或,分隔；excludeExportPackage元素定义了spider节点作为服务器时排除自动对外发布的spider服务的包路径，以;或,分隔。只要服务端在该参数上设置了相关路径，java客户端只要在serviceProxyPackage参数上设置对应路径就可以直接通过@Autowired依赖注入方式调用远程服务端包下提供的相应服务。

服务代理：对于java客户端而言，spider提供了自动代理功能，开发者可像调用本地Spring服务一样调用远程服务器上的spider服务，开发者只需要在spider.localService插件下的serviceProxyPackage元素上定义需要自动代理的spider服务的包路径，以;或,分隔；excludeProxyPackage元素上定义需要排除自动代理的spider服务的包路径，以;或,分隔。对于非java客户端比如C#或C++客户端而言，开发者需要调用对应的SDK Client。

发布的服务和代理的服务不能有交叉。如果一个服务既需要在本地处理，又需要spider代理转发给下游服务器，则配置在发布列表。此时这些服务无法通过用户编程来远程调用。通常这些服务是用于特殊目的，且被标记为broadcast。

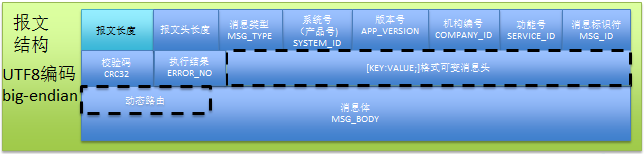
PS：从技术上来说，只要客户端和服务端同一个服务的方法签名和服务号相同就能够调用（也就是方法名无关），不过不建议这么做，在将来的版本中也可能会要求完全匹配。

还需要注意的是，如果某个spider节点希望作为NB的角色，在serviceExportPackage配置了发布路径，同时又包含了某个服务的实现，如果该服务的路由条目被解析到本地处理，则该服务会在该NB节点被处理，而不会继续往后转发，这一点需要注意。所以，对于作为NB的角色，建议除了广播的服务，不要配置在serviceExportPackage下，而是配置在serviceProxyPackage。

## spider报文结构

spider协议实现采用big-endian（标准的网络字节序），编码格式可配置。

Spider报文由四部分组成，分别为报文长度+固定报文头+可变报文头+报文体。如下所示：



报文长度：4字节（整型），最大支持16M报文

固定报文头：79字节

可变报文头：报文头长度-79字节

报文体：报文长度-报文头长度

### spider固定报文头

固定报文头由多部分组成，分别为：

* 报文头长度：4字节（整型字符串），不对应用系统开放（该实现在将来的版本可能会更改）
* 消息类型：1字节，ASCII字符串，1：请求；2：应答；3：消息推送。不对应用系统开放
* 消息类别：1字节。1：业务请求；0：spider内置请求。不对应用系统开放
* 系统编号/产品编号：2字节（ASCII字符串），对应用系统开放
* ~~连接亲和性标识符：8字节，对应用系统开放，1.0.8版本。~~
* 应用版本号：8字节（ASCII字符串），对应用系统开放xx.xx.xx
* 机构编号：6字节（ASCII字符串），对应用系统开放
* 功能号/服务号：12字节（ASCII字符串），00000000-00000199：spider内部保留功能号；对应用系统开放
* 消息标识符：32字节（UUID字符串），不对应用系统开放（该实现在将来的版本可能会更改）
* 校验码：10字节（ASCII字符串），不对应用系统开放（该实现在将来的版本可能会更改）。CRC32为long类型，故不调整。
* 执行结果：3字节（整数字符串），不对应用系统开放。为兼容原来，不做调整。

总共79字节，java客户端同步升级，c#/c++等其他接入需对应修改）

### spider可选报文头

json格式。必须符合JSON RFC（www.json.org）规范。

已完成开发。

已测试（基于并行执行插件）。

可选报文头默认会随着主请求发给下游服务器，如果不想发送给下游服务器，则需要具体的插件自行负责对可选报文头进行过滤。

### spider报文体结构

报文支持多种序列化机制，支持能被google gson/ackson/protobuf/protostuff序列化的所有对象，包括但不限于plain pojo，list,map,set,array以及嵌套。典型的报文体举例如下。

* + 简单的string

“string” --注意，需要包括双引号

* + 简单的数字

888

* + 字符串列表/数组

[“string1”,”string2”,”string3”]

* + 复杂对象

{“date”:”20160914170209”,”params”:{“key3”:”value3”,”key2”:”value2”,”key1”:”value1”},”tt”:”20160914170209”,”ints”:[1,2,3,4,5,6,7],”routes”:[{“serviceId”:”1”,”systemId”:”1”,”SUBSYS”:”1”,”appVersion”:”1”,”companyId”:”1”,”clusterName”:”1”},{“serviceId”:”2”,”systemId”:”2”,”SUBSYS”:”2”,”appVersion”:”2”,”companyId”:”2”,”clusterName”:”2”},{“serviceId”:”3”,”systemId”:”3”,”SUBSYS”:”3”,”appVersion”:”3”,”companyId”:”3”,”clusterName”:”3”}]}

格式化后为：

{

“date”: “20160914170209”,

“params”: {

“key3”: “value3”,

“key2”: “value2”,

“key1”: “value1”

},

“tt”: “20160914170209”,

“ints”: [

1,

2,

3,

4,

5,

6,

7

],

“routes”: [

{

“serviceId”: “1”,

“systemId”: “1”,

“SUBSYS”: “1”,

“appVersion”: “1”,

“companyId”: “1”,

“clusterName”: “1”

},

{

“serviceId”: “2”,

“systemId”: “2”,

“SUBSYS”: “2”,

“appVersion”: “2”,

“companyId”: “2”,

“clusterName”: “2”

