JEUS Scheduler 안내서

Document info

Document name: JEUS Scheduler 안내서

Document date: 2005-06-06

Manual release version: 3

Software Version: JEUS 5

차 례

그림 목차

매뉴얼에 대해서

매뉴얼의 대상

본 문서는 JEUS Scheduler 기능을 사용하고자 하는 관리자나 프로그래머를대상으로 한다.

매뉴얼의 전제 조건

본 문서를 읽기 전에 JEUS 소개나 JEUS 설치 안내서를 읽어보길 권한다. 그리고 JEUS Getting Started Tutorial 도 읽어보면 도움이 된다.

그리고, 원격 Scheduler 를 사용하기 위해 RMI 와 JNDI 에 대한 지식이 필요하다. 이러한 배경 지식에 대해서는 자바 관련 문서를 읽어보기를 권한다.

기본적으로 본 문서에서는 J2EE 와 Java 스펙에 대한 것은 설명하지 않고, JEUS 관련 내용만 설명한다.

관련 매뉴얼

본 문서를 읽은 후에는 다음 문서를 읽어보기를 권한다.

- JEUS Server 안내서
- JEUS 웹 관리자 안내서
- JEUS Client Application 안내서

일러두기

표기 예	내용

표기 예	내용
텍스트	본문, 12 포인트, 바탕체 Times New Roman
텍스트	본문 강조
CTRL+C	CTRL 과 동시에 C 를 누름
`public class myClass { }`	Java 코드
`<system-config>`	XML 문서
참조: / 주의:	참조 사항과 주의할 사항
Configuration 메뉴를 연다	GUI 의 버튼 같은 컴포넌트
JEUS_HOME	JEUS 가 실제로 설치된 디렉토리 예)c:\jeus50
`jeusadmin nodename`	콘솔 명령어와 문법
[파라미터]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨. 예)<node name>은 실제 hostname 으로 변경해서 사용
|	선택 사항. 예) A|B: A 나 B 중 하나
…	파라미터 등이 반복되어서 나옴
?, +, *	보통 XML 문서에 각각 "없거나, 한 번", "한 번 이상", "없거나, 여러 번"을 나타낸다.

표기 예	내용
. . .	XML 이나 코드 등의 생략
<<*FileName.ext*>>	코드의 파일명
1.1 그림 1.	그림 이름이나 표 이름

OS 에 대해서

본 문서에서는 모든 예제와 환경 설정을 Microsoft Windows™의 스타일을 따랐다. 유닉스같이 다른 환경에서 작업하는 사람은 몇 가지 사항만 고려하면 별무리 없이 사용할 수 있다. 대표적인 것이 디렉토리의 구분자인데, Windows 스타일인 "\"를 유닉스 스타일인 "/"로 바꿔서 사용하면 무리가 없다. 이외에 환경 변수도 유닉스 스타일로 변경해서 사용하면 된다.

그러나 Java 표준을 고려해서 문서를 작성했기 때문에, 대부분의 내용은 동일하게 적용된다.

용어 설명

다음에 소개되는 용어는 본 문서 전체에 걸쳐서 사용되는 용어이다. 용어가 이해하기 어렵거나 명확하지 않을 때는 아래 정의를 참조하기 바란다.

용어	정의
Fixed-delay	작업이 고정된 간격으로 수행되는 것을 말한다.
Fixed-rate	작업이 고정된 비율로 수행되는 것을 말한다.
JEUS 노드 Scheduler Service	JEUS 노드상에서 구동되는 JEUS Scheduler 로 원격 사용자에게 서비스된다.

용어	정의
Stand-alone Scheduler	JEUS 의 Scheduler Service 형태가 아닌 어플리케이션 내에서 라이브러리처럼 동작하는 Scheduler

연락처

Korea
Tmax Soft Co., Ltd
18F Glass Tower, 946-1, Daechi-Dong, Kangnam-Gu
Seoul 135-708
South Korea
Email: info@tmax.co.kr
Web (Korean):http://www.tmax.co.kr

USA
Tmax Soft, Co.Ltd.
2550 North First Street, Suite 110
San Jose, CA 95131
USA
Email: info@tmaxsoft.com
Web (English): http://www.tmaxsoft.com

Japan
Tmax Soft Japan Co., Ltd.
6-7 Sanbancho, Chiyoda-ku,
Tokyo 102-0075
Japan
Email: info@tmaxsoft.co.jp
Web (Japanese): http://www.tmaxsoft.co.jp

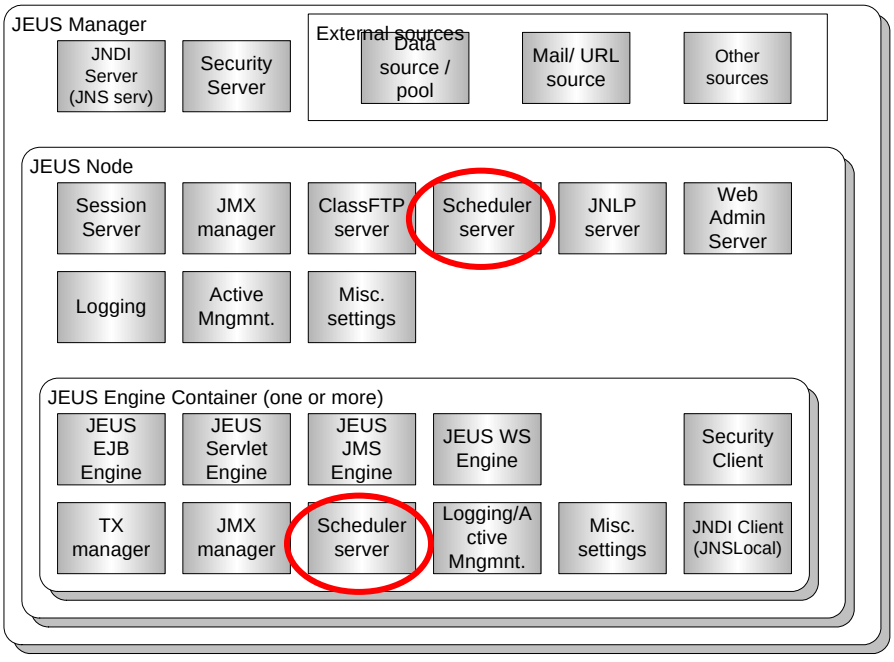
China
Beijing Silver Tower, RM 1507, 2# North Rd Dong San Huan,
Chaoyang District, Beijing, China, 100027
Tel: 86-10-64106148 Fax: 86-10-64106144
E-mail : info@tmaxchina.com.cn
Web (Chinese): http://www.tmaxchina.com.cn

1 소개

JEUS Scheduler 는 JEUS 의 확장된 기능으로 정해진 시간에 혹은 주기적으로 작업을 수행해야 할 때 사용한다.

Scheduler 는 다양한 용도로 사용이 가능하다. 자동적으로 temporary file 를 삭제한다거나, DB connection check 등 system 유지에 관련된 업무를 예로 들어 설명할 수 있다. 다른 예로서는 business 업무와 관련된 것으로 복권의 자동 출력이나, 확정된 상품 품목을 E-mail 로 자동적으로 보내게 할 수 있다.

이 문서에서는 JEUS Scheduler 의 사용방법에 대해서 설명한다.



1.1 그림 1. JEUS 에서의 Scheduler 컴포넌트.

2 JEUS Scheduler 개요

2.1 개요

JEUS Scheduler 는 정해진 시간에 수행되거나 반복적으로 수행되는 작업을 스케쥴링할 수 있는 기능을 제공한다.

J2EE 환경에서 Timer Service 를 이용하고자 할 때 J2SE Timer(java.util.Timer)를 직접 쓸 수 없으므로 EJB Timer Service 를 쓰거나 JEUS 에서 제공하는 JEUS Scheduler 를 사용해야 한다.

