**SCF使用文档**

**2016年4月**

目 录

[前言 4](#_Toc316647520)

[第一章 SCF概述 5](#_Toc316647521)

[1.1 什么是SCF 5](#_Toc316647522)

[1.2 发展历史 5](#_Toc316647523)

[第二章 服务端 6](#_Toc316647524)

[2.1 流程图 6](#_Toc316647525)

[2.2 参数说明 6](#_Toc316647526)

[2.3 SCF文件依赖 8](#_Toc316647527)

[2.4 开发规范 8](#_Toc316647528)

[第三章 客户端 12](#_Toc316647529)

[3.1 流程图 12](#_Toc316647530)

[3.2 参数说明 12](#_Toc316647531)

[3.3 文件依赖 13](#_Toc316647532)

[3.4 开发规范 13](#_Toc316647533)

[第四章 协议 15](#_Toc316647534)

[第五章 序列化 16](#_Toc316647535)

[5.1 Object序列化 16](#_Toc316647536)

[5.2 Array/List/Map序列化 16](#_Toc316647537)

[5.3 基本类型 17](#_Toc316647538)

[5.4 String 17](#_Toc316647539)

[第六章 服务部署 18](#_Toc316647540)

[6.1 设置 18](#_Toc316647541)

[6.2 目录结构 18](#_Toc316647542)

[6.2.1 根目录 18](#_Toc316647543)

[6.2.2 SCF\_v3.4/bin 19](#_Toc316647544)

[6.2.3 service托管服务 20](#_Toc316647545)

[6.3 服务升级 20](#_Toc316647546)

[附录A 常见问题解决方案 21](#_Toc316647547)

[附录B SCF服务监测 22](#_Toc316647548)

前 言

**SCF约定**

本书针对SCF最新版进行讲解，本书针对Java版进行说明，.NET版本暂不包括。

SCF最新版本:

com.bj58.spat.dao.0.0.1

com.bj58.spat.scf.client.1.4.3

com.bj58.spat.scf.protocol.1.4.1

com.bj58.spat.scf.serializer.1.4.4

com.bj58.spat.scf.server-3.4.4

Maven私服:

<http://10.58.120.19:8058/nexus>

**使用约定**

等宽字体

用于示例代码和代码段、类、变量、方法名称、文本内使用的Java关键字、SQL命令、表名、列名及XML元素和标记等。

等宽黑体

某些代码示例中用于强调。

# SCF概述

## 什么是SCF

SCF是服务通信基础(Service Communication Foundation)是具有高性能、高可靠性，提供异步、多协议、事件驱动的网络应用程序框架。

特点：

1）跨平台

2）高效、安全

3）高性能、异步

4) 支持TCP与 HTTP传输协议

## 发展历史

* scf\_v1版

跨平台序列化(支持json格式)

手动生成代理

* scf\_v2版

序列化升级

自动生成代理

客户端重构

telnet监测

* scf\_v3版

http协议支持

热部署

服务“容器化”