},

{

“serviceId”: “3”,

“systemId”: “3”,

“SUBSYS”: “3”,

“appVersion”: “3”,

“companyId”: “3”,

“clusterName”: “3”

}

]

}

注意：传输时需要采用非格式化格式。

* + 日期

“20160914170330”

* + map

{“key3”:”value3”,”key2”:”value2”,”key1”:”value1”}

* + 整数数组/列表

[1,2,3,4,5,6,7]

对于java，建议采用jackson 2.x。

对于c++，可参考<https://github.com/miloyip/nativejson-benchmark>\ <http://www.oschina.net/question/12_91812>。

对于c#，可基于.net框架自带的System.Web.Script进行包装和完善。

### 各种数据类型支持情况

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 参数支持 | 返回值支持 | 格式和取值 | java | C# | C++ |
| void | 无参 |  |  |  |  |  |
| byte |  |  |  |  |  |  |
| Short |  | √ |  | short |  |  |
| Integer |  | √ |  | Int |  |  |
| Long |  | √ |  | Long |  |  |
| Float |  | √ |  | Float |  |  |
| Double |  | √ |  | Double |  |  |
| Date |  | √ | yyyyMMddHH24mmss | java.util.date |  |  |
| Char |  | √ |  | Char |  |  |
| String |  | √ |  | String |  |  |
| 枚举 |  | 暂不支持 | -- | -- | -- | -- |
| Object |  | √ |  |  |  |  |
| List |  | √ |  | Java.util.list |  |  |
| Map |  | √ |  | Java.util.map |  |  |
| Map<String,HashMap<String,Myinfo>> |  |  |  |  |  |  |
| Array |  | √ |  |  |  |  |
| List<Pojo> |  | √ |  | Java.util.list<E> |  |  |
| 上述对象的嵌套 |  | √ |  |  |  |  |
| 上述对象作为参数枚举 |  |  |  |  |  |  |

例如，可以支持如下格式的对象传输：

**Class T <T>** TDate {

**private** Date date;

**private** Map<String,String> params;

**private** Timestamp tt;

**private** **int**[] ints;

**private** List<T> routes;

}

## 服务接口定义要求

因为本系统的目标不是直接兼容现有系统，也不是现有系统不经过任何接口改造即可直接套用。综合考虑灵活性，性能，开发效率，建议服务参数和返回值均采用对象（现有的RPC框架基本都采用这种模式，比如gRPC，ICE），同时参数继承SpiderBizHead类（其中包含了设置动态路由等相关信息）。如下所示：

入参DTO：

public class ServiceReq extends SpiderBizHead {/\*如果无需动态路由功能，则不必继承

SpiderBizHead\*/

}

服务签名：

@ServiceModule()

public interface DemoService {

@Service()

public ServiceResp addServer(ServerReq req);

}

还要考虑C++以及其他在用编程语言实现动态代理的简易性。

对于C#，可参考System.Runtime.Remoting.Proxies.RealProxy类实现。

从spider 1.0.5版本开始（2016年11月28日发布），也支持请求服务入参为空，例如：

@ServiceModule()

public interface DemoService {

@Service()

public ServiceResp queryClusters ();

}

## 上下文信息（1.0.10）

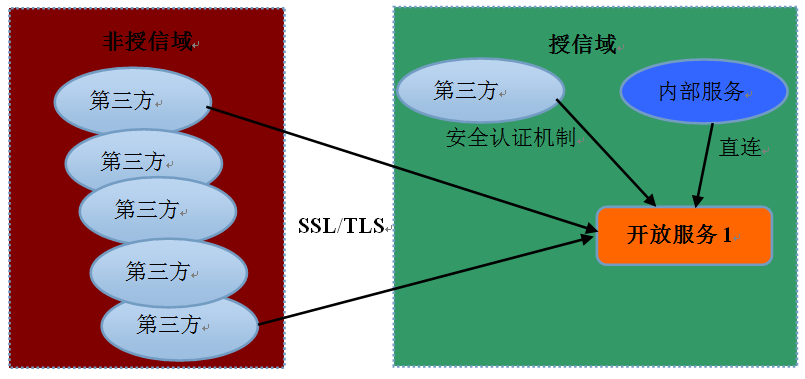
新增RpcContext上下文信息，对于不希望通过业务接口传递，但是又无法增加到SpiderBizHead中的一些通用参数信息比如token，亦或是不希望或无法继承、聚合SpiderBizHead的场景，可以放置在RpcContext中进行传递，RpcContext是一个内置的ThreadLocal，因此业务方可以透明的在上下文中进行传递。无论是调用方还是被调用方，都能够通过RpcContext得到这些信息。

RpcContext中的信息，除了标准报文头的信息外，将通过可选报文头的方式进行传递。

同时这也适合于在接口由第三方提供，无法修改，但是逻辑上下文中又需要传递额外信息时使用。

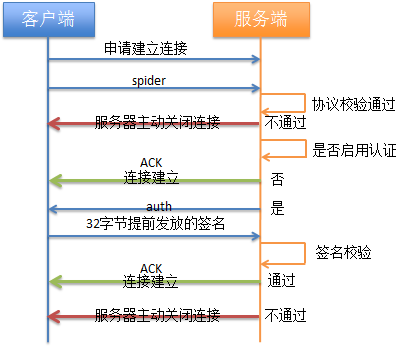
## 安全性

spider提供了多重安全保障机制，目前主要支持接入握手/SSL（1.0.9）校验，报文完整性校验，报文加密，报文长度检查四种机制。



### 接入认证

spider使用两次握手校验，其握手流程如下：



原始报文如下：



签名将采用私钥加密，公钥解密非对称加密或者AES对称加密的方式实现。

1.0.4之前的版本采用AES 128算法。

1.0.6版本开始采用RSA算法。

1.0.9版本开始支持SSL认证，客户端对服务端认证，服务端不对客户端认证。

非SSL模式，license信息存储在classpath\*:spider.dat文件中，在程序中固定。对于C#客户端，license信息存储在spider.xml中的license属性中，具体参见C#客户端开发指南。

