一、基本环境搭建

1、安装 zeromq

```
unzip zeromq-2.1.9.zip
```

```
cd zeromq-2.1.9
```

```
./configure
```

出现：

```
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking how to create a ustar tar archive... gnutar
checking for gcc... no
checking for cc... no
checking for cl.exe... no
configure: error: in `/home/lijianjun/zeromq-2.1.9':
configure: error: no acceptable C compiler found in $PATH
See `config.log' for more details
```

这是因为没有安装 C 编译器

解决方法：

```
yum install gcc
```

之后，再重新 编译

出现：

```
configure: error: Unable to find a working C++ compiler
```

解决方法：

```
yum install gcc-c++
```

之后，再重新编译

出现：

```
error: cannot link with -luuid, install uuid-dev.
```

解决方法：

```
yum install uuid*
```

```
yum install libuuid*
```

```
yum install e2fsprogs*
```

之后就编译无误啦！

```
make
```

```
make install
```

```
sudo ldconfig
```

这样，zeromq 就安装成功啦！

2、安装 jzmq

先安装 git

```
yum install git
```

再

```
git clone git://github.com/nathanmarz/jzmq.git
```

```
cd jzmq
```

```
./autogen.sh
```

出现：

```
autogen.sh: error: could not find libtool.  libtool is required to run autogen.sh.
```

解决办法：

```
yum install libtool
```

之后，再 ./autogen.sh 就没有错误了！

```
./configure
```

出现：

```
configure: error: the JAVA_HOME environment variable must be set to your JDK location.
```

解决方法：

1、先卸载了 centos 自带的 JDK,然后安装 SUN 的 JDK

参考：

http://hermosa-young.iteye.com/blog/1798026

http://www.cnblogs.com/hitwtx/archive/2012/02/13/2349752.html

```
rpm -qa|grep java
```

// 查看 jdk 的信息

```
yum -y remove  java-1.6.0-openjdk-1.6.0.0-1.7.b09.el5
```

// 卸载

下载 SUN 的 JDK

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

jdk-7u71-linux-i586.rpm

在/usr 下建立一个 java 目录，以备将 java 程序安装在此目录下

```
mkdir /usr/java
```

```
cd /usr/java
```

```
rpm  -ivh jdk-7u71-linux-i586.rpm
```

解压后，在/usr/java 目录下就会生成一个新的目录 jdk1.7.0_71，该目录下存放的是解压后的文件。

为了以后设置方便，我们该生成的目录 jdk1.7.0_71 改名为 jdk

```
mv jdk1.7.0_71 jdk
```

最后进行环境变量的设置

```
vi /etc/profile
```

```
export JAVA_HOME=/usr/java/jdk
```

```
export PATH=$PATH:$JAVA_HOME/bin
```

```
export CLASSPATH=.:$JAVA_HOME/jre/lib/rt.jar:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools
```

执行配置文件，令其立刻生效

```
source /etc/profile
```

验证是否安装成功

```
java -version
```

```
java version "1.7.0_71"
```

Java(TM) SE Runtime Environment (build 1.7.0_71-b14)

Java HotSpot(TM) Client VM (build 24.71-b01, mixed mode, sharing)

之后，再

```
./configure
```

```
make
```

```
make install
```

都没有出现任何错误。

Jzmq 就安装成功啦！

由于之前参考的是：

http://blog.csdn.net/wind520/article/details/9308809

走了一些弯路，不过对于也就当学习了！

Centos 6.4 自带 OpenJDK 的环境，但是需要手动配置 JAVA_HOME 的环境变量。

可以通过 `java -version` 查看当前已装版本

java version "1.7.0_45"

OpenJDK Runtime Environment (rhel-2.4.3.3.el6-i386 u45-b15)

OpenJDK Client VM (build 24.45-b08, mixed mode, sharing)

安装包放在哪了？

openJDK 安装好后的目录位于：
/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.45

可以在“java-1.7.0-openjdk-1.7.0.45”目录下看到“bin”，在 bin 下可以找到 javac 文件，说明这就是 JDK 了！

其他版本都在/usr/lib/jvm 下，包括 jre 和 jdk

```
vim /etc/profile
```

```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.45

export CLASSPATH=.:$JAVA_HOME/jre/lib/rt.jar:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar

export PATH=$PATH:$JAVA_HOME/bin
```

这样我们就设置好了 JDK，

在输入 source /etc/profile 就可以生效了

但是，还是不行

提示：找不到 javac 文件

确实找遍了所有 java 目录和 jvm 都没有找到 javac，只能卸了自带的，重装！

3、安装 Python2.7.2

```
wget http://www.python.org/ftp/python/2.7.2/Python-2.7.2.tgz
tar zxvf Python-2.7.2.tgz
cd Python-2.7.2
```

```
./configure
make
make install
vi /etc/ld.so.conf
```