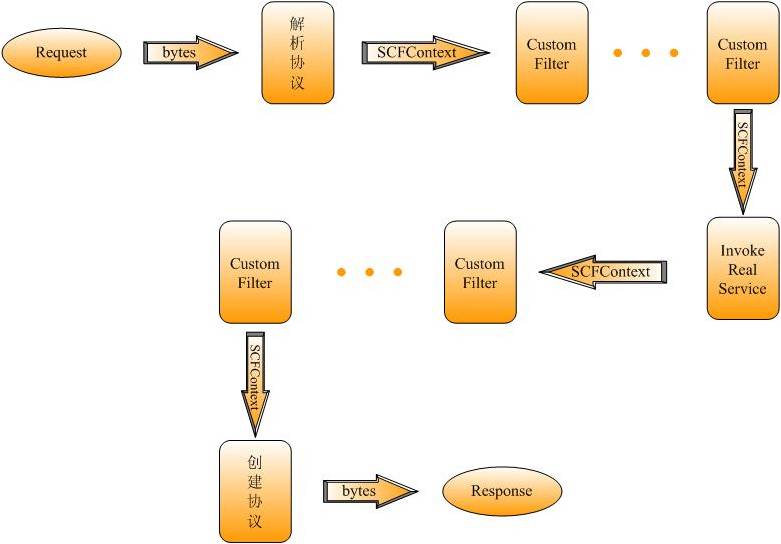
权限控制

服务重启客户端优雅切换

服务异步化

# 服务端

## 流程图



## 参数说明

|  |  |
| --- | --- |
| **字段** | **说明** |
| scf.service.name | 服务名 |
| scf.encoding | 编码格式(默认utf-8) |
| scf.hotdeploy | 是否启动热部署 |
| scf.filter.global.request | request全局过滤 |
| scf.filter.global.response | response全局过滤 |
| scf.filter.connection | 连接过滤 |
| scf.iptable.allow.iplist | 允许访问IP |
| scf.iptable.forbid.iplist | 禁止访问IP |
| scf.init | 初始化类 |
| scf.log.udpserver.ip | UDP服务IP |
| scf.log.udpserver.port | UDP监听端口 |
| scf.log.exectime.limit | UDP日志输出间隔 |
| scf.proxy.invoker.implement | 服务端异步执行类 |
| scf.async.worker.count | 服务端异步执行线程数 |
| scf.servers | 启动服务集合 |
| scf.server.tcp.enable | 是否启动TCP服务 |
| scf.server.tcp.listenPort | TCP监听端口 |
| scf.server.tcp.listenIP | TCP服务IP |
| scf.server.tcp.receiveBufferSize | 返回字节大小(默认1024 \* 64) |
| scf.server.tcp.sendBufferSize | 发送字节大小(默认1024 \* 64) |
| scf.server.tcp.frameMaxLength | TCP服务最大支持长度(默认1024 \* 512) |
| scf.server.tcp.workerCount | TCP服务工作线程数 |
| scf.server.tcp.implement | TCP 启动服务类 |
| scf.server.http.enable | 是否启动HTTP服务 |
| scf.server.http.listenPor | HTTP监听端口 |
| scf.server.http.listenIP | HTTP服务IP |
| scf.server.http.receiveBufferSize | 返回字节大小(默认1024 \* 64) |
| scf.server.http.sendBufferSize | 发送字节大小(默认1024 \* 64) |
| scf.server.http.frameMaxLength | HTTP服务最大支持长度(默认1024 \* 512) |
| scf.server.http.workerCount | HTTP服务工作线程数 |
| scf.server.http.implement | HTTP启动服务类 |
| cf.server.telnet.enable | 是否启动telnet监测服务 |
| scf.server.telnet.listenPort | telnet监听端口 |
| scf.server.telnet.listenIP | telnet服务IP |
| scf.server.telnet.receiveBufferSize | 返回字节大小(默认1024 \* 64) |
| scf.server.telnet.sendBufferSize | 发送字节大小(默认1024 \* 64) |
| scf.server.telnet.frameMaxLength | telnet服务最大支持长度(默认1024 \* 512) |
| scf.server.telnet.workerCount | telnet服务工作线程数 |
| scf.server.telnet.implement | telnet启动服务类 |
| scf.secure | 是否启动权限认证 |

## SCF文件依赖

com.bj58.spat.scf.server-3.4.4-SNAPSHOT.jar   
com.bj58.spat.scf.serializer-1.4.4-SNAPSHOT.jar  
com.bj58.spat.scf.protocol-1.4.1-SNAPSHOT.jar  
com.bj58.spat.utility-3.4.5-SNAPSHOT  
com.bj58.spat.dao-0.0.1-SNAPSHOT.jar

## 开发规范

服务对外暴露的方法签名中有涉及基本类型数组的，必须使用与其对应的对象类型。ex:  
int[] -> Integer[]  
long[] -> Long[]

**数据契约规范**

/\*\*标记表名为T\_News，当DB中的表名与类名不一致时需要指定\*/

@Table(name=“T\_News”)

/\*\*是否需要序列化 \*/

@SCFSerializable(name="XXXXXXX")

@NotDBColumn /\*\*标记该字段不是数据库字段\*/

@Id /\*\*标记该字段为主键字段(不可插入，不可修改)\*/

@SCFMember /\*\*标记该字段为需要序列化字段\*/

**示例:**

@Table(name=“T\_News”) /\*\*标记表名为T\_News \*/

@SCFSerializable

public class News {

/\*\*标记该字段不是数据库字段\*/

@NotDBColumn

private static final long serialVersionUID = 1L;