### 报文完整性校验

spider对报文进行CRC32完整性校验以验证报文在传输过程中不会被篡改，对于被篡改的报文拒绝处理。

### 报文长度检查

spider对每个接收到的报文会检查其最小长度是否大于报文最小的可能长度，如果小于最小长度，则会强行断开客户端连接。

### 报文加密

spider支持对报文做加密，目前支持AES128加密算法。

### SSL配置说明

1.0.9版本，开始支持SSL加密通信。

客户端和服务端spider.xml配置参见配置文件一节。

java证书的生成步骤可参见<http://www.cnblogs.com/zhjh256/p/6488668.html>。

## spider高级特性

### 多租户

spider原生支持多租户部署，spider报文头对外开放了机构号、系统号两个属性用于支持多租户场景下的路由。

多租户场景下的路由可以支持下述几种模式：

* + 系统号；
  + 系统号+服务号（模式匹配）；
  + 机构号+系统号；
  + 机构号+系统号+服务号（模式匹配）；

要启用多租户模式，必须确保至少有一条路由的系统号或机构号或版本号不为\*。

### 灰度升级

spider原生支持灰度升级，spider报文头对外开放了版本号属性用于支持灰度升级场景下的路由。

一般来说，建议在接入服务器中维护应用的版本可最简化管理。为此，spider在配置文件中增加了一个appVersion的属性，如果在配置文件中设置了该属性，在报文头版本号属性为空时自动注入到spider请求中。

### 高可用与集群

spider原生支持高可用和集群特性，在集中式管理模式下，具有相同节点名的服务器节点会自动组成集群并通知相应的上游节点。在独立管理模式下，通过将需要组成集群的服务器节点配置在spider.xml文件相同的spider.channel->server元素下，各服务器节点将自动组成集群。

在任何模式下，只要下游节点出现故障，spider均会自动将其从集群列表中隔离，并定期重试。如果出现故障的节点需要永久性删除，则可以通过服务中心进行删除或者在独立管理模式下使用相应的restful api进行删除。

### 运行时扩展

1.0.11版本开始，通过服务中心实现服务注册和发现，反向注册不再受支持。

### 流控机制

为了防止服务器异常或者客户端异常时可能出现的雪崩现象，1.0.11版本开始，支持客户端和服务端流控。

### 负载均衡

为什么负载均衡是必要的，由于JDK 1.9之前默认gc parallelgc,而java gc stop the world的特性，因此对于不容容忍可能的pending app的应用，除非服务器配置完全足够，确保任何时候都不会发生full GC，否则HA将大大缓解full GC时的影响。

虽然JDK 1.9采用了g1 gc，其最小化了pause时间，但cms总体目标和g1类似。如果更关心响应时间，cms更合适，优化后的cms会比g1更合适。g1的长期目标是替换cms。

spider原生支持负载均衡，无需任何配置。

在当前版本中，spider仅支持轮训模式的负载均衡。1.0.8版本开始，支持多种负载均衡模式。

### 运行状态检查

spider提供了一系列接口可用于查询当前节点运行状态信息，这些接口以RESTFUL API的方式提供给维护人员和开发人员使用。具体参见RESTFUL API一节。

### 可信模式

spider可运行于可信和非可信两种模式，任何时候，系统只能运行在其中一种状态。

可信模式是指spider只要接收到客户端的请求，就会第一时间先把请求持久化，随后执行接收到的请求，理论上，在可信模式下，只要接收到请求，就肯定会被执行，主要适合于报盘等业务场景，在可信模式下，可设置需要缓存的最大请求数量，默认为10000。

非可信模式和通常的应用服务器相同，请求以接收到的顺序排队被处理，当所有的线程均满负荷运行时，请求会在线程队列中排队，此时如果节点宕机，重启后，未处理的请求将会丢失。

默认情况下，spider运行在非可信模式。

1.0.11版本开始，增加配置支持某些功能、某些业务维度上必须串行执行。

### 广播请求

在很多的分布式系统中，视对性能的要求不同，系统间缓存数据的同步会采用下列机制：

* 分布式缓存（如redis、memcached、couchbase、ehcache等）；
* MQ（RabbitMQ、ActiveMQ）；

很多时候，出于性能考虑，我们会选择在各应用节点间之间数据同步的变化，比如系统参数、股票代码、字典项等。虽然上述机制都可以实现，但上述每种机制都会导致需要维护额外的第三方软件、且需要配置HA（否则会有额外的SFP），开发则需要专门学习API，会增加额外更多的硬件和维护成本。为此，为了最小化各种成本，spider中间件在服务级别提供几乎没有额外学习成本的支持。

spider支持服务以广播模式执行，所有本节点能够连接到目标节点都将收到请求。通常，用户可能会在client节点发起广播请求，也可能会在server节点发起广播请求，这两种模式的差别在于，server节点本身也是该服务的提供者。纯技术角度而言，他们没有区别。不过容器的实现往往最常用的场景作为默认值，特殊场景需要额外的配置。Spider采用类似模式，对于需要以广播方式执行的服务，需要在SericeModule和Service同时设置broadcast属性大于0，此时这些服务会按照名称进行注入，而非默认的按类型注入。同时，对于这些服务，在spider.xml中，需要作为自动代理的服务（serviceProxyPackage），而不是发布的服务，只有这样，spider运行时才会为这些服务创建远程代理。

