# **一、批处理**

我们在一些项目中如：银行、保险、零商业门店系统中的对帐、结帐、核算、日结等操作中经常会碰到一些"批处理“作业。

这些批处理经常会涉及到一些大数据处理，同时处理一批增、删、改、查等SQL，往往涉及到好几张表，这边取点数据那边写点数据，运行一些存储过程等。

批处理往往耗时、耗资源，往往还会用到多线程去设计程序代码，有时处理不好还会碰到内存泄漏、溢出、不够、CPU占用高达99%，服务器被严重堵塞等现象。

笔者曾经经历过一个批处理的3次优化，该批处理笔者按照数据库连接池的原理实现了一个线程池，使得线程数可以动态设定，不用的线程还可还回线程池，其过程经历了加入cache等操作，最后连负载截均衡都想到了。

最终虽然取得了较满意的结果，但是当中不断的优化程序、算法上耗时彼多，尤其是想到了负截匀衡。大家知道，一个web或者是一个app容器当达到极限时可以通过加入集群以及增加节点的手段来提高总体的处理能力，但是这个批处理往往是一个应用程序，要把应用程序做成集群通过增加节点的方式来提高处理能力可不是一件简单的事，对吧？

当然我不是说不能实现，硬代码通过SOCKET通讯利用MQ机制加数据库队列是一种比较通用的设计手段，可以做到应用程序在处理大数据大事务时的集群这样的能力。

但是对于一些较大型的商业客户尤其是银行、保险、大型零售行业或者是电信等客户，他们都是有成品套件的，你自己设计的具有集群能力的批处理固然有成就感，但是从另外一个方面来说，从稳定性、可用性、维护性来说相信同类的现成的成熟商业用品一定要超过你自己编写的批处理程序吧?

因为我这套架构师的道路走的是企业级架构师路线，因此也不得不经常提到一些商业级解决方案，对于自己设计这个具有集群扩展能力的批处理我会放到以后去讲，今天主要讲的就是使用商业成品来完成你的批处理的集成。

# **二、商业级解决方案**

来看一个批处理的需求：

1.能够做批处理

2. 能够通过增加节点来提高批处理的能力，相当于集群

3. 具有错误重跑的能力

4. 断点处理能力，比如说5000批次作业，我跑了2000批后失败了，这时后面3000批在我做了一些调整后会接着前面的2000批继续跑下去

5. 完善的日志、监督、暂停（挂起）、定时（嘿嘿，这个夸张）跑批处理的能力

我们先用传统的自己手工来做批处理的设计思想来考虑这个需求，至少要用到下面几个技术：

1. 我们得写个线程池，就和我上面提到过的自己按照jdbc connection pool的原理去写这个线程池，快手的话2周至少要得吧，对吧？

2. 集成JMS或者是MQ机制，使得该程序具有节点间通讯能力这样就能做到负载均衡了。

3. 将任务做任务记录持久化到数据库，这样可以做到错误记录、断点、重跑等功能

4. 得要用Quartz类似的组件来实现这个”定时，定期跑批”的功能吧

好了，上述这些需求就够你做一个工程了，不是吗？

我现在告诉你，这个批处理的需求是一个最低层次的商业级批处理作业的需求，如果用我以前工程上涉及到的完整需求，这可写上10几页（仅需求部分）。

大家知道在做大型客户如：银行、保险等项目时，这些客户是怎么考虑的吗?

首先，你如果告诉他你需要1个月左右来做这个批处理（包含测试，这个已经是非常快的速度了），它会告诉你它只给你1周或者2周时间（最多了）。

因为客户认为，批处理无非就是输入->处理->输出就完了，怎么处理的，有这么复杂吗，嘿嘿，客户一般认为它的需求都是很简单的，开发商往往都是喜欢多要时间这样可以多算人力成本。

但是看官们请注意，客户这些考虑也是没有错的，是的，批处理从大的方面来说就是read->handle->write是这样的，至于read数据库+read文件再+read socket对于客户来说，它就认为只有一个read，这其中的苦处啊，只有我们程序员自己知道。

现在好了，目前已经有成熟的商业成品套件来实现上面所有的这些需求了，它让程序员只需要关注：

* how to read
* how to handle(your own logic)
* how to write

就行了，至于其它的事，它来帮你做。

开源的有spring batch，不过spring batch走的是云计算道路，商业产品中比较有名的就是这个IBM Websphere Compute Grid。

# **三、IBM Websphere Compute Grid介绍**

## **3.1 基本概念**

这边是compute而不是computer啊，不要搞错了。

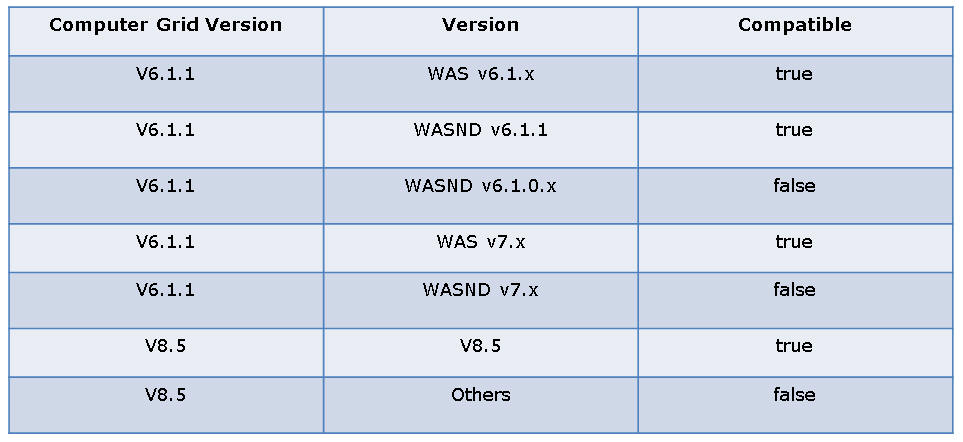
这个IBM Websphere Compute Grid简称WAS XD CG，是属于IBM Websphere套件中的附属品，它是架构在WASND的基础上来实现批处理作业的.

它利用WASND的功能，因为IBM的WASND是支持网格规划的，WASND不仅可以把单个节点与节点间进行联合，甚至可以把几个联结起来的节点-称为cell（单元），cell与cell之间再进行联合。

而IBM Compute Grid就是架构在这样的基础之上，对于提高批处理的能力来说无非就是再布署一个基于WASND的节点或者是WASND的单元。

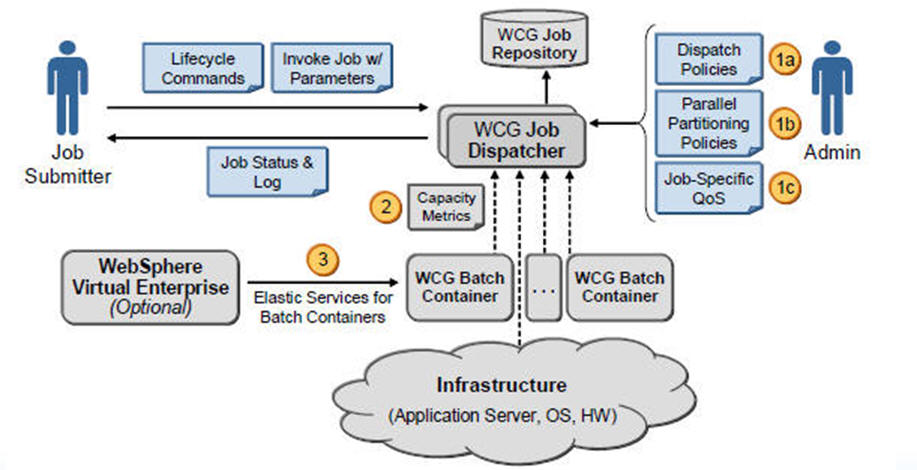
所以这边有一个概念需要注意啦，它是基于WASND的。

而IBM Compute Grid的安装也与相对应的WASND的版本必须绝对的对应的，哪怕两者间的小版本号有差异都不能安装，下面是经本人测试了各个WAS与IBM Compute Grid后得出的一个IBM Compute Grid与WAS产品的版本对应关系：



当然，如果你手上比较含酸，没有WASND只有WAS也可以用WAS来安装IBM Compute Grid，但是版本一定要按照我的这个对照表去下载IBM Compute Grid啊。

## **3.2 IBM Compute Grid的运作原理**



挺复杂一张图，没什么难看懂的，抓关键。

1. 它的运作原理就是作业流程自定义，相当于一个批处理作业写完后，它有先后的执行步骤，a, b, c, d，我现在的执行步骤是a->b->c->d，但下次我的执行顺序是：a->c->d->b，对吧。

这个作业步骤或者我们说流程是可以自定义的，你不要和我说你因为作业步骤的执行顺序变了你去改你的代码改你的if else哦，这个就是传统的思维了。

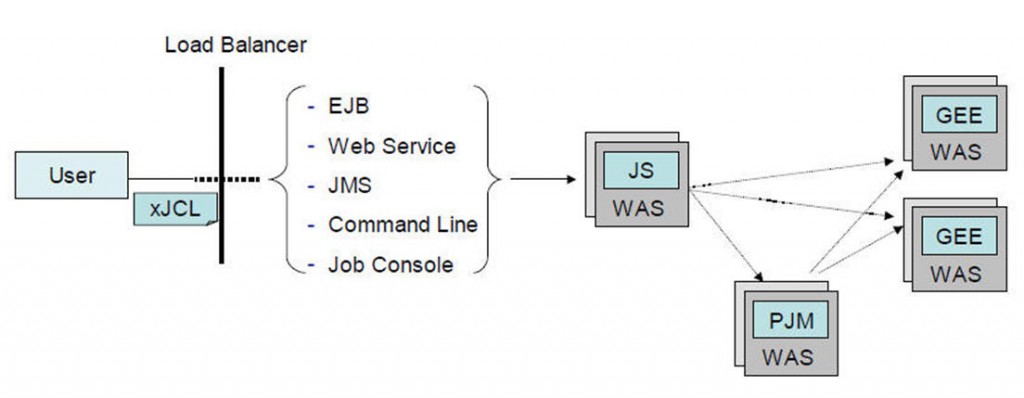
商业的吗，它讲究的就是一个：

****业务快速实现、业务直接可由业务人员制定、业务直接转换成IT资产并且快速推向市场、垂直的业务解决方案能力****

请牢牢记住上面这一长溜红色加粗的方案，它是你作为一个企业级架构师将来奋斗的目标，死写代码是没有用的，理念同样重要。

1. 并行的作业处理能力，就是我说的具有通过增加一个节点就可以提高你的作业处理能力的功能。
2. 它的作业流程的制定其实和制定我们的webservice一样，是一个描述性文档，将这个描述性文档（通常是一个xml文件）即我们的作业描述上传到IBM Compute Grid作业中心后，即可实现定时啦、重跑啦、断点续跑啦等等等等功能。
3. 批处理作业也作为了“企业IT资产”的一部分-IT ASSET，在WAS中我们的IT ASSET典型的为一个.ear布署包，即你的批处理啊，以后我们的批处理就是是一个.ear的包，要布署到WAS中去的。

