LTS 使用说明文档

一、*项目结构描述*

项目使用 maven 构建：

1. lts-admin: LTS 的 WEB 管理后台
2. lts-core: LTS 核心包，各个模块基本上都引用这个
3. lts-jobclient: LTS 中 JobClient 角色需要引入的工程
4. lts-jobtracker: LTS 中 JobTracker 角色需要引入的工程
5. lts-tasktracker: LTS 中 TaskTracker 角色需要引入的工程
6. lts-logger: 任务日志记录工程，中间包括 api 和 console, mysql, mongo 三种实现
7. lts-queue: LTS 的任务存储队列，包括 api 和 mysql, mongo 两种实现
8. lts-example: 测试例子，包括 JobClientTest, JobTrackerTest, TaskTrackerTest 启动这三个测试类，就可以查看任务执行效果

二、*怎么使用 jar*

1. 如果你的项目是 maven 构建的话，需要将上面这些工程的 jar 包上传到你们自己的 maven 服务器上。然后再引入相应的 jar 即可。上传的话，可以直接在工程的 pom.xml 文件中添加 distributionManagement 即可。譬如:

```
<distributionManagement>
    <repository>
        <id>nexus-releases</id>
        <name>Local Nexus Repository</name>
        <url>http:// maven.xxx.com /nexus/content/repositories/releases</url>
    </repository>
    <snapshotRepository>
        <id>nexus-snapshots</id>
        <name>Local Nexus Repository</name>
    <url>http://maven.xxx.com/nexus/content/repositories/snapshots</url>
    </snapshotRepository>
</distributionManagement>
```

然后通过 maven 的 deploy 命令上传上去即可。

2. 如果你是直接使用 jar 包的方式，那么你先需要安装 maven，然后通过 maven 来编译整个工程，然后将编译出来的 jar 包引入即可。

三、*部署建议以及 jar 包引入*

注册中心： 二选一

根据参数设置决定

```
setRegistryAddress("zookeeper://127.0.0.1:2181")
setRegistryAddress("redis://127.0.0.1:6379")
```

zookeeper:

可选： zkclient 和 curator 两种

```
<dependency>
    <groupId>com.github.sgroschupf</groupId>
    <artifactId>zkclient</artifactId>
```

```
                    <version>${zkclient.version}</version>
                </dependency>
                <dependency>
                    <groupId>com.netflix.curator</groupId>
                    <artifactId>curator-framework</artifactId>
                     <version>${curator.version}</version>
                </dependency>
          配置: jobTracker.addConfig("zk.client", "zkclient");
                  // 可选值：zkclient(默认), curator
        redis:
                <dependency>
                    <groupId>redis.clients</groupId>
                    <artifactId>jedis</artifactId>
                    <version>${jedis.version}</version>
                </dependency>
```

1. 对于 JobTracker 工程，建议独立部署。

需要引入 jar 包：

```
<dependency>
    <groupId>com.lts</groupId>
    <artifactId>lts-jobtracker</artifactId>
    <version>${lts.version}</version>
</dependency>
```

日志记录，三选一：

具体使用哪个由传入配置决定：

```
jobTracker.addConfig("job.logger", "mongo");
// 可选值: mongo, mysql, console (默认)
<dependency>
    <groupId>com.lts</groupId>
    <artifactId>lts-logger-mongo</artifactId>
    <version>${lts.version}</version>
</dependency>
<dependency>
    <groupId>com.lts</groupId>
    <artifactId>lts-logger-console</artifactId>
    <version>${lts.version}</version>
</dependency>
<dependency>
    <groupId>com.lts</groupId>
    <artifactId>lts-logger-mysql</artifactId>
    <version>${lts.version}</version>
</dependency>
```

队里存储，二选一：

具体使用哪个由传入配置决定：

```
jobTracker.addConfig("job.queue", "mongo");
// 可选值: mongo, mysql (默认)
<dependency>
    <groupId>com.lts</groupId>
    <artifactId>lts-queue-mongo</artifactId>
    <version>${lts.version}</version>
</dependency>
<dependency>
    <groupId>com.lts</groupId>
    <artifactId>lts-queue-mysql</artifactId>
    <version>${lts.version}</version>
</dependency>
```