同时，在业务开发时，也需要区分调用上下文是client还是server，一般来说作为web接入时相当于client，否则为server。（其实如果作为纯路由，我们应该区别对待的（只不过当前并没有明确区分，因为在此之前的实现中，我们仅对于spider内置管理和监控功能提供广播模式的功能））。作为client执行广播请求的时候，由于client本身不提供业务服务，所以SericeModule的broadcast属性无论为何值都可以。作为server执行广播请求的时候，由于server本身不提供业务服务，所以SericeModule的broadcast属性必须大于0（具体取决于是否有返回值，1：没有返回值；2：有返回值。一般建议业务类有返回值或者有相应的轮训机制），同时自动注入的时候，并且使用注入的时候，必须按名称注入，不能使用按类型注入。定义实现的时候，应该设置服务名称为Introspector.decapitalize(shortClassName)，spider运行时在查找本地实现时，会根据这个名称进行获取。

还需要注意的是，在主动发起的节点如果也需要调用实现，则必须自行调用，spider只负责调用后续节点，不负责调用自身。但是很多时候网络拓扑是client->BSNP->MSNP，这个时候，如果client发起广播请求，则BSNP和MSNP都会被更新；如果是BSNP发起，则只有MSNP会被更新。简单的规则就是，更新本地的时候调用本地bean的实现，更新远程的时候调用远程bean的实现，同时要更新的时候，在一个父逻辑中分别调用。对于BSNP发起发起广播的情况，需要注意的是平级的BSNP视部署结构，可能不会被更新。

因为在前后台分离的架构中，client（web端）基本上没有任何的业务逻辑，而广播本身通常是在完成某些持久化的变更之后或者对于一些类似行情的不需要持久化的操作（应该增加一个接入层），所以，对于需要广播的请求，为了最小化重复广播，建议在CNB的外面一层控制广播的逻辑。

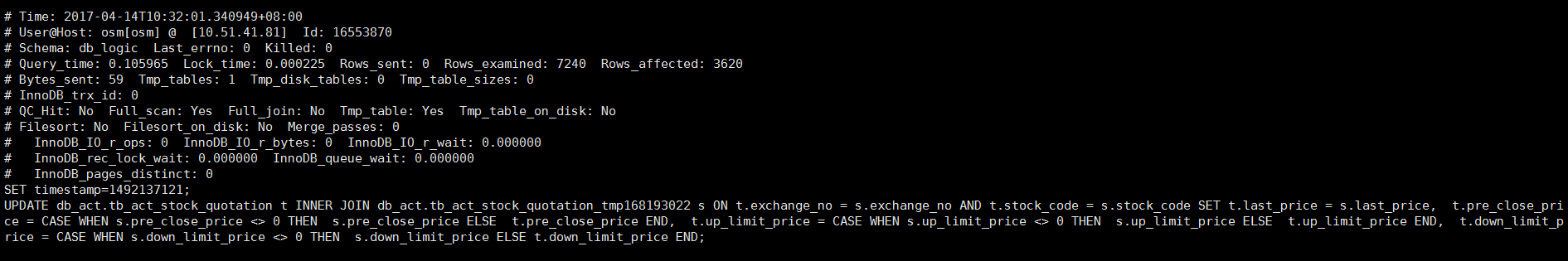
具体示例可参考spider开发指南。

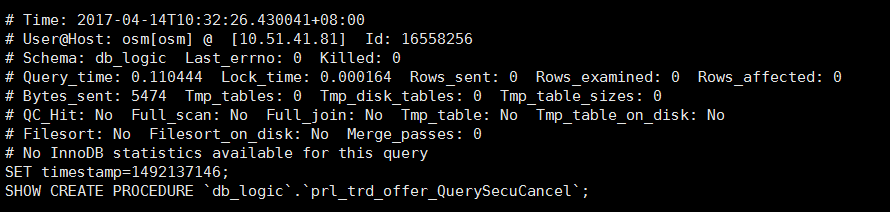
### 支持服务的当前库和历史库属性

支持除了通过功能号外，根据功能的当前、历史库特性进行路由，极大地简化功能号配置和分配的管理。1.0.10。

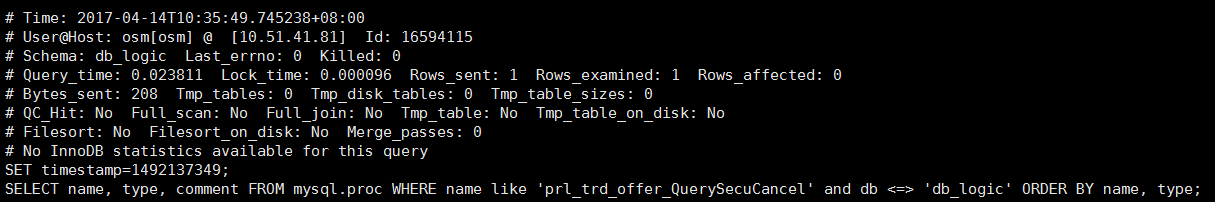
### 守护进程

spider可以以java service wrapper包装器方式支持在守护模式下运行，当进程通过包装器启动时，守护进程将轮训检测进程是否存在，如果进程不存在，则会自动尝试启动确保尽可能避免人为失误或者OS误杀可能导致的系统异常。









## 服务动态拦截与抓包

为了方便开发、测试以及某些情况下在生产环境下提高排查问题的效率，spider支持在运行期间动态抓包以及拦截服务的执行。被拦截的服务在本节点停止处理（包括那些将由本节点最终处理的请求）和往后面转发，通知返回响应的错误信息给调用客户端。