喏，下面这张是IBM Compute Grid的部件图

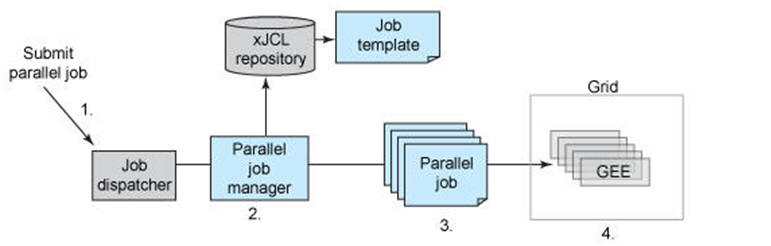


此处的xJCL就是一个XML格式的作业描述文件，它是以EJB形式并且以.ear包布署入WAS的。

它不是类似于web service吗，web service的入口被称为end point对吧？它也有end point，它叫grid end point即GEE。

GEE通过PJM即Parallel Job Manager并行作业管理器来协调和实现集群的，这一切都是布署在WAS上的。

下面这张是IBM Compute Grid的作业逻辑流程图



# **四、IBM Compute Grid的安装、开发与布署**

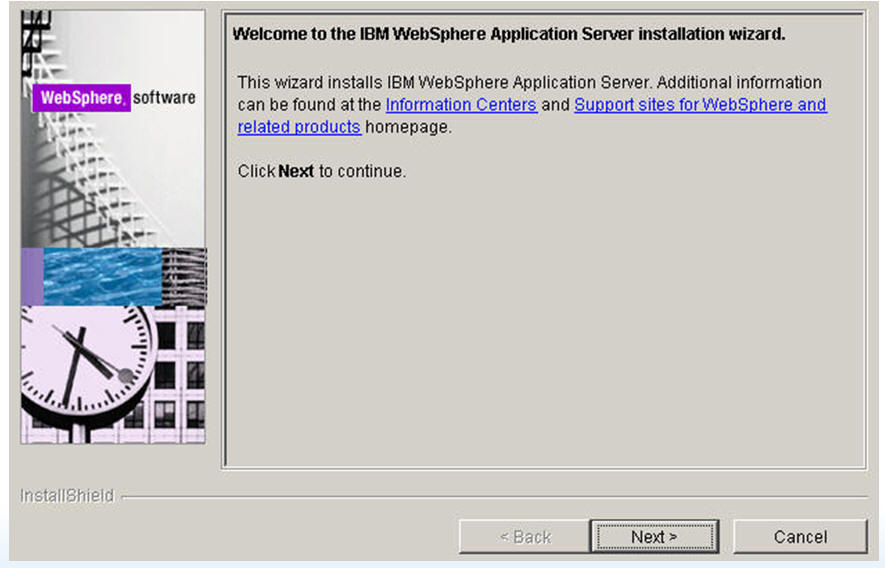
## **4.1 安装IBM Compute Grid**

在这个例子里我们使用WAS7.x+IBM Compute Grid6.1.x来做，都可以在IBM的官网上下载到Trial Version，当然你也可以使用WASND8.0 trial version与IBM Compute Gridv8.0来做测试。

这边我们需要安装三样东西：

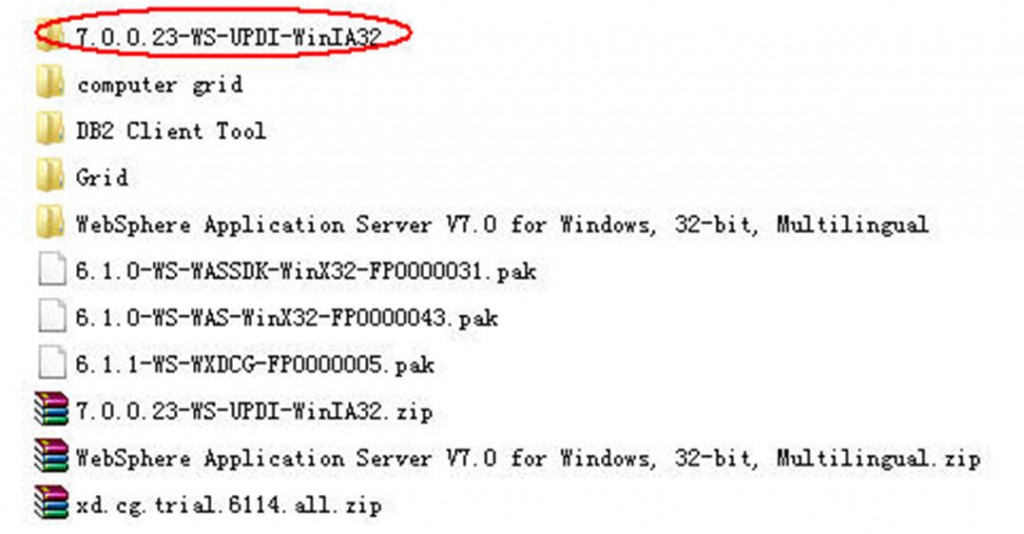
* IBM WAS7.0
* IBM Compute v6114
* IBM Compute Grid的开发框架，该框架使用ANT可帮助你通过ANT STEP1, ANT STEP2, ANT STEP3这样一步步实现你的xJCL文件，生成EAR，自动布署，简化你的开发，使得程序开发人员只要专注于写它们的reader, handler, writer就行了。

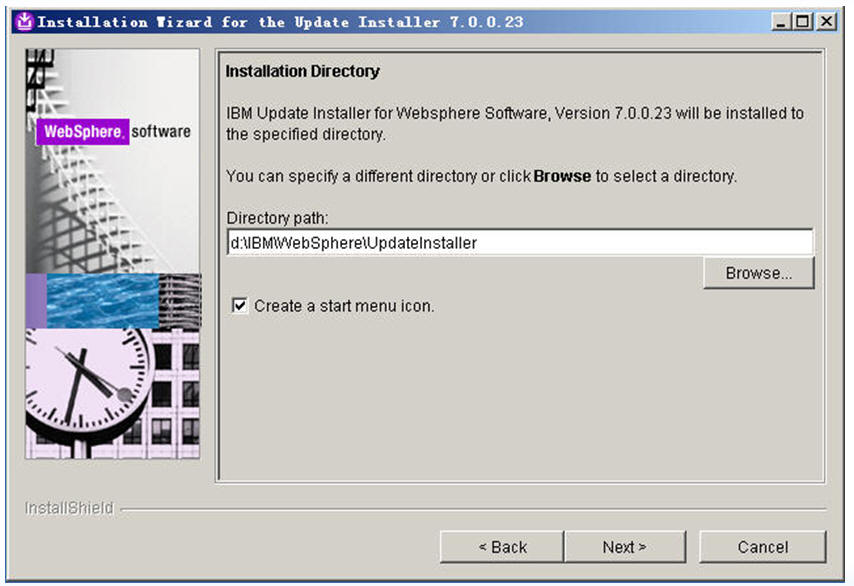
### **4.1.1 安装WAS**



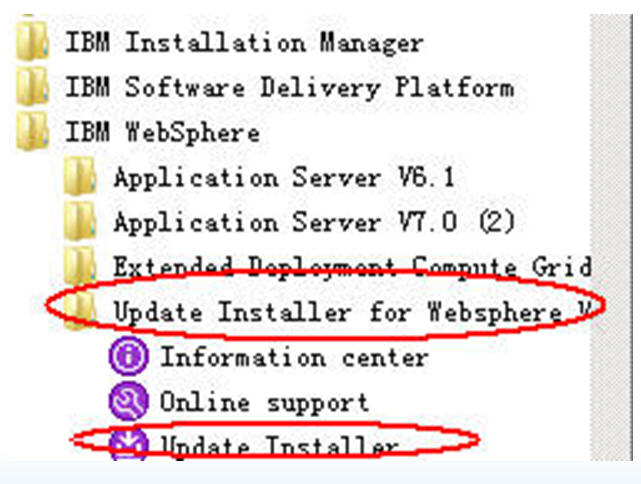
安装完WAS后不要忘了建一个profile。

然后把我们的WAS打一下补丁，打补丁前不要忘了先装IBM WAS 7.0.0.23 Updater





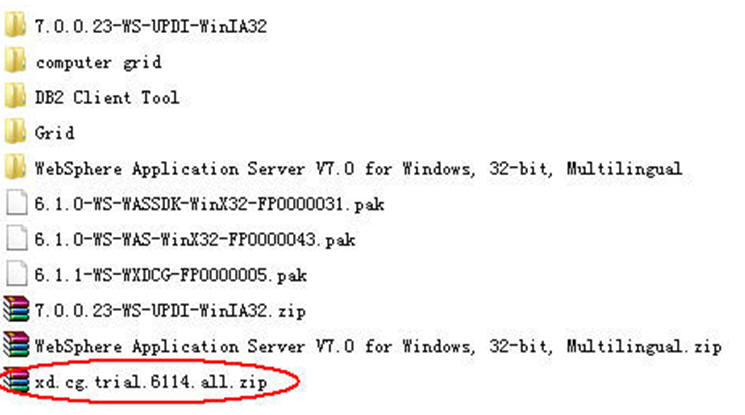
装完后在开始菜单选择这个选项来把WAS7.X的最新补丁（IBM官网去下）打上：



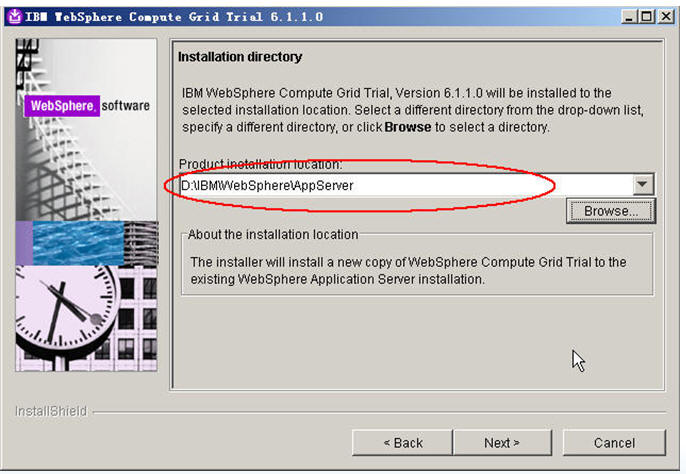
打完补丁后就可以安装IBM Compute Grid了.