追加 /usr/local/lib/

补充知识：参考

http://blog.csdn.net/yjkwf/article/details/7545002

/etc/ld.so.conf 此文件记录了编译时使用的动态库的路径，也就是加载 so 库的路径。

默认情况下，编译器只会使用/lib 和/usr/lib 这两个目录下的库文件，而通常通过源码包进行安装时，如果不指定--prefix 会将库安装在/usr/local 目录下，而又没有在文件/etc/ld.so.conf 中添加/usr/local/lib 这个目录。这样虽然安装了源码包，但是使用时仍然找不到相关的.so 库，就会报错。也就是说系统不知道安装了源码包。

```
sudo ldconfig
```

4、安装 zookeeper

```
wget
http://labs.mop.com/apache-mirror/zookeeper/zookeeper-3.4.5/zo
okeeper-3.4.5.tar.gz
tar -zxvf zookeeper-3.4.5.tar.gz
cp -R zookeeper-3.4.5 /usr/local/
mv  zookeeper-3.4.5  zookeeper
```

设置 ZOOKEEPER_HOME 和 ZOOKEEPER_HOME/bin

```
vim /etc/profile
export ZOOKEEPER_HOME="/usr/local/zookeeper"
```

```
export PATH=$PATH:$ZOOKEEPER_HOME/bin

source /etc/profile
```

用 zoo_sample.cfg 制作$ZOOKEEPER_HOME/conf/zoo.cfg

```
cp /usr/local/zookeeper/conf/zoo_sample.cfg
/usr/local/zookeeper/conf/zoo.cfg
```

```
sudo mkdir /tmp/zookeeper

sudo mkdir /var/log/zookeeper
```

zookeeper 的单机安装已经完成了

5、安装 storm

```
unzip storm-0.8.2.zip
mv  storm-0.8.2  /usr/local/   移动
mv  storm-0.8.2  storm   重命名
```

```
vim /etc/profile
export STORM_HOME=/usr/local/storm

export PATH=$PATH:$STORM_HOME/bin

source /etc/profile
```

这样单机版的环境就弄好了，接下来：

二、虚拟环境下 demo

本地运行测试程序 storm-start，可在 win7 环境下或在 linux 下完成。

参考：

http://blog.csdn.net/yjkwf/article/details/7545002

按照 https://github.com/nathanmarz/storm-starter，执行这个程序需要用 lein，这里介绍的方法用 eclipse 代替 lein 的作用。

1） 安装 twitter4j

```
mkdir twitter4j

cd twitter4j

wget http://twitter4j.org/en/twitter4j-2.2.5.zip

unzip twitter4j-2.2.5.zip
```