EJB Timer Service 는 EJB 환경에서만 사용할 수 있는데 반하여, JEUS Scheduler 는 모든 J2EE 환경에서 사용할 수 있고 일반 J2SE 애플리케이션 에서도 사용이 가능하다.

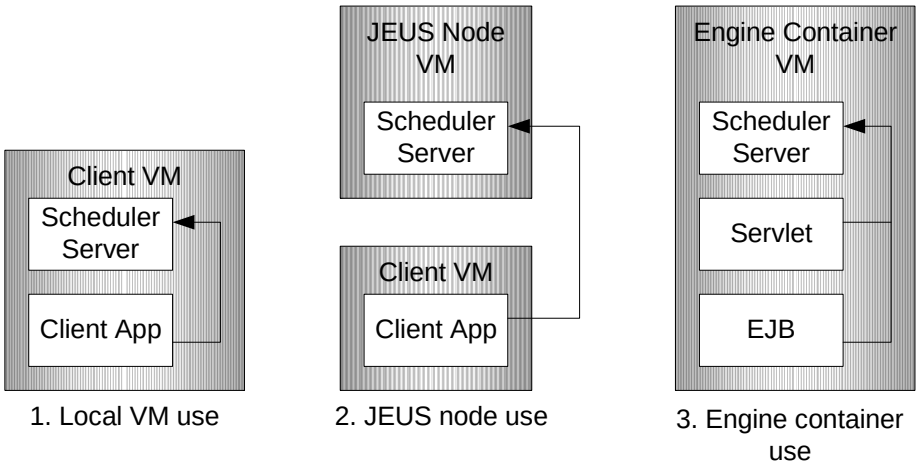
JEUS Scheduler 는 J2SE Timer(java.util.Timer)와 유사하기 때문에 J2SE Timer 에 익숙하다면 JEUS Scheduler 를 쉽게 사용할 수 있다. 또한 J2SE Timer 에는 없는 작업 종료 시점(end time)과 최대 수행 횟수(maxcount)를 지정할 수 있는 기능도 제공한다.

JEUS Scheduler 는 사용자 애플리케이션 내에서 사용할 수도 있고 JEUS Server 에서 사용될 수도 있다. JEUS Server 는 원격에서 접속할 수 있는 Scheduler Service 를 구동시킬 수 있으며 설정 파일을 통해 스케쥴링 될 작업을 등록할 수도 있다.

JEUS Scheduler 는 다음과 같이 다양한 환경에서 사용이 가능하다.

- J2SE 애플리케이션에서 Stand-alone Scheduler 사용
- J2EE 클라이언트 애플리케이션에서 Stand-alone Scheduler 사용
- JEUS 노드 Scheduler Service 를 원격에서 접속하여 사용
- JEUS 노드 설정파일에 Job 을 등록해서 사용
- JEUS Engine Container Scheduler Service 를 J2EE 컴포넌트에서 사용

- JEUS Engine Container 설정파일에 Job 을 등록해서 사용



2.2 그림 2. Scheduler Server 의 수행 방식들

각각의 Scheduler Service 를 사용하는 경우를 살펴보자.

클라이언트 어플리케이션에서 Stand-alone 방식으로 구동되는 JEUS Scheduler Service 는 클라이언트 어플리케이션 내부에서 어떠한 작업을 주기적으로 수행하고 싶을 때 주로 사용한다. 또한 클라이언트 어플리케이션이 종료되면 더 이상 수행될 필요가 없는 작업인 경우에 사용한다.

JEUS 노드 Scheduler Service 는 클라이언트 어플리케이션에 관계없이 항상 주기적으로 수행되어야 할 작업을 스케쥴링하고자 할 때 주로 사용한다. 서버에서 항상 수행되어야 할 주기적인 작업이 있다면 Job-list 에 미리 등록하여 서버 구동시에 수행되도록 할 수도 있다.

JEUS Engine Container Scheduler Service 는 주로 J2EE 컴포넌트들(Servlet/JSP, EJB 등)이 주기적으로 수행되는 작업을 스케쥴링하고자 할 때 주로 사용한다.

2.3 결론

지금까지 JEUS Scheduler 의 기본적인 내용에 대해서 알아보았다. 다음 장에서부터는 JEUS Scheduler 프로그래밍 하는 방법과 설정하는 방법에 대해서 알아보기로 한다.

3 JEUS Scheduler 프로그래밍

3.1 소개

이번 장에서는 JEUS Scheduler 프로그래밍에 필요한 기본지식과 사용 방법에 대해서 다룬다.

먼저 JEUS Scheduler 를 구성하고 있는 클래스들을 살펴보고 스케쥴링 작업을 정의하는 방법, 작업을 등록하는 방법, 등록된 작업을 제어하는 방법에 대해 설명한다. 이후에 JEUS Scheduler 를 어떤 경우에 사용할 수 있는 지를 보여준다.

JEUS 5.0 에서는 이전에 사용되던 방식에 비해 여러가지가 변경되거나 추가되었다. 따라서 기존 JEUS Scheduler 를 사용하던 프로그래머는 다음 사항을 고려해야 한다.

- 작업을 정의하기 위한 인터페이스로 ScheduleListener 인터페이스가 추가되었다. ScheduleListener 인터페이스는 최상위 작업 인터페이스로 모든 작업은 이 인터페이스를 구현(implements) 해야 한다. 기존에 존재하던 Schedule 작업 클래스도 이제는 ScheduleListener 를 구현하고 있다. 따라서 이제는 Schedule 작업 클래스를 상속해서 작업을 정의하지 않아도 ScheduleListener 를 구현해서 작업을 정의할 수 있게 되었다.

- 기존에 사용되던 Client 와 Server 용 SchedulerManager 는 더 이상 사용되지 않으며(deprecated) 새롭게 SchedulerFactory 클래스가 추가되었다. SchedulerFactory 는 Scheduler 객체를 얻는데 사용되며 Scheduler 객체를 통해 Client 환경, Server 환경, Remote Client 환경의 구별없이 작업을 등록할 수 있게 되었다.

- Scheduler 인터페이스에는 작업을 등록하는 다양한 메소드들이 추가되었다. 이제 작업 등록시에 시작시간, 주기, 종료시간, 최대 수행 횟수를 지정할 수 있게 되었다.

- 기존의 JEUS Scheduler 는 기본적으로 싱글쓰레드 방식으로 작업을 수행시켰지만 새로운 JEUS Scheduler 는 쓰레드풀(thread-pool)

을 이용하여 각 작업을 별도의 쓰레드로 동작시킨다. 따라서 어떤 한 작업이 수행중에 block 되더라도 다른 작업 수행에 영향이 없도록 하였다.

- 프로그래밍 방식으로 작업을 등록하지 않고 JEUS 서버 설정에 작업을 등록할 수 있는 Job-list 기능이 추가되었다.

하지만 JEUS 5.0 Scheduler 는 기존 버전과 하위 호환성을 유지하고 있다. 따라서 기존에 작성된 프로그램도 별도의 수정없이 운영 가능하다.

이번 장에서 다루는 모든 예제는 JEUS samples 디렉토리에서 찾아볼 수가 있다. 예제 파일들은 JEUS_HOME/samples/scheduler 디렉토리를 참조하기 바란다.

3.2 JEUS Scheduler 클래스 개요

이번 절에서는 JEUS Scheduler 를 구성하고 있는 주요 인터페이스와 클래스에 대해서 알아보도록 하자.