2. 对于 TaskTracker 工程，这个因为是跑任务的，具体看业务场景，一般情况下也是独立部署

```xml
<dependency>
    <groupId>com.lts</groupId>
    <artifactId>lts-tasktracker</artifactId>
    <version>${lts.version}</version>
</dependency>
```

FailStore, 三选一：

```xml
<dependency>
    <groupId>com.sleepycat</groupId>
    <artifactId>je</artifactId>
    <version>${sleepycat.version}</version>
</dependency>
<dependency>
    <groupId>org.rocksdb</groupId>
    <artifactId>rocksdbjni</artifactId>
    <version>${rocksdbjni.version}</version>
</dependency>
<dependency>
    <groupId>org.fusesource.leveldbjni</groupId>
    <artifactId>leveldbjni-all</artifactId>
    <version>${leveldbjni.version}</version>
</dependency>
```

配置：

```
jobClient.addConfig("job.fail.store", "leveldb");
// 可选值: leveldb, rocksdb, berkeleydb
```

3. 对于 JobClient 工程，这个是提交任务的工程，一般是和业务相关的，所以会放在业务工程中，当然也要看业务场景

```xml
<dependency>
    <groupId>com.lts</groupId>
    <artifactId>lts-jobclient</artifactId>
    <version>${lts.version}</version>
</dependency>
```

FailStore, 三选一：

```xml
<dependency>
    <groupId>com.sleepycat</groupId>
    <artifactId>je</artifactId>
    <version>${sleepycat.version}</version>
</dependency>
<dependency>
    <groupId>org.rocksdb</groupId>
    <artifactId>rocksdbjni</artifactId>
    <version>${rocksdbjni.version}</version>
</dependency>
<dependency>
    <groupId>org.fusesource.leveldbjni</groupId>
    <artifactId>leveldbjni-all</artifactId>
    <version>${leveldbjni.version}</version>
</dependency>
```

配置：

```
jobClient.addConfig("job.fail.store", "leveldb");
```

```
// 可选值: leveldb, rocksdb, berkeleydb
```

4. 对于 lts-admin 工程，这个是一个 web 控制台，可以选择部署也可以选择不部署。非必要

配置 lts-admin-config.properties 文件既可

四、Quick Start

1. JobTracker 示例

```
JobTracker jobTracker = new JobTracker();
// 节点信息配置
jobTracker.setRegistryAddress("zookeeper://127.0.0.1:2181");
// jobTracker.setRegistryAddress("redis://127.0.0.1:6379");
// jobTracker.setListenPort(35002); // 默认 35001
jobTracker.setClusterName("test_cluster");   // 三种节点都要保持一致

// master 节点变化监听器，当有集群中只需要一个节点执行某个事情的时候，可以监听这个事件
jobTracker.addMasterChangeListener(new MasterChangeListenerImpl());

// 设置业务日志记录，  可选值: mongo, mysql , console, 推荐使用 mongo
jobTracker.addConfig("job.logger", "mongo");
// 任务队列用 mongo
jobTracker.addConfig("job.queue", "mongo");
// mongo 配置
jobTracker.addConfig("mongo.addresses", "127.0.0.1:27017");     // 多个地址用逗号分割
jobTracker.addConfig("mongo.database", "lts");
// mysql 配置
// jobTracker.addConfig("jdbc.url", "jdbc:mysql://127.0.0.1:3306/lts");
// jobTracker.addConfig("jdbc.username", "root");
// jobTracker.addConfig("jdbc.password", "root");

// 这个是对于 返回给客户端 任务的 老数据删除策略
jobTracker.setOldDataHandler(new OldDataDeletePolicy());
// 设置 zk 客户端用哪个，  可选 zkclient(默认), curator
jobTracker.addConfig("zk.client", "zkclient");
// 启动节点
jobTracker.start();
```