要进行运行时抓包和拦截，必须在服务端将tcpDump参数设置为true，同时可调用下列spider服务开启抓包、拦截和停止。

00000025：开启服务动态拦截或抓包，请求报文结构参见io.spider.manage.api.InterceptManage接口。

00000032：停止服务动态拦截或抓包，请求报文结构参见io.spider.manage.api.InterceptManage接口。

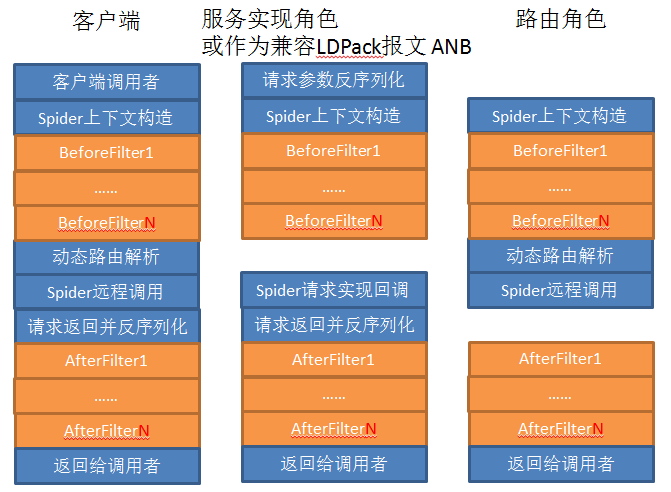
同时在客户端监听00000033请求报文，spider对于经过的符合条件的spider报文通过该服务号推送给开启抓包的客户端。

计划在spider 1.0.5版本开始，提供专门的spider管理客户端控制台。

在第1版本中，支持根据报文头配置抓包规则，在第2版本中，将开始支持基于报文内容配置抓包规则。

## 过滤器

spider支持在请求执行前或完成后进行特殊处理，比如安全性检查、敏感字段混淆等等。为此，spider提供了BeforeFilter和AfterFilter。其执行位置如下图所示：



流水线插件配置在spider.xml配置文件的<plugin pluginId=”spider.filter”>节点下，如下所示：

<plugin pluginId=”spider.filter”>

<filter>io.spider.demo.filter.BeforeFilterImpl1 </filter>

<filter>io.spider.demo.filter.BeforeFilterImpl2</filter>

<filter> io.spider.demo.filter.AfterFilterImpl1 </filter>

<filter> io.spider.demo.filter.AfterFilterImpl2 </filter>

</plugin>

其中io.spider.demo.filter.BeforeFilterImpl1为完整类名，放置在classpath\*目录下即可。过滤器类在配置文件中的顺序为执行顺序，spider运行时会根据给定类实现了BeforeFilter或者AfterFilter自动确定其为前置或后置过滤器，没有实现任何一个接口的类会自动在启动时抛出异常并中止。

为最佳性能和灵活性，建议仅在主动调用客户端实现前置或后置过滤器，服务实现端实现前置或后置过滤器，尽可能避免在路由节点使用过滤器（当节点对于某功能充当路由节点时，请求参数以及返回值将以未解密JSON StringBuilder的格式传递）。

### 前置过滤器

要进行前置处理，必须实现BeforeFitler过滤器，其签名如下所示：

**package** io.spider.filter;

**import** io.spider.meta.SpiderPacketHead;

**public** **interface** BeforeFilter {

**public** **void** doFilter(SpiderPacketHead head,Object origParam);

}

head spider上下文信息。

origParam为序列化之前的原始请求参数。

### 后置过滤器

要进行后置处理，必须实现AfterFitler过滤器，其签名如下所示：

**package** io.spider.filter;

**public** **interface** AfterFilter {

public void doFilter(SpiderPacketHead head,Object retObj);

}

head spider上下文信息。

retObj为反序列化之后的返回值。

### 内置过滤器

K/V翻译/更新（支持同义词，比如order\_id，order\_return\_id）。

金额、数字、日期字段等格式化。

## 插件化体系

为了满足灵活扩展的需要，spider支持灵活的自定义插件扩展，从功能上来说，插件和过滤器的差别在于过滤器不会阻止请求的执行同时对于主程序不会有API上的影响（比如servlet 过滤器和监听器）（最多改变后一步的执行路径），而插件不仅可以改变请求后一步的执行路径，还可以完全阻止请求往后继续执行，在插件内部完成所有的处理，甚至可能会要求主程序不得不做出变动（比如j2ee 7中新增的websocket api），当然也可以什么都不做。

从本质上来说，插件的地位同等于目标服务器或者spider.localService，从执行流程上来说，请求找到路由目标或者spider.localService后，在本节点的处理就已经终止，但是插件可以决定请求的下一处理者仍然是插件（自己除外），在插件中，甚至可以和正常的spider客户端一样，调用远程服务。

在配置了自定义插件的情况下，插件、过滤器、本地处理器、路由的执行顺序如下：

<plugins pipeline=” customPluginName1;customPluginName2;spider.beforeFilter; spider.localService;spider.channel;spider.afterFilter”>（尚未实现，下周需要实现）

其中spider.filter、spider.localService、spider.channel为内置约定的最佳顺序。自定义插件顺序可通过事先配置，也可以在实现的doService()接口中通过不同的返回码运行时改变插件执行顺序或者跳过某些插件进入spider.localService或spider.channel、甚至直接终止请求。如果没有在流水线中配置至少第一个应执行的插件，则所有相关插件均不会被执行；否则，插件实现可运行时指定下一个处理插件或者顺序执行所有后续配置的插件。如果配置了超过一个自定义插件，则不管实际实现了几个插件，最多只会执行流水线中指定的插件。所以，建议要么只配置第一个自定义插件，其它运行时指定；要么，列出所有的自定义插件，运行时除非必要，否则按照配置的顺序往后执行。