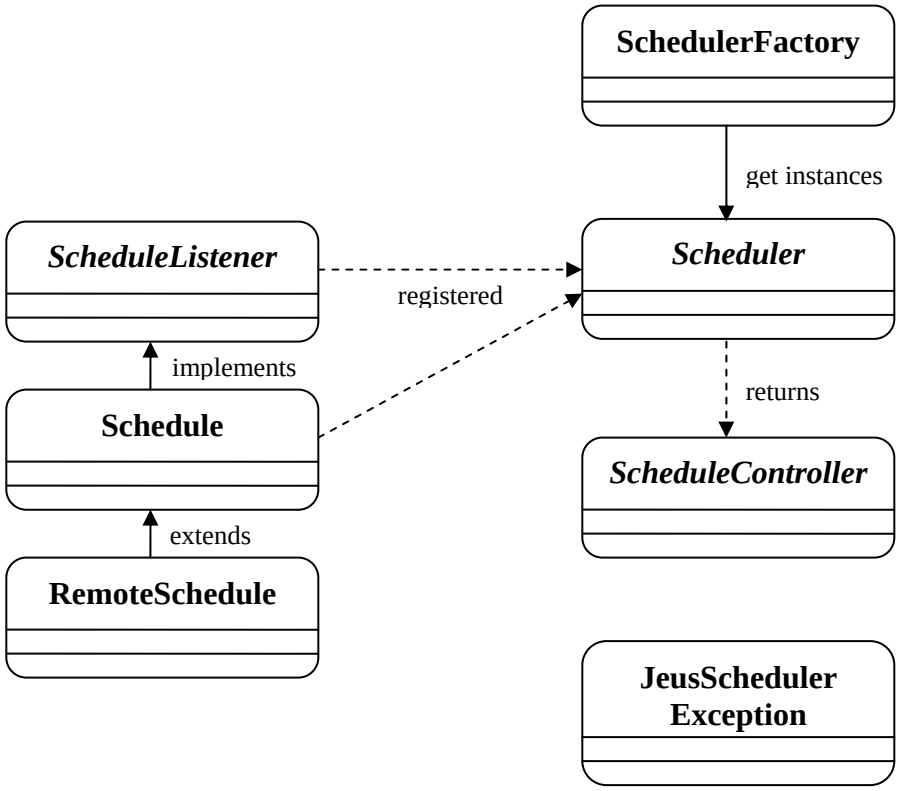
기본적으로 JEUS Scheduler 는 J2SE Timer 와 개념적으로 유사할 뿐만 아니라 유사한 인터페이스를 가지고 있다. 작업을 나타내는 java.util.TimerTask 클래스는 JEUS Scheduler 의 jeus.schedule.ScheduleListener 인터페이스와 같고, 작업을 등록하는 java.util.Timer 클래스는 JEUS Scheduler 의 jeus.schedule.Scheduler 인터페이스와 같다. 따라서, 이러한 유사성을 고려하여 JEUS Scheduler 를 사용한다면 좀더 쉽게 익숙해 질 수 있을 것이다.

package jeus.schedule

JEUS Scheduler 의 모든 클래스와 인터페이스는 jeus.schedule 패키지와 그 하위 패키지에 소속되어 있다.

interface ScheduleListener

정해진 시간에 수행되어야 할 작업은 ScheduleListener 인터페이스를 구현(implements)하여 클래스로 정의된다. ScheduleListener 는 하나의 Callback 메소드인 onTime()을 가지고 있어 해당 시간이 되었을 때 이 메소드가 호출된다.



abstract class Schedule

Schedule 클래스는 ScheduleListener 를 구현한 추상 클래스로 JEUS 5.0 이전에 사용하던 작업 클래스이다. 이 클래스는 onTime() 이외에도 nextTime() Callback 메소드가 있어서 작업 등록시에 호출될 시간을 예약하는 것이 아니라 작업 등록 후에 동적으로 다음 호출될 시간을 결정할 수 있다.

abstract class RemoteSchedule

RemoteSchedule 추상 클래스는 특수한 Schedule 클래스로 initialize() Callbak 메소드를 가지고 있어 객체 생성시에 초기화 파라미터 값을 받을 수 있다.

class SchedulerFactory

SchedulerFactory 클래스는 실제(concrete) Scheduler 객체를 얻어오기 위해 사용된다.

interface Scheduler

Scheduler 인터페이스는 JEUS Scheduler 에 작업을 등록하기 위한 핵심 인터페이스이다. 다양한 registerSchedule() 메소드를 정의하고 있다.

interface ScheduleController

Scheduler 에 작업을 등록하면 ScheduleController 인터페이스를 구현한 객체를 리턴받는다. 이 핸들 객체는 작업에 대한 정보를 얻거나 작업을 취소시키는데 사용된다.

exception JeusSchedulerException

JEUS Scheduler 에 작업 등록시나 취소시에 내부적으로 문제가 발생하는 경우 JeusSchedulerException 이 발생할 수 있다.

각 인터페이스나 클래스에 대한 자세한 설명은 JEUS Scheduler Javadoc API 를 참조하도록 한다.

3.3 작업 정의 하기

ScheduleListener interface 사용하기

정해진 시간에 수행되어야 할 작업은 ScheduleListener 인터페이스를 구현(implements)하여 클래스로 정의된다. ScheduleListener 는 하나의 Callback 메소드인 onTime()을 가지고 있어 해당 시간이 되었을 때 이 메소드가 호출된다. 따라서 작업 클래스를 정의하기 위해서는 onTime() 메소드를 구현하고 메소드 내에서 작업을 수행하도록 프로그램을 작성한다.

Task object example:

```
public class SimpleTask implements ScheduleListener {
   private String name;
   private int count;

   public SimpleTask(String name) {
      this.name = name;
   }

   public void onTime() {
      count++;
```

```
        echo("##### " + name + " is waked on " + new Date());
    }
...
}
```

Schedule class 사용하기

ScheduleListener 를 바로 구현하지 않고 Schedule 클래스나 RemoteSchedule 클래스를 상속하여 작업 클래스를 정의할 수도 있다. Schedule 클래스나 RemoteSchedule 클래스는 JEUS 5.0 이전 Scheduler 에서 사용되던 작업 클래스이다. JEUS 5.0 에서는 일반적인 작업을 정의할 때는 ScheduleListener 를 구현하도록 하고 있지만 하위 호환성을 위해 기존 작업 클래스를 그대로 제공하고 있다.

Schedule 추상 클래스는 onTime() 이외에도 nextTime() 이라는 Callback 메소드가 있어서 작업 등록시에 호출될 시간을 예약하지 않고 작업 클래스 내에서 다음 호출될 시간을 결정하도록 한다. 따라서 고정 주기를 갖는 작업보다는 가변적인 주기를 갖는 작업의 경우에 좀더 사용될 수 있다.

JEUS Scheduler 는 Schedule 작업 객체의 처음 수행시간을 결정하기 위해 작업 객체 등록후에 먼저 nextTime()을 호출하여 처음 수행시간을 정한다. 그런 다음 해당시간이 되면 onTime()을 호출하여 작업을 수행하고 onTime()이 종료되면 다시 nextTime()을 호출하여 다음 수행시간을 정하게 된다. nextTime() 은 다음번에 작업이 수행될 절대시간을 milli-second 값으로 넘겨주어야 한다. 이 때 0 을 리턴하면 작업이 더 이상 수행되지 않게 된다.

Schedule 작업 객체는 onTime()이 수행된 후에 nextTime()을 호출하기 때문에 onTime()에서 작업을 수행한 시간만큼 nextTime() 호출이 지체된다. 따라서 정확한 간격으로 작업을 호출하도록 프로그래밍 하기가 쉽지 않다. 그렇기 때문에 되도록이면 작업 내의 nextTime()을 통해 반복 주기를 구현하기 보다는 작업 등록시에 반복 주기를 주는 것이 좋다.

