

File Queue 자바 개발자 매뉴얼

(FileQueue Middleware ver14.x.x 기준)

리눅스 버전

최종 작성일: 2024/04/02

개요

메시지 기반 미들웨어인 FileQueue 는 C언어로 개발되었으며, JAVA 개발자를 위한 JNI 인터페이스를 제공한다. 본 제품은 리눅스와 유닉스 OS에서 작동하며, 윈도우는 지원하지 않는다.

본 문서에서는 자바 개발자가 FileQueue 사용을 경험하기 위하여 리눅스에서 본 미들웨어를 설치하는 과정과 설치 후 java 프로그래밍을 통한 FileQueue 접근 API를 설명한다.

실제 현장에서는 현업의 미들웨어 담당자가 설치, 라이선스 발급 및 환경설정을 하게 되기 때문에 java 개발자는 설치과정, 환경설정, 큐생성 및 접근확인 절차는 숙지하지 않아도 된다.

따라서, 자바 개발자는 앞의 내용은 참고만 하고 문서의 마지막 내용(9페이지: 자바 개발 및 테스트)인 java 프로그래밍 과정으로 바로 넘어 가도 된다.

준비사항

- 리눅스 서버 (삼성증권은 레드햇 계열 8.7 을 사용중입니다. 본 패키지는 레드햇 8.7 에서 빌드되었습니다.)
- 패키지 다운로드 및 설치
- 자바 설치

FileQueue 미들웨어는 라이선스 파일이 설치되어 있어야만 동작하며, 라이선스를 얻기 위해서는 설치 하고자 하는 서버의 운영체제 버전과 호스트 이름을 패키지 공급사에 전달하여, 패키지와 라이선스파일을 전달받아 설치하여야 한다.(단, 개발단계에서는 1개월의 유효기간이 있는 Trial 라이선스가 공급하는 패키지 안에 포함되어 있음)

패키지가 동작을 위한 사전 설치 소프트웨어는 아래와 같다.

- jdk 1.8 이상
- 제공되는 모니터링 유틸리티 동작을 위한 ncurses 라이브러리.
-

패키지 공급

패키지는 2가지 형태로 공급된다.

- GitHub
- Package 전달(온라인 또는 오프라인)

서버가 인터넷에 연결되어 있는 경우에는 git 명령을 통해 GitHub에서 clone을 다운 받을 수 있다. 설치하고자 하는 디렉토리로 이동 후 아래의 명령을 수행한다.

```
# git clone http://github.com/c3cc3/java\_package.git <enter>
```

```
[ums@Lenovo-P1 tmp]$ git clone http://github.com/c3cc3/java_package.git
Cloning into 'java_package'...
warning: redirecting to https://github.com/c3cc3/java_package.git/
remote: Enumerating objects: 2469, done.
remote: Counting objects: 100% (2469/2469), done.
remote: Compressing objects: 100% (1688/1688), done.
remote: Total 2469 (delta 669), reused 2448 (delta 648), pack-reused 0
Receiving objects: 100% (2469/2469), 16.53 MiB | 6.38 MiB/s, done.
Resolving deltas: 100% (669/669), done.
```

명령을 수행하고 나면 현재 디렉토리에 java_package 라는 디렉토리가 생성된다.

```
[ums@Lenovo-P1 tmp]$ cd java_package/
[ums@Lenovo-P1 java_package]$ ll
total 3580
-rw-r--r-- 1 ums ums 55 Apr 3 21:17 Debian.txt
-rw-r--r-- 1 ums ums 61 Apr 3 21:17 README.md
-rw-r--r-- 1 ums ums 76 Apr 3 21:17 RHEL.txt
drwxr-xr-x 2 ums ums 4096 Apr 3 21:17 bin
drwxr-xr-x 2 ums ums 4096 Apr 3 21:17 conf
drwxr-xr-x 2 ums ums 4096 Apr 3 21:17 enmq
-rwxr-xr-x 1 ums ums 486 Apr 3 21:17 env.sh
drwxr-xr-x 3 ums ums 4096 Apr 3 21:17 include
drwxr-xr-x 2 ums ums 4096 Apr 3 21:17 lib
drwxr-xr-x 2 ums ums 4096 Apr 3 21:17 logs
-rw-r--r-- 1 ums ums 3612591 Apr 3 21:17 ncurses-6.4.tar.gz
drwxr-xr-x 5 ums ums 4096 Apr 3 21:17 share
drwxr-xr-x 2 ums ums 4096 Apr 3 21:17 src
drwxr-xr-x 5 ums ums 4096 Apr 3 21:17 utl
```

온/오프라인을 통해 패키지 파일을 전달받은 경우에는 패키지 파일을 서버에 옮긴 후, 아래 명령

으로 패키지를 푼다.

```
$ tar xvfz java_package.tar.gz <ENTER>
```

디렉토리 내용은 위와 동일하다.

파일을 풀고 나면 java_package 디렉토리가 자동으로 생성되며 그 아래에는 다음과 같은 디렉토리들과 파일이 존재한다.