实现一个自定义插件包含两个步骤：

实现io.spider.plugin. SpiderPlugin接口。SpiderPlugin的签名如下：

**public** **interface** SpiderPlugin {

/\*\*

\* 解析spider自定义插件

\* **@param** customPluginEle

\*/

**public** **void** doParse(Element customPluginEle);

/\*\*

\* 插件处理逻辑实现

\* **@param** packet

\* **@return**

\*/

**public** SpiderPacketPluginResp doService(SpiderPacketPluginReq packet);

}

1. 在spider.xml中配置该插件。建议所有的自定义插件都遵循如下的配置格式：

<customPlugin name=”” class=””>

<args>

<arg name=”factorType”></arg>

<arg name=”factors”></arg>

<arg name=”degree”></arg>

</args>

</customPlugin>

其中，customPlugin为固定名称，标识该节点为自定义插件，name属性指定插件的名称，class属性指定插件的实现类。args以及其子节点arg为插件自定义参数，开发者自行定义和扩展。

为了避免潜在的业务不一致性，当插件加载失败时，spider将中止启动。

注1：广播请求不受自定义插件影响。

注2：抓包在自定义组件之前。

### 内置插件

spider内置了一些实用的插件，这里做简单的介绍。

#### 会话校验插件

Spider通过插件的方式支持进行会话校验，对于有安全性的需求，可以确保所有的请求都是已经登录的。同时支持单点登录和超时时间控制。

由于spider天然的集群特性，会话校验插件支持spider集群内节点间会话共享。

一般来说，spider只需要配置在NB层即可。

会话校验插件包含如下下配置参数：

<customPlugin name=”sessionPlugin” class=””>

<args>

<arg name=”sessionTimeout”>1800</arg>

<arg name=”enableParallelLogin”>true</arg>

<arg name=”loginServiceId”></arg>

<arg name=”logoutServiceId”></arg>

<arg name=”enableGlobalSession”>false</arg> <!--一般建议启用，线上资管系统必须启用🡪

<arg name=”redisServerAddress”></arg>

<arg name=”password”></arg>

</args>

</customPlugin>

#### 并行执行插件

为了获得某些批处理场景下更低的响应时间，spider支持灵活的并行执行架构，在当前的版本中，系统支持以下两种并行计算因子的指定：

1. 客户端枚举指定；
2. 调用功能号自动计算；

并行计算支持配置自定义维度，其通过配置在customPlugin元素的参数中，该插件没有配置参数，如下所示：

<customPlugin name=”” class=””></customPlugin>

客户端可以通过在可变报文头中指定并行计算的相关参数，具体参见io.spider.meta.SpiderBizHead。目前支持如下参数：

* + factorType：<!--1:客户端枚举指定;2: 调用功能号自动计算-->
  + factors：<!--并行因子，对于枚举指定，以”,”分隔的字符串；对于调用功能号，则指定功能号（该功能号必须能从本节点路由到达）。
  + degree：<!—并行度，默认max(4, cpu数量/4)-->
  + factorDriver：<!—当factorType=2时，调用功能号的查询条件。factorType=1时自动被忽略，可传可不传-->
  + condJsonPath：指定并行查询的条件字段，用于计算出并行条件后插件会写到请求参数，这样可以保证服务的设计与并行执行完全弱相关。

基于CNB/ANB作为高速路由的考虑，并行执行插件仅在第一个NP节点才开始进行并行化（在到达随后的节点时，并行计算参数自动清除），节点的角色可通过配置文件中的role属性进行设置。

由于并行计算是一个fork and join的流程，因此为了尽可能地简化应用层的特殊考虑，spider在BaseResp的基础上扩展了一个ParallelBaseResp，用于并行计算。当用户希望使用并行计算特性时，需要使用该类作为基类，同时增加一个data字段作为业务执行结果，data可以是任何类型（字段名必须是data，放在其他字段上的信息均会丢失），如下所示：

**public** **class** SpiderBaseResp {

**private** String errorNo = SpiderErrorNoConstant.***ERROR\_NO\_SUCCESS***;

**private** String errorInfo = SpiderErrorNoConstant.***ERROR\_INFO\_SUCCESS***;

}

**public** **class** ParallelBaseResp **extends** SpiderBaseResp {

**private** String clusterName = "-1";

**private** String ip = "-1";

**private** **int** port = -1;

}

客户端可以得到每个并行执行子进程的执行结果以及业务返回值。

由于并行计算本身会引入额外的进程或线程间交互成本，因此建议仅在运行超过10秒以上的场景中使用并行执行特性。

并行执行的具体使用可参考spider开发指南。

## 日志中心

Spider RPC框架包含三种类型的日志信息：

* 请求/应答日志。通过log4j支持，具体参考*logMsgIdPrefix属性。*
* 性能统计日志。具体参考
* 慢请求日志。

从1.0.10开始，spider在原来的基础上增加了集中式日志存储，在集中式存储模式下，日志存储在mongodb中，因此如果要使用集中式日志模式，需要事先在相关服务器上安装mongodb。

PS：具体采用什么技术作为集中式存储方案在99%的应用中并没有多大区别，最重要的是要定期清理不必要的日志，以及日志格式设计的在出现问题时能够方便地查找到所有的上下关联的信息。对于分布式系统而言尤其如此，其日志通常跨越多个服务器，确保每个环节的日志能够衔接起来才是日志服务器的关键，纯粹的将日志从分散存储归集到集中式存储除了增加系统不稳定性和成本外，没有任何益处。