Schedule object example:

```
public class SimpleSchedule extends Schedule {
    private String name;
    private int count;
    private long period = 2000; // 2 seconds
```

```
   public SimpleSchedule(String name) {
      this.name = name;
   }

   public void onTime() {
      count++;
      echo("##### " + name + " is waked on " + new Date());
   }

   public long nextTime(long currentTime) {
      return currentTime + period;
   }
...
}
```

RemoteSchedule class 사용하기

RemoteSchedule 클래스는 원격으로 작업을 등록할 때 초기화 변수를 지정할 수 있는 Schedule 객체이다. 주로 원격에서 클래스 이름을 통해 작업 객체를 등록할 때 사용된다.

이 클래스는 initialize() Callback 메소드를 가지고 있는데 작업 등록후에 초기화 파라미터로 이 메소드가 한번 호출된다. 따라서 원격에서 클래스 이름으로 작업을 등록할 때 초기화 값을 주고 싶을 때 사용될 수 있다.

initialize() Callback 은 Scheduler.registerSchedule(classname, hashtable, daemon_flag) 메소드를 이용하여 RemoteSchedule 작업 객체를 등록하는 경우에만 호출된다.

RemoteSchedule object example:

```
public class SimpleRemoteSchedule extends RemoteSchedule {
   private String name;
   private int count;
   private long period;

   // this should have no-arg constructor
   // because this will be instantiated by Class.newInstance()
   public SimpleRemoteSchedule() {
```

```
    }

    // this is called by scheduler after creation
    public void initialize(Hashtable parameters) {
        name = (String) parameters.get("name");
        Long interval = (Long) parameters.get("interval");
        if (interval != null)
            period = interval.longValue();
        else
            period = 2000;
    }

    public void onTime() {
        count++;
        echo("##### " + name + " is waked on " + new Date());
    }

    public long nextTime(long currentTime) {
        return currentTime + period;
    }
...
}
```

3.4 Scheduler 객체 얻기

JEUS Scheduler 는 로컬 환경과 원격 환경에서 모두 구동될 수 있다.

로컬 환경에서 구동된다는 것은 프로그램이 구동되고 있는 로컬 JVM 내에 Scheduler 인스턴스가 생성되며 등록된 모든 작업이 같은 JVM 내에서 구동된다는 것을 말한다. JEUS Scheduler 는 JVM 내에서 현재 하나의 인스턴스만 생성되며 이 인스턴스를 Default Scheduler 라고 한다. 따라서 현재는 JVM 내에 모든 클라이언트 들이 Default Scheduler 를 공유하게 된다. 로컬 환경의 JEUS Scheduler 는 일반 J2SE 애플리케이션이나 J2EE 애플리케이션 클라이언트, J2EE 컴포넌트 등에서 사용된다.

원격 환경에서 구동된다는 것은 프로그램이 구동되고 있는 JVM 이 아닌 원격의 다른 JVM 에서 Scheduler 인스턴스가 생성되고 등록된 모든 작업이 원격 JVM 내에서 구동된다는 것을 말한다. 원격 Scheduler 는

RMI 객체 형태로 나타나기 때문에 클라이언트는 RMI 콜을 통해 JEUS Scheduler 를 사용하게 된다. JEUS 환경에서는 JEUS Node 에 원격 Scheduler Service 가 구동된다. 원격 환경의 JEUS Scheduler 는 원격 클라이언트가 JEUS Node 에 작업을 등록할 때 사용된다.

로컬 환경의 JEUS Scheduler 를 사용하기 위해서 SchedulerFactory 를 이용한다. 아래와 같이 간단하게 Default Scheduler 인스턴스를 얻을 수 있다.

```
// Get the default scheduler
Scheduler scheduler = SchedulerFactory.getDefaultScheduler();
```

JEUS Node 에 있는 원격 JEUS Scheduler 를 사용하기 위해서 JNDI Lookup 을 이용한다. 아래와 같이 JEUS 노드 Scheduler 인스턴스(스텁)를 얻을 수 있다.

```
// Get the remote scheduler
InitialContext ic = new InitialContext();
Scheduler scheduler = (Scheduler)ic.lookup(Scheduler.NODE_SCHEDULER_NAME);
```

참고: JEUS 5.0 이전에 사용되던 jeus.schedule.server.SchedulerManager와 jeus.schedule.client.SchedulerManager는 더 이상 사용되지 않는다(deprecated). 대신 SchedulerFactory를 통해 Scheduler 객체를 얻어서 사용하기를 권한다. 하지만 하위 호환성을 유지하기 위해 위 클래스들은 JEUS 5.0 에도 그대로 제공된다.

3.5 작업 등록하기

이번 절에서는 Scheduler 인터페이스를 이용하여 작업을 등록하는 방법에 대해서 알아보도록 한다.

로컬 환경에서든 원격 환경에서든 Scheduler 인스턴스를 얻어왔다면 작업을 등록 방법은 동일하다. 단지 원격 JEUS Scheduler 는 작업 객체가 원격으로 전송(serialization)되어 원격에서 돌아간다는 점만 차이가 있다.

한번 수행되는 작업 등록하기

단지 특정 시간에 한번만 수행되어야 할 작업의 경우에는 하나의 수행 시간만 주어 작업을 등록할 수 있다. 이 때 수행시간은 java.util.Date 객체로 절대 시간을 줄 수도 있고, milli-second 값으로 현재 시간 이후 얼

마의 delay 후에 수행되어야 하는지 줄 수도 있다. 이 경우 다음 메소드를 이용하여 작업을 등록한다.

```
registerSchedule(ScheduleListener task, Date time, boolean isDaemon)

registerSchedule(ScheduleListener task, long delay, boolean isDaemon)
```

참고: isDaemon 파라미터는 추후에 설명한다.

다음은 작업을 등록하는 예제 코드이다.

```
SimpleTask task1 = new SimpleTask("task1");
Date firstTime1 = new Date(System.currentTimeMillis() + 2000);
ScheduleController handle1
  = scheduler.registerSchedule(task1, firstTime1, false);
```

반복되는 작업 등록하기

반복되는 작업의 경우 첫 수행시간, 주기, 종료시간, 최대 수행횟수 등을 주어 작업을 등록할 수 있다. 반복되는 작업의 특성에 따라 반복 주기가 fixed-delay 방식으로 반복될 지 아니면 fixed-rate 방식으로 반복될 지를 결정해야 한다.

참고: Fixed-delay vs. Fixed-rate

Fixed-delay – 작업이 수행되는 간격이 일정하게 유지된다. 작업의 다음 수행시간은 이전 수행시간과 주기에 의해서 결정된다. 만약 작업의 수행이 지체(작업 수행시간이 오래걸리거나 garbage collection 과 같은 외부 이유에 의해서 지체되는 경우)되어 다음 작업이 수행되어야 할 시기가 넘어간 경우에 다음 작업은 바로 수행되며 그 이후에 수행되는 작업들은 그만큼 지체된다. 따라서 장기적으로는 작업의 수행시간이 조금씩 뒤쳐질 수 있다.

Fixed-rate – 작업이 수행되는 비율이 일정하게 유지된다. 작업의 다음 수행시간은 첫 수행시간과 주기에 의해서 결정된다. 작업의 수행이 지체되더라도 다음 작업은 바로 뒤따라 수행되며 시간당 수행되는 비율을 유지한다. 장기적으로 작업의 수행시간이 초기에 지정한 주기에 따라 계속 유지된다.

JEUS Scheduler 는 Fixed-rate 로 작업을 등록하면 비교적 정확한 주기적인 호출시간을 보장 해주기 위해 작업 수행이 지체되더라도 시간이 되면 다른 쓰레드에의해 작업을 호출한다. 따라서 작업 수행이 지체되는 경우에 같은 작업이 동시에(concurrently) 돌게 된다. 따라서 이러한 경우에는 작업 객체가 thread-safe 한지 고려해야 한다.