## **4.2 安装IBM Compute Grid**

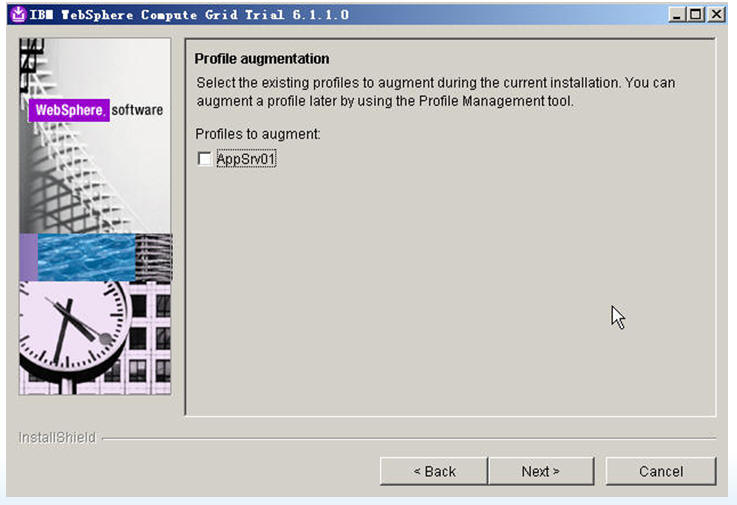
去官网下载这个包：



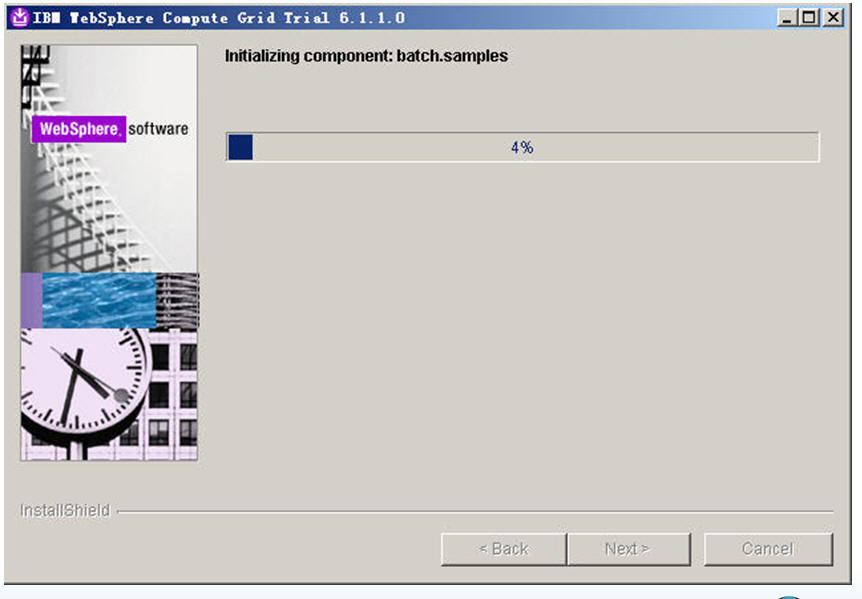
安装时它会自动认出你在WAS中建立的profile，一般WAS的profile都是带有一个AppServer的对吧？喏，它会自动认出



在下面这一步请先把这个自动认出的WAS中的profile里的AppServer的自动勾选改成“不勾选”，即我们不需要在Compute Grid安装时自动augment(扩展）我们的节点：

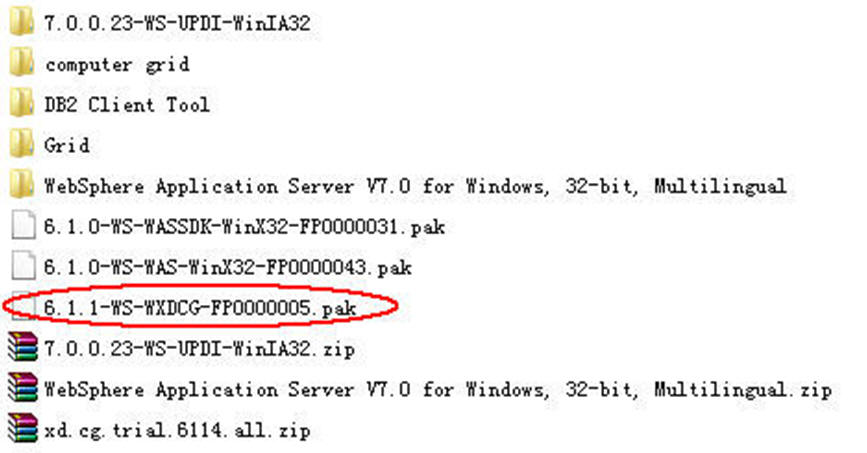


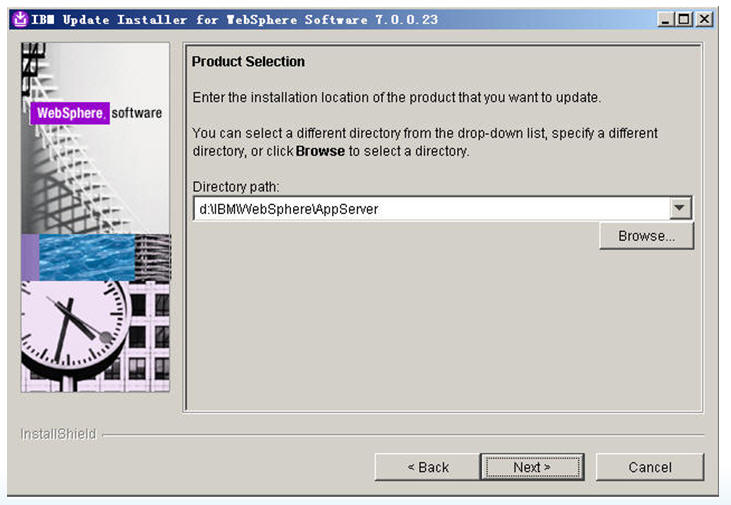
下一步下一步直到开始安装



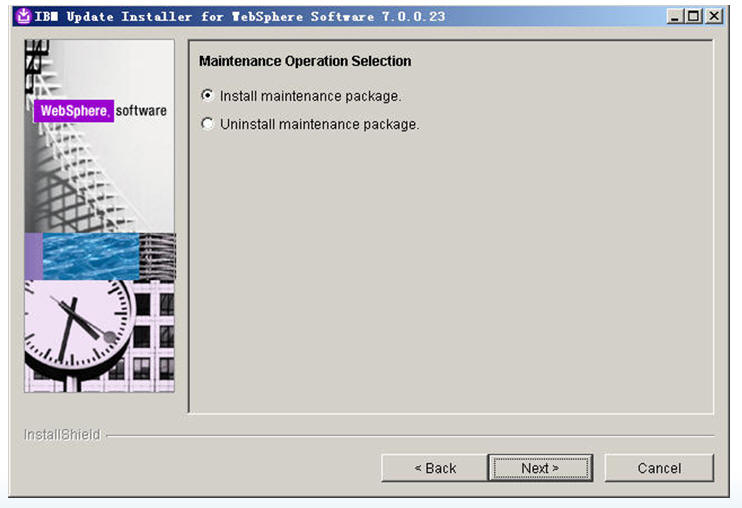
安装结束后不要勾选“start my first step to validation”这一选项，把这个选项取掉后可以直接点“finish"按钮。

装完后把IBM Compute Gridv6114升级到6115，也用我们的IBM WAS Updater来安装，你要在官网先下载这个补丁：

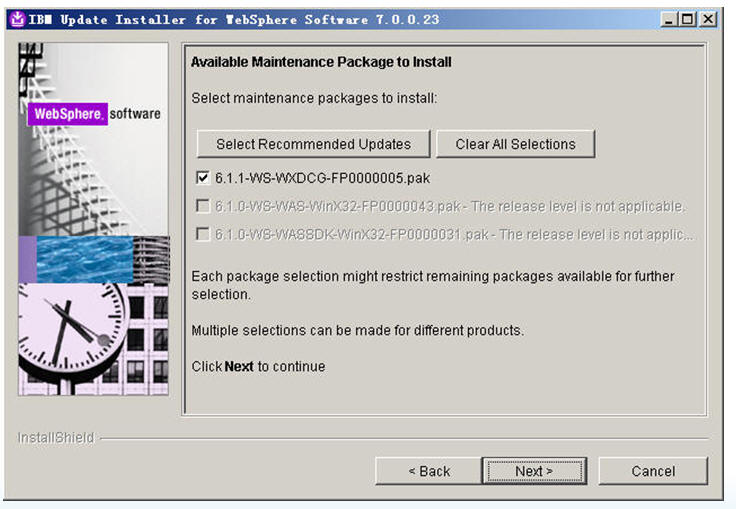




点下一步



然后把刚才下载的IBM Compute Grid 6115补丁（一个.pak文件）选入进updater manager后进行打补丁的操作



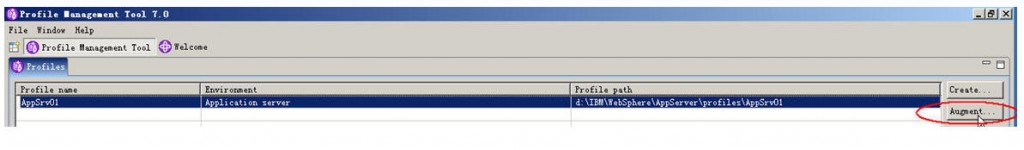
上面步骤全部执行完毕后，在开始菜单选择IBM WAS的菜单，在菜单中选择下面这一项：