2） 追加源文件 jar

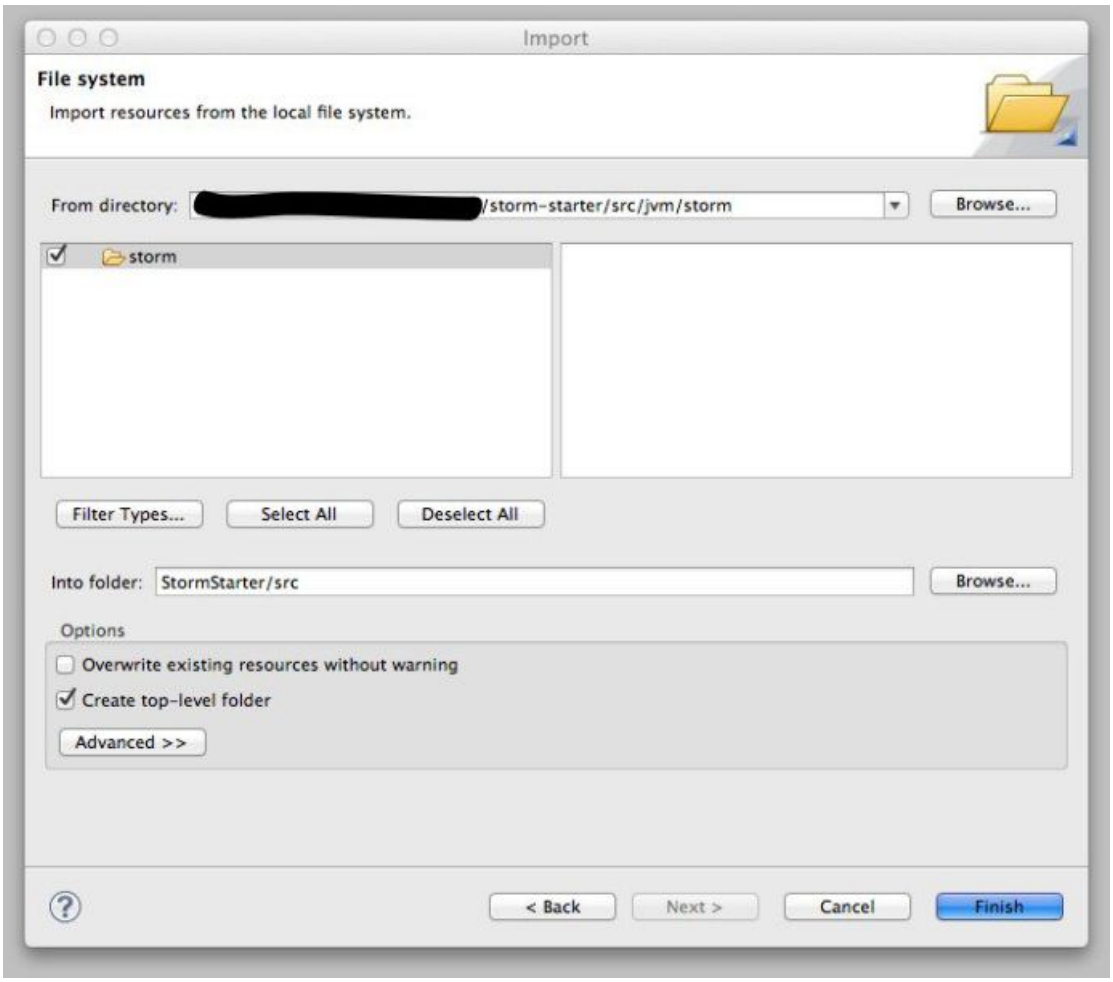
使用 eclipse 建立 java projectb 并追加 twitter4j 和 storm 的 jar 文件。

File-> New -> Java Project ->随便取个名字-> Next -> Libraries -> add External JARs...-> 追加 twitter4j 和 storm 的 jar 文件（/path/to/twitter4j/lib/*.jar 和/path/to/storm/lib/*.jar 和 /path/to/storm/storm-{version}.jar）-> Finsh

导入 storm-start

File -> Import -> General -> File System -> Next -> Browse(From directory) -> /path/to/storm-start/src/jvm/storm -> Browse(Info floder) -> xxx -> src -> OK -> “storm” 和 “Create top-level folder” 前打勾 -> Finish