- bin: 미들웨어 유틸리티 파일
- enmq: 파일큐와 라이선스/info 파일 가 존재하는 위치
- src: java source examples 및 java classs build script
- conf: 예제 소스를 구동 시키기 위한 config 파일
- utl : 예제소스 테스트를 위한 프로그램들
- include: include 파일들
- lib: FileQueue 라이브러리 파일
- logs: 어플리케이션 구동시 미들웨어처리 관련 로그파일 위치
- setup.txt : (파일) : 사용절차 매뉴얼(본 문서 대신 이 파일을 참고하여 설치 진행을 해도 됨)
- env.sh: (파일) 환경변수 값 셋팅 shell script

환경값을 로그인 관련 rc 파일(.bashrc, .profile 등)에 셋팅하지 않고 일시적으로 환경값을 변경하여 테스트 및 개발을 원할 경우 이 스크립트를 내 환경에 맞게 vi 편집하여 수행한 후 테스트 및 개발을 진행해도 된다.

env.sh 파일 내용은 아래와 같다.

```
[ums@Lenovo-P1 java_package]$ cat env.sh
#export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
#export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.402.b06-2.0.1.el8.x86_64
export JAVA_HOME=/opt/jdk-17
export FQ_WORK_HOME=/tmp/java_package
export FQ_DATA_HOME=/tmp/java_package/enmq
export FQ_LIB_HOME=/tmp/java_package/lib
export PATH=$PATH:/tmp/java_package/bin:$JAVA_HOME
export UMS_COMMON_CONF_FILE=/tmp/java_package/conf/ums_common.conf
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/tmp/java_package/lib:$JAVA_HOME/lib:
[ums@Lenovo-P1 java_package]$
```

환경파일 내용은 실제 설치할 내용으로 편집한다. 위는 /tmp/java_package 에 설치하는 예제이자.

그리고, JAVA가 설치되어 있는 위치를 확인 후, JAVA_HOME 값을 맞게 수정한다.

수정이 끝나면 아래 명령으로 수행시킨다.

```
# source env.sh <ENTER>
```

환경값이 적용되었는지 확인하려면 아래 명령으로 확인한다.

```
# echo $FQ_DATA_HOME <ENTER>  
/tmp/java_package/enmq ->큐파일이 생성된 위치가 확인되면 정상
```

사용절차 요약

1. 환경변수 값 셋팅
2. 라이선스 발급 및 설치 (패키지에는 30일 유효기간의 Trial 라이선스가 동봉되어 있음)
3. 큐 생성
4. 큐 접근 확인
5. Java 개발 및 테스트

사용절차 상세

1. 환경 변수값 셋팅

미들웨어가 정상 동작하기 위한 환경 셋팅이며, 본 문서에서는 리눅스에서 사용자 홈에 존재하는 .bashrc 또는 .profile에 추가하여야 하는 내용을 설명한다.

설치 디렉토리의 env.sh 파일의 내용을 복사하여 .bashrc 또는 .profile에 붙여넣기를 한다.

```
export JAVA_HOME=/opt/jdk-17  
export FQ_WORK_HOME=/tmp/java_package  
export FQ_DATA_HOME=/tmp/java_package/enmq  
export FQ_LIB_HOME=/tmp/java_package/lib  
export PATH=$PATH:/tmp/java_package/bin:$JAVA_HOME  
export UMS_COMMON_CONF_FILE=/tmp/java_package/conf/ums_common.conf  
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/tmp/java_package/lib:$JAVA_HOME/lib:
```

JAVA_HOME은 서버의 JAVA 설치 환경에 맞게 수정한다. 나머지 항목은 사용자 환경에 맞게 수정 하여야 한다

.bashrc 수정 후에는 터미널 로그아웃 후, 재 로그인을 해야 한다. (또는 \$source .bashrc <ENTER> 수행)

재로그인 후에 환경변수 확인

```
$ echo $FQ_DATA_HOME <ENTER>  
출력)  
/home/javauser/java_package/enmq
```

위와 같이 출력되면 정상이다

/home/javauser 는 실제 패키지를 푼 홈 디렉토리로 나타난다.

2. 정식/임시 라이선스 발급 및 설치

라이선스 발급을 위해서는 공급사에 설치하고자 하는 서버의 운영체제 환경과 호스트이름을 전달 하여야 한다. 발급받은 라이선스 파일을 패키지 설치 디렉토리의 enmq 디렉토리 밑에 이동시키 면 된다.

제공된 패키지내에는 제공일로부터 1개월간 유효한 Trial 라이선스가 동봉되어 짐.

```
-rw-r--r-- javauser javauser 28 May 3 03:06 DESKTOP-G7IIL9O.license
```

위의 예시는 DESKTOP-G7IIL9O 라는 호스트이름을 가진 서버의 정식 라이선스가 발급된 경우이며 임시 라이선스가 발급된 경우에는 아래와 같이 나타남.

```
-rw-r--r-- 1 ums ums 28 6월 10 21:08 trial
```

정식 라이선스 파일명은 호스트이름.license 로 존재한다. Trial 일 경우 Trial 파일이 존재.

만약, Trial 라이선스가 만료되면 큐 동작에 오류가 발생한다. 필요시 공급자에게 연락하면 재발급 된 Trial 파일이 공급된다.

3. 큐생성

큐는 여러 디렉토리에 생성될 수 있다.

이 때, 큐가 설치된 모든 디렉토리를 지정하는 파일이 `~/enmq/ FQ_directories.list` 파일이다.

이 파일에는 1개 이상의 큐설치 디렉토리가 지정되어 있어야 한다.

`/tmp/java_package/enmq`

위는 파일 내용이다.

큐를 생성하기 위해서는 `~/bin/fqm` 이라는 유틸리티를 사용한다.

큐생성 방법에는 2가지 방법이 있다.

여러 개의 큐를 한번의 명령으로 생성하고자 할 때에는 큐 리스트 파일을 작성하고

개별로 하나씩 생성하고 한다면 info파일 생성과 명령을 수행한다.

먼저, 여러 개의 큐를 한번에 생성하는 방법이다.

Package 안의 enmq 폴더 아래에는 샘플 리스트파일(Samsung.list)이 존재한다.

이 파일을 열어 첫번째 칼럼의 디렉토리 부분을 실제로 맞게 수정한다.

2번째 칼럼은 큐이름을 적는다 큐이름은 16자 이내의 영문/숫자/_/- .문자만 지원한다.

3번째 칼럼은 맵핑숫자이다. 특별한 경우가 아니면 100을 사용한다.

4번째 칼럼은 3번째 값에 곱하기를 하게 되는 수이며 이 값과 위의 맵핑값이 곱하여져서 실제 생성되는 큐 방의 크기가 된다. 즉, 아래 예시에서는 $100 * 10 = 1000$ 이되어 1000개의 큐방이 만들어 진다.