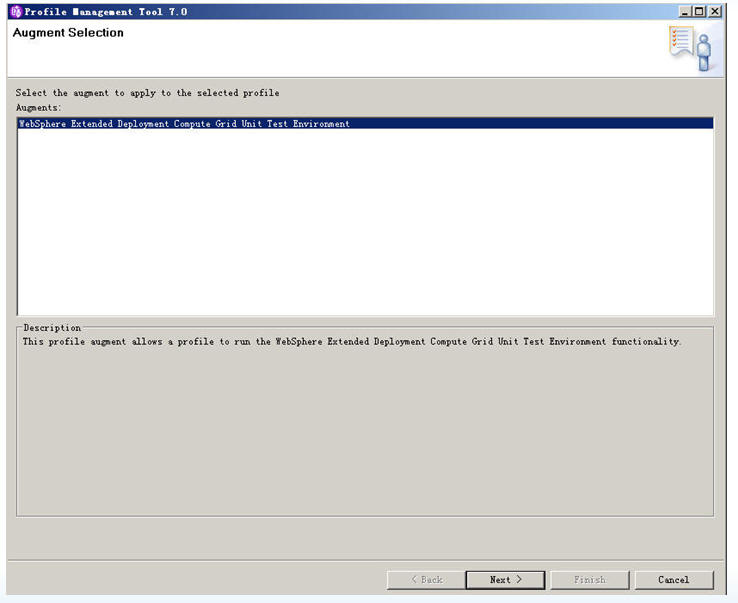
弹出下面这个界面



点“launch profile management tool”这个按钮

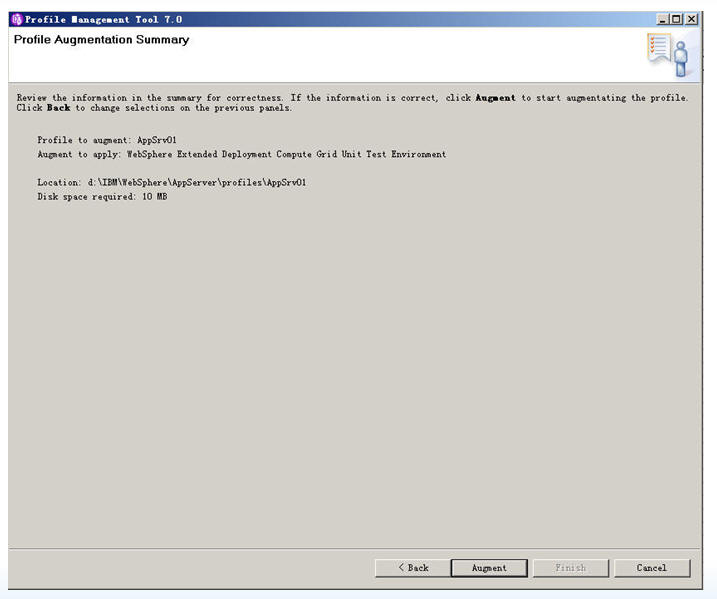


选择你要扩展的profile中的AppServer后点“Augment”按钮，然后下一步

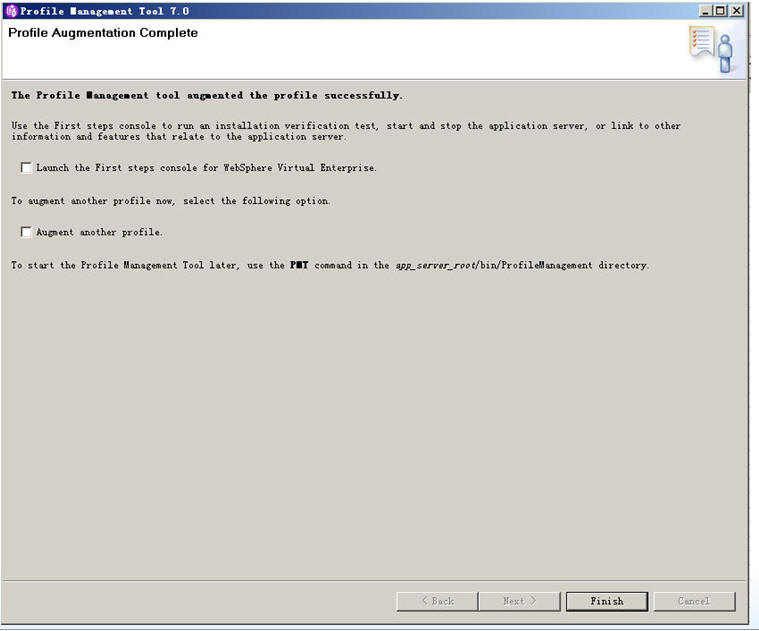


下一步

最后点"Augment"开始在我们的IBM WAS的profile中安装IBM Compute Grid组件



最后到了这个界面，按照我界面的选项，不要勾选任何选项，然后点“finish”完成安装



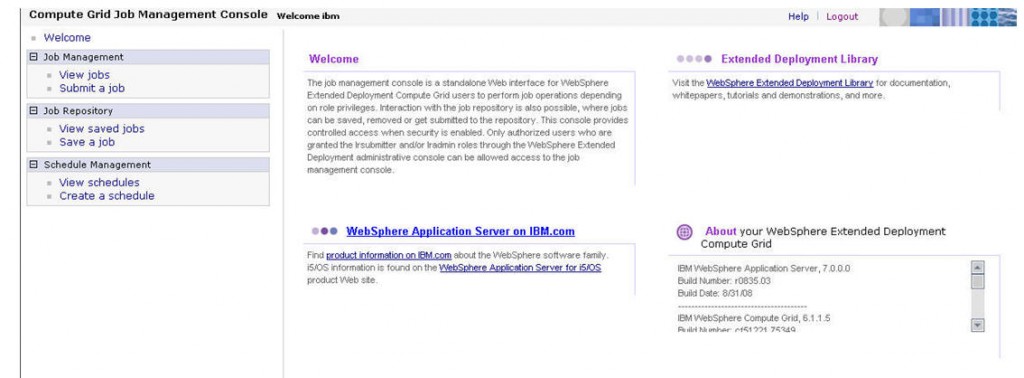
安装完后启动你的WAS服务，然后输入：[http://localhost:9060/ibm/console](http://localhost:9060/ibm/console" \t "http://blog.csdn.net/lifetragedy/article/details/_blank)。

这时你可以看到你的WAS中除了原有的你建立的profile外，还会多出一项：



再输入：[http://localhost:9080/jmc](http://localhost:9080/jmc" \t "http://blog.csdn.net/lifetragedy/article/details/_blank)

可以进入IBM Compute Grid的job management console



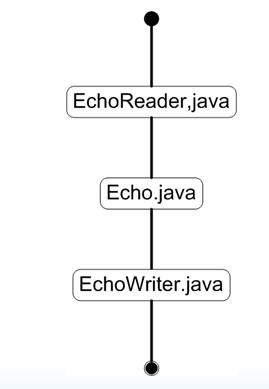
这就代表了我们的IBM Compute Grid已经正确的安装在了我们的WAS中了。

## **4.2 开发我们的第一个基于IBM Compute Grid的批处理程序**

这个程序很简单：

1. 读一个文件文件，几十万行。
2. 进行简单的整理程序，即把读入的文本文件中的列按照数据库内的表结构整理成一个java bean，java bean中每个字段对应着数据库表中的每一个字段。
3. 将读入的几十万行写入我们的数据库中去。

下面是我们的批处理的流程，够简单吧。



### **4.2.1 下载IBM Compute Grid开发框架IBM Compute Grid Batch Development Framework v.6.1.1.1.1**

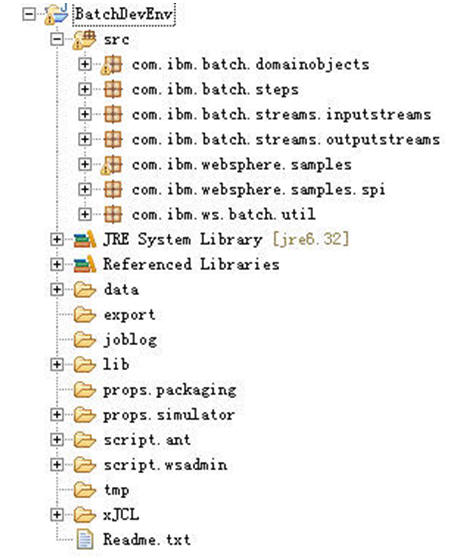
你可以通过官网下载，下载网址如下：

[http://www.ibm.com/developerworks/forums/thread.jspa?threadID=228339&tstart=0](http://www.ibm.com/developerworks/forums/thread.jspa?threadID=228339&tstart=0" \t "http://blog.csdn.net/lifetragedy/article/details/_blank)

我回头也会上传到我的博客的资源中去.