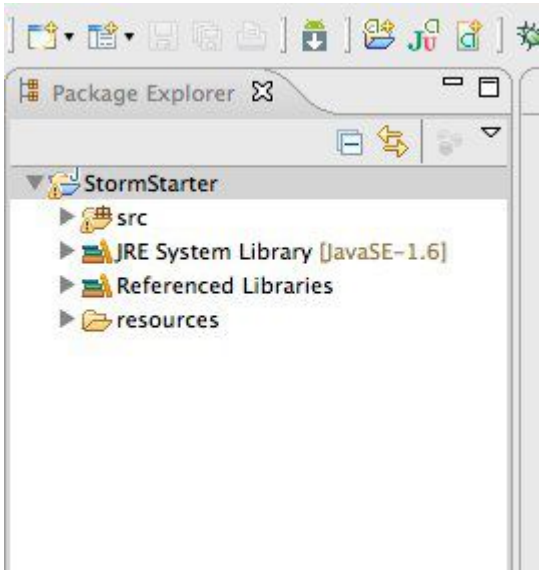
完成之后如图：



追加 resources（python 文件 word count 用）

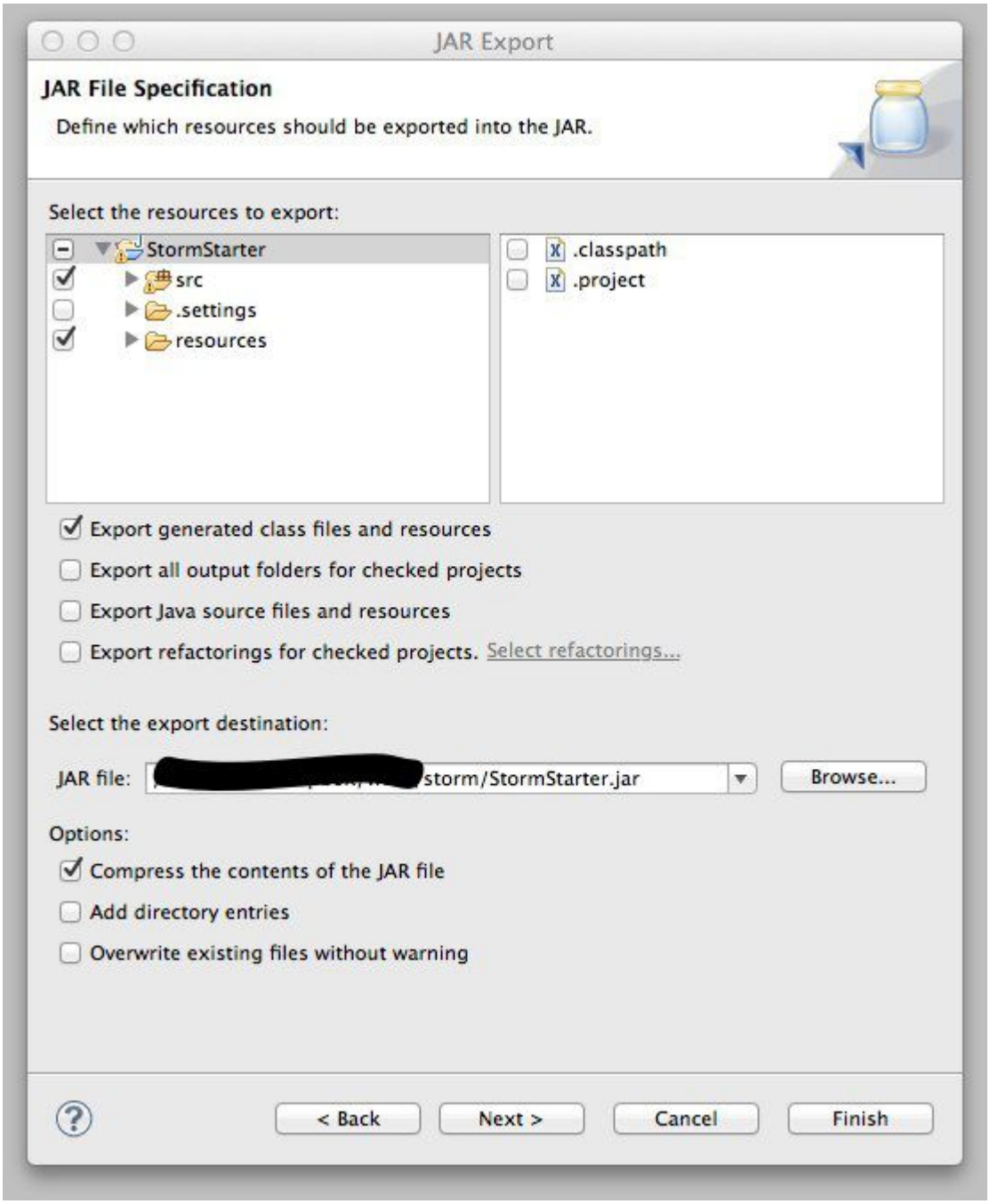
File -> Import -> General -> File System -> Next -> Browse(From directory) -> /path/to/storm-start/multilang/resources -> Browse(Info floder) -> xxx -> OK -> check “resources” and “Create top-level folder” -> Finish

2 个源文件都追加好之后，eclipse 左边显示如下图：



4） JAR export

File -> Export -> JAR -> JAR file -> 取消“.classpath”，“.project”和“<.settings”->的勾 browse -> path/to/export/name.jar -> Finish (忽视 warnings)



5） 执行刚才编译的文件

```
# storm jar StormStarter.jar storm.starter.ExclamationTopology
```

如果出现类似下面的文字，说明运行成功！

```
....
```

```
11367 [Thread-25] INFO  backtype.storm.daemon.task  - Emitting:
class storm.starter.ExclamationTopology$ExclamationBolt source:
2:3, stream: 1, id: {}, [golda!!!]

....
```

三、本地测试 storm

要注意上面的本地模式运行 ExclamationTopology 只是一个 storm 的虚拟环境下测试 demo。那我们怎样将程序运行在刚刚搭建的单机版的环境里面呢？

注意看官方实例中 WordCountTopology 类如果不带参数其实是执行的本地模式， 也就是刚说的虚拟的环境，

带上参数就是将 jar 发送到了 storm 执行了。

1、启动 zookeeper：

```
/usr/local/zookeeper/bin/zkServer.sh
```

单机版直接启动， 不用修改什么配置， 如集群就需要修改 zoo.cfg 另一篇文章会讲到

```
zkServer.sh status
```
#查看 zkserver 是否成功启动