## 服务注册和查找中心

在集中管理模式下，服务中心用于监控整个环境的运行状态，以及动态发布新增加或者删除的服务器，集群等。

### 类RESTFUL API

为了在独立管理模式下尽可能提供高可用性，spider中间件提供了一系列restful api用于动态管理当前节点的路由，下游节点等。目前支持的RESTFUL API如下所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 功能 | 服务号 | HTTP方法 | RESTFUL地址 | 已发布 |
| 查询路由信息 | 00000001 | GET | http://ip:port/spider/routes.html | 是 |
| 查询下游服务器 | 00000002 | GET | [http://ip:port/spider/clusters](http://api.spider-rpc.com/v1/clusters).html | 是 |
| 查询客户端连接 | 00000003 | GET | [http://ip:port/spider/clients](http://api.spider-rpc.com/v1/clients).html | 是 |
| 查询发布的服务 | 00000004 | GET | [http://ip:port/spider/exports](http://api.spider-rpc.com/v1/exports).html | 是 |
| 查询代理的服务 | 00000005 | GET | [http://ip:port/spider/proxies](http://api.spider-rpc.com/v1/proxies).html | 是 |
| 查询当前节点概述信息 | 00000006 | GET | [http://ip:port/spider/myinfo](http://api.spider-rpc.com/v1/myinfo).html | 是 |
| 查询动态路由缓存 | 00000007 | GET | [http://ip:port/spider/dyn-route-caches](http://api.spider-rpc.com/v1/dyn-route-caches).html | 是 |
| 新增服务器节点 | 00000008 | POST | [http://ip:port/spider/node/add](http://api.spider-rpc.com/v1/node/add) | 是 |
| 新增服务器集群 | 00000009 | POST | [http://ip:port/spider/cluster/add](http://api.spider-rpc.com/v1/cluster/add) | 是 |
| 新增路由条目 | 00000010 | POST | [http://ip:port/spider/routes](http://api.spider-rpc.com/v1/routes)/add | 是 |
| 删除路由条目 | 00000011 | POST | [http://ip:port/spider/routes/delete](http://api.spider-rpc.com/v1/routes/delete) | 是 |
| 删除服务器节点 | 00000012 | POST | [http://ip:port/spider/node/delete](http://api.spider-rpc.com/v1/node/delete) | 是 |
| 删除服务器集群 | 00000013 | POST | [http://ip:port/spider/cluster/delete](http://api.spider-rpc.com/v1/cluster/delete) | 是 |
| 查询服务运行时性能指标 | 00000014 | GET | [http://ip:port/spider/metrics](http://api.spider-rpc.com/v1/metrics).html | 是 |
| 查询可信模式待处理请求队列 | 00000015 | GET | [http://ip:port/spider/queues](http://api.spider-rpc.com/v1/queues).html | 是 |
| 清空服务运行时性能指标 | 00000016 | POST | [http://ip:port/spider/metric/reset](http://api.spider-rpc.com/v1/metric/reset)-metrics.html | 是 |
| 注册到服务中心 | 00000017 | POST | [http://ip:port/spider/register](http://api.spider-rpc.com/spider/register) | 否 |
| 生成license文件 | - | POST | http://ip:port/spider/generate-license-key.html | 是 |
| 待返回给客户端的请求 |  |  | http://ip:port/spider/pending-requests.html | 是 |
| spider拓扑树 | 00000034 | GET | http://ip:port/spider/node-tree.html | 是 |

PS：通过RESTFUL API可以得到的功能，在服务中心均可用，反之不然。

### 服务中心节点管理功能

|  |  |
| --- | --- |
| 功能 | 服务号 |
| 查询所有受管服务器 | 00000103 |
| 查询服务器概述信息 | 00000106 |
| 推送新增服务器节点 | 00000108 |
| 推送新增服务器集群 | 00000109 |
| 推送新增路由条目 | 00000110 |
| 推送删除服务器节点 | 00000112 |
| 推送删除服务器集群 | 00000113 |
| 查询服务运行时性能指标 | 00000114 |
| 查询慢日志 | 00000128 |
| 执行shell命令（当节点作为server运行时，调用该命令可以在所在节点执行shell命令） | 00000129 |
| 发送文件 | 00000130 |

### 服务中心spider.xml说明

服务中心本身也是一个spider实例，运行在server模式下，其配置除了role属性外，和常规的spider服务端并无差别，如下所示：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<spider>

<nodeName value=*"sc"* cloud=*"false"* role=*"sc"* serviceDefineType=*"ld"*

serviceCenter=*"0.0.0.0"* appVersion=*""* charset=*"UTF-8"* dev=*"true"*/>

<plugins>

<plugin pluginId=*"spider.localService"* serviceTimeout=*"10000"*

zlibCompress=*"false"* encrypt=*"false"* anonymous=*"true"*

serviceProxyPackage=*""*>

<server enable=*"true"* port=*"7070"*

serviceExportPackage=*"io.spider.sc.center.api"* />

</plugin>

<plugin pluginId=*"spider.channel"*>

</plugin>

<plugin pluginId=*"spider.filter"*>

</plugin>

</plugins>

<routeItems consistent=*"true"*>

<routeItem serviceId=*"\*"* clusterName=*"spider.localService"* />

</routeItems>

</spider>

一般来说，尽量不要修改除port和dev外的其他属性值。

## spider内部服务列表

spider中间件保留了一批内部服务号，这些服务号仅供内部使用，应用程序不得使用这些服务号，如若使用了，会导致中间件异常。预留的内部服务号从00000000-00000199。下面是完整的服务列表及其含义：