/\*\*标记该字段为主键字段(不可插入,不可修改)\*/

@Id

/\*\*标记该字段为需要序列化字段\*/

@SCFMember

private int newsID;

@Column(name = "newsTitle") /\*\*标记该字段在数据库中的字段名为newsTitle \*/

@SCFMember

private String title;

public int getNewsID() {

return newsID;

}

public void setNewsID(int newsID) {

this.newsID = newsID;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

}

**备注:**

/\*\*标记为主健字段，可更新，可插入\*/

@Id(insertable=true, updatable=true)

**接口定义**

/\*\*标记该接口对外提供服务\*/

@ServiceContract

public interface INewsService {

@OperationContract /\*\*标记该方法对外暴露 \*/

public News getNewsByID(int newsID) throws Exception;

@OperationContract /\*\*标记该方法对外暴露 \*/

public List<News> getNewsByCateID(int cateID,Out<Integer> totalCount) throws Exception;

}

/\*\*标记该类对外提供服务，服务契约为INewsService \*/

@ServiceBehavior

public class NewsService implements INewsService {

@Override

public List<News> getNewsByCateID(int cateID, Out<Integer> totalCount) throws Exception {

List<News> list = DBHelper.daoHelpers.sql

.getListByCustom(News.class, "\*", "cateid=" + cateID, "");

return list;

}

@Override

public News getNewsByID(int newsID) throws Exception {

return (News) DBHelper.daoHelpers.sql.get(News.class, newsID);

}

}

**错误示例**

@ServiceContract /\*\*标记该接口对外提供服务 \*/

public interface INewsService<**T**> {

@OperationContract /\*\*标记该方法对外暴露\*/

public **T** getNewsByID(int newsID) throws Exception;

}

**备注说明**

1、老版服务端实现时接口和接口实现类不需要@ServiceContract、@ServiceBehavior等注解，通过服务文件中的serviceframe.xml一一匹配。新版服务端通过注解的方式实现，不需要在serviceframe.xml中配置，**注意不要两者一起混合使用。**

2、针对有些服务需要写filter时，在服务对应的scf\_config.xml中增加配置

<property>

<name>scf.filter.global.request</name>

<value append="true">具体实现类</value>

</property>

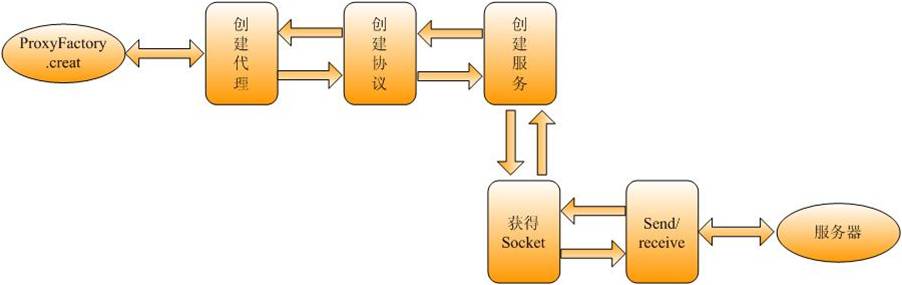
3、@SCFSerializable(name="XXXXX")

如果一个项目中有多个相同类名的实体类需要序列化时，要增加别名name=”XXXXX”

如果服务端实体类增加了name="XXXXX"属性，客户端实体类也需要增加该属性。

# 客户端

## 流程图



## 参数说明

**Service**

|  |  |
| --- | --- |
| name | 远程服务名称 |
| id | 服务ID |
| maxThreadCount | 最大线程连接数 |

**SocketPool**

|  |  |
| --- | --- |
| bufferSize | 网络receive传输流缓存大小(最小10KB) |
| minPoolSize | 连接池中最小连接数量 |
| maxPoolSize | 连接池中最大连接数量 |
| nagl | 是否启用Nagle算法 |
| autoShrink | 是否启动连接自动回收 |
| sendTimeout | 发送超时时间 |
| receiveTimeout | 接收超时时间 |
| waitTimeout | 当连接池中没有连接时需要等待释放连接的时间 |
| maxPakageSize | 网络send传输流缓存大小(默认50KB) |
| reconnectTime | 超时重新连接次数 |
| protected | 是否根据请求数量自动调节超时时间.默认为true |