10000개의 방을 만들고자 한다면 $100 * 100$ 의 값을 넣거나 $10 * 1000$ 을 넣으면 된다.

아래 예시는 총 5개의 큐를 만드는 예시이다.

5번째 칼럼은 메시지 최대 길이 이다.

만약, 예시값의 8192 바이트 보다 더 큰 메시지를 처리하고자 한다면 이 값을 2의 배수값으로 변경하면 된다.

```
[ums@Lenovo-P1 enmq]$ cat samsung.list
# 1-th column: path
# 2-th column: qname
# 3-th      : mapping_num
# 4-th      : multi_num
# 5-th      : msglen
#
/tmp/java_package/enmq|LG_SM_ONL|100|10|8192
/tmp/java_package/enmq|KT_SM_ONL|100|10|8192
/tmp/java_package/enmq|SK_SM_ONL|100|10|8192
/tmp/java_package/enmq|IB_SM_ONL|100|10|8192
/tmp/java_package/enmq|RESULT_ONL|100|10|8192
[ums@Lenovo-P1 enmq]$
```

수정이 끝났으면 다음과 같은 명령을 수행한다.

```
$ fqm -f samsung.list FQ_CREATE <ENTER>
```

```
[ums@Lenovo-P1 enmq]$ fqm -f samsung.list FQ_CREATE
count = [5]

[0]-th queue: [/tmp/java_package/enmq] [LG_SM_ONL] [100] [10] [8192]
  - info file('LG_SM_ONL.Q.info' creation successful!
  - Successfully added a new queue('LG_SM_ONL') to the list file(ListFQ.info)!
CMD:[0] path= [/tmp/java_package/enmq] qname=[LG_SM_ONL] ...
  - Check if header file already does exist .....[OK ]
  - create header dummy file.....[OK ]
  - header mmapping.....[OK ]
  - initialize header file.....[OK ]
  - change mode body file.....[OK ]
  - close all files safely.....[OK ]
  - Finalize.....[OK ]
  - Check if data file already does exist .....[OK ]
  - create dummy data file.....[OK ]
result: rc=[0]
  - Successfully created a new queue('/tmp/java_package/enmq', 'LG_SM_ONL').

[1]-th queue: [/tmp/java_package/enmq] [KT_SM_ONL] [100] [10] [8192]
  - info file('KT_SM_ONL.Q.info' creation successful!
  - Successfully added a new queue('KT_SM_ONL') to the list file(ListFQ.info)!
CMD:[0] path= [/tmp/java_package/enmq] qname=[KT_SM_ONL] ...
  - Check if header file already does exist .....[OK ]
  - create header dummy file.....[OK ]
  - header mmapping.....[OK ]
  - initialize header file.....[OK ]
  - change mode body file.....[OK ]
  - close all files safely.....[OK ]
  - Finalize.....[OK ]
  - Check if data file already does exist .....[OK ]
  - create dummy data file.....[OK ]
result: rc=[0]
  - Successfully created a new queue('/tmp/java_package/enmq', 'KT_SM_ONL').
```

위 화면은 생성이 성공적으로 이루어질 때의 화면이다.


```
[ums@Lenovo-P1 enmq]$ ls -rlt
total 60
-rw-r--r-- 1 ums ums      28 Apr 3 21:23 trial
-rwxr-xr-x 1 ums ums      60 Apr 3 21:24 recreate.sh
-rw-r--r-- 1 ums ums     343 Apr 3 21:25 samsung.list
-rw-r--r-- 1 ums ums       0 Apr 3 21:52 FQ_external_env.conf
-rw-r--r-- 1 ums ums      23 Apr 3 21:53 FQ_directories.list
-rw-r--r-- 1 ums ums       0 Apr 3 21:54 ListFQ.info,temp__
-rw-r--r-- 1 ums ums     217 Apr 3 21:54 LG_SM_ONL.Q.info__
-rw-rw-rw- 1 ums ums     1032 Apr 3 21:54 LG_SM_ONL.h
-rw-rw-rw- 1 ums ums    8192000 Apr 3 21:54 LG_SM_ONL.r
-rw-r--r-- 1 ums ums     217 Apr 3 21:54 KT_SM_ONL.Q.info
-rw-rw-rw- 1 ums ums     1032 Apr 3 21:54 KT_SM_ONL.h
-rw-rw-rw- 1 ums ums    8192000 Apr 3 21:54 KT_SM_ONL.r
-rw-r--r-- 1 ums ums     217 Apr 3 21:54 SK_SM_ONL.Q.info
-rw-rw-rw- 1 ums ums     1032 Apr 3 21:54 SK_SM_ONL.h
-rw-rw-rw- 1 ums ums    8192000 Apr 3 21:54 SK_SM_ONL.r
-rw-r--r-- 1 ums ums     217 Apr 3 21:54 IB_SM_ONL.Q.info
-rw-rw-rw- 1 ums ums     1032 Apr 3 21:54 IB_SM_ONL.h
-rw-rw-rw- 1 ums ums    8192000 Apr 3 21:54 IB_SM_ONL.r
-rw-r--r-- 1 ums ums     219 Apr 3 21:54 RESULT_ONL.Q.info
-rw-r--r-- 1 ums ums      51 Apr 3 21:54 ListFQ.info
-rw-rw-rw- 1 ums ums     1032 Apr 3 21:54 RESULT_ONL.h
-rw-rw-rw- 1 ums ums    8192000 Apr 3 21:54 RESULT_ONL.r
-rw-r--r-- 1 ums ums       0 Apr 3 21:54 SK_SM_ONL.backup
-rw-r--r-- 1 ums ums       0 Apr 3 21:54 RESULT_ONL.backup
-rw-r--r-- 1 ums ums       0 Apr 3 21:54 LG_SM_ONL.backup
-rw-r--r-- 1 ums ums       0 Apr 3 21:54 KT_SM_ONL.backup
-rw-r--r-- 1 ums ums       0 Apr 3 21:54 IB_SM_ONL.backup
[ums@Lenovo-P1 enmq]$
```

생성된 후에는 위와 같이 여러 파일들이 존재하게 된다.