2. TaskTracker 示例

```
TaskTracker taskTracker = new TaskTracker();
 // 任务执行类，实现 JobRunner 接口
taskTracker.setJobRunnerClass(TestJobRunner.class);
taskTracker.setRegistryAddress("zookeeper://127.0.0.1:2181");
// taskTracker.setRegistryAddress("redis://127.0.0.1:6379");
taskTracker.setNodeGroup("test_trade_TaskTracker"); // 同一个 TaskTracker 集群这个名字相同
taskTracker.setClusterName("test_cluster");
taskTracker.setWorkThreads(20);
// 反馈任务给 JobTracker 失败，存储本地文件路径
// taskTracker.setFailStorePath(Constants.USER_HOME);
// master 节点变化监听器，当有集群中只需要一个节点执行某个事情的时候，可以监听这个事件
taskTracker.addMasterChangeListener(new MasterChangeListenerImpl());
// 业务日志级别
// taskTracker.setBizLoggerLevel(Level.INFO);
// 可选址  leveldb(默认), rocksdb, bekeleydb
// taskTracker.addConfig("job.fail.store", "leveldb");
taskTracker.start();


// 任务执行类
```

```
public class TestJobRunner implements JobRunner {
    private static final Logger LOGGER = LoggerFactory.getLogger(TestJobRunner.class);
    @Override
    public Result run(Job job) throws Throwable {
        try {
            LOGGER.info("我要执行：" + job);

            BizLogger bizLogger = LtsLoggerFactory.getBizLogger();
            // 会发送到 LTS (JobTracker 上)
            bizLogger.info("测试，业务日志啊啊啊啊啊");

            Thread.sleep(1000L);

            if (job.getRetryTimes() > 5) {
                return new Result(Action.EXECUTE_FAILED, "重试次数超过 5 次了，放过你吧!");
            }
            if (SystemClock.now() % 2 == 1) {
                return new Result(Action.EXECUTE_LATER, "稍后执行");
            }
        } catch (Exception e) {
            LOGGER.info("Run job failed!", e);
            return new Result(Action.EXECUTE_LATER, e.getMessage());
        }
        return new Result(Action.EXECUTE_SUCCESS, "执行成功了，哈哈");
    }
}
```

3. JobClient 实例

```
// 推荐使用RetryJobClient
JobClient jobClient = new RetryJobClient();
jobClient.setNodeGroup("test_jobClient");
jobClient.setClusterName("test_cluster");
jobClient.setRegistryAddress("zookeeper://127.0.0.1:2181");
// jobClient.setRegistryAddress("redis://127.0.0.1:6379");
// 任务重试保存地址，默认用户目录下
// jobClient.setFailStorePath(Constants.USER_HOME);
// 任务完成反馈接口
jobClient.setJobFinishedHandler(new JobFinishedHandlerImpl());
// master 节点变化监听器，当有集群中只需要一个节点执行某个事情的时候，可以监听这个事件
jobClient.addMasterChangeListener(new MasterChangeListenerImpl());
// 可选址  leveldb(默认), rocksdb, bekeleydb
// taskTracker.addConfig("job.fail.store", "leveldb");
jobClient.start();
```

提交任务

```
Job job = new Job();
// 必填，尽量taskId 有一定规律性，能让自己识别
job.setTaskId(StringUtils.generateUUID());
// 任务的参数，value 必须为字符串
job.setParam("shopId", "111");
// 是否接收执行反馈消息 jobClient.setJobFinishedHandler(new
JobFinishedHandlerImpl()); 中接受
job.setNeedFeedback(true);
// 执行节点的group 名称
job.setTaskTrackerNodeGroup("test_trade_TaskTracker");
// 这个是 cron expression 和 quartz 一样，可选
// job.setCronExpression(cronExpression);
// 这个是指定执行时间，可选
// job.setTriggerTime(new Date());
// 当 cronExpression 和 triggerTime 都不设置的时候，默认是立即执行任务
// response 返回提交成功还是失败
Response response = jobClient.submitJob(job);
```

五、参数说明

参见 （四）中的例子中的参数注释

六、扩展

LTS 使用 SPI 扩展机制

1. lts-logger 扩展

step1:

引入

```
<dependency>
    <groupId>com.lts</groupId>
    <artifactId>lts-logger-api</artifactId>
    <version>${lts.version}</version>
</dependency>
```

step2:

实现 JobLogger 和 JobLoggerFactory 接口

step3:

在 resources 下的 META-INF/lts 文件夹中增加文件

com.lts.biz.logger.JobLoggerFactory,

文件内容为: xxx=com.lts.biz.logger.xxx.XxxJobLoggerFactory

step4:

使用：

```
jobTracker.addConfig("job.logger", "xxx");
```

你可以实现接口将日志输出到你想输出的地方，譬如用 mq 发出去，然后分析什么的

2. lts-queue 扩展

这个类似，可以参考 lts-queue-mongo 或者 lts-queue-mysql 的实现

3. 其他的扩展，后续文档会跟进来