**Protocol**

|  |  |
| --- | --- |
| serialize | 序列化方式(scf、json) |
| encoder | 序列化采用的编码格式 |
| compressType | 是否压缩(当前版本不支持压缩，默认为UnCompress) |

**Server**

|  |  |
| --- | --- |
| deadTimeout | 服务器心跳检测间隔时间 |
| name | 服务名称 |
| host | 服务IP |
| port | 服务端口 |
| maxCurrentUser | 最大当前用户数，超过此数将服务器设置为busy状态 |

**Secure**

|  |  |
| --- | --- |
| info | Key内容 |

## 文件依赖

com.bj58.spat.scf.client-1.4.4-SNAPSHOT.jar   
com.bj58.spat.scf.serializer-1.4.4-SNAPSHOT.jar  
com.bj58.spat.scf.protocol-1.4.1-SNAPSHOT.jar

## 开发规范

1、客户端服务接口与方法名必须一致

1. 实体里边的字段必须一一对应，大小写需一致

3、Java实体里对boolean/Boolean类型的字段get/set方法需使用完整名。 ex：

private boolean isClose;

public boolean getIsClose(){…}

说明: 这里的方法名必须得getIsClose()不能是isClose

public void setIsClose(){…}

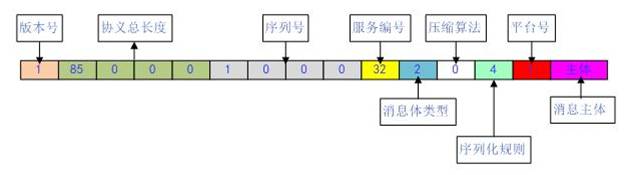
1. Java接口中方法签名的参数类型数组必须用对象类型。

示例: Long[],Integer[]而不能用long[],int[]

**示例:**SCFInit.init(“E:/scf.config”);/\*\* 设置客户端读取配置文件路径\*/   
INewsService newsService =   
ProxyFactory.create(INewsService.class, "tcp://demo/NewsService");

**说明:**  
1、INewsService 接口类  
2、tcp://demo/NewsService(demo:服务名 NewsService:接口实现类)

# 协议



|  |  |
| --- | --- |
| **编码** | **说明** |
| 009 | tab |
| 010 | 换行键(line feed) LF |
| 013 | 回车键 CR |
| 017 | ctrl键 键盘控制码 |
| 018 | alt 键 键盘控制码 |

版本号：默认为1

序列号：SessionId 每次自加,当数值大于1024 \* 1024 \* 1024时重新赋值为1

服务编号：客户端配置文件中serviceid

消息体类型：Response(1),Request(2),Exception(3),Config(4),Handclasp(5),Reset(6);

压缩算法：不压缩UnCompress(0),SevenZip(1), DES(2);

序列化规则：JSON(1),JAVABinary(2),XML(3),SCFBinary(4);

平台号：Dotnet(0),Java(1),C(2);

# 序列化

## Object序列化

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Object** | | | | | | |
| 4 | 1 | 4 | 1 | 4N(N>0) | 属性类型Id | 数据 |
| 属性类型Id | 数据 |
| 类型Id | 是否引用 | Hashcode | 泛型参数数量 | 泛型参数类型Id | 数据 | |
| 备注:如果第5个字节是1那么只写入该对象的Hashcode, 如果类型是泛型时才输出灰色部分 | | | | | | |

## Array/List/Map序列化

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Array/List/Map** | | | | | |
| 4 | 1 | 4 | 4 | 属性类型Id | 数据 |
| 属性类型Id | 数据 |
| 类型编码 | 是否引用 | Hashcode | 数组长度 | 数据 | |

**ArraySerializer**

**WriteObject**

1、写入4byte的类型编码

2、写入1byte判断是否为引用类型，1为引用类型(如果输入值为null,写入1,0返回,获取当前输入值的hashcode ,如果包含当前hashcode则写入1，hashcode并返回,如果不包含写入0，hashcode)