다음은 개별로 큐를 생성하는 절차이다.

큐 생성순서는 다음과 같다.

- 첫번째: fqm 유틸리티를 이용하여 기본 info 파일 생성
- 두번째: 편집기(vi)를 이용하여 생성된 info 파일 수정
- 세번째: fqm 유틸리티를 이용하여 실제 큐 생성

첫번째 info 파일생성 명령은 아래와 같다.

```
$ fqm -q {큐생성 디렉토리명} {큐명} FQ_GEN_INFO <ENTER>
```

예시) 디렉토리(라이선스 파일이 복사된 위치)에 TST이라는 큐명을 생성하고자 할 때 실제 명령을 다음과 같다.

```
$ fqm -q /tmp/java_package/enmq TST FQ_GEN_INFO <ENTER>
```

```
[ums@Lenovo-P1 enmq]$ fqm -q /tmp/java_package/enmq TST FQ_GEN_INFO  
ch=[q] optarg=[(null)] optind=[2]  
cmd = [17]  
Success generated 'TST.Q.info and added qname in 'ListFQ.info'!
```

성공시 디렉토리(/enmq)에는 TST.Q.info 라는 이름이 생성되어 진다.

두번째 절차는 이 파일을 편집하는 것이다.

```
QNAME = "TST"  
QPATH = "/home/javauser/java_package/enmq" -> 필요시 변경필요  
MSGLEN = "4096"  
MMAPPING_NUM = "10"  
MULTI_NUM = "100"  
DESC = "temporary"  
XA_MODE_ON_OFF = "ON"  
WAIT_MODE_ON_OFF = "OFF"  
MASTER_USE_FLAG = "0"  
MASTER_HOSTNAME = "0"
```

만들어진 파일의 내용은 위와 같다.

만약, 큐 메시지 크기를 8192로 변경하고자 하면 4096을 8192로 변경한다. 생성되는 큐의 레코드량은 MMAPPING_NUM 와 MULTI_NUM 의 곱의 값이 된다. 즉 위에서는 1000개의 레코드가 만들어 진다. 만약 10만개의 레코드를 생성코자 한다면 10 -> 1000으로 변경하면 된다.

그리고 DESC (Description: 설명) 을 원하는 문자열로 변경한다.

나머지 항목을 특별한 경우가 아니면 그대로 둔다.

세번째, info파일 편집을 마쳤으면 실제 큐를 생성한다. 명령은 아래와 같다.

```
$ fqm -q /tmp/java_package/enmq TST1 FQ_CREATE <ENTER>
```

```
[ums@Lenovo-P1 enmq]$ fqm -q /tmp/java_package/enmq TST FQ_CREATE
ch=[q] optarg=[(null)] optind=[2]
cmd = [0]
CMD:[0] path=[/tmp/java_package/enmq] qname=[TST] ...
- Check if header file already does exist .....[OK ]
- create header dummy file.....[OK ]
- header mmappping.....[OK ]
- initialize header file.....[OK ]
- change mode body file.....[OK ]
- close all files safely.....[OK ]
- Finalize.....[OK ]
- Check if data file already does exist .....[OK ]
- create dummy data file.....[OK ]
result: rc=[0]
success.
[ums@Lenovo-P1 enmq]$
```

성공적으로 큐가 생성되면 위와 같은 메시지가 모두 OK로 출력된다. 만약 이중에 NOK가 포함 되어 있다면 실패한 것이고 이때는 /tmp/fqm.log 파일은 확인하여 실패 원인을 추적할 수 있다.

큐가 정상적으로 생성되면 디렉토리에 아래와 같은 파일이 생성된다.

```
-rw-rw-rw- 1 javauser javauser 1032 May  3 18:33 TST.h
-rw-rw-rw- 1 javauser javauser 4096000 May  3 18: TST.r
```

4. 접근확인

정상적으로 생성이 되었을 경우, 모니터링 프로그램을 통해 큐 파일의 상태를 확인할 수 있다.

모니터링을 위한 프로그램은 패키지 설치 위치의 /bin 에 위치한다.

```
[ums@Lenovo-P1 bin]$ ls -l
total 1056
-rwxr-xr-x 1 ums ums 69688 Apr  3 21:17 TestFQ
-rwxr-xr-x 1 ums ums 53528 Apr  3 21:17 TestSQ
-rwxr-xr-x 1 ums ums 61872 Apr  3 21:17 fqm
-rwxr-xr-x 1 ums ums 87 Apr  3 22:02 mon.sh
-rwxr-xr-x 1 ums ums 113640 Apr  3 21:17 monitor
-rw-r--r-- 1 ums ums 72040 Apr  3 21:17 monitor.log
-rwxr-xr-x 1 ums ums 425904 Apr  3 21:17 nmon
-rwxr-xr-x 1 ums ums 52136 Apr  3 21:17 qvf
-rwxr-xr-x 1 ums ums 110120 Apr  3 21:17 shm_monitor
-rwxr-xr-x 1 ums ums 102304 Apr  3 21:17 wmonitor
[ums@Lenovo-P1 bin]$
```

모니터링 유틸은 2가지 이며, monitor 와 qvf 이다.

먼저 monitor는 시스템에 설치되어 있는 모든 큐를 실시간 모니터링하는 프로그램이다.