没启动的话：

```
zkServer.sh start
```
多次尝试

直到出现

```
Starting zookeeper ... STARTED
```

查看状态时：zkServer.sh status

出现

JMX enabled by default

Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg

Mode: standalone

2、配置 storm

vim /usr/local/storm/conf/storm.yaml

```
storm.zookeeper.servers:
     - "192.168.241.128"  //本机 IP 地址
 nimbus.host: "192.168.241.128"
 storm.zookeeper.port: 2181
 storm.local.dir: “/tmp/storm”
 supervisor.slots.ports:
    - 6700
    - 6701
    - 6702
    - 6703
```

3、接着启动 zkServer nimbus， supervisor 和 ui 几个服务

storm nimbus&

要等待一会直到出现---- backtype.storm.daemon.nimbus

storm supervisor&

要等待一会直到出现---- storm.daemon.supervisor

storm ui&

要等待一会直到出现---- properties backtype.storm.ui.core

jps 可以查看各个进程的运行状态

```
[root@lijianjun lijianjun]# jps
24756 core
24459 QuorumPeerMain
24846 Jps
24725 nimbus
24736 supervisor
```

core 对应的进程是 Storm UI

Jps 对应的进程是 Java jps

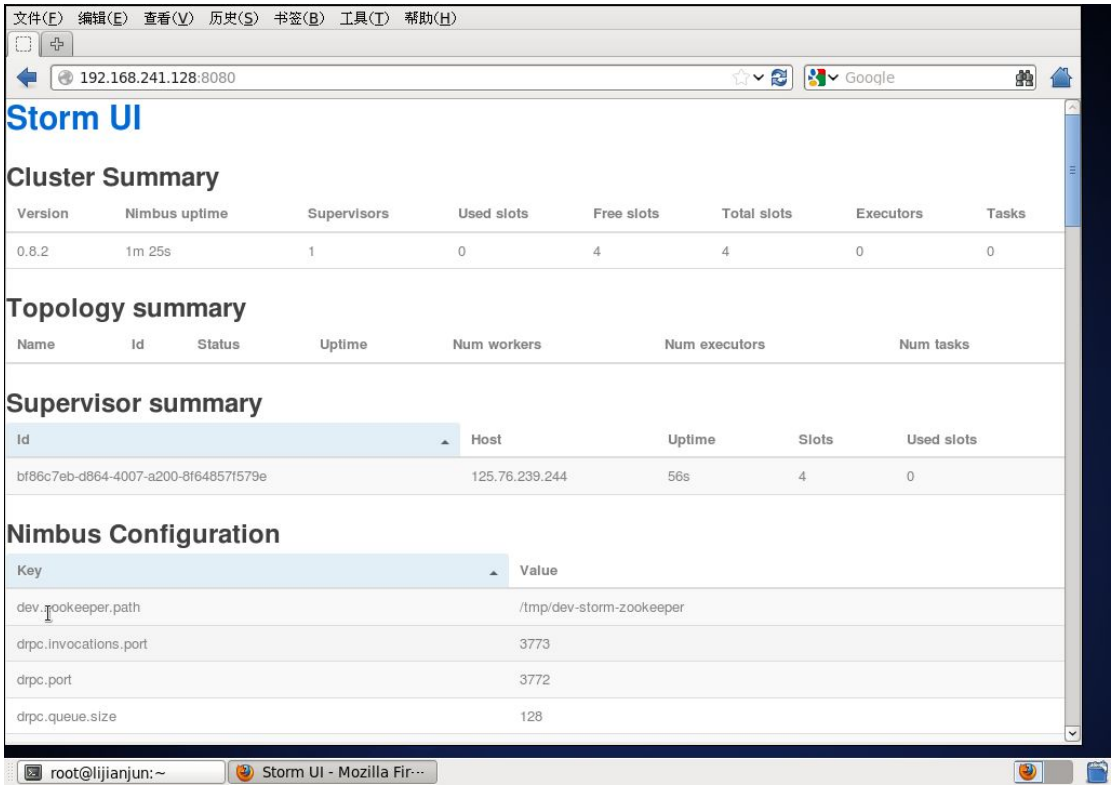
nimbus 对应的进程是 Storm nimbus

supervisor 对应的进程是 Storm supervisor

QuorumPeerMain 对应的进程是 zkServer.sh

也可在浏览器中查看状态

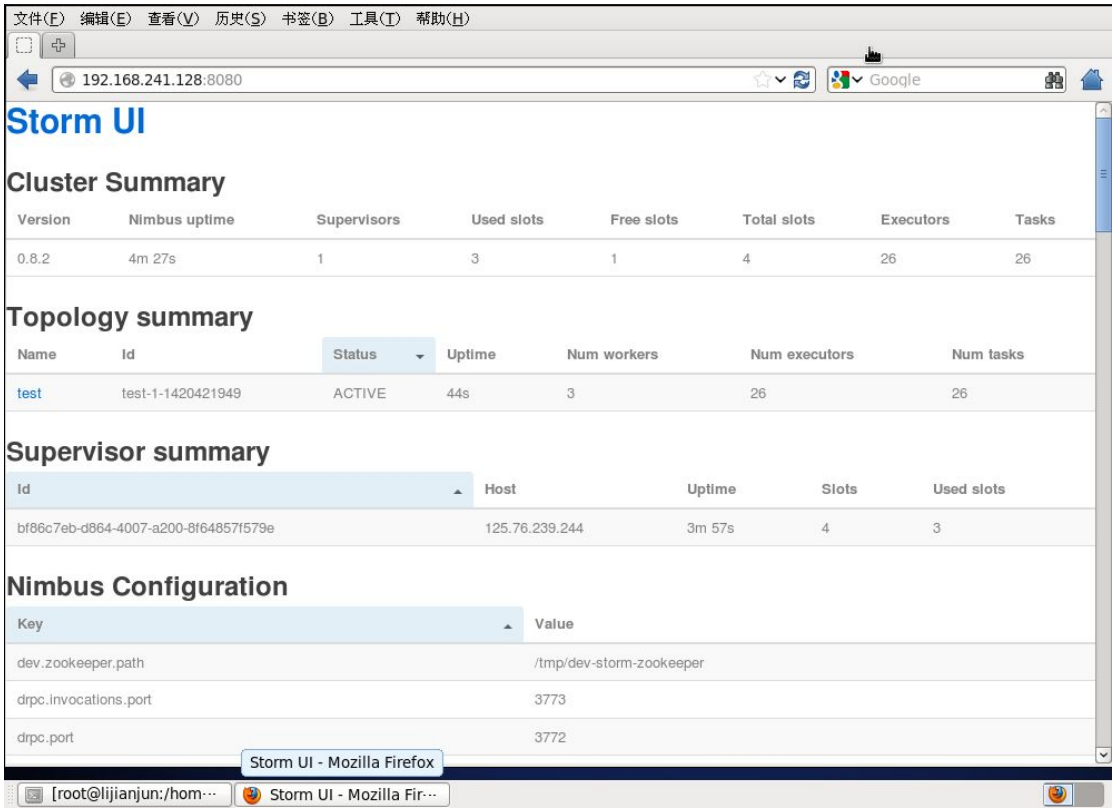
http://192.168.241.128:8080/



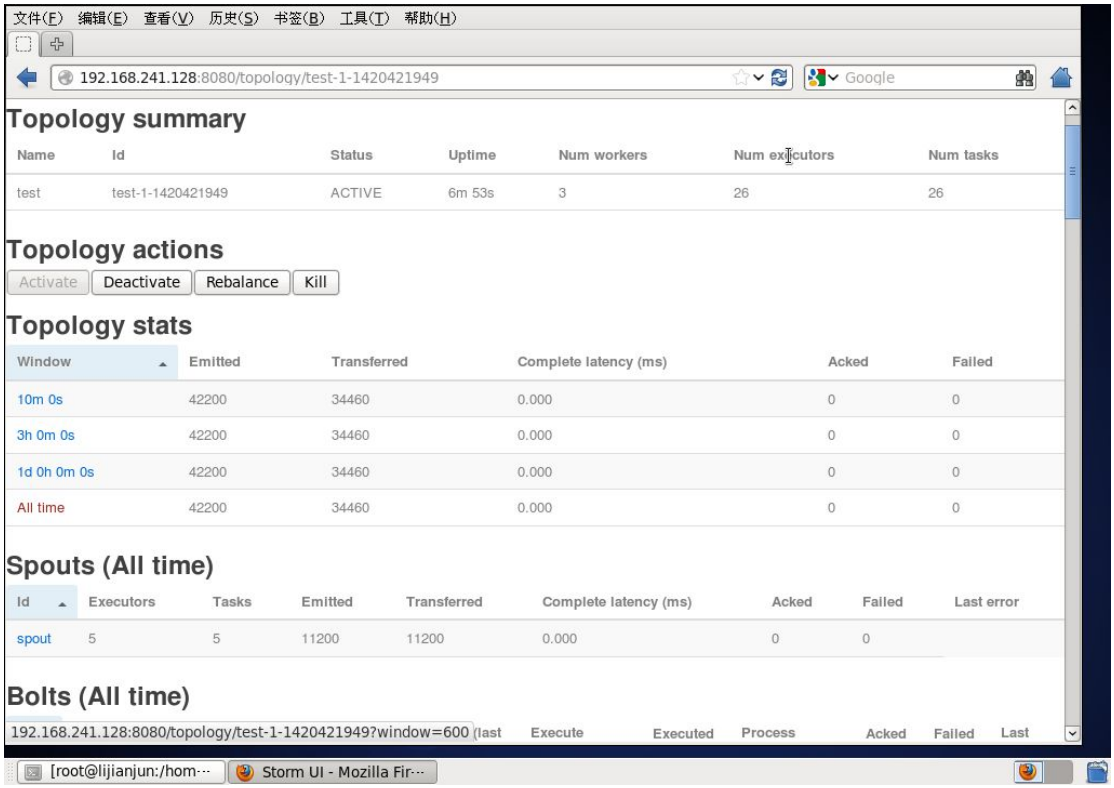
4、最后，提交拓扑

```
storm jar secondstorm.jar storm.starter.WordCountTopology test
```