3、根据类型编码获取class,判断属于什么类型,根据不同类型分别调用不同类型的序列化方式

4、写入数据长度

5、写入数据内容

**ReadObject**

1、读取4byte获得类型编码，如果类型编码为0返回null

2、读取1byte判断判断是否为引用类型，如果获取值大于0为则引用类型,返回当前hashcode对应object

3、读取4byte获得hashcode

4、根据类型编码获取class,判断属于什么类型，根据不同类型分别调用不同类型的序列化方式

6、获取数据长度

7、获取数据内容

## 基本类型

|  |  |  |  |
| --- | --- | --- | --- |
| **基本类型** | | | |
| Byte | 1 | Int16 | 2 |
| Int32 | 4 | Int64 | 8 |
| Float | 8 | Double | 8 |
| Char | 2 | DateTime | 8 |
| Boolean | 1**(**0:false 1:true**)** | | |

**EnumSerializer**

**WriteObject**

1、写入4byte类型编码

2、调用String序列化方式进行序列化

**ReadObject**

1、读取4byte类型编码

2、根据类型编码获得class

3、根据class类型,String发序列化方式读取

4、Enum.valueOf(classType, value)返回

## String

|  |  |  |  |
| --- | --- | --- | --- |
| **String** | | | |
| 1 | 4 | 4 | N |
| 是否引用 | Hashcode | 总长度 | 数据 |

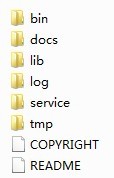
# 服务部署

## 设置

* 设置Java环境变量
* 安装scf\_v3.4
* 部署服务 scf\_v3.4/server/deploy/服务名/
* 启动服务

## 目录结构

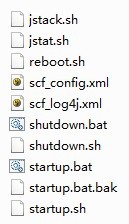
### 根目录



说明:

1. bin:启动脚本，配置文件
2. docs:文档
3. SCF服务框架jar
4. log:日志文件
5. service:托管服务
6. tmp:服务PID文件
7. COPYRIGHT:版权说明
8. README:说明

### SCF\_v3.4/bin



jstack.sh：输出堆栈信息脚本

Describe

Print stack info

jstack.sh [<server-name | pid>] [<print count>] [<print url>]

<server-name> this server name

<pid> this server pid

<print-count> stack info print count

<print-url> stack info print url ex:/opt/scf\_v3.4/log

if url is ex so print real url is /opt/scf\_v3.4/log/jt$PID\_$COUNT

jstat.sh：输出JVM堆信息脚本

Describe

Print heap info

jstak.sh [-<option>] [<server-name | pid>] [<interval-time> ms] [<count>]

<option> -gc | -class | -gccause | -gcutil | -gcpermcapacity | -gccapacity

-gcnew | -gcnewcapacity | -gcold | -gcoldcapacity

-compiler | -printcompilation

default -gc

<server-name> this server name

<pid> this server process id

<interval-time> Sampling interval The following forms are allowed:<n>['ms']

Where <n> is an integer and the suffix specifies the units as milliseconds('ms') .

The default units are 'ms'.

reboot.sh：重启脚本

reboot this server

reboot.sh <service-startscript> <time> s

service-startscript:this script name is servicestartscript name

example:

umc\_run.sh (reboot.sh umc\_run 10)

imc\_start.sh (reboot.sh imc\_start 10)

scf\_config.xml：服务全局配置文件

scf\_log4j.xml：服务全局log4j配置文件

shutdown.sh/bat：关闭服务脚本

startup.sh/bat：启动服务脚本

### service托管服务

deploy：热布署目录

lib：所有服务公用组件(jdbc驱动, memcache\_client等)

## 服务升级

1、升级scf\_v3.4

将scf\_v3.4拷贝至服务端

2、将原SCF容器\service\deploy\路径下各服务文件拷贝至scf\_v3.4\service\deploy\路径

3、修改scf\_v3.4\service\deploy\路径下各服务文件中的 scf\_log4j.xml 日志输出路径

4、将原SCF容器\bin\路径下 \*\*\*\_start.sh或\*\*\*\_run.sh 文件拷贝至scf\_v3.4\bin\目录

5、shutdown原服务。

6、启动scf\_v3.4\bin\目录下对应服务.