인자 없이 셸에서

```
$ monitor <ENTER>
```

를 치면 수행된다. 정상적으로 수행되면 아래와 같이 나타난다. 종료시 'q'를 치면 된다.


```
1 oracle linux 9
2024/04/03 22:13:30 687370 -refresh interval(1), rows(70) cols(146), Press key: Fefresh:<F1> quit<F4>
- Hostname: Lenovo-P1, - IP address: 0.0.0.0 - Number of Queues: 6
FQ_DATA_HOME: /tmp/java_package/enmq, Allocated Disk: 45056000
Available Disk Space: 249191391 blocks.(1 block=4096)
[ M&Wise UMS Distributor FileQueue Status. Library version 14.8.0 ]
=====
NO      QNAME      MAX_RECO  GAP      Usage    enQ_lock  deQ_lock  BIG      TPS(en)  TPS(de)  en_sum    de_sum    Q_path
0(0)    LG_SM_ONL   1000      0         0.00     1          0          0          0          0          0          0          /tmp/java_package/enmq
1(1)    KT_SM_ONL   1000      0         0.00     1          0          0          0          0          0          0          /tmp/java_package/enmq
2(2)    SK_SM_ONL   1000      0         0.00     1          0          0          0          0          0          0          /tmp/java_package/enmq
3(3)    IB_SM_ONL   1000      0         0.00     1          0          0          0          0          0          0          /tmp/java_package/enmq
4(4)    RESULT_ONL  1000      0         0.00     1          1          0          0          0          0          0          /tmp/java_package/enmq
5(5)    TST         1000      0         0.00     2          0          0          0          0          0          0          /tmp/java_package/enmq
=====
```

단, monitor 프로그램은 ncurses 동적 라이브러리를 사용한다. 따라서, 이 라이브러리가 서버에 설치되지 않은 경우에는 정상적으로 동작하지 않는다. ncurses라이브러리를 설치할 수 없는 경우에는 아래에서 설명하는 qvf 유틸리티를 이용하면 된다.

만약 ncurses를 시스템에 설치하고자 하면 다음 명령을 이용한다.


Ubuntu 또는 Debian 기반 시스템에서는 다음과 같이 설치할 수 있습니다:

arduino

 Copy code


```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

Red Hat 또는 CentOS 기반 시스템에서는 다음과 같이 설치할 수 있습니다:

 Copy code


```
sudo yum install ncurses-devel
```

Fedora 기반 시스템에서는 다음과 같이 설치할 수 있습니다:

 Copy code

```
sudo dnf install ncurses-devel
```

macOS에서는 **ncurses**를 설치하기 위해 Homebrew를 사용할 수 있습니다:

 Copy code

```
brew install ncurses
```

참고) 위 화면에서 IP address가 정확하게 나타나게 하는 방법

큐가 생성된 위치에(~/enmq) FQ_external_env.conf 파일이 존재하며 이 파일에 Ethernet card 의 정확한 명칭을 XML 형식으로 작성한 후 monitor를 다시 실행하면 실제 IP가 정확하게 표기됨.

Ifconfig 명령 또는 ip addr show 명령을 이용하면 현재 서버의 Ethernet card 에 할당된 IP 리스트가 아래와 같이 모두 나타나며 여기서 한 개의 Ethernet 명칭을 FQ_external_env.conf 파일에 편집한다.

```
$ ip addr show <enter>
2: enp11s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 90:2b:34:58:8d:46 brd ff:ff:ff:ff:ff:ff
    inet 172.30.9.20/24 brd 172.30.9.255 scope global noprefixroute dynamic enp11s0
```

```

        valid_lft 2865sec preferred_lft 2865sec
        inet6 fe80::76b5:bc17:64a3:9a03/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: eno1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 90:2b:34:58:8d:44 brd ff:ff:ff:ff:ff:ff
    inet 172.30.9.56/24 brd 172.30.9.255 scope global noprefixroute dynamic eno1
        valid_lft 2745sec preferred_lft 2745sec
        inet6 fe80::8ef0:e00c:e221:fbcd/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
4: wlp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default qlen 1000
    link/ether 74:e5:43:b8:0b:ad brd ff:ff:ff:ff:ff:ff
    inet 172.30.9.30/24 brd 172.30.9.255 scope global noprefixroute dynamic wlp8s0
        valid_lft 2666sec preferred_lft 2666sec
        inet6 fe80::2751:bc04:536e:efb/64 scope link noprefixroute

```

아래는 FQ_external_env.conf 파일 내용

```
<eth>enp11s0</eth>
```

예시이다.

모니터링을 하는 또 하나의 유틸리티 프로그램은 qvf 명령이다. 이 프로그램은 하나의 큐에 관한 상세내역을 실시간 출력하는 프로그램이다. 이 프로그램은 curses 라이브러리를 사요하지 않기 때문에 별도의 설치 없이도 텍스트 기반으로 동작한다.

이 프로그램의 실행 편의를 위해 mon.sh 이 제공된다.

mon.sh 을 구동하게 되면 TST 라는 큐의 상세내역이 아래 화면과 같이 1초간격으로 refresh 되면서 화면에 나타난다.(큐설치 위치에 따라 vi 편집 필요)

```
javauser@DESKTOP-G7IIL9O: ~/fq_14.3.6_java_package/bin
The path of queue file      :/home/javauser/fq_14.3.6_java_package/enmq
The queue Name              :TST
File size                   :4096000 (bytes)
The created queue spaces    :1000
The number of incoming msg(en-queue) :0 (0 TPS)
The number of outgoing msg(de-queue) :0 (0 TPS)
A gap between en and de     :0 ( The remaining space =[1000] )
Untreated big messages's count :0
The buffer size for a message(no-big) :4096 (bytes)
last en queue time          :20230503 183345
last de queue time          :20230503 183345
latest en queue time (micro sec) :20230503 183345
latest de queue time (micro sec) :20230503 183345
mapping length(area size) bytes :40960
en: from_offset-to_offset    :0-0
de: from_offset-to_offset    :0-0
Current BIG files            :0
Master-Assign-Flag          :0
Master-Assign-Host          :0
```

종단을 위해서는 Control + 'C'로 중단한다.

모니터링에 생성된 큐의 상황이 잘 표시된다면 큐 생성도 완전하게 된 것이면 접근도 가능한 상태가 확인된 것이다. 만약, 다른 큐의 내용을 보려면 이 파일을 큐명을 인자로 받도록 수정하여

```
$ mon.sh [큐명] <enter>
```

형식으로 수행하도록 수정하면 된다.

여기 까지가 환경설정/큐생성/접근확인 과정이다.

요약하면 패키지를 디렉토리에 푼 후, 내 환경에 맞게 편집 해야 할 목록은 아래와 같다.