这是一个eclipse工程， 可以直接导入eclipse的workspace中去。

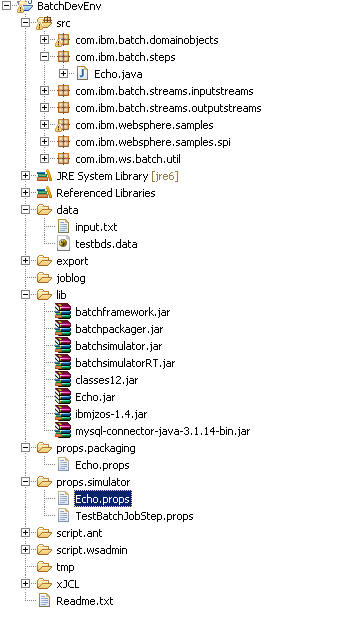


我们要读入的文本文件放在data目录，里面有一个input.txt文件，内容如下：

Date,Open,High,Low,Close,Volume,Adj Close                                        
2008-02-19,106.94,107.62,104.64,105.00,7376400,105.00                            
2008-02-15,105.27,106.25,105.00,106.16,6235600,106.16                            
2008-02-14,107.94,108.50,105.50,106.13,7340600,106.13                            
2008-02-13,107.13,108.93,106.80,108.42,6608200,108.42                            
2008-02-12,105.16,107.33,104.70,106.53,7650200,106.53                            
2008-02-11,103.05,105.55,102.87,105.14,6098300,105.14                            
2008-02-08,102.19,103.67,102.07,103.27,6085600,103.27                            
2008-02-07,102.89,104.00,100.60,102.34,11255800,102.34                           
2008-02-06,105.05,106.49,103.58,103.59,8265700,103.59                            
2008-02-05,107.06,108.05,104.68,105.02,9048900,104.62                            
2008-02-04,108.67,109.00,107.23,107.93,5985500,107.52                            
2008-02-01,107.16,109.40,105.86,109.08,8047100,108.66                            
2008-01-31,104.21,107.97,103.70,107.11,9054600,106.70                            
2008-01-30,105.85,107.65,104.86,105.65,7431100,105.25                            
2008-01-29,105.50,106.80,104.60,106.10,6616200,105.70                            
2008-01-28,104.44,105.77,103.83,104.98,7858500,104.58

我差不多定义了60万行吧。

在这边我使用的是mysql数据库，因此我把mysql相关的jdbc jar放入到了工程的lib目录中去了



接下去我要修改props.simulator文件夹中的Echo.props文件

**[plain]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "copy)

1. # (C) Copyright IBM Corp. 2008 - All Rights Reserved.
2. # DISCLAIMER:
3. # The following source code is sample code created by IBM Corporation.
4. # This sample code is provided to you solely for the purpose of assisting you
5. # in the  use of  the product. The code is provided 'AS IS', without warranty or
6. # condition of any kind. IBM shall not be liable for any damages arising out of your
7. # use of the sample code, even if IBM has been advised of the possibility of
8. # such damages.
10. job-name=Echo
11. application-name=Echo
13. #The following property references the WebSphere XD Compute Grid provided batch controller EJB
14. #when run in the batch simulator, this actually specifies a pojo wrapper class to the batch step.
15. #When you deploy this to a batch container running within an application server, this JNDI name
16. #has to be updated to reference the controller EJB for this step (which is generated for you by
17. #the batch packager).
19. controller-jndi-name=ejb/com/ibm/ws/batch/EchoBatchController
21. ##################################################################
22. # The utilityjars property specifies libraries required by
23. # this job.
24. #
25. # NOTE: this property is used only by the WSBatchPackager utility,
26. #       which is used to create an ear file for deploying this
27. #       batch application.
28. #
30. utilityjars=../lib/batchframework.jar;../lib/ibmjzos-1.4.jar
32. checkpoint-algorithm=com.ibm.wsspi.batch.checkpointalgorithms.RecordbasedBase
33. checkpoint-algorithm-prop.recordcount=1000
35. #Input Stream declarations
36. bds.inputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader
37. bds-prop.inputStream.PATTERN\_IMPL\_CLASS=com.ibm.batch.streams.inputstreams.EchoReader
38. bds-prop.inputStream.FILENAME=${echo.data}/input.txt

41. bds-prop.inputStream.debug=false
42. bds-prop.inputStream.EnablePerformanceMeasurement=false
43. bds-prop.inputStream.EnableDetailedPerformanceMeasurement=false
45. #data transformation declarations
47. batch\_bean-name=IVTStep1
48. batch-bean-jndi-name=ejb/GenericXDBatchStep
49. batch-step-class=com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep
51. #batch-bean-jndi-name=ejb/com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep
53. prop.BATCHRECORDPROCESSOR=com.ibm.batch.steps.Echo
54. prop.debug=false
55. prop.EnablePerformanceMeasurement=false
56. prop.EnableDetailedPerformanceMeasurement=false
58. #Output stream declarations
59. #bds.outputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteWriter
60. bds.outputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.LocalJDBCWriter
61. bds-prop.outputStream.PATTERN\_IMPL\_CLASS=com.ibm.batch.streams.outputstreams.EchoWriter
62. #oracle
63. #bds-prop.outputStream.jdbc\_url=jdbc:oracle:thin:@localhost:1521:ymkorcl
64. #bds-prop.outputStream.jdbc\_driver=oracle.jdbc.OracleDriver
65. #bds-prop.outputStream.userid=ymk
66. #bds-prop.outputStream.pswd=password\_1
68. bds-prop.outputStream.jdbc\_url=jdbc:mysql://localhost:3306/eltdb?useUnicode=true&characterEncoding=utf8
69. bds-prop.outputStream.jdbc\_driver=com.mysql.jdbc.Driver
70. bds-prop.outputStream.userid=elt
71. bds-prop.outputStream.pswd=password\_1
73. bds-prop.outputStream.tablename=t\_grid\_output\_test
74. bds-prop.outputStream.FILENAME=${echo.data}/output.txt
75. bds-prop.outputStream.AppendJobIdToFileName=false
76. bds-prop.outputStream.EnablePerformanceMeasurement=false
77. bds-prop.outputStream.EnableDetailedPerformanceMeasurement=false
78. bds-prop.outputStream.debug=false

注意这几句：

#Input Stream declarations  
bds.inputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader  
bds-prop.inputStream.PATTERN\_IMPL\_CLASS=com.ibm.batch.streams.inputstreams.EchoReader  
****bds-prop.inputStream.FILENAME=${echo.data}/input.txt****

****bds.outputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.LocalJDBCWriter****  
bds-prop.outputStream.PATTERN\_IMPL\_CLASS=com.ibm.batch.streams.outputstreams.EchoWriter

bds-prop.outputStream.jdbc\_url=jdbc:mysql://localhost:3306/eltdb?useUnicode=true&characterEncoding=utf8  
****bds-prop.outputStream.jdbc\_driver=com.mysql.jdbc.Driver****

bds-prop.outputStream.userid=myuserid  
bds-prop.outputStream.pswd=password\_1

通过这个properties文件我们可以得知我们的程序主要由3个类组成,它们分别是：

* com.ibm.batch.streams.inputstreams.EchoReader
* com.ibm.batch.streams.outputstreams.EchoWriter
* com.ibm.batch.steps.Echo

这三个类的调用是离散的，只是通过Echo.props文件中的描述：

****第一步：读入数据****

#Input Stream declarations  
bds.inputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader  
bds-prop.inputStream.PATTERN\_IMPL\_CLASS=com.ibm.batch.streams.inputstreams.EchoReader  
bds-prop.inputStream.FILENAME=${echo.data}/input.txt  
  
  
  
  
bds-prop.inputStream.debug=false  
bds-prop.inputStream.EnablePerformanceMeasurement=false  
bds-prop.inputStream.EnableDetailedPerformanceMeasurement=false  
  
  
#data transformation declarations  
  
  
batch\_bean-name=IVTStep1  
batch-bean-jndi-name=ejb/GenericXDBatchStep  
batch-step-class=com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep  
  
  
#batch-bean-jndi-name=ejb/com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep  
  
****第二步：处理数据****  
prop.BATCHRECORDPROCESSOR=com.ibm.batch.steps.Echo  
prop.debug=false  
prop.EnablePerformanceMeasurement=false  
prop.EnableDetailedPerformanceMeasurement=false  
  
****第三步：写数据****  
#Output stream declarations  
#bds.outputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteWriter  
bds.outputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.LocalJDBCWriter  
bds-prop.outputStream.PATTERN\_IMPL\_CLASS=com.ibm.batch.streams.outputstreams.EchoWriter

来定义的，读者可以根据这个规划自己定义自己的Step即工作步骤，并且在properties里任意调整各个步骤的顺序

下面给出

com.ibm.batch.streams.inputstreams.EchoReader  
com.ibm.batch.streams.outputstreams.EchoWriter  
com.ibm.batch.steps.Echo

