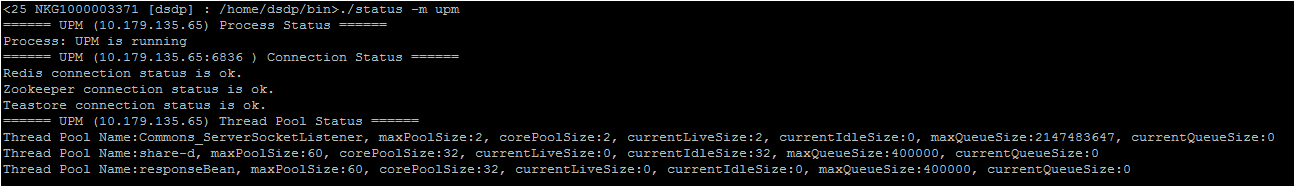
# 组件简介

## 功能介绍

status命令增加输出线程池和队列的使用情况，打印格式如下：



# 涉及到的外部组件

## 无

# 配置文件和API

## 配置文件改动

### dsf.properties配置改动

dsf.properties文件中增加如下两个配置项，两个配置项配置用来获取DSF侧的线程池使用情况:

*############################################*

*# DSF threadpool monitor config*

*############################################*

threadpool.monitor.switch**=**true

*# unit(s)*

threadpool.monitor.interval**=**30

### Spring配置文件加载

配置加载如下路径的spring配置文件：classpath:commons/utils/spring/ commons.utils.service.xml.如果以前已经加载过或者已有的通配能够加载到的话，则忽略此步骤。

### pom.xml改动

commons新增了一个jar包，添加依赖该jar包：

<dependency>

<groupId>com.huawei.jaguar</groupId>

<artifactId>commons.utils</artifactId>

<version>V300R003C60B608-SNAPSHOT</version>

</dependency>

## 组件侧线程池代码改动

为了在commons侧能够获取组件自己维护的线程池对象，进而获取最大线程数，活动的线程数等数据，组件侧使用线程池对象的地方需要做如下改动：

目前各个组件获取线程池对象的方式主要是如下四种，每种方式都需要改动下，改成Commons组提供的获取线程池对象的方式：

1, 代码中通过调用Executors.newFixedThreadPool(…),Executors.newCachedThreadPool(…)等JDK的Executors中的静态方法的方式来获取线程池对象

代码改成：CommonsExecutors.newFixedThreadPool(threadPoolName,…),CommonsExecutors. newCachedThreadPool(threadPoolName,…)等通过调用Commons组提供的CommonsExecutors中的静态方法来获取线程池对象。

2, 代码中通过new ThreadPoolExecutor()或者new ScheduledThreadPoolExecutor()等直接new线程池对象的方式获取

代码改成：CommonsExecutors.newFixedThreadPool(threadPoolName,…),CommonsExecutors. newCachedThreadPool(threadPoolName,…)等通过调用Commons组提供的CommonsExecutors中的静态方法来获取线程池对象。

3, 配置一个ThreadPoolExecutor或者ScheduledThreadPoolExecutor类型的Bean的方式

Bean配置方法改成：

<bean id=*"threadPools"*

class=*"*com.huawei.jaguar.commons.utils.concurrent*.*CommonsExecutors*"*

factory-method=*"*newFixedThreadPool*"*>

<constructor-arg name=*"*threadPoolName*"* value=*"moudlename\_xxxx\_threadpool"* />

<constructor-arg name=*"maxPoolSize"* value=*"${up.xxxx.maxPoolSize}"* />

<constructor-arg name=*"corePoolSize"* value=*"${up.xxxx.corePoolSize}"* />

……

</bean>

*Moudlename代表组件名字，xxxx代表线程池的用途*

4，配置一个ThreadPoolTaskExecutor类型的Bean(ThreadPoolTaskExecutor是Spring提供的)的方式

Bean配置方法改成：

<bean id=*"threadPools"*

class=*"*com.huawei.jaguar.commons.utils.concurrent*.*CommonsExecutors*"*

factory-method=*"*newThreadPoolTaskExecutor*"*>

<constructor-arg name=*"*threadPoolName*"* value=*"moudlename\_xxxx\_threadpool"* />