```
~/mnwise_package/env.sh
-> JAVA_HOME 편집
-> LD_LIBRARY_PATH
-> FQ_DATA_HOME
-> FQ_INSTALL_HOME
-> PATH
```

~/enmq/

->FQ_directories.list : 압축을 푼 디렉토리로 변경

➔ 다수개의 큐를 만드는 방법 또는 한 개의 큐를 만드는 방법을 통해 큐 생성

monitor 또는 qvf 명령을 통해 생성된 큐 접근확인.

여기까지 준비가 되었으면 설치는 잘 이루어진 것이다.

예제 개발과정을 살펴본다.

5. 자바 개발 및 테스트

자바언어를 이용하여 큐에 접근하기 위한 모든 예제들은 설치 디렉토리의 examples 폴더 밑에 존재한다.

여기서는 TST라는 큐가 사전에 생성되어 있다는 전제조건하에서 이 큐에 데이터를 넣을 때 사용되는 en-queue 예제와 큐에서 데이터를 꺼낼 때 사용되는 de-queue 과정을 제공하는 예제 소스를 참고로 설명한다.

설치 디렉토리 examples에는 미들웨어 사용을 위한 여러 예제 소스들이 포함되어 있으나, 실제 java 어플리케이션 개발자가 사용하게 될 en-queue/de-queue 소스만을 설명한다.

- en-queue 예제 소스 : TestEnQ.java
- de-queue 예제 소스: TestDeQ.java / TestDeQXA.java
- 멀티쓰레드 예제 소스: MultiThread_DeQ_Demo.java / Multithread_example.java

먼저, 큐에 데이터를 꺼내는 과정인 de-queue 예제 소스는 아래와 같다.

개발하려는 소스에 필수로 포함되어야 할 주요 부분은 적색으로 표기되어 있다.

```
1 import com.clang.fq.*;
2
3 /*
4 ** Warning: max buffer size is 65536
5 */
6 public class TestDeQXA {
7     static {
8         System.loadLibrary("jfq"); // Load native library at runtime
9                                     // hello.dll (Windows) or libhello.so (Unixes)
10        System.loadLibrary("fq");
```

```

11     }
12     TestDeQXA () {
13
14     }
15
16     // Test Driver
17     public static void main(String[] args) {
18         int rc;
19
20         // make a FileQueueJNI instance with naming test.
21         // 3-th argument is loglevel. (0: trace, 1: debug, 2: info, 3: Warning, 4: error, 5: emerg,
6: request)
22         // Use 1 in dev and 4 real.
23         String Qpath = System.getenv("FQ_DATA_HOME");
24         FileQueueJNI test = new FileQueueJNI( 0, "/tmp/jni.log", 1, Qpath, "TST");
25
26         rc = test.open(); // 프로그램 구동시에만 1회 호출
27         if( rc < 0 ) {
28             System.out.println("open failed: " + test.logname + "," + test.path + "," +
test.qname + "," + " rc: " + rc);
29             return;
30         }
31         System.out.println("open success: " + test.logname + "," + test.path + "," +
test.qname + "," + " rc: " + rc);
32
33         for(;;) { // polling file queue.
34
35             rc = test.readXA(); // XA read 반복호출
36             if( rc < 0 ) {
37                 System.out.println("readXA failed: " + test.path + "," + test.qname + "," + "
rc: " + rc);
38                 break;
39             }
40             else if( rc == 0 ) {
41                 System.out.println("empty: " + test.path + "," + test.qname + "," + " rc: " +
rc);
42

```

```

43         try {
44             Thread.sleep(1000); // Pause for 1 second
45         }
46         catch(InterruptedException ex) {
47             Thread.currentThread().interrupt();
48         }
49         continue;
50     }
51     else {
52         String data = test.get_out_msg();
53         long out_seq = test.get_out_seq();
54         String out_unlink_filename = test.get_out_unlink_filename();
55         long out_run_time = test.get_out_run_time();
56
57         System.out.println("read success: " + test.path + "," + test.qname + "," +
" rc: " + rc + " msg: " + data + " seq: " + out_seq + " unlink_filename: " + out_unlink_filename +
" run_time(micro seconds): " + out_run_time);
58
59         // input your jobs in here
60         //
61         //
62
63         if( rc > 10) { // normal data
64             rc = test.commitXA();
65             System.out.println("commit success: rc: " + rc);
66         }
67         else { // abnormal data
68             rc = test.cancelXA();
69             System.out.println("cancel success: rc: " + rc);
70             break;
71         }
72         continue;
73     }
74 }
75 test.close(); // 프로그램 종료시에만 1회 호출
76 }
77 }

```

예제프로그램 개요)

본 예제는 큐에 들어오는 메시지를 기다리다 데이터가 들어오면 처리하고 다시 꺼내는 작업을 반복하는 데몬 서버프로그램의 예제 이다. 큐에서 데이터를 꺼내는 방법은 두 가지가 존재한다.

하나는 NON-XA 모드 이고 하나는 XA 모드 이다. 본 예제는 XA 모드의 예제이다.

XA모드의 꺼내기는 데이터 분실을 방지하는 모드이며, 데이터를 꺼내어도 commit을 수행하기 이전에는 데이터가 큐에서 삭제되지 않기 때문에 데이터를 꺼낸 후 처리 중 프로세스가 비정상 중 단되어도 데이터가 사라지지 않는다. NON-XA모드에서는 꺼낸 순간 데이터가 큐에서 사라지기 때문에 데이터의 분실을 어플리케이션 프로그램이 책임져야 한다.

주요 코드 설명)

6~11 라인: file queue 동적 라이브러리 로딩

23 String Qpath = System.getenv("FQ_DATA_HOME");

 환경설정값에서 큐위치를 읽어 온다.

24 FileQueueJNI test = new FileQueueJNI(0, "/tmp/jni.log", 1, Qpath, "TST");

FileQueue 객체생성:

인자값:

- 0- th: 0 멀티 쓰레드일 경우 스레드별 다른 인덱스를 0, 1, 2 .. 값으로 지정하고 싱글 일 경우에는 0을 사용하면 된다.
- 1- th: 로그파일 위치 및 이름.:
- 2- th: 로그레벨 (0: trace, 1: debug, 2: info, 3: Warning, 4: error, 5: emerg, 6: request) 6종 류이며 0:trace 레벨이 가장 상세한 내역이 로그에 저장된다. 통상적으로 개발중에는 trace 또는 debug를 사용하고 운영단계에서는 error를 지정한다.
- 3- 큐위치: 큐가 생성된 디렉토리명을 전달한다.
- 4- 큐이름: 큐이름을 전달한다.

5-