这三个类的源码：

### **com.ibm.batch.streams.inputstreams.EchoReader**

**[java]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "copy)

1. **public** **class** EchoReader **implements** FileReaderPattern, JDBCReaderPattern,
2. ByteReaderPattern, RecordOrientedDatasetReaderPattern {
4. **protected** BDSFWLogger logger;
5. **protected** EchoDataHolder echoDataHolder;
7. // Properties for reading from a JDBC input source
8. **protected** String tableNameKey = "tablename";
9. **protected** String tableName = "ALG.TIVPWXD0";
10. **protected** String echoQuery = "select \* from ";
12. // Properties for reading bytes from a Byte or Dataset input source
13. **protected** String RecordLengthKey = "EchoReader\_record\_length";
14. **protected** String defaultRecordLength = "80";
15. **protected** **int** recordLength;
16. **protected** **byte**[] buf;
18. /\*\*
19. \* Initialize method is driven upon object creation. The properties object
20. \* passed in contains the properties specified for the input stream in the
21. \* xJCL.
22. \*/
23. **public** **void** initialize(Properties props) {
25. logger = **new** BDSFWLogger(props);
27. **if** (logger.isDebugEnabled())
28. logger.debug("entering EchoReader.initialize()");
30. // initialize any JDBC properties that have been defined
31. tableName = props.getProperty(**this**.tableNameKey, tableName);
32. echoQuery += tableName;
33. // -----------------------------------------------------
35. // initialize any byte reader properties that have been defined.
36. recordLength = Integer.valueOf(
37. props.getProperty(RecordLengthKey, defaultRecordLength))
38. .intValue();
39. buf = **new** **byte**[recordLength];
40. // -----------------------------------------------------
42. **if** (logger.isDebugEnabled())
43. logger.debug("exiting EchoReader.initialize()");
44. }
46. // The fetchHeader method exposes any header data that has been parsed for
47. // this input stream.
48. **public** Object fetchHeader() {
49. // no header data to expose.
50. **return** **null**;
51. }
53. // File read methods - FileReaderPattern implementation
55. // the processHeader method's task is to parse the data in the buffered
56. // reader and extract any header information that could be used for this
57. // stream.
58. // the java io methods of mark() and reset() should be used here to ensure
59. // that the BufferedReader, upon completion of parsing the header data, is
60. // positioned
61. // at the start of the first data record to be processed.
63. **public** **void** processHeader(BufferedReader arg0) **throws** IOException {
64. // n/a for this example.
65. }
67. // the task of the fetchRecord() method is to parse the raw data and map it
68. // to the domain object. The java io methods of mark() and reset() should be
69. // used
70. // to ensure the buffered reader, upon completion of obtaining the record,
71. // is at the starting position of the next data record to be processed.
73. **public** Object fetchRecord(BufferedReader reader) **throws** IOException {
74. String line = reader.readLine();
75. **if** (line != **null**) {
76. **if** (logger.isDebugEnabled())
77. logger.debug("EchoReader.fetchRecord(bufferedReader)- line= "
78. + line);
79. **return** **new** EchoDataHolder(line);
80. } **else** {
81. **if** (logger.isDebugEnabled())
82. logger.debug("EchoReader.fetchRecord(bufferedReader)- returning null");
83. **return** **null**;
84. }
85. }
87. // -----------------------------------------------------
89. // JDBC read methods - JDBCReaderPattern implementation
91. // the task for getInitialLookupQuery is to return the SQL query to be
92. // executed the first time this job is every invoked. Note, a seperate
93. // method is used
94. // for obtaining the query that should be executed if the job is restarted.
95. // For example: select \* from table 1;
96. **public** String getInitialLookupQuery() {
97. **return** echoQuery;
98. }
100. // the task for the getRestartQuery method is to create the SQL string that
101. // should be executed if the job is restarted. The String parameter to the
102. // method
103. // contains the data returned in getRestartTokens(). This data was persisted
104. // by the Batch container on behalf of this stream during a checkpoint.
105. // Upon restart, that data should be used to determine where to reposition
106. // the stream to (for example, as arguments for a WHERE clause in your SQL
107. // query).
108. // for example, select \* from table1 where recordName is between B and Z
109. **public** String getRestartQuery(String arg0) {
110. // TODO Auto-generated method stub
111. **return** **null**;
112. }
114. // the task for the getRestartTokens() method is to return any data that
115. // should be stored during a checkpoint for this stream. The data stored
116. // would be used
117. // to reposition the tream upon restart of the job. Think of this as
118. // returning data that you would need to populate the WHERE clause of an SQL
119. // query.
120. // For example, select \* from table1 where recordName is between B and Z
121. **public** String getRestartTokens() {
122. // restart logic for echo JDBC reader is not implemented.
123. **return** **null**;
124. }
126. // This method maps the columns of a database row to a hashmap.
127. // The hashmap can get be queried by the record processor for fields, where
128. // key = column name, value = column value.
130. **public** Object fetchRecord(ResultSet resultSet) {
132. **if** (logger.isDebugEnabled())
133. logger.debug("entering EchoReader.fetchRecord(resultSet)");
135. **try** {
136. ResultSetMetaData rsmd = resultSet.getMetaData();
137. **int** columnCount = rsmd.getColumnCount();
139. HashMap<String, Object> dbMap = **new** HashMap<String, Object>();
140. **for** (**int** i = 1; i <= columnCount; i++) {
141. String columnName = rsmd.getColumnName(i);
142. Object columnValue = resultSet.getObject(i);
143. dbMap.put(columnName, columnValue);
144. }
146. **if** (logger.isDebugEnabled()) {
147. logger.debug("EchoReader.fetchRecord(resultSet)- dbmap = "
148. + dbMap);
149. logger.debug("exiting EchoReader.fetchRecord(resultSet)");
150. }
152. **if** (dbMap.size() == 0)
153. **return** **null**;
154. **else**
155. **return** **new** EchoDataHolder(dbMap);
157. } **catch** (Throwable t) {
158. **throw** **new** RuntimeException(t);
159. }
160. }
162. // -----------------------------------------------------
164. // Byte read methods - ByteReaderPattern implementation
166. // the processHeader method's task is to parse the data in the
167. // bufferedInputStream reader and extract any header information that could
168. // be used for this stream.
169. // the java io methods of mark() and reset() should be used here to ensure
170. // that the bufferedInputStream, upon completion of parsing the header data,
171. // is positioned
172. // at the start of the first data record to be processed.
173. **public** **void** processHeader(BufferedInputStream arg0) **throws** IOException {
174. // n/a for this example
175. }
177. // the task of the fetchRecord() method is to parse the raw data and map it
178. // to the domain object. The java io methods of mark() and reset() should be
179. // used
180. // to ensure the buffered reader, upon completion of obtaining the record,
181. // is at the starting position of the next data record to be processed.
182. **public** Object fetchRecord(BufferedInputStream reader) **throws** IOException {
184. buf = **new** **byte**[recordLength];
185. **int** nread = reader.read(buf);
187. **if** (logger.isDebugEnabled())
188. logger.debug("fetchRecord(bufferedInputStream)\nbuf: " + buf
189. + "\nnread=" + nread);
191. **if** (nread > 0) {
192. String inputStr = (**new** String(buf, "utf-8")).trim();
193. // return new EchoDataHolder(buf, nread);
194. **return** **new** EchoDataBean(inputStr);
195. } **else** {
196. **return** **null**;
197. }
198. }
200. // -----------------------------------------------------
202. // MVS Dataset read methods - RecordOrientedDatasetReaderPattern
203. // implementation
205. // the processHeader method's task is to parse the data in the
206. // bufferedInputStream reader and extract any header information that could
207. // be used for this stream.
208. // the java io methods of mark() and reset() should be used here to ensure
209. // that the bufferedInputStream, upon completion of parsing the header data,
210. // is positioned
211. // at the start of the first data record to be processed.
212. **public** **void** processHeader(ZFile arg0) **throws** IOException {
213. // n/a for this example
214. }
216. // the task of the fetchRecord() method is to parse the raw data and map it
217. // to the domain object. The io methods of mark() and reset() should be used
218. // to ensure the buffered reader, upon completion of obtaining the record,
219. // is at the starting position of the next data record to be processed.
220. **public** Object fetchRecord(ZFile reader) **throws** IOException {
221. System.out.println("========>read record ZFile reader");
222. **byte**[] buf = **new** **byte**[reader.getLrecl()];
223. **int** nread = reader.read(buf);
225. **if** (logger.isDebugEnabled())
226. logger.debug("fetchRecord(zfile)\nbuf: " + buf + "\nnread=" + nread);
228. **if** (nread > 0) {
229. **if** (logger.isDebugEnabled())
230. logger.debug("nread is > 0, returning object");
231. **return** **new** EchoDataHolder(buf, nread);
232. } **else** {
233. **if** (logger.isDebugEnabled())
234. logger.debug("nread is < 0, returning null");
235. **return** **null**;
236. }
237. }
239. // -----------------------------------------------------
240. }

### **com.ibm.batch.streams.outputstreams.EchoWriter**

**[java]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "copy)

1. **public** **class** EchoWriter **implements** FileWriterPattern, ByteWriterPattern,
2. JDBCWriterPattern, RecordOrientedDatasetWriterPattern {
4. **protected** BDSFWLogger logger;
5. **protected** EchoDataHolder echoDataHolder;
6. **protected** String jobid;
7. **protected** String jobIdKey = "JobStepId";
8. **protected** **int** counter = 0;
10. // Properties for writing to a JDBC output source
11. **protected** String tableNameKey = "tablename";
12. **protected** String tableName = "t\_grid\_output\_test";
13. **protected** String sqlQueryPreTablename = "insert into ";
15. /\* oracle insert sql\*/
16. //protected String tableValues = "(pk\_id, my\_date, open,high,low,close,volume,adj\_close)";
17. //protected String sqlQueryPostTablename = " values (seq\_test\_output\_id.nextval, ?, ?, ?, ?, ?, ?,?)";
19. /\* mysql insert sql\*/
20. **protected** String tableValues = "(my\_date, open,high,low,close,volume,adj\_close)";
21. **protected** String sqlQueryPostTablename = " values (?, ?, ?, ?, ?, ?,?)";
23. **public** **void** initialize(Properties props) {
24. logger = **new** BDSFWLogger(props);
25. jobid = props.get(**this**.jobIdKey).toString();
27. **if** (logger.isDebugEnabled())
28. logger.debug("EchoWriter.initialize()");
30. // initialize any JDBC properties that have been defined
31. tableName = props.getProperty(**this**.tableNameKey, tableName);
32. }
34. // -----------------------------------------------
36. // File write methods - FileWriterPattern implementation
38. // the task for the writeHeader(bufferedWriter) method is to write any
39. // header data to the output stream prior to writing the output data
40. // records.
41. // This method is only called once, which is upon initialization of the
42. // first execution of this job. This means the header will not be written
43. // again
44. // if the job is restarted.
45. **public** **void** writeHeader(BufferedWriter arg0) **throws** IOException {
46. // n/a, no header data to write.
47. }
49. // the task for the writeHeader(bufferedWriter) method is to write the
50. // header object passed to this stream
51. // to the output stream prior to writing the output data records.
52. // This method is only called once, which is upon initialization of the
53. // first execution of this job. This means the header will not be written
54. // again
55. // if the job is restarted.
56. **public** **void** writeHeader(BufferedWriter arg0, Object arg1)
57. **throws** IOException {
58. }
60. // the task for the writeRecord method is to write the processed domain
61. // object to the output stream.
62. **public** **void** writeRecord(BufferedWriter out, Object record)
63. **throws** IOException {
64. **if** (counter != 0) {
65. out.newLine();
66. }
67. counter++;
68. **if** (logger.isDebugEnabled())
69. logger.debug("EchoWriter.writeRecord(BufferedWriter)- record= "
70. + record);
72. out.write(record.toString());
73. }
75. // ------------------------------------------------------
77. // byte writing methods - ByteWriterPattern implementation
79. // The task for the writeHeader(bufferedOutputStream) method is to write any
80. // header data to the output stream prior to writing the output data bytes.
81. // This method is only called once, which is upon initialization of the
82. // first execution of this job. This means the header will not be written
83. // again
84. // if the job is restarted.
85. **public** **void** writeHeader(BufferedOutputStream arg0) **throws** IOException {
86. // n/a, no header data to write.
88. }
90. // the task for the writeRecord method is to write the processed domain
91. // object to the output stream.
92. **public** **void** writeHeader(BufferedOutputStream arg0, Object arg1)
93. **throws** IOException {
94. }
96. // the task for the writeRecord method is to write the processed domain
97. // object to the output stream.
98. **public** **void** writeRecord(BufferedOutputStream out, Object record)
99. **throws** IOException {
101. EchoDataHolder holder = ((EchoDataHolder) record);
103. **if** (logger.isDebugEnabled())
104. logger.debug("writeRecord(bufferedOutputStream)\nbuf: " + holder
105. + "\nnread=" + holder.nread);
107. out.write(holder.getByteData(), 0, holder.nread);
108. }
110. // ------------------------------------------------------
112. // JDBC writing methods - JDBCWriterPattern implementation
114. // the task for the getSQLQuery method is to return an SQL string that is
115. // will be used to store this domain object in the database.
116. **public** String getSQLQuery() {
118. String sqlQuery = sqlQueryPreTablename + tableName + tableValues
119. + sqlQueryPostTablename;
120. // System.out.println("sqlQuery====" + sqlQuery);
121. **return** sqlQuery;
122. }
124. // the task for thie writeRecord(pstmt, record) method is to map the domain
125. // object to the prepared statement. The bds framework then manages
126. // executing that
127. // that prepared statement for you (because then we can do things like JDBC
128. // batching).
129. **public** PreparedStatement writeRecord(PreparedStatement pstmt, Object record) {
130. **try** {
131. **if** (logger.isDebugEnabled()) {
132. logger.debug("EchoWriter.writeRecord(PreparedStatement)- record= "
133. + record);
134. }
135. EchoDataBean echoData = (EchoDataBean) record;
136. //          System.out.println("mydate===" + echoData.getMyDate() + "  open==="
137. //                  + echoData.getOpen());
138. pstmt.setDate(1, StringUtil.convertStrToDate(echoData.getMyDate()));
139. pstmt.setDouble(2, echoData.getOpen());
140. pstmt.setDouble(3, echoData.getHigh());
141. pstmt.setDouble(4, echoData.getLow());
142. pstmt.setDouble(5, echoData.getClose());
143. pstmt.setDouble(6, echoData.getVolume());
144. pstmt.setDouble(7, echoData.getAdjClose());
145. counter++;
146. } **catch** (Throwable t) {
147. **throw** **new** RuntimeException(t);
148. }
149. **return** pstmt;
150. }
152. // -------------------------------------------------------
154. // writing bytes to a fixed-block MVS dataset -
155. // RecordOrientedDatasetWriterPattern implementation
157. // The task for the writeHeader(ZFile) method is to write any header data to
158. // the output stream prior to writing the output data bytes.
159. // This method is only called once, which is upon initialization of the
160. // first execution of this job. This means the header will not be written
161. // again
162. // if the job is restarted.
163. **public** **void** writeHeader(ZFile arg0) **throws** IOException {
164. // n/a, no header data to write.
165. }
167. // the task for the writeRecord method is to write the processed domain
168. // object to the output stream.
169. **public** **void** writeHeader(ZFile arg0, Object arg1) {
170. // no header data to write.
171. }
173. // the task for the writeRecord method is to write the processed domain
174. // object to the output stream.
175. **public** **void** writeRecord(ZFile out, Object record) **throws** IOException {
177. EchoDataHolder holder = ((EchoDataHolder) record);
179. **if** (logger.isDebugEnabled())
180. logger.debug("writeRecord(zFile)\nbuf: " + holder + "\nnread="
181. + holder.nread);
183. out.write(holder.getByteData(), 0, holder.nread);
184. }
185. }