이 경우 다음 메소드를 이용하여 작업을 등록한다.

```
registerSchedule(ScheduleListener task, Date firstTime, long period,
 Date endTime, long maxcount, boolean isDaemon)

registerSchedule(ScheduleListener task, long delay, long period,
 Date endTime, long maxcount, boolean isDaemon)

registerScheduleAtFixedRate(ScheduleListener task, Date firstTime,
 long period, Date endTime, long maxcount, boolean isDaemon)

registerScheduleAtFixedRate(ScheduleListener task, long delay,
 long period, Date endTime, long maxcount, boolean isDaemon)
```

작업 등록시에 사용하는 파라미터 들은 다음과 같다.

- Date firstTime(시작시간): 처음 수행될 시간.
- long delay(시작시간): 현재 이후에 처음 수행될 시간 (milli-seconds)
- long period(반복주기): 반복 수행 주기(milli-seconds)
- Date endTime(종료시간): 종료 시간으로 이 시간 이후에는 작업이 더이상 수행되지 않는다. null 인 경우에는 종료시간에 제약이 없다.
- long maxcount (수행횟수): 최대 수행 횟수. Scheduler.UNLIMITED 인 경우에는 제한이 없다.
- isDaemon: 원격 클라이언트의 경우에만 사용되며 true 이면 연결이 끊어지면 작업을 자동종료 한다.
- isThreaded: 더 이상 사용되지 않는다(deprecated).

Daemon flag 에 대해:

원격으로 Schedule 을 등록하는 경우에만 의미가 있으며 true 값으로 세팅하면 클라이언트와의 연결이 종료되었을 때 작업이 종료된다. 하지만 현재 RMI Runtime 의 DGC(Distrubuted Garbage Collection) 정책에 의해 클라이언트가 연결이 종료되었음을 판단하기 때문에 실제로 클라이언트의 연결이 종료되고 15 분 정도가 지나야 종료되었음을 탐지하게 되어 스케쥴링이 취소된다.

아래는 작업을 등록하는 예제 코드이다.

```
SimpleTask task2 = new SimpleTask("task2");
ScheduleController handle2 = scheduler.registerSchedule(task2, 2000, 2000,
  null, Scheduler.UNLIMITED, false);

SimpleTask task3 = new SimpleTask("task3");
Date firstTime3 = new Date(System.currentTimeMillis() + 2000);
Date endTime3 = new Date(System.currentTimeMillis() + 10 * 1000);
ScheduleController handle3 = scheduler.registerScheduleAtFixedRate(task3,
  firstTime3, 2000, endTime3, 10, false);
```

Schedule 작업 객체 등록하기

Schedule 이나 RemoteSchedule 작업 객체를 등록하는 것은 하위 호환성을 유지하기 위해 제공된다.

작업의 처음 수행시간과 이후에 반복되는 수행시간은 Schedule 작업 객체의 nextTime() 메소드를 이용하기 때문에 등록시에는 별도의 파라미터를 줄 필요가 없다. 이 경우 다음 메소드를 이용하여 작업을 등록한다.

registerSchedule(Schedule task, boolean isDaemon)

registerSchedule(String classname, Hashtable params, boolean isDaemon)

아래는 작업을 등록하는 예제 코드이다.

```
Hashtable params = new Hashtable();
params.put("name", "task3");
params.put("interval", new Long(3000));
ScheduleController handle3 = scheduler.registerSchedule(
  "samples.scheduler.SimpleRemoteSchedule", params, true);

SimpleSchedule task4 = new SimpleSchedule("task4");
```

```
ScheduleController handle4
  = scheduler.registerSchedule(task4, true);
```

3.6 작업 제어하기

JEUS Scheduler 에 작업을 등록하면 핸들(handle)인 ScheduleController 객체를 리턴한다. 이 객체는 등록된 작업 하나당 만들어지는데 등록된 작업에 제어하기 위해 사용될 수 있다.

이 핸들을 이용하여 작업에 대한 정보를 얻어오거나 작업을 취소시킬 수 있다.

아래의 예제는 ScheduleController.cancel() 메소드를 호출하여 작업을 취소시키는 것을 보여주고 있다.

```
SimpleTask task2 = new SimpleTask("task2");
ScheduleController handle2 = scheduler.registerSchedule(task2, 2000, 2000,
   null, Scheduler.UNLIMITED, false);

Thread.sleep(10 * 1000);
handle2.cancel();
```

3.7 Stand-alone JEUS Scheduler 사용하기

일반 J2SE 애플리케이션이나 J2EE 애플리케이션 클라이언트에서 JEUS Scheduler 를 사용할 수 있다. 이 경우에는 JEUS Server 와 별개로 JEUS Scheduler 를 J2SE Timer 와 같이 라이브러리처럼 사용할 수 있다.

Scheduler 객체를 얻기 위해서는 SchedulerFactory 클래스를 이용하면 된다. 또한 로컬 환경에서는 작업 등록시에 daemon flag 는 사용되지 않으므로 어떤 값을 넣어도 무방하다.

다음은 Stand-alone 클라이언트 예제 코드이다.

```
public class StandAloneClient {
   public static void main(String args[]) {
      try {
         // Get the default scheduler
```

```
        Scheduler scheduler = SchedulerFactory.getDefaultScheduler();

        // Register SimpleTask which runs just one time
        echo("Register task1 which runs just one time...");
        SimpleTask task1 = new SimpleTask("task1");
        Date firstTime1 = new Date(System.currentTimeMillis() + 2000);
        ScheduleController handle1
           = scheduler.registerSchedule(task1, firstTime1, false);

        Thread.sleep(5 * 1000);
        echo("");

        // Register SimpleTask which is repeated with fixed-delay
        echo("Register task2 which is repeated until it is canceled...");
        SimpleTask task2 = new SimpleTask("task2");
        ScheduleController handle2
           = scheduler.registerSchedule(task2, 2000, 2000, null,
Scheduler.UNLIMITED, false);

        Thread.sleep(10 * 1000);
        handle2.cancel();
        echo("");

        // Register SimpleTask which is repeated with fixed-rate
        echo("Register task3 which is repeated for 10 seconds...");
        SimpleTask task3 = new SimpleTask("task3");
        Date firstTime3 = new Date(System.currentTimeMillis() + 2000);
        Date endTime3 = new Date(System.currentTimeMillis() + 10 * 1000);
        ScheduleController handle3
           = scheduler.registerScheduleAtFixedRate(task3, firstTime3, 2000,
endTime3, 10, false);

        Thread.sleep(12 * 1000);
        echo("");

        // Register SimpleSchedule which is repeated every 2 seconds
        echo("Register task4 which is repeated every 2 seconds...");
        SimpleSchedule task4 = new SimpleSchedule("task4");
        ScheduleController handle4
```

```
            = scheduler.registerSchedule(task4, false);

         Thread.sleep(10 * 1000);
         echo("");

         // Cancel all tasks
         echo("Cancel all tasks registerd on the scheduler...");
         scheduler.cancel();
         Thread.sleep(5 * 1000);

         System.out.println("Program terminated.");
      } catch (Exception e) {
         e.printStackTrace();
      }
   }

   private static void echo(String s) {
      System.out.println(s);
   }

}
```

J2EE 애플리케이션 클라이언트에서 JEUS Scheduler를 사용하는 경우에는 Deployment Descriptor(jeus-client-dd.xml) 에 JEUS Scheduler의 쓰레드풀에 관련된 설정을 할 수가 있다. 설정에 대해서는 4.4 절을 참조하기 바란다.

주의: JEUS Scheduler 를 사용한 코드를 컴파일하거나 구동시키기 위해서는 JEUS 관련 클래스(jeus.jar 등)가 classpath 에 지정되어 있어야 한다.

3.8 JEUS 노드 Scheduler Service 사용하기

JEUS 노드에 Scheduler Service가 구동되어 있다면 원격 클라이언트가 이를 사용할 수 있다. 이를 위해서는 먼저 JEUS 노드에 Scheduler Service가 구동되도록 설정되어 있어야 한다. 설정에 대해서는 4.2절을 참고하도록 한다.