```
26         rc = test.open();
```

큐를 오픈한다 리턴값(rc)가 < 0 일 경우, 오류발생으로 중단한다.

```
35         rc = test.readXA(); // XA read
```

XA 모드로 데이터를 읽어 온다.

리턴은 0, <0, >0 3가지 형태이며, 0일경우 들어온 데이터가 없는 empty 를 의미하고,

0> 경우는 정상적으로 데이터를 꺼낸것이며 정상적인 업무를 수행하면 된다.

< 0 경우는 읽기 오류이며 만약 그값이 -99 라면, 관리자에 의한 큐사용 중지가 발생한것이기 때문에 1초간 sleep후에 다시 읽기를 수행반복하거나 중지하여야 한다.

이때의 동작은 운영설계에 따라 달라진다.

```
64         rc = test.commitXA()
```

데이터를 꺼낸 후 정상적인 처리가 이루어 졌을 경우에는 commitXA()를 호출하고 다음 읽기를 수행한다.

```
68         rc = test.cancelXA();
```

만약 업무처리가 실패하였다면 cancelXA()를 호출한다.

```
75         test.close();
```

서버를 종료하고자 할 때, 오픈된 큐를 닫는다.

큐에 데이터를 넣는 과정인 en-queue 예제 소스는 아래와 같다.

```
1 import com.clang.fq.*;
2
3 public class TestEnQ {
```

```

4    static {
5        System.loadLibrary("jfq"); // Load native library at runtime
6                                     // hello.dll (Windows) or libhello.so (Unixes)
7        System.loadLibrary("fq");
8    }
9    TestEnQ () {
10
11    }
12
13    // Test Driver
14    public static void main(String[] args) {
15        int rc;
16
17        // make a FileQueueJNI instance with naming test.
18        // 3-th argument is loglevel. (0: trace, 1: debug, 2: info, 3: Warning, 4: error, 5: emerg,
6: request)
19        // Use 1 in dev and 4 real.
20
21        String Qpath = System.getenv("FQ_DATA_HOME");
22        FileQueueJNI test = new FileQueueJNI( 0, "/tmp/jni.log", 1, Qpath, "TST");
23
24        rc = test.open();
25        if( rc < 0 ) {
26            System.out.println("open failed: " + test.logname + "," + test.path + "," +
test.qname + "," + " rc: " + rc);
27            return;
28        }
29        System.out.println("open success: " + test.logname + "," + test.path + "," +
test.qname + "," + " rc: " + rc);
30
31        float usage;
32        usage = test.usage();
33        System.out.println("Current Usage: " + usage);
34
35
36        // String kind_media = "LM"; /* kind_media  kakao:KA */

```

```

37      String kind_media = "SM"; /* kind_media  kakao:KA */
38      // String kind_media = "PU"; /* kind_media  Push:PU */
39      // String kind_media = "EM"; /* kind_media  Email:EM */
40      // String kind_media = "SM"; /* kind_media  SMS:Short Message */
41      String phone_no = "01072021516";
42
43      String      send_msg      =
"012345678901234567890123456789012345678901234567890123456789012345678
9012345678901234567890123456789012345678901234567";
44      String      template      = "Hello  %var1%  !  my  name
is %var2%_____";
45
46      String var_data = "Choi|Gwisang";
47
48      String SendData = kind_media+phone_no+send_msg+template+var_data;
49
50      System.out.println( "Send Data: " + SendData );
// 중요 : Write과정은 성공할때까지 재시도 하는 것이 보통이다.
// 따라서, 아래 문장을 while(true) 로 감싸고
// 성공시에는 break 로 빠져나오로,
// Full 시에는 1초 sleep 후 continue로 재시도 하기를 권장한다.
51      rc = test.write( SendData );
52
53      // rc = test.write("01072021516This is a send
msg.....");
54
55      if( rc < 0 ) {
56          System.out.println("Write failed: " + test.path + "," + test.qname + "," + " rc: "
+ rc);
57          test.close();
58          return;
59      }
60      else if( rc == 0 ) {
61          System.out.println("full: " + test.path + "," + test.qname + "," + " rc: " + rc);
62          test.close();
63          return;
64      }
65      else {
66          long out_seq = test.get_out_seq();
67          long out_run_time = test.get_out_run_time();

```

```

64
65         System.out.println("Write success: " + test.path + "," + test.qname + "," + " rc:
" + rc + "," + " seq: " + out_seq + " run_time(micro seconds): " + out_run_time);
66         test.close(); // 반복적으로 enqueue 할 경우 이부분의 루프 밖으로 뺀다.
67         return;
68     }
69 }
70 }

```

예제 프로그램 개요)

위 프로그램은 1건의 메시지를 큐에 넣는 과정을 설명한 소스이다.

만약, 반복적으로 메시지를 넣는다면 매번 큐를 오픈 할 필요 없이 한번만 오픈하고, write() 만을 반복하면 된다.

소스설명)

1~30 라이까지는 de-queue과정과 동일하다.