|  |  |  |
| --- | --- | --- |
| 服务号 | 功能 | 完成状态 |
| 00000001 | 查询路由信息 | 完成 |
| 00000002 | 查询下游服务器 | 完成 |
| 00000003 | 查询客户端连接 | 完成 |
| 00000004 | 查询发布的服务 | 完成 |
| 00000005 | 查询代理的服务 | 完成 |
| 00000006 | 查询当前节点概述信息 | 完成 |
| 00000007 | 查询动态路由缓存 | 完成 |
| 00000008 | 新增服务器节点 | 完成 |
| 00000009 | 新增服务器集群 | 完成 |
| 00000010 | 新增路由条目 | 完成 |
| 00000011 | 删除路由条目 | 完成 |
| 00000012 | 删除服务器节点 | 完成 |
| 00000013 | 删除服务器集群 | 完成 |
| 00000014 | 查询服务运行时性能指标 | 完成 |
| 00000015 | 查询待处理请求队列 | 完成 |
| 00000016 | 清空服务运行时性能指标 | 完成 |
| 00000017 | 服务器节点注册 | 完成 |
| 00000018 | 上报性能指标（会自动过滤0记录） | 完成 |
| 00000019 | 协议验证 | 完成 |
| 00000020 | 客户端身份认证 | 完成 |
| 00000021 | 心跳检测 | 完成 |
| 00000022 | 导出已完成可信请求 | 已开发 |
| 00000023 | 导出已完成可信请求并删除 | 已开发 |
| 00000024 | 导出性能指标到本地文件 | 已开发 |
| 00000025 | 开启服务动态拦截与抓包 | 完成 |
| 00000026 | 上报慢日志 | 已开发 |
| 00000027 | HA持久化存储超时或断开异常通知 | 已开发 |
| 00000028 | 查询慢日志 |  |
| 00000029 | 执行shell命令（调用该命令可以在所在节点执行shell命令） | 已开发 |
| 00000031 | 生成license key | restful已发布 |
| 00000032 | 停止服务动态拦截与抓包 | 完成 |
| 00000033 | 抓包时服务端推送给客户端消息 | 完成 |
| 00000034 | 查询供广播使用的节点基本信息（用于发起者画节点树） | 完成 |
| 00000036 | 查询服务器业务线程请求积压 | 完成 |
| 00000037 | 增加抓包查询，原计划2016年11月25日调整为由原先实现的主推调整为主动请求，最后还是决定新增一个功能号主动查询的方式，这样两这都能兼容。对于C#/python开发的客户端，适合于拉的方式，对于web容器适合于推的方式，由spider.xml配置参数控制哪者生效 | 完成 |
| 00000038 | 查询当前服务器是否进入流控状态。 |  |

## 二进制包与开发SDK

spider完整的发布包包含下列二进制组件包：

|  |  |  |
| --- | --- | --- |
| 包名 | 功能和用途 | Maven坐标 |
| io.spider.jar | spider核心包 | <dependency>  <groupId>io.spider</groupId>  <artifactId>spider</artifactId>  <version>2.0.0-RELEASE</version>  </dependency> |
| io.spider.ext.jar | Spider核心扩展包，比如管理、监控spider运行状态，与服务中心交互等等 | <dependency>  <groupId>io.spider</groupId>  <artifactId>spider.ext</artifactId>  <version>2.0.0-RELEASE </version>  </dependency> |
| io.spider.sc.client.api.jar | 服务中心管理模式下，客户端提供的主要管理功能接口 | <dependency>  <groupId>io.spider</groupId>  <artifactId>spider.sc.client.api</artifactId>  <version>2.0.0-RELEASE</version>  </dependency> |
| io.spider.sc.center.api.jar | 服务中心管理模式下，服务中心端提供的主要功能接口 | <dependency>  <groupId>io.spider</groupId>  <artifactId>spider.sc.center.api </artifactId>  <version>2.0.0-RELEASE </version>  </dependency> |
| ~~io.spider.center.war~~ | ~~spider服务中心应用~~ |  |
| ~~libspiderclientcpp.dll~~ | ~~Spider C++客户端SDK~~ |  |
| ~~libspiderclient.dll~~ | ~~Spider C# SDK~~ |  |

## spider性能测试

### 测试环境部署结构



### 测试用例

|  |  |
| --- | --- |
| 类别 | 说明 |
| 请求报文 | 194字节（{"systemId":"PL","appVersion":"qq.qq.qq","companyId":"12","sourceIp":"DESKTOP-NDOHT79","destIp":"DESKTOP-NDOHT79","spiderOpts":{},"companyName":"机构1","createDate":"20160101","companyType":1}） |
| 响应报文 | 178字节（{"lists":[{"nodeId":100001,"nodeName":"机构100001","companyId":"1200","createDate":"20160101"},{"nodeId":100001,"nodeName":"机构100001","companyId":"1200","createDate":"20160101"}]}）。 |
| 客户端用例 | @RequestMapping(value="/1001.html")  **public** @ResponseBody String t1001(Model model,HttpServletRequest request){  PlatformReq req = **new** PlatformReq().createDemo();  req.setCompanyId("12");  req.setSystemId("PL");  // 之所以每次HTTP请求调用5次spider请求，是因为一开始用单次跑，客户端很快就到cpu 100%，以至于spider NB一直压不到CPU 100%，故调整。  platformService.doQuery(req);  platformService.doQuery(req);  platformService.doQuery(req);  platformService.doQuery(req);  platformService.doQuery(req);  **return** JsonUtils.*toJson*(platformService.doQuery(req));  } |
| 服务端用例 | @Service  **public** **class** PlatformServiceImpl **implements** PlatformService {  **public** PlatformQuery doQuery(PlatformReq req) {  //System.out.println("收到spider请求: " + req.toString());  PlatformQuery query = **new** PlatformQuery();  **return** query.createDemo();  }  } |
| 并发数 | 200 |

### 测试结果

|  |  |
| --- | --- |
| 项目名称 | 结果 |
| 客户端运行次数 | 约1500万 |
| Spider调用次数 | 约7500万 |
| 平均响应时间 | 90毫秒 |
| 客户端QPS | 2199次/秒 |
| Spider NB/NP QPS | 约11000次/秒（2199\*5=10995） |
| Spider客户端负载 | 95%~100% |
| Spider NB负载 | 90%-95% |
| Spider NP负载 | 60%左右 |
| 失败率 | <0.01% |

### 原始数据