JEUS 노드 Scheduler Service 는 RMI Scheduler 객체를 JNDI 에 등록해두고 있다. 따라서 클라이언트에서는 JNDI Lookup 을 통해서 원격

Scheduler 객체(실제론 Stub 객체)를 얻을 수 있다. 이 객체의 JNDI 이름은 “jeus_service/Scheduler”로 Scheduler.NODE_SCHEDULER_NAME 상수을 사용하도록 한다.

일단 Scheduler 객체를 얻으면 작업을 등록하는 방법은 동일하다. 단지 등록된 작업 객체는 전송(Serialization)되어 원격 Scheduler 에서 실제로 돌아가게 된다. 즉, JEUS 노드에서 수행된다.

이 경우 등록시에 daemon flag 는 의미가 있으며 daemon flag 를 true 로 작업을 등록하면 앞 절에서 이야기 한대로 원격 클라이언트가 종료되면 원격 작업도 종료되도록 한다.

다음은 Remote 클라이언트 예제 코드이다.

```
public class RemoteClient {
   public static void main(String args[]) {
      try {
         // Get the remote scheduler
         InitialContext ic = new InitialContext();
         Scheduler scheduler = (Scheduler)
ic.lookup(Scheduler.NODE_SCHEDULER_NAME);

         // Register SimpleTask which runs just one time
         echo("Register task1 which runs just one time...");
         SimpleTask task1 = new SimpleTask("task1");
         Date firstTime1 = new Date(System.currentTimeMillis() + 2000);
         ScheduleController handle1
            = scheduler.registerSchedule(task1, firstTime1, true);

         Thread.sleep(5 * 1000);
         echo("");

         // Register SimpleTask which is repeated with fixed-delay
         echo("Register task2 which is repeated until it is canceled...");
         SimpleTask task2 = new SimpleTask("task2");
         ScheduleController handle2
            = scheduler.registerSchedule(task2, 2000, 2000, null,
Scheduler.UNLIMITED, true);

         Thread.sleep(10 * 1000);
```

```
            handle2.cancel();
            echo("");

            // Register SimpleRemoteSchedule which is repeated every 3 seconds
            echo("Register task3 which is repeated every 3 seconds...");
            Hashtable params = new Hashtable();
            params.put("name", "task3");
            params.put("interval", new Long(3000));
            ScheduleController handle3
               =
scheduler.registerSchedule("samples.scheduler.SimpleRemoteSchedule", params,
true);

            Thread.sleep(10 * 1000);
            echo("");

            // Cancel all tasks
            echo("Cancel all tasks registerd on the scheduler...");
            scheduler.cancel();
            Thread.sleep(5 * 1000);

            System.out.println("Program terminated.");

         } catch (Exception ex) {
            ex.printStackTrace();
         }
      }

      private static void echo(String s) {
         System.out.println(s);
      }
}
```

주의: 위 예제를 실행하려면, 해당 작업 클래스 파일을 jar 파일로 묶어서 JEUS_HOME \lib\application 에 복사해야 한다. JEUS 노드가 해당 작업을 수행하기 위해서는 해당 클래스를 로딩해야하기 때문이다.

3.9 J2EE 컴포넌트에서 JEUS Scheduler 사용하기

EJB 나 Servlet 과 같은 J2EE 컴포넌트에서 JEUS Scheduler 를 사용할 수 있다.이 때 JEUS Scheduler 는 JEUS Engine Container 에서 구동되게 된다.

EJB 2.1 표준에는 EJB Timer Service 를 명시하고 있으며 JEUS 5.0 에서도 EJB Timer Service 를 제공하고 있다. 따라서 EJB 컴포넌트의 경우 J2EE 표준을 준수하려면 JEUS Scheduler 보다는 EJB Timer Service 를 쓰는 것을 권장한다. 하지만 EJB 외의 J2EE 컴포넌트에서는 EJB Timer Service 를 쓸 수 없으므로 JEUS Scheduler 를 사용해야 한다.

J2EE 컴포넌트에서 JEUS Scheduler 를 사용하는 것은 Stand-alone JEUS Scheduler 를 사용하는 방식과 동일하다. SchedulerFactory 클래스를 이용하여 Engine Container 에서 구동되는 Scheduler 객체를 얻어온 후에 필요한 등록 메소드를 호출하여 작업을 등록하면 된다.

다음은 EJB 에서 JEUS Scheduler 사용 예제이다.

```
public class HelloEJB implements SessionBean {
   private SimpleTask task;
   private ScheduleController taskHandler;
   private boolean isStarted;

   public HelloEJB() {
   }

   public void ejbCreate() {
      task = new SimpleTask("HelloTask");
      isStarted = false;
   }

   public void trigger() throws RemoteException {
      if (!isStarted) {
         Scheduler scheduler = SchedulerFactory.getDefaultScheduler();
         taskHandler
            = scheduler.registerSchedule(task, 2000, 2000, null,
Scheduler.UNLIMITED, false);
      }
   }
```

```
    public void ejbRemove() throws RemoteException {
        if (isStarted)
            taskHandler.cancel();
    }

    public void setSessionContext(SessionContext sc) {
    }

    public void ejbActivate() {
    }

    public void ejbPassivate() {
    }
}
```

```
public class HelloClient {

    public static void main(String args[]) {
        try {
            InitialContext ctx = new InitialContext();

            HelloHome home = (HelloHome) ctx.lookup("helloApp");
            Hello hello = (Hello) home.create();
            hello.trigger();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

JEUS Engine Container에서 동작하는 JEUS Scheduler 에 대해 쓰레드풀과 Job-list 설정을 할 수가 있다. 설정에 대해서는 4.3절을 참조하기 바란다.

3.10 Job-list 사용하기

JEUS 서버 에 프로그래밍 방식으로 작업을 등록하지 않고 설정파일을 이용하여 작업을 등록할 수 있다. 이것을 Job-list 라고 한다.

Job-list 는 JEUS 노드 Scheduler 와 JEUS Engine Container Scheduler 에 등록할 수 있다.

Job-list 로 작업을 등록하면 작업은 fixed-rate 방식으로 반복 수행된다.

Job-list를 등록하기 위해 필요한 설정은 4.5절을 참조하기 바란다.

3.11 결론

이것으로 JEUS Scheduler 프로그래밍에 관련된 내용을 마친다. 지금까지 작업을 정의하는 법, Scheduler 객체를 얻는법, 작업 객체를 등록하는 법 그리고 등록된 작업을 제어하는 방법에 대해서 알아보았다.

이번 장을 통하여 JEUS Scheduler 를 사용하기 위한 대부분의 기술을 배울 수 있었다.

4 JEUS Scheduler 설정

4.1 소개

이번 장에서는 JEUS 설정파일이나 Deployment Descriptor 에 JEUS Scheduler 를 설정하는 방법에 대해서 다룬다.

JEUS 노드 Scheduler Service 를 사용하기 위해서는 JEUSMain.xml 설정파일에 JEUS Scheduler 를 사용하도록 설정되어 있어야 한다. 그외에도 쓰레드풀과 Job-list 에 대해서 설정할 수 있다.

JEUS 노드 Scheduler Service 외에도 Engine Container Scheduler Service 에 대한 쓰레드 풀과 Job-list 를 설정할 수 있다. 또한, J2EE 애플리케이션 클라이언트에서 사용할 JEUS Scheduler 에 대해서도 Deployment Descriptor 에서 설정할 수 있다.

기본적으로 JEUS Scheduler 에 대한 설정을 하지 않으면 기본 값으로 JEUS Scheduler 가 초기화되기 때문에 JEUS 노드 Scheduler Service 외에는 무설정으로 JEUS Scheduler 를 사용할 수 있다.

4.2 JEUS Node Scheduler Service 설정

원격에서 JEUS 노드 Scheduler Service 에 접근하는 경우나 Job-list 를 이용하여 JEUS Server 에서 주기적인 작업을 수행시키려고 한다면 JEUS 노드 Scheduler Service 를 활성화 시켜야 한다.

JEUS 노드 Scheduler Service 에 대한 설정은 JEUSMain.xml 설정 파일의 <node> element 안의 <scheduler> element 에서 할 수 있다.

JEUS 노드 Scheduler Service 를 활성화 시키는 것은 아래와 같이 <scheduler> element 에서 <enabled> element 값을 true 로 세팅하면 된다.

<<JEUSMain.xml>>

```
<jeus-system>
    <node>
```

```
      ...
       <scheduler>
         <enabled>true</enabled>
       </scheduler>
        ...
   </node>
   ...
</jeus-system>
```

설정 파일 수정 후에 JEUS Server 를 재시작해야 반영된다.

또한 아래와 같이 JEUS Scheduler 에서 사용하는 쓰레드 풀에 대한 세밀한 설정도 할 수 있다.