此命令的作用就是用 storm 将 jar 发送给 storm 去执行， 后面的 test 是定义的 toplogy 名称



点击 topology summary 下面的 test 出现



可以看到 spouts 和 bolts 的状态，说明提交成功，本地模式启动 OK!

Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Last error
spout	5	5	11200	11200	0.000	0	0	

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Last error
count	12	12	7740	0	0.074	11.121	7760	6.756	7720	0	
split	8	8	23260	23260	0.016	3.506	3600	4982.215	3620	0	

点击 spout 出现

Spout stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
All time	23160	23160	0.000	0	0
1d 0h 0m 0s	23160	23160	0.000	0	0
3h 0m 0s	23160	23160	0.000	0	0
10m 0s	23160	23160	0.000	0	0

Output stats (All time)

Stream	Emitted	Transferred	Complete latency (ms)	Acked	Failed
default	23160	23160	0	0	0

Executors (All time)

Id	Uptime	Host	Port	Emitted	Transferred	Complete latency (ms)	Acked	Failed
[22-22]	9m 41s	125.76.239.244	6703	4760	4760	0.000	0	0
[23-23]	9m 42s	125.76.239.244	6702	4440	4440	0.000	0	0
[24-24]	9m 41s	125.76.239.244	6701	4780	4780	0.000	0	0
[25-25]	9m 41s	125.76.239.244	6703	4720	4720	0.000	0	0
[26-26]	9m 42s	125.76.239.244	6702	4460	4460	0.000	0	0

点击 split 出现

Component summary

Id	Topology	Executors	Tasks
split	test	8	8

Bolt stats

Window	Emitted	Transferred	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed
10m 0s	55940	55940	1.155	8540	1350.799	8760	0
3h 0m 0s	60540	60540	1.639	9420	1920.823	9480	0
1d 0h 0m 0s	60540	60540	1.639	9420	1920.823	9480	0
All time	60540	60540	1.639	9420	1920.823	9480	0

Input stats (All time)

Component	Stream	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed
spout	default	1.639	9420	1920.823	9480	0

Output stats (All time)

Stream	Emitted	Transferred
default	60540	60540

Executors

Id	Uptime	Host	Port	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed
[14-14]	11m 51s	125.76.239.244	6702	9020	9020	0.009	3.871	1400	2302.859	1420	0
[15-15]	11m 52s	125.76.239.244	6701	4800	4800	0.002	1.541	740	289.526	760	0
[16-16]	11m 48s	125.76.239.244	6703	9520	9520	0.000	1.987	1500	2442.568	1480	0
[17-17]	11m 51s	125.76.239.244	6702	9060	9060	0.003	1.800	1400	2011.930	1420	0
[18-18]	11m 52s	125.76.239.244	6701	4780	4780	0.001	0.649	740	120.324	740	0
[19-19]	11m 48s	125.76.239.244	6703	9500	9500	0.000	0.947	1500	3332.959	1480	0
[20-20]	11m 51s	125.76.239.244	6702	9020	9020	0.001	0.714	1400	2162.521	1420	0
[21-21]	11m 52s	125.76.239.244	6701	4840	4840	0.000	0.649	740	203.632	760	0