```

31         float usage;
32         usage = test.usage();
33         System.out.println("Current Usage: " + usage);

```

현재 큐의 사용량을 가져온다. 리턴값은 33.5 와 같이 float 값이다.

현재 사용량에 따라 동작을 달리 할 필요가 있는 경우 이 함수 호출을 이용하면 된다.

41~46 enqueue 하고자 하는 message 를 String으로 만든다.

```

49         rc = test.write( SendData );

```

메시지를 enqueue한다.

리턴값은 0, 0>, <0 세가지 형태이다.

0일경우는 큐가 full되어 있는 경우이다. 이 때는 정책에 따라 일정 시간을 sleep후 다시 49 write 를 호출하거나 중단하면 된다.

0> 경우에는 정상적으로 en-queue 된 것이다.

< 0 : 경우에는 오류가 발생한 것이며, 중단하고 로그파일의 내용을 확인하고 조치하여야 한다.

참고) 실제 package에 동봉된 소스와 본 페이지의 내용은 약간 다를 수 있습니다.

빌드과정

환경셋팅이 정상적으로 되었을 경우 패키지/examples 디렉토리 밑에 존재하는 build.sh 을 수행하면 동봉된 모든 예제들이 재 빌드 된다.

만약, 개발자가 새로운 my_exercise.java 를 생성하였을 경우.

```
$ javac -classpath ./FileQueueJNI.jar my_exercise.java <ENTER>
```

명령을 수행하여 빌드한다.

프로그램 구동

패키지에 제공된 en-queue(TestEnQ.java) / de-queue(TestDeQXA.java) 예제프로그램으로 동작을 확인 하는 과정은 다음과 같다.

먼저, 터미널 3개를 연다.

1번 터미널에서 위에서 설명한 monitor 또는 mon.sh 을 수행한다.

2번 터미널에서

```
$ java TestDeQXA <ENTER>
```

를 구동하면 데이터가 없음을 알리는 로그가 화면에 1초단위로 스크롤 된다.

3번 터미널에서

```
$ java TestEnQ<ENTER>
```

를 구동하면 1건의 데이터가 큐에 들어가고 2번 터미널에서 이 메시지를 꺼내어 화면에 꺼낸 데이터를 출력할 것이다.

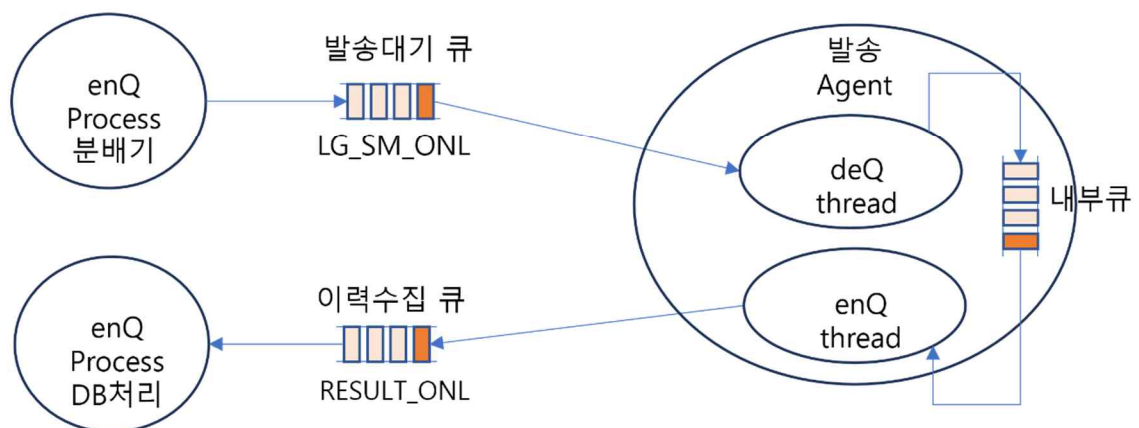
이 때, 1번 터미널에서의 변화를 보면 순간적으로 변화가 보일 것이다.

실제 Agent 개발에 참고할 소스

실제 에이전트 개발에 참고 할 수 있도록 /src 와 /utl 두개의 디렉토리에 참고할 소스가 포함되어 있습니다.

참고 소스의 프로세스 동작 구조는 다음과 같다

프로세스 구조



/utl/enQ 디렉토리에는 분배기가 발송큐에 넣어주는 것처럼 JSON형식으로 큐에 넣는 프로그램이 포함되어 있고,

/util/deQ 디렉토리에는 이력처리가 결과큐에서 꺼내는 프로그램 처럼 동작하는 기능이 포함되어 있다.

그리고 /src/에는 Agent처럼 동작하는 가상 Agent의 JAVA 프로그램이니 마련되어 있다.

Agent개발자는 이 소스를 참고하여 개발하면 도움이 될 것이다.

시뮬레이션 하기 위해서는 터미널 4개를 열고

1번터미널에서 monitor를 구동시켜 놓고,

2번터미널에서 deQ run.sh 의 디렉토리 를 수정 후 구동하고,

3번터미널에서 /src/ 밑에서 xml 파일을 편집후 run.sh 로 구동하고,

4번터미널에서 enQ run.sh을 편집 후 구동하면

메지지의 흐름이 모니터를 통해 보이게 된다.

패키지 공급 및 소스 문의

연락처:

전화: 010-7202-1516 최귀상

이메일: gwisang.choi@gmail.com

감사합니다.