```
<jeus-system>
    <node>
      ...
       <scheduler>
         <enabled>true</enabled>

          <!-- Scheduler Thread-pool settings -->
          <thread-pool>
            <min>2</min>
            <max>30</max>
            <period>3600000</period>
          </thread-pool>

       </scheduler>
        ...
   </node>
   ...
</jeus-system>
```

자세한 사항은 부록의 JEUSMain.xml 설정 Reference 을 참조하기 바란다.

4.3 JEUS Engine Container Schedule Service 설정

J2EE 컴포넌트에서 사용하는 JEUS Scheduler Service 를 역시 JEUSMain.xml 설정파일에서 설정할 수 있다.

설정하는 방식은 JEUS 노드 Scheduler Service 와 동일하며 설정하는 위치가 <engine-container> element 밑에 <scheduler>에서 한다는 점이 차이가 있다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    <node>
       ...
            <engine-container>
          ...
          <!-- Engine-Container Scheduler Service -->
          <scheduler>
            <enabled>true</enabled>

            <!-- Scheduler Thread-pool settings -->
            <thread-pool>
              <min>2</min>
              <max>30</max>
              <period>3600000</period>
            </thread-pool>

          </scheduler>
            ...
       </engine-container>
         ...
   </node>
   ...
</jeus-system>
```

설정 파일 수정 후에 JEUS Server 를 재시작해야 반영된다.

4.4 Client Container Scheduler Service 설정

J2EE 애플리케이션을 사용하는 경우 Client Container 에서 구동되는 JEUS Scheduler 에 대한 설정을 할 수 있다.

애플리케이션 client을 위한 JEUS deployment descriptor인 jeus-client-dd.xml 파일에 다음과 같이 `<scheduler>` 설정을 넣는다.

<<*jeus-client-dd.xml*>>

```
<?xml version="1.0"?>
<jeus-client-dd>
 <module-info>
        ...
 </module-info>
 ...
 <scheduler>
       <enabled>true</enabled>

       <!-- Scheduler Thread-pool settings -->
       <thread-pool>
         <min>2</min>
         <max>30</max>
         <period>3600000</period>
       </thread-pool>

     </scheduler>

</jeus-client-dd>
```

Client Container 와 애플리케이션 Client 에 대해서는 JEUS Client Application 안내서를 참조하기 바란다.

4.5 Job-list 설정

프로그램 코드에서 작업을 등록하는 것 이외에 JEUS 설정 파일에 작업을 등록하여 JEUS 서버 구동시에 작업이 자동으로 스케쥴링 되게 할 수 있다.

여기서 Job 이란 스케쥴링 될 하나의 작업 단위를 말한다. 실제로 jeus.schedule.ScheduleListener 를 구현한 클래스를 Job 으로 등록할 수 있다.

Job 은 JEUS 노드에 등록하거나 JEUS Engine Container 에 등록할 수 있다. 어느 곳에 등록하느냐에 따라 수행되는 위치가 달라진다. JEUS 노드에 등록하면 JEUS 노드 Scheduler Service 에서 작업이 수행이 되며, JEUS Engine Container 에 등록하면 JEUS Engine Container JVM 에서 작업이 수행된다. 따라서 사용자의 필요에 따라 적절한 위치에서 Job 을 등록해야 한다.

Job 에 해당하는 클래스는 반드시 jeus.schedule.ScheduleListener 를 구현해야 하며 해당 클래스와 관련 클래스를 jar 파일로 묶어 JEUS_HOME\lib\application 디렉토리 밑에 놓아야 한다.

Job-list 설정은 JEUSMain.xml 설정파일에 <scheduler> element 내에서 할 수 있다. JEUS 노드 Job-list 의 경우에는 <node> element 밑에 있는 <scheduler> element 에 <job-list>를 추가하고, Engine Container Job-list 의 경우에는 <engine-container> element 밑에 있는 <scheduler> element 에 <job-list>를 추가한다.

다음은 하나의 Job 을 등록하는 예제이다.

```
<scheduler>
...
 <job-list>
    <job>
       <class-name>samples.ScheduleJob</class-name>
       <name>ScheduleJob</name>
       <begin-time>2005-02-01T00:00:00+09:00</begin-time>
       <end-time>2005-03-01T00:00:00+09:00</end-time>
       <interval>
          <minutely>30</minutely>
       </interval>
       <count>-1</count>
    </job>
 </job-list>
</scheduler>
```

각각의 파라미터의 의미는 아래와 같다.

- class-name: 작업 클래스 파일이름 (fully-qualified class name)
- name: (optional) 작업에 대한 이름을 지정한다.
- description: (optional) 작업에 대한 설명을 지정한다.
- begin-time: (optional) 작업이 최초로 수행될 시간. XML dateTime type 으로 Timezone 까지 입력한다(한국은 +09:00).주어지지 않으면 JEUS Server 시작시에 시작된다.
- end-time: (optional) 작업이 끝날 시간. XML dateTime type 으로 Timezone 까지 입력한다(한국은 +09:00). 주어지지 않으며 종료하지 않는다.
- interval: 작업 주기로 <millisecond>, <minutely>, <hourly>, <daily> element 를 이용하여 입력할 수 있다.
- count: (optional) 작업의 최대 수행 횟수, 주어지지 않거나 -1 인 경우 최대 수행 횟수에 제한이 없다.

주의: 만약 등록한 작업의 begin-time 이 과거이면 주기적으로 현재시간 이후에 최초의 수행되어야 할 시간에 최초 수행되도록 조정된다. end-time이 과거인 경우에는 작업은 한번도 수행되지 않는다.

Job-list 방식으로 등록된 작업은 fixed-rate 방식으로 반복된다. 따라서 비교적 정확한 시간에 호출되지만 작업 수행이 오래 걸릴 때는 작업이 동시에 진행될 수 있기 때문에 작업이 thread-safe 하도록 고려해야 한다.

4.6 결론

이번 장에서는 JEUS Scheduler 를 설정하는 것에 대해서 배웠다.

좀더 자세한 설정 항목과 설명은 부록의 Reference 를 참조하도록 한다.

5 결론

이상으로 JEUS Scheduler 의 개념과 이를 사용하는 방법에 대해서 다루었다..

이 문서외에도 자세한 사항을 알기 위해서 JEUS Scheduler Javadoc API 를 참조하기 바란다.

6 마치며

지금까지 JEUS Scheduler 의 사용법에 대해서 알아보았다.

지금까지 본 매뉴얼을 정독하고 따라 해보면서 JEUS Scheduler 를 사용하는데 필요한 기술을 충분히 습득했을 것이다.

본 매뉴얼에서 자세히 언급하지 못한 부분은 다음 매뉴얼을 참고하기 바란다.

