

Scheduler 안내서

JEUS 9

TMAXSOFT

저작권 공지

Copyright 2024. TmaxSoft Co., Ltd. All Rights Reserved.

제한된 권리

이 소프트웨어(JEUS®) 사용설명서와 프로그램은 저작권법과 국제 조약에 의해 보호됩니다. 사용설명서와 프로그램은 TmaxSoft Co., Ltd.와의 사용권 계약 하에서만 사용할 수 있으며, 사용설명서는 사용권 계약의 범위 내에서만 배포 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부를 TmaxSoft의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단으로 전송, 복제, 배포하거나 2차적 저작물을 작성할 수 없습니다.

이 소프트웨어 사용설명서와 프로그램의 사용권 계약은 어떠한 경우에도 사용설명서 및 프로그램과 관련된 지적 재산권(등록 여부를 불문)을 양도하는 것으로 해석되지 않으며, 브랜드나 로고, 상표 등을 사용할 권한을 부여하지 않습니다. 사용설명서는 오로지 정보 제공만을 목적으로 하며, 이로 인한 계약상의 직접적 또는 간접적 책임을 지지 않습니다. 또한 사용설명서 상의 내용이 법적 또는 상업적인 특정 조건을 만족시킬 것을 보장하지 않습니다. 사용설명서는 제품의 업그레이드나 수정에 따라 예고 없이 변경될 수 있으며, 내용상의 오류가 없음을 보장하지 않습니다.

상표 공지

JEUS®는 TmaxSoft Co., Ltd.의 등록 상표입니다.

Java, Solaris는 Oracle Corporation 및 그 자회사, 관계회사의 등록 상표입니다.

Microsoft, Windows, Windows NT는 Microsoft Corporation의 등록 상표 또는 상표입니다.

HP-UX는 Hewlett Packard Enterprise Company의 등록 상표입니다.

AIX는 International Business Machines Corporation의 등록 상표입니다.

UNIX는 X/Open Company, Ltd.의 등록 상표입니다.

Linux는 Linus Torvalds의 등록 상표입니다.

Noto는 Google Inc.의 상표입니다. Noto 글꼴은 오픈 소스입니다. 모든 Noto 글꼴은 SIL Open Font License, 버전 1.1에 따라 게시됩니다. (<https://www.google.com/get/noto/>)

기타 모든 제품들과 회사 이름은 각각 해당 소유주의 상호, 상표, 또는 등록 상표입니다.

본 사용설명서에 기재된 회사, 시스템, 제품 이름 등에 반드시 상표 표시(™, ®)를 하지는 않습니다.

오픈 소스 소프트웨어 공지

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다.: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

관련 상세 정보는 제품의 다음 디렉터리에 기재된 사항을 참고하시기 바랍니다. :

\${INSTALL_PATH}/license/oss_licenses

기술 지원

홈페이지: <https://www.tmaxsoft.com>

기술 서비스 센터: 1544-8629

안내서 이력

제품 버전	안내서 버전	발행일	비고
JEUS 9	3.1.1	2024-09-27	-

목차

1. 소개	1
1.1. 개요	1
1.2. Scheduler 컴포넌트 구조	1
1.3. Scheduler Server	2
2. Scheduler 프로그래밍	4
2.1. 개요	4
2.2. JEUS Scheduler 클래스	5
2.3. 작업 정의	6
2.3.1. ScheduleListener 인터페이스 구현	6
2.3.2. Schedule 클래스 상속	7
2.3.3. RemoteSchedule 클래스 상속	7
2.4. Scheduler 객체 얻기	8
2.4.1. 로컬 환경에서 Scheduler 객체 얻기	8
2.4.2. 원격 환경에서 Scheduler 객체 얻기	9
2.5. 작업 등록	9
2.5.1. 한 번 수행되는 작업 등록	10
2.5.2. 반복되는 작업 등록	10
2.5.3. Schedule 작업 객체 등록	11
2.6. 작업 제어	12
2.7. Scheduler 사용	12
2.7.1. Standalone 환경에서 사용	12
2.7.2. JEUS 서버에서 사용	14
2.7.3. Jakarta EE 컴포넌트에서 사용	16
2.8. Job-list 사용	17
3. Scheduler 설정	18
3.1. 개요	18
3.2. 서버 스케줄러 설정	18
3.3. Job-list 설정	19
3.4. Thread Pool 설정	21
3.4.1. 공용 Thread Pool	21
3.4.2. 전용 Thread Pool	21
3.5. 클라이언트 컨테이너 설정	22

1. 소개

본 장에서는 JEUS Scheduler의 기능과 사용 가능한 환경, 수행 방식에 대해 설명한다.

1.1. 개요

JEUS Scheduler는 정해진 시간에 수행되거나 반복적으로 수행되는 작업을 scheduling할 수 있는 기능을 제공한다.