点击 count 出现

Component summary

Id	Topology	Executors	Tasks
count	test	12	12

Bolt stats

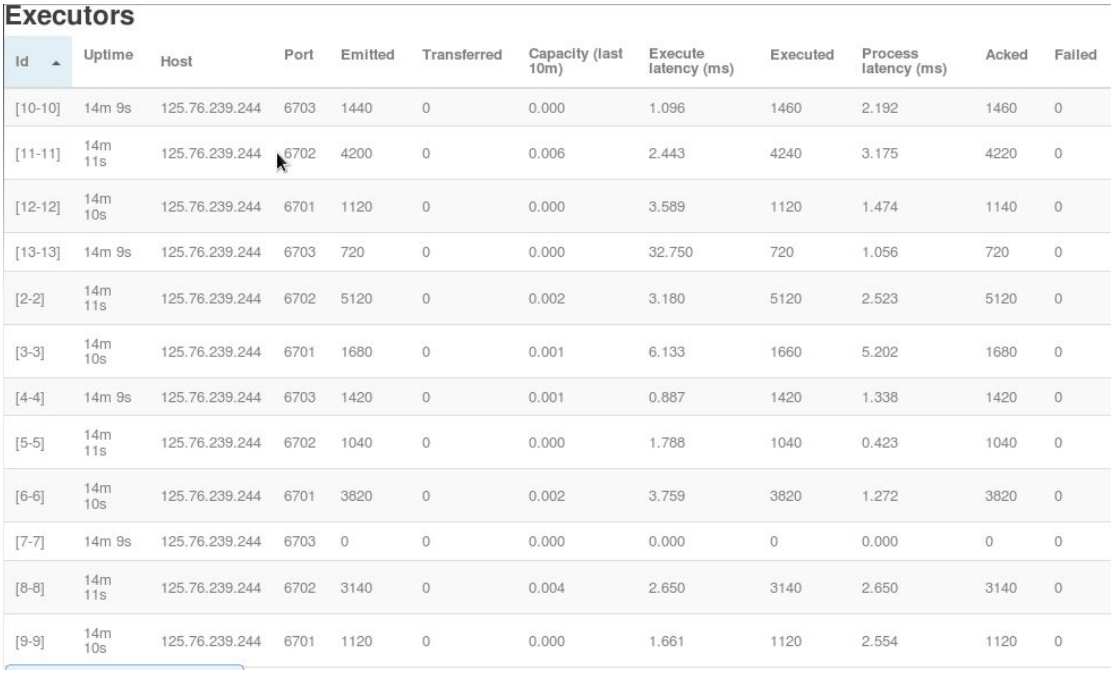
Window	Emitted	Transferred	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed
10m 0s	18900	0	0.582	18900	0.441	18920	0
3h 0m 0s	24820	0	3.768	24860	2.375	24880	0
1d 0h 0m 0s	24820	0	3.768	24860	2.375	24880	0
All time	24820	0	3.768	24860	2.375	24880	0

Input stats (All time)

Component	Stream	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed
split	default	3.768	24860	2.375	24880	0

Output stats (All time)

Stream	Emitted	Transferred
default	24820	0



Executors

Id	Uptime	Host	Port	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed
[10-10]	14m 9s	125.76.239.244	6703	1440	0	0.000	1.096	1460	2.192	1460	0
[11-11]	14m 11s	125.76.239.244	6702	4200	0	0.006	2.443	4240	3.175	4220	0
[12-12]	14m 10s	125.76.239.244	6701	1120	0	0.000	3.589	1120	1.474	1140	0
[13-13]	14m 9s	125.76.239.244	6703	720	0	0.000	32.750	720	1.056	720	0
[2-2]	14m 11s	125.76.239.244	6702	5120	0	0.002	3.180	5120	2.523	5120	0
[3-3]	14m 10s	125.76.239.244	6701	1680	0	0.001	6.133	1660	5.202	1680	0
[4-4]	14m 9s	125.76.239.244	6703	1420	0	0.001	0.887	1420	1.338	1420	0
[5-5]	14m 11s	125.76.239.244	6702	1040	0	0.000	1.788	1040	0.423	1040	0
[6-6]	14m 10s	125.76.239.244	6701	3820	0	0.002	3.759	3820	1.272	3820	0
[7-7]	14m 9s	125.76.239.244	6703	0	0	0.000	0.000	0	0.000	0	0
[8-8]	14m 11s	125.76.239.244	6702	3140	0	0.004	2.650	3140	2.650	3140	0
[9-9]	14m 10s	125.76.239.244	6701	1120	0	0.000	1.661	1120	2.554	1120	0

5、停止 Storm Topology：

```
storm kill {toponame}
```

```
storm kill test
```

出现：

```
backtype.storm.command.kill_topology test

0    [main] INFO  backtype.storm.thrift  - Connecting to Nimbus at localhost:6627

1088 [main] INFO  backtype.storm.command.kill-topology  - Killed topology: test
```

则关闭成功！

其他常用命令：

1、提交/部署 TOPOLOGY

storm jar /jar 文件所在目录.jar firststorm.WordCountTopology（拓扑

名） /words.txt(参数所在路径)

2、删除 TOPOLOGY

```
storm  kill  {toponame}
```

3、激活 TOPOLOGY

```
storm  active  {toponame}
```

4、不激活 TOPOLOGY

```
storm  deactive  {toponame}
```

5、列出所有 TOPOLOGY

```
storm  list
```