- JEUS Scheduler Javadoc API

A. JEUSMain.xml Scheduler 설정 Reference

A.1 소개

본 부록의 레퍼런스는 JEUS 의 메인 설정 파일인 JEUSMain.xml 에서 Scheduler 태그에 대해서 설명하고 있다.

이 파일의 xsd 파일은 "JEUS_HOME\config\xsds" 디렉토리의 "jeus-main.xsd" 파일이다.

본 레퍼런스는 3 부분으로 나눠져있다.

1. **XML Schema/XML 트리**: XML 설정 파일의 모든 태그 리스트를 정리했다. 각 노드의 형식은 다음과 같다.

    a. 태그 레퍼런스로 빨리 찾아보기위해서 각 태그마다 인덱스 번호(예: (11))를 붙여놓았다. 태그 레퍼런스에서는 이 번호 순서로 설명한다.

    b. XML Schema 에서 정의한 XML 태그명을 <**tag name**> 형식으로 표시한다.

    c. XML Schema 에서 정의한 Cardinality 를 표시한다. "?" = 0 개 나 1 개의 element, "+" = 1 개 이상의 element, "*" = 0 개 이상의 element, (기호가 없음) = 정확히 1 개의 element

    d. 몇몇 태그에는 "P" 문자를 붙여놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝할 때 사용된다.

2. **태그 레퍼런스**: 트리에 있는 각 XML 태그를 설명한다. 이런 테이블은 다음 하위-항목들을 포함하고 있다.

    a. **Description**: 태그에 대한 간단한 설명

    b. **Value Description**: 입력하는 값과 타입

c. **Value Type**: 값의 데이터 타입. 예) String

d. **Default Value**: 해당 XML 을 사용하지 않았을 때 기본적으로 사용되는 값

e. **Defined values**: 이미 정해져 있는 값

f. **Example**: 해당 XML 태그에 대한 예

g. **Performance Recommendation**: 성능 향상을 위해서 추천하는 값

h. **Child Elements**: 자신의 태그 안에 사용하는 태그

3. **샘플 XML 파일**: "JEUSMain.xml"에 대한 완전한 예제

A.2 XML Schema/XML 트리

```
(91) <scheduler>?
       (92) <enabled>? P
       (93) <thread-pool>?
              (94) <min>? P
              (95) <max>? P
              (96) <period>? P
       (97) <job-list>?
              (98) <job>*
                     (99) <class-name>
                     (100) <name>?
                     (101) <description>?
                     (102) <begin-time>?
                     (103) <end-time>?
                     (104) <interval>
                            (105) <millisecond>
                            (106) <minutely>
                            (107) <hourly>
                            (108) <daily>
                     (109) <count>? P
```

A.3 Element Reference

(91) <jeus-system> <node> <engine-container> **<scheduler>**	
Description	JEUS Scheduler 에 관련된 설정을 담고있다.
Child Elements	(92)enabled? (93)thread-pool? (97)job-list?

(92) <jeus-system> <node> <engine-container> <scheduler> **<enabled>**	
Description	Scheduler Service 를 구동시킬지를 설정한다.
Value Type	boolean
Default Value	true

(93) <jeus-system> <node> <engine-container> <scheduler> **<thread-pool>**	
Description	scheduler 에서 multi-thread 로 job 을 실행할때 사용하는 thread pool 을 설정한다.
Child Elements	(94)min? (95)max? (96)period?

(94) <jeus-system> <node> <engine-container> <scheduler> <thread-pool> **<min>**	
Description	pooling 되는 객체의 최소값을 지정한다.
Value Type	nonNegativeIntType
Value Type Description	0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.
Default Value	2

(95) <jeus-system> <node> <engine-container> <scheduler> <thread-pool> **<max>**	
Description	pooling 되는 객체의 최대값을 지정한다.
Value Type	nonNegativeIntType
Value Type Description	0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.
Default Value	30

(96) <jeus-system> <node> <engine-container> <scheduler> <thread-pool> **<period>**	
Description	pooling 되는 객체를 정리하는 시간을 지정한다.
Value Type	long
Default Value	3600000

Performance Recommendation 이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

```
(97) <jeus-system> <node> <engine-container> <scheduler> <job-list>
```

Description scheduler 에 등록할 job list 을 지정한다.

Child Elements
```
(98)job*
```

```
(98) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job>
```

Description scheduler 에 등록할 하나의 job 을 지정한다.

Child Elements
```
(99)class-name
(100)name?
(101)description?
(102)begin-time?
(103)end-time?
(104)interval
(109)count?
```

```
(99) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <class-name>
```

Description job 을 수행하는 class 의 fully qualified name 이다.

Value Type token

```
(100) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <name>
```

Description 이 job 의 이름을 지정한다.

Value Type token

```
(101) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <description>
```

Description 이 job 의 설명을 적을 수 있다.

Value Type string

```
(102) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <begin-time>
```

Description 이 job 의 시작 시간을 지정한다. 만약 시작 시간이 주어지지 않으면 job 은 바로 시작된다.

Value Type dateTime

```
(103) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <end-time>
```

Description 이 job 의 종료 시간을 지정한다. 만약 종료 시간이 주어지지 않으면 job 은 종료되지 않는다.

Value Type dateTime

```
(104) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <interval>
```

Description 이 job 이 수행되는 주기를 지정한다.

Child Elements
```
(105)millisecond
(106)minutely
(107)hourly
(108)daily
```

```
(105) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <interval> <millisecond>
```

Description 주기를 millisecond 단위로 지정한다.

Value Type long

```
(106) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <interval> <minutely>
```

Description 주기를 분 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

```
(107) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <interval> <hourly>
```

Description 주기를 시간 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

```
(108) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <interval> <daily>
```

Description 주기를 날짜 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

```
(109) <jeus-system> <node> <engine-container> <scheduler> <job-list>
<job> <count>
```

Description 이 job 이 수행되는 횟수를 지정한다.

Value Type long

Default Value -1

Defined Value -1
수행되는 횟수를 제한하지 않는다.

A.4 JEUSMain.xml 샘플 파일

<<JEUSMain.xml>>

```
<?xml version="1.0"?>
<jeus-system xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <node>
    <name>johan</name>

    <!-- Node Scheduler Service -->
    <scheduler>
      <enabled>true</enabled>

      <!-- Scheduler Thread-pool settings -->
      <thread-pool>
        <min>2</min>
        <max>30</max>
        <period>3600000</period>
      </thread-pool>

      <!-- Jobs to be executed after booting -->
      <job-list>
        <job>
          <class-name>sample.scheduler.ScheduleJob</class-name>
          <name>My Task</name>
          <description>This task runs in every 30 minutes</description>
          <begin-time>2005-02-01T09:00:00</begin-time>
          <end-time>2005-02-28T23:59:59</end-time>
          <interval>
            <minutely>30</minutely>
          </interval>
          <count>-1</count>
        </job>
      </job-list>

    </scheduler>

    <engine-container>
      <name>container1</name>
```

```
    <!-- others are omitted -->

    <!-- Engine-Container Scheduler Service -->
    <scheduler>
      <enabled>true</enabled>

      <!-- Scheduler Thread-pool settings -->
      <thread-pool>
        <min>2</min>
        <max>30</max>
        <period>3600000</period>
      </thread-pool>

      <!-- Jobs to be executed after booting -->
      <job-list>
        <job>
          <class-name>samples.ScheduleJob2</class-name>
          <name>My Task</name>
          <description>This task runs in every 2 hours</description>
          <begin-time>2005-02-01T09:00:00</begin-time>
          <end-time>2005-02-28T23:59:59</end-time>
          <interval>
            <hourly>2</hourly>
          </interval>
          <count>-1</count>
        </job>
      </job-list>

    </scheduler>
  </engine-container>

  <!-- others are omitted -->

 </node>

</jeus-system>
```

색 인