Scheduler는 JEUS의 확장된 기능으로 정해진 시간이나 주기적으로 작업을 수행해야 할 때 사용할 수 있다. EJB에서 사용할 수 있는 Timer Service와 같은 기능을 제공하는데 이와 유사한 목적을 지니고 있으며 EJB 환경이 아니더라도 사용할 수 있다. 예를 들어 주기적으로 임시 파일을 삭제한다거나, 주기적으로 데이터베이스 커넥션을 체크하는 등의 시스템 관리에 사용할 수도 있다.

Jakarta EE 환경에서 Timer Service를 이용할 때 Java SE Timer(`java.util.Timer`)를 직접 사용할 수 없으므로 EJB Timer Service를 사용하거나 JEUS에서 제공하는 JEUS Scheduler를 사용해야 한다. EJB Timer Service는 EJB 환경에서만 사용할 수 있는데 반하여 JEUS Scheduler는 모든 Jakarta EE 환경에서 사용할 수 있고 일반 Java SE 애플리케이션에서도 사용이 가능하다.

JEUS Scheduler는 Java SE Timer(`java.util.Timer`)와 유사하기 때문에 Java SE Timer에 익숙하다면 JEUS Scheduler를 쉽게 사용할 수 있다. 또한 Java SE Timer에는 없는 작업 종료 시점(end time)과 최대 수행 횟수(max count)를 지정할 수 있는 기능도 제공한다.

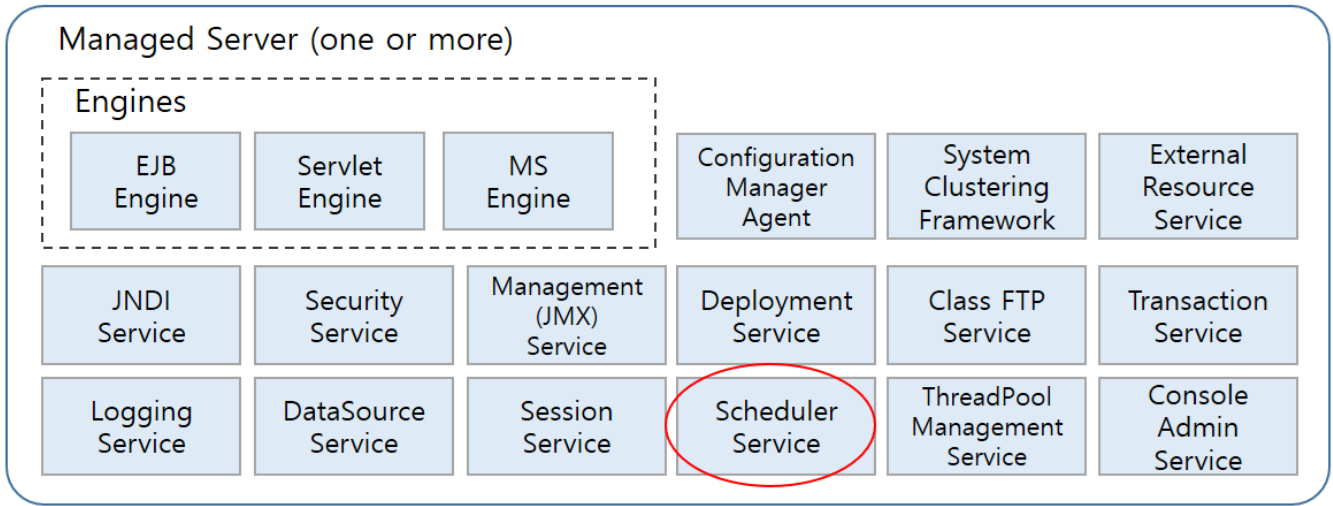
JEUS Scheduler는 다음과 같이 다양한 환경에서 사용이 가능하다.

- Java SE 애플리케이션에서 Standalone Scheduler 사용
- Jakarta EE 애플리케이션 클라이언트에서 Standalone Scheduler 사용
- JEUS 서버 Scheduler Service를 원격에서 접속하여 사용
- JEUS 서버 설정 파일에 Job을 등록해서 JEUS 서버 내에서 사용
- JEUS 서버 Scheduler Service를 Jakarta EE 컴포넌트(Servlet, JSP, EJB 등)에서 사용

1.2. Scheduler 컴포넌트 구조

JEUS Scheduler는 사용자 애플리케이션 내에서 사용할 수도 있고 JEUS 서버에서 사용할 수도 있다. JEUS 서버는 원격에서 접속할 수 있는 Scheduler Service를 구동시킬 수 있으며 설정 파일을 통해 scheduling 될 작업을 등록할 수도 있다.

JEUS Scheduler의 컴포넌트는 다음과 같은 구조를 갖는다.