### **com.ibm.batch.steps.Echo**

**[java]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "copy)

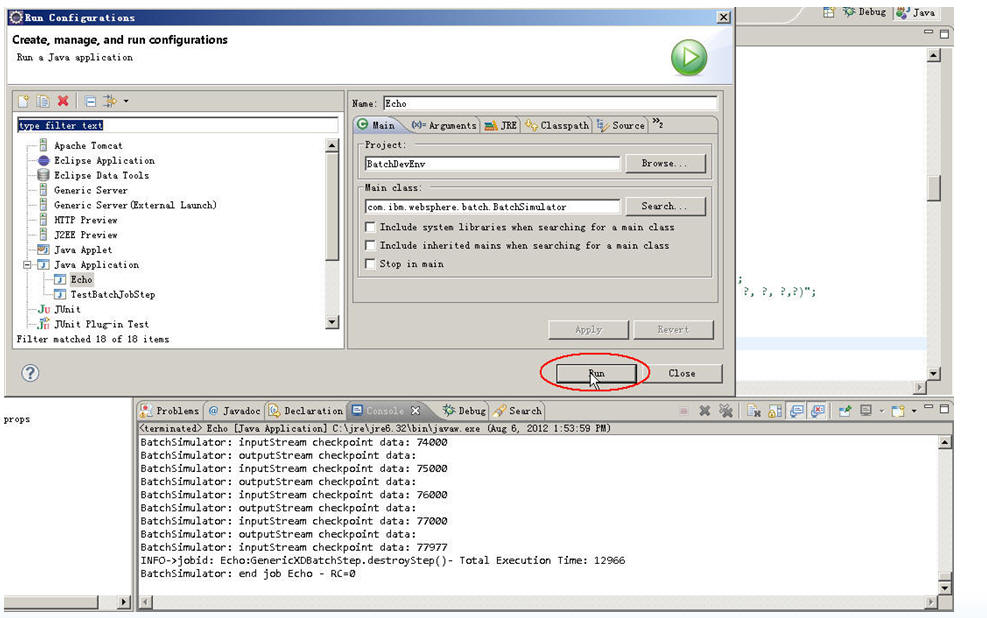
1. **public** **class** Echo **implements** BatchRecordProcessor {
3. **protected** BDSFWLogger logger;
4. Integer time= 0;
5. Integer count= 0;
7. // this method is called once by GenericXDBatchStep.initializeJobStep()
8. **public** **void** initialize(Properties arg0) {
10. logger = **new** BDSFWLogger(arg0);
11. **if** (logger.isDebugEnabled())
12. logger.debug("initialize.");

15. }
17. // this method is called repeatedly by GenericXDBatchStep.processJobStep()
18. **public** Object processRecord(Object domainObject) **throws** Exception {
20. **if** (logger.isDebugEnabled())
21. logger.debug("processing record: " + domainObject);
23. // Since this is an echo step, just return the domain object that was passed in.
24. **return** domainObject;
25. }
27. // this method is called once by GenericXDBatchStep.destroyJobStep()
28. **public** **int** completeProcessing() {
29. **try** {
30. }
31. **catch**(Exception e) {
32. e.printStackTrace();
33. }
34. **if** (logger.isDebugEnabled())
35. logger.debug("completed processing.");
36. **return** 0;
37. }
38. }

## **4.3 布署批处理作业至IBM WAS Compute Grid**

### **4.3.1 布署前的程序调试**

一般我们找到我们的"handler“，然后在eclipse里使用右键run as->把main class填成：com.ibm.websphere.batch.BatchSimulator来进行测试。



### **4.3.2 生成布署包**

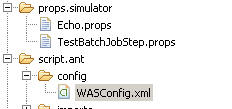
一切在eclipse中调试通过后我们就可以开始生成我们的布署包了。

### **第一步：WAS设置**

定位到script.ant.config文件夹中的WASConfig，设置你将要布署的WAS的相关连接参数：

**[html]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "copy)

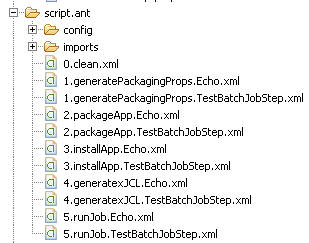
1. **<project** name="WASConfig" default="checkIfHomeSet"**>**
2. **<property** name="WAS\_profile\_home\_default" value="D:/IBM/WebSphere/AppServer/profiles/AppSrv01" **/>**
3. **<property** name="WAS\_appserver\_host\_default" value="localhost" **/>**
4. **<property** name="WAS\_appserver\_port\_default" value="9082" **/>**
6. **<property** name="WAS\_adminserver\_host\_default" value="${WAS\_appserver\_host\_default}" **/>**
7. **<property** name="WAS\_adminserver\_port\_default" value="unset" **/>**
8. **<property** name="deploymentTarget\_default" value=""**/>**
10. **<property** name="wsadminScript\_default" value="${basedir}/../script.wsadmin/installApp.py" **/>**
12. **<target** name="setDefaults"**>**
13. **<condition** property="WAS\_profile\_home" value="${WAS\_profile\_home\_default}"**>**
14. **<not>**
15. **<isset** property="${WAS\_profile\_home}" **/>**
16. **</not>**
17. **</condition>**
18. **<condition** property="WAS\_appserver\_host" value="${WAS\_appserver\_host\_default}"**>**
19. **<not>**
20. **<isset** property="${WAS\_appserver\_host}" **/>**
21. **</not>**
22. **</condition>**
23. **<condition** property="WAS\_appserver\_port" value="${WAS\_appserver\_port\_default}"**>**
24. **<not>**
25. **<isset** property="${WAS\_appserver\_port}" **/>**
26. **</not>**
27. **</condition>**
28. **<condition** property="WAS\_adminserver\_host" value="${WAS\_adminserver\_host\_default}"**>**
29. **<not>**
30. **<isset** property="${WAS\_adminserver\_host}" **/>**
31. **</not>**
32. **</condition>**
33. **<condition** property="WAS\_adminserver\_port" value="${WAS\_adminserver\_port\_default}"**>**
34. **<not>**
35. **<isset** property="${WAS\_adminserver\_port}" **/>**
36. **</not>**
37. **</condition>**
38. **<condition** property="deploymentTarget" value="${deploymentTarget\_default}"**>**
39. **<not>**
40. **<isset** property="${deploymentTarget}" **/>**
41. **</not>**
42. **</condition>**
43. **<condition** property="wsadminScript" value="${wsadminScript\_default}"**>**
44. **<not>**
45. **<isset** property="${wsadminScript}" **/>**
46. **</not>**
47. **</condition>**
48. **<echo** message="WASConfig.xml set WAS\_profile\_home=${WAS\_profile\_home}" **/>**
49. **<echo** message="WASConfig.xml set WAS\_appserver\_host=${WAS\_appserver\_host}" **/>**
50. **<echo** message="WASConfig.xml set WAS\_appserver\_port=${WAS\_appserver\_port}" **/>**
51. **<echo** message="WASConfig.xml set WAS\_adminserver\_host=${WAS\_adminserver\_host}" **/>**
52. **<echo** message="WASConfig.xml set WAS\_adminserver\_port=${WAS\_adminserver\_port}" **/>**
53. **<echo** message="WASConfig.xml set deployementTarget=${deployementTarget}" **/>**
54. **<echo** message="WASConfig.xml set wsadminScript=${wsadminScript}" **/>**
55. **</target>**
57. **<target** name="checkIfHomeSet" depends="setDefaults"**>**
58. **<echo** message="Checking if default WAS home set ..."**/>**
59. **<condition** property="homeNotSet"**>**
60. **<equals** arg1="${WAS\_profile\_home}" arg2="unset"**/>**
61. **</condition>**
62. **<fail** message="Must set property WAS\_profile\_home\_default in script.ant/config/WASConfig.xml" if="homeNotSet"**/>**
63. **<echo** message="WAS profile home set to ${WAS\_profile\_home}"**/>**
64. **</target>**
66. **</project>**