# 附录A 常见问题解决方案

1、操作数据库找不到DAOBase类

请在当前的项目中导入com.bj58.spat.dao-1.0.0-SNAPSHOT.jar

2、客户端读取本地配置文件空指针

System.out.println(SCFConst.CONFIG\_PATH);打印配置文件路径，请查看路径是否正确，或者客户端scf\_config.xml配置是否正确

3、客户端调用问题

tcp://服务名/接口实现类

服务名：与scf\_config.xml中服务器配置服务名相同

1. SCF jar包版本冲突

查看当前项目是否引用多个版本的SCF jar包。maven项目之间依赖要注意依赖项目之间jar包版本差别。

5、java.lang.classNotFoundExceptoon:java.lang.Long,java.lang.Boolean

签名方法包含泛型格式

1. 枚举类型序列化异常 (java.lang.IllegalArgumentException: No enum const class)

Enum.valueof(类，String 参数),String 必须为枚举值而不是代表的数字，

或者枚举类中有对toString的重载返回的是数字类型

all(2) 参数必须为all 而不能为2，或者不能重载toString 返回数字2

1. 序列化异常com.bj58.spat.scf.serializer.component.exception.com.bj58.spat.scf.serializer.component.exception.DisallowedSerializeException

序列化没有标注序列化注解的类就报如上错误

1. com.bj58.spat.scf.server.contract.entity.Out无法找到该类

请更换为com.bj58.spat.scf.protocol.sdp.Out,当前类在协议jar中。

1. java.lang.NoClassDefFoundError:com/bj58/spat/scf/server/protocol/sdp/SDPBase

更换com.bj58.spat.scf.server.contract-1.1-SNAPSHOT为协议jar

10、org.xml.sax.SAXParseException: Content is not allowed in prolog 出现该问题的原因是读取的xml中可能有非法字符或特殊字符，或者编码格式不同照成的，如果是读取scf.config引起的该问题，请先确定scf.config的路径，然后再排查scf.config编码格式和字符编码问题。System.out.println(SCFConst.CONFIG\_PATH);输出当前项目读取scf.config的路径。

# 附录B SCF服务监测

1. **SCF分析功能**

count[|second num|method methodName]

\* show method call times in num seconds

\* second : in num seconds statistics once (num default 1)

\* method : for statistics method

\* example : count

\* example : count|second 3

\* example : count|second 3|method getInfo

time|grep abc[|group num|column -tkda]

\* show method execute time

\* grep : condition

\* group : method called num times show statistics once

\* column : show column a->all t->time k->key d->description

\* example: time|grep getInfo

\* example: time|grep getInfo|group 10|column -tk

exec|top

|netstat -na

\* exec command (at present only allow:top or netstat)

\* example: exec|top

control \* use for control scf-server

help \* show help

quit \* quit monitor

**使用示例**

使用方式以imc为例，imc对外提供的监控端口为26003，该端口可以通过查看 scf/service/deploy/服务名/scf\_config.xml

<!-- telnet server listent port -->

<property>

<name>scf.server.telnet.listenPort</name>

<value>26003</value>

</property>

1、telnet 192.168.9.120 26003 先telnet进去

2、查看getInfo方法的执行情况：

time|grep getInfo回车

3、查看所有方法的执行情况：

time|grep \_回车

输出结果如下：

time:3ms--key:InvokeRealService\_InfoProviderWithIndex.GetInfo--description:protocol version:1

fromIP:/10.3.12.20:49545

lookUP:InfoProviderWithIndex

methodName:GetInfo

params:

--key:Long[]

--value:[Ljava.lang.Long;@40880562

--key:String

--value:\*

--key:String

--value:

--key:String

--value:

4、查看服务并发数：  
count回车

5、退出quit

6、要想保存结果可以在telnet的时候把输出重定向到一个文件中

例：telnet 192.168.9.120 26003 > tmp

1. **JVM检测**

**2.1**、使用scf\_v3.4/bin下jstat.sh脚本查看JVM各生代使用情况，具体使用请查看具体脚本。

**2.2、jmap**

jmap -heap <pid> 查看当前进程堆内存使用情况

jmap -histo:live <pid> 存活对象情况

jmap -histo <pid> 对象数量大小

jmap -permstat <pid> PermGen情况

jmap dump:format=b,file= 输出对象情况分析

备注:

 jmap -dump:live 及 jmap -histo:live 会执行full gc

1. **JVM线程监控**

使用scf\_v3.4/bin下jstack.sh脚本查看线程信息，具体使用请查看具体脚本。

1. **pidstat**

**该命令需要安装sysstat软件包,**

pidstat 2 5 #每隔2秒显示5次所有活动进程的CPU 使用情况

pidstat -p 1643 2 5 #每隔2秒显示5次PID为1643的进程的CPU使用情况显示

pidstat -p 1643 2 5 - r #每隔2秒显示5次PID为1643的进程的内存使用情况显示

备注:pidstat命令没有保存性能数据的功能

1. **netstat命令**

功能说明：Netstat用于显示与IP、TCP、UDP和ICMP协议相关的统计数据，一般用于检验本机各端口的网络连接情况。

语　　法：netstat [-acCeFghilMnNoprstuvVwx][-A<网络类型>][--ip]

补充说明：利用netstat指令可让你得知整个Linux系统的网络情况。

参　　数：

-a或–all 显示所有连线中的Socket。

-A<网络类型>或–<网络类型> 列出该网络类型连线中的相关地址。

-c或–continuous 持续列出网络状态。

-C或–cache 显示路由器配置的快取信息。

-e或–extend 显示网络其他相关信息。

-F或–fib 显示FIB。

-g或–groups 显示多重广播功能群组组员名单。

-h或–help 在线帮助。

-i或–interfaces 显示网络界面信息表单。

-l或–listening 显示监控中的服务器的Socket。

-M或–masquerade 显示伪装的网络连线。

-n或–numeric 直接使用IP地址，而不通过域名服务器。

-N或–netlink或–symbolic 显示网络硬件外围设备的符号连接名称。

-o或–timers 显示计时器。

-p或–programs 显示正在使用Socket的程序识别码和程序名称。

-r或–route 显示Routing Table。

-s或–statistice 显示网络工作信息统计表。

-t或–tcp 显示TCP传输协议的连线状况。

-u或–udp 显示UDP传输协议的连线状况。

-v或–verbose 显示指令执行过程。

-V或–version 显示版本信息。

-w或–raw 显示RAW传输协议的连线状况。

-x或–unix 此参数的效果和指定”-A unix”参数相同。

–ip或–inet 此参数的效果和指定”-A inet”参数相同

netstat 的一些常用选项

netstat –s

本选项能够按照各个协议分别显示其统计数据。如果我们的应用程序（如Web浏览器）运行速度比较慢，或者不能显示Web页之类的数据，那么我们就可以用本选项来查看一下所显示的信息。我们需要仔细查看统计数据的各行，找到出错的关键字，进而确定问题所在。

netstat –e

本选项用于显示关于以太网的统计数据。它列出的项目包括传送的数据报的总字节数、错误数、删除数、数据报的数量和广播的数量。这些统计数据既有发送的数据报数量，也有接收的数据报数量。这个选项可以用来统计一些基本的网络流量）。

netstat –r

本选项可以显示关于路由表的信息，类似于后面所讲使用route print命令时看到的 信息。除了显示有效路由外，还显示当前有效的连接。

netstat –a

本选项显示一个所有的有效连接信息列表，包括已建立的连接（ESTABLISHED），也包括监听连接请（LISTENING）的那些连接。

netstat –n

显示所有已建立的有效连接。

1. 序列化对象必须有空构造方法

Ex:

Public 类名(){}

HTTP 服务规范

1. 接口和接口实现类 要遵循SCF整体规范

@ServiceContract //标记该接口对外提供服务

public interface INewsService {

@OperationContract //标记该方法对外暴露

public News getNews(int newsID) throws Exception;

}

@ServiceBehavior

public class RestService implements INewsService {

@Override

public News getNews(int newsID) throws Exception {

return null;

}

}

1. A
2. A