<constructor-arg name=*"maxPoolSize"* value=*"${up.xxxx.maxPoolSize}"* />

<constructor-arg name=*"corePoolSize"* value=*"${up.xxxx.corePoolSize}"* />

……

</bean>

*Moudlename代表组件名字，xxxx代表线程池的用途*

上述四种方式都是调用的commons组的CommonsExecutors中的静态方法来去获取线程池对象，CommonsExecutors的本质也是调用JDK的Executors来获取线程池对象，只不过多了一个线程池名称（threadPoolName）的参数，该参数用来唯一标识一个线程池，线程池名称不能重复，各个组件在给线程池起名字时需要遵循如下原则：*组件名称\_线程池的用途（哪个业务功能中用到的线程池）\_threadpool*

### 配置文件内容示范

### 配置文件说明

## 获取连接池使用数据代码实现

Status命令还需要打印Oracle，Redis连接池的使用情况，从收集各个组件的Oracle，Redis连接池对象在代码中的使用情况来看，使用方式，线程池对象是否对外暴露各个组件都不太一样，Commons侧很难自动获取这些连接池对象，从而自动获取活动连接数等数据，只能放到组件侧自己获取，然后Commons来取。组件侧需要按照如下方式改动：

1. 配置maven依赖Commons的commons.utils.jar
2. 实现该包中的com.huawei.jaguar.commons.utils.connectionpool.monitor. IConnectionPoolMonitorService接口中的两个方法

实现类的大概实现举例如下，各个组件根据实际情况调整实现：

public class ConnectionPoolMonitorServiceImpl implements IConnectionPoolMonitorService

{

//从收集代码来看，组件用到Oracle数据库的，连接池都是直接或者间接（自己又封装了下）使用的BasicDataSource，把连接池对象通过Spring注入或者其他方式传入进来。

private org.apache.commons.dbcp.BasicDataSource oracleDataSource;

//从收集代码来看，各个组件的连接Redis的客户端对象都是继承的JedisClientCluster接口，该接口中的Pool<? **extends** ShardedJedis> getShardedJedisPool()方法可以获取连接池对象，Pool类中有对应的方法可以获取活动连接数等数据

private redis.clients.util.Pool redisPool;

@Override

List<ConnectionPoolMonitorResponse> getRedisConnPoolMonitorResponse()

{

//如果组件没有用到redis，则直接返回null

//有多少redis连接池就要返回多少个ConnectionPoolMonitorResponse

List<ConnectionPoolMonitorResponse> list = new ArrayList< ConnectionPoolMonitorResponse>(10);

ConnectionPoolMonitorResponse rsp = new ConnectionPoolMonitorResponse();

rsp.setConnectionPoolName(connPoolName); ---连接池的名字遵循:*组件名称\_redis\_连接池的用途（哪个业务功能中用到的连接池）\_connectionpool，如果组件中只有一个Redis连接池，也可以直接起名为：组件名称\_redis\_connectionpool，有多个连接池的话需要用“连接池的用途（哪个业务功能中用到的连接池）“进一步区分下。*

*rsp.*setMaxActive(redisPool.internalPool.getMaxIdle());

*…*..封装其他属性

*rsp*.setCurrentActive(redisPool.getNumActive());

….. 封装其他属性

list.add(rsp);

return list;

}

@Override

List<OracleConnPoolMonitorResponse> getOracleConnPoolMonitorResponse()

{

//如果组件没有用到oracle，则直接返回null

//有多少oracle连接池就要返回多少个OracleConnPoolMonitorResponse

List<OracleConnPoolMonitorResponse > list = new ArrayList< OracleConnPoolMonitorResponse>(10);

OracleConnPoolMonitorResponse rsp = new OracleConnPoolMonitorResponse();

rsp.setConnectionPoolName(connPoolName); ---连接池的名字遵循:*组件名称\_oracle\_连接池的用途（哪个业务功能中用到的连接池）\_connectionpool，如果组件中只有一个Oracle连接池，也可以直接起名为：组件名称\_oracle\_connectionpool，有多个连接池的话需要用“连接池的用途（哪个业务功能中用到的连接池）“进一步区分下。*

*rsp.*setMaxActive(oracleDataSource.getMaxIdle());

*…*..封装其他属性

return list;

}  
  
}

1. 将实现类配置一个Spring Bean，并加载到Spring上下文中，Commons侧会通过Spring的getBeansOfType(IConnectionPoolMonitorService.**class**)从上下文中查找该Bean，然后调用接口中的方法获取Oralce和Redis的连接池数据。

## 使用指导

# Maven依赖

场景样例为组件插件的一类功能实现过程，包括场景描述、开发过程。

## 依赖说明

## 依赖示例

# 附件

## 一些示范配置文件

# 测试

## 测试

执行status –m all命令打印组件连接状态和线程池使用情况的信息。

其中threadpoolname为share-d和responesebean的两个线程池是DSF侧的线程池。