### **第二步：启动WAS**

### **第三步：**

在eclipse中定位到下面这个文件夹，看到一堆的xml文件了吗？



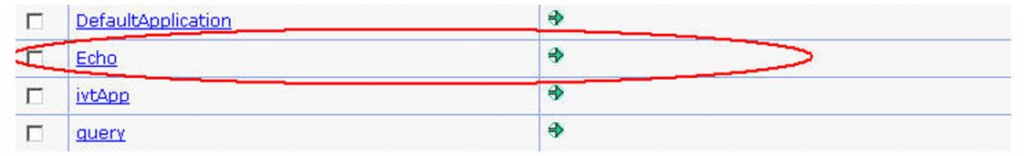
依下面的次序把每个xml文件打开后用ANT运行一下：

* clean.xml
* generatePackagingProps.Echo.xml
* packageApp.Echo.xml
* installApp.Echo.xml
* generatexJCL.Echo.xml

### **第四步：重启你的WAS**

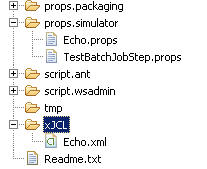
此时，进入这个网址：[http://localhost:9060/ibm/console](http://localhost:9060/ibm/console" \t "http://blog.csdn.net/lifetragedy/article/details/_blank)

在WAS的企业程序菜单中你可以看到有一个ear的应用已经布署并应该随着你的第三步中的重启WAS后而处于启动阶段了：



怎么执行我们的批处理脚本呢？

来，看



看到这个Echo.xml文件了吗？这个是根据Echo.props文件在进行AntgeneratexJCL.Echo.xml的过程中自动生成的批处理脚本，即.xJCL文件

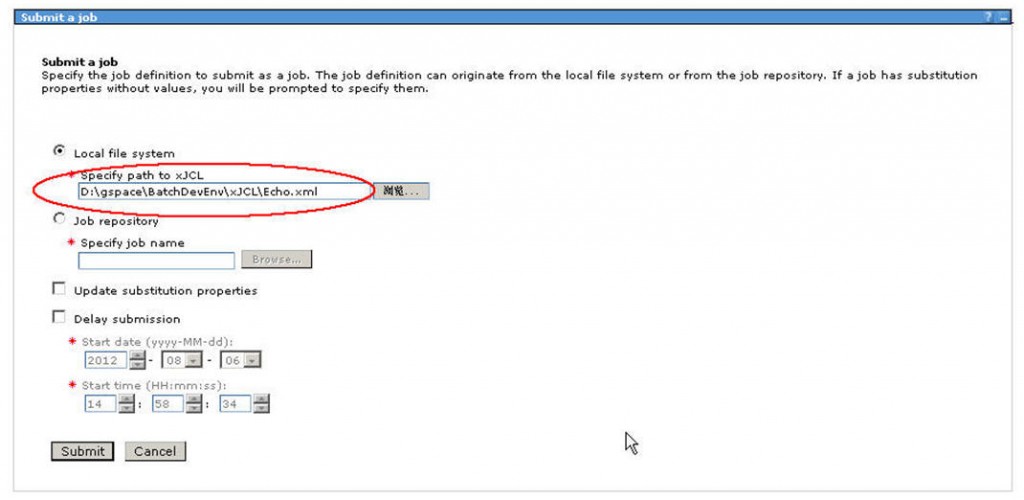
Echo.xml文件（xJCL）

**[html]** [view plain](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "view plain) [copy](http://blog.csdn.net/lifetragedy/article/details/8507252" \o "copy)

1. **<?xml** version="1.0" encoding="UTF-8" **?>**
2. <!--
3. #
4. # WebSphere Batch xJCL
5. #
6. # This file generated on 2012.08.06 at 14:12:13 CST by:
7. #   BatchSimulator Version: WXD611 [cf20947.48960]
8. #
9. --**>**
10. **<job** name="Echo" default-application-name="Echo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**>**
11. **<jndi-name>**ejb/com/ibm/ws/batch/EchoBatchController**</jndi-name>**
12. **<step-scheduling-criteria>**
13. **<scheduling-mode>**sequential**</scheduling-mode>**
14. **</step-scheduling-criteria>**
15. **<checkpoint-algorithm** name="chkpt"**>**
16. **<classname>**com.ibm.wsspi.batch.checkpointalgorithms.RecordbasedBase**</classname>**
17. **<props>**
18. **<prop** name="recordcount" value="1000"**/>**
19. **</props>**
20. **</checkpoint-algorithm>**
21. **<results-algorithms>**
22. **<results-algorithm** name="jobsum"**>**
23. **<classname>**com.ibm.wsspi.batch.resultsalgorithms.jobsum**</classname>**
24. **</results-algorithm>**
25. **</results-algorithms>**
26. **<job-step** name="Step1"**>**
27. **<jndi-name>**ejb/GenericXDBatchStep**</jndi-name>**
28. **<checkpoint-algorithm-ref** name="chkpt"**/>**
29. **<results-ref** name="jobsum"**/>**
30. **<batch-data-streams>**
31. **<bds>**
32. **<logical-name>**outputStream**</logical-name>**
33. **<props>**
34. **<prop** name="pswd" value="password\_1"**/>**
35. **<prop** name="AppendJobIdToFileName" value="false"**/>**
36. **<prop** name="debug" value="false"**/>**
37. **<prop** name="EnableDetailedPerformanceMeasurement" value="false"**/>**
38. **<prop** name="jdbc\_driver" value="com.mysql.jdbc.Driver"**/>**
39. **<prop** name="EnablePerformanceMeasurement" value="false"**/>**
40. **<prop** name="userid" value="elt"**/>**
41. **<prop** name="FILENAME" value="${echo.data}/output.txt"**/>**
42. **<prop** name="tablename" value="t\_grid\_output\_test"**/>**
43. **<prop** name="jdbc\_url" value="jdbc:mysql://localhost:3306/eltdb"**/>**
44. **<prop** name="PATTERN\_IMPL\_CLASS" value="com.ibm.batch.streams.outputstreams.EchoWriter"**/>**
45. **</props>**
46. **<impl-class>**com.ibm.websphere.batch.devframework.datastreams.patterns.LocalJDBCWriter**</impl-class>**
47. **</bds>**
48. **<bds>**
49. **<logical-name>**inputStream**</logical-name>**
50. **<props>**
51. **<prop** name="FILENAME" value="${echo.data}/input.txt"**/>**
52. **<prop** name="EnablePerformanceMeasurement" value="false"**/>**
53. **<prop** name="EnableDetailedPerformanceMeasurement" value="false"**/>**
54. **<prop** name="debug" value="false"**/>**
55. **<prop** name="PATTERN\_IMPL\_CLASS" value="com.ibm.batch.streams.inputstreams.EchoReader"**/>**
56. **</props>**
57. **<impl-class>**com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader**</impl-class>**
58. **</bds>**
59. **</batch-data-streams>**
60. **<props>**
61. **<prop** name="EnablePerformanceMeasurement" value="false"**/>**
62. **<prop** name="EnableDetailedPerformanceMeasurement" value="false"**/>**
63. **<prop** name="debug" value="false"**/>**
64. **<prop** name="BATCHRECORDPROCESSOR" value="com.ibm.batch.steps.Echo"**/>**
65. **</props>**
66. **</job-step>**
67. **</job>**

****在这个文件中我们可以任意改动其们批处理的步骤，调整我们的批处理脚本的各个步骤的执行顺序而不需要改动源码与再打包。****

打开我们的IBM Compute Grid的job management console界面,输入这个网址[http://localhost:9080/jmc](http://localhost:9080/jmc" \t "http://blog.csdn.net/lifetragedy/article/details/_blank)



看到了米有？ 把这个xml文件填入Specify path to xJCL这一栏，点”submit“按钮，就可以执行该批处理了，通过这个界面，想想我在上面”商业级解决方案“中提到过的5点需求，是不是它都可以执行了，嘿嘿，大家看看单节点执行的效率吧。

# **五、通过WASND来实现批处理网格计算**

这个留给大家自行练习了。

你不需要改动任何程序代码，只要：

1. 布署一个WASND
2. 安装IBM Compute Grid后在Augment profile时选择WASND的主控域
3. IBM Compute Grid会根据指定的WASND的主控域自动安装到各个WASND中的节点
4. 设置需要布署的WASND的相关参数
5. 依次运行我们的一系列ANT用的XML
6. 重启WASND
7. 还是通过[http://localhost:9080/jmc](http://localhost:9080/jmc" \t "http://blog.csdn.net/lifetragedy/article/details/_blank)进入IBM Compute Grid控制台
8. 提交xJCL

此时，这个批处理就会使用WASND中的各个布署有IBM Compute Grid的节点来进行”****网格计算了****”.

注意：

***这边的网格计算和传统的集群和load balance还是有区别的，这边的网格计算更类似于原来我有一个CPU，4G内存处理100万笔记录，通过WASND增加了一个节点后我就拥有1\*2个CPU以及4GB\*2的内存来运行同样的100万笔记录了，然后我再WASND一个节点，那么就再增加我批处理的处理能力。***