ns4_gear_watchdog 监控项目说明文档

该项目主要实现功能有两部分：

1、 系统类数据监控

ns4_gear_watchdog 是一个 java 项目，作为一个主线程启动， watchdogServer 作为入口，启动该 main 方法即可。

服务的启动 main 方法:

```
package com.creditease.ns4.gear.watchdog.monitor;

import com.creditease.ns.log.NsLog;
import com.creditease.ns4.gear.watchdog.common.log.NsLogger;
import com.creditease.ns4.gear.watchdog.monitor.jmx.Jmx;
import com.creditease.ns4.gear.watchdog.monitor.process.WatchdogChildTask;
import com.creditease.ns4.gear.watchdog.monitor.process.shutdown.ShutdownHandler;

/**
 * @author outman
 * @description Watchdog 服务入口
 * @date 2019/1/15
 */
public class WatchdogServer {
    private static final NsLog logger = NsLogger.getWatchdogLogger();
    private Long startTime = System.currentTimeMillis();
    private static WatchdogServer server = new WatchdogServer();
    public static WatchdogServer instance() {
        return server;
    }

    public void start() {
        // 注册停止进程处理信号
        new ShutdownHandler().registerSignal();
        // 启动子进程服务
        WatchdogChildTask.getInstance().init();
        // 启动 JMX 服务
        Jmx.getInstance().start();
        // 输出启动 info
        logger.info("============== WatchdogServer started ==============");
    }

    public static void main(String[] args) {
        WatchdogServer.instance().start();
    }
```

```
    public WatchdogChildTask getWatchdogChildTask() {
        return WatchdogChildTask.getInstance();
    }

    public Long getStartTime() {
        return this.startTime;
    }
}
```

在 ns4_gear_watchdog 启动时会首先初始化一些参数，jvmOpts 虚拟机参数设置、appName 应用名、serverMain 方法名等相关参数。这些参数的就是目标程序的运行配置，也就是子线程。ns4_gear_watchdog 可以控制子线程的创建和销毁，并且使用 jmx 实现通讯，进而拿到系统运行的相关参数。

2、 业务类数据监控

业务监控采用的是非植入代码式监控方法，这就要求在不植入代码的情况下获取到程序运行中的业务数据，本项目采用，编写 plugin 插件形式代码，由 NS4.0 框架启动加载的方式，通过业务顺序流转过程中的操作，框架在处理业务流程时候，将业务数据获取出来，以实现业务数据的获取。

业务监控类必须实现 MonitorPlugin 接口，否则将不能接收数据，二期要实现 load 方法。创建私有的 Listener 监听的方式获取框架发送来的数据。

BusinessPlugin 业务数据接收代码示例：

```
public class BusinessPlugin extends AbstractPlugin implements MonitorPlugin {
    private static final NsLog logger = NsLogger.getWatchdogPluginLogger();
    private List<MonitorListener> monitorListeners;

    @Override
    public List<MonitorListener> getMonitorListeners() {
        return monitorListeners;
    }
    @Override
    public void load(NSEnvironment nsEnvironment) {
        logger.info("BusinessPlugin.load");
        monitorListeners = new ArrayList<>();
        MonitorTradeListeners tradeListeners = new MonitorTradeListeners();
        monitorListeners.add(tradeListeners );
        this.loaded = true;
    }
    @Override
```

```
    public void unload() {
        logger.info("卸载完毕");
    }
}
```

监听类也要实现 MonitorListener 接口，并实现 processMonitorEvent 方法。然后根据自己的业务需求,将数据解析和保存需要的格式,或者通过 jmx 展示出来。

```
public class MonitorTradeListeners implements MonitorListener {
    public static Map<String,List<MonitorEvent>> tradeMessageMap;
    private static final NsLog logger = NsLogger.getWatchdogPluginLogger();
    @Override
    public void processMonitorEvent(MonitorEvent monitorEvent) {
        MonitorDataFrame dataFrame = MonitorDataFrame.getInstance();
        //业务处理代码
        ……..
    }
}
```