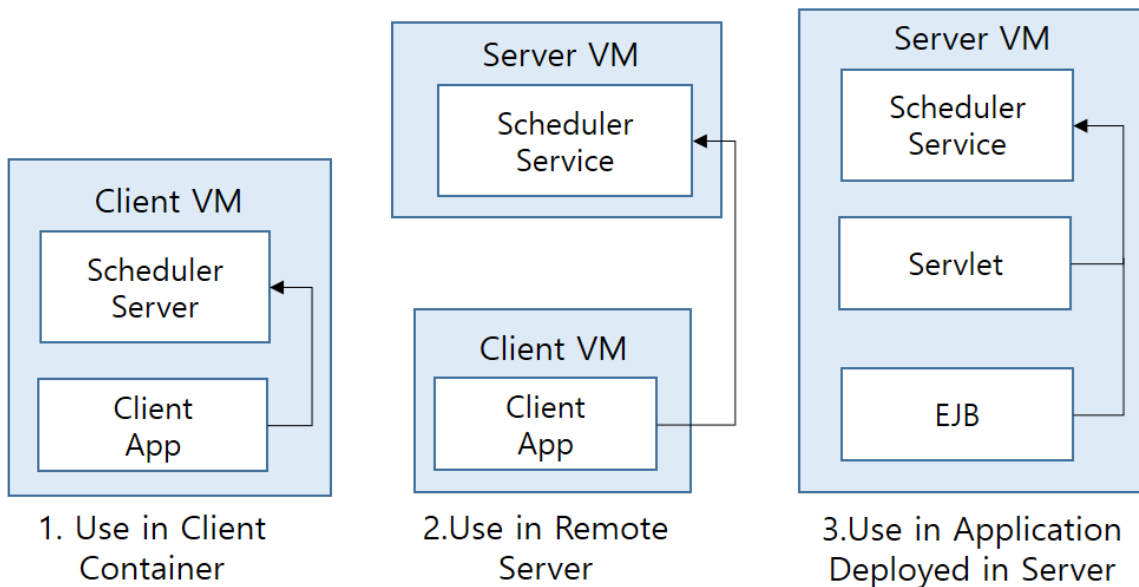


JEUS에서의 Scheduler 컴포넌트

1.3. Scheduler Server

Scheduler Service가 동작하는 서버 형태에 따라 Scheduler의 수행 방식이 달라지고, Scheduler Service가 구분된다.

다음은 Scheduler Server의 종류에 따른 수행 방식을 나타낸다.



Scheduler 서버 형태별 수행 방식

다음은 각각의 Scheduler Service를 사용하는 경우에 대한 설명이다.

- Server Scheduler Service

항상 주기적으로 수행되어야 할 작업을 scheduling할 때 사용한다. 서버에서 항상 수행되어야 할 주기적인 작업이 있다면 Job-list에 미리 등록하여 서버가 구동할 때 수행되도록 할 수도 있다.

Jakarta EE 컴포넌트들(Servlet, JSP, EJB 등)이 주기적으로 수행되는 작업을 scheduling할 때 주로 사용한다. 또한 JNDI 저장소에 등록되어서 원격 클라이언트에서도 사용할 수 있다.

- Client Scheduler Service

애플리케이션 클라이언트에서 Standalone 방식으로 구동되는 JEUS Scheduler Service는 애플리케이션 클라이언트 내부에서 어떠한 작업을 주기적으로 수행하는 경우 주로 사용한다. 또한 애플리케이션 클라이언트가 종료되면 더 이상 수행될 필요가 없는 작업인 경우에 사용한다.

2. Scheduler 프로그래밍

본 장에서는 JEUS Scheduler 프로그래밍에 필요한 기본 지식과 사용 방법에 대해서 설명한다.

2.1. 개요

JEUS 5 이후 버전에서는 이전(4.x)에 사용되던 방식과 비교하여 다음과 같이 몇 가지 사항이 변경 및 추가되었다. 하지만 JEUS 9 Scheduler는 기존 버전과 호환성을 유지하고 있기 때문에 기존에 작성된 프로그램도 별도의 수정 없이 운영이 가능하다.

- SchedulerListener 인터페이스 추가

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 ScheduleListener에서 구현하고 있다. 따라서 Schedule 작업 클래스를 상속해서 작업을 정의하지 않아도 ScheduleListener를 구현해서 작업을 정의할 수 있다.

- SchedulerFactory 클래스 추가

기존에 사용되던 클라이언트와 서버용 SchedulerManager는 더 이상 사용되지 않으며(deprecated), 새롭게 SchedulerFactory 클래스가 추가되었다. SchedulerFactory는 Scheduler 객체를 얻는 데 사용되며 Scheduler 객체를 통해 클라이언트 환경, 서버 환경, 리모트 클라이언트 환경의 구별 없이 작업을 등록할 수 있다.

- 다양한 메소드 추가

Scheduler 인터페이스에는 작업을 등록하는 다양한 메소드들이 추가되었다. 이제 작업을 등록할 때 시작시간, 주기, 종료시간, 최대 수행 횟수를 지정할 수 있다.

- Thread Pool을 이용한 멀티 스레드 방식으로 변경

기존의 JEUS Scheduler는 기본적으로 싱글 스레드 방식으로 작업을 수행했지만 새로운 JEUS Scheduler는 Thread Pool을 이용하여 각 작업을 별도의 스레드로 동작시킨다. 따라서 어떤 한 작업이 수행 중에 블록되더라도 다른 작업 수행에 영향이 없다.

- Job-list 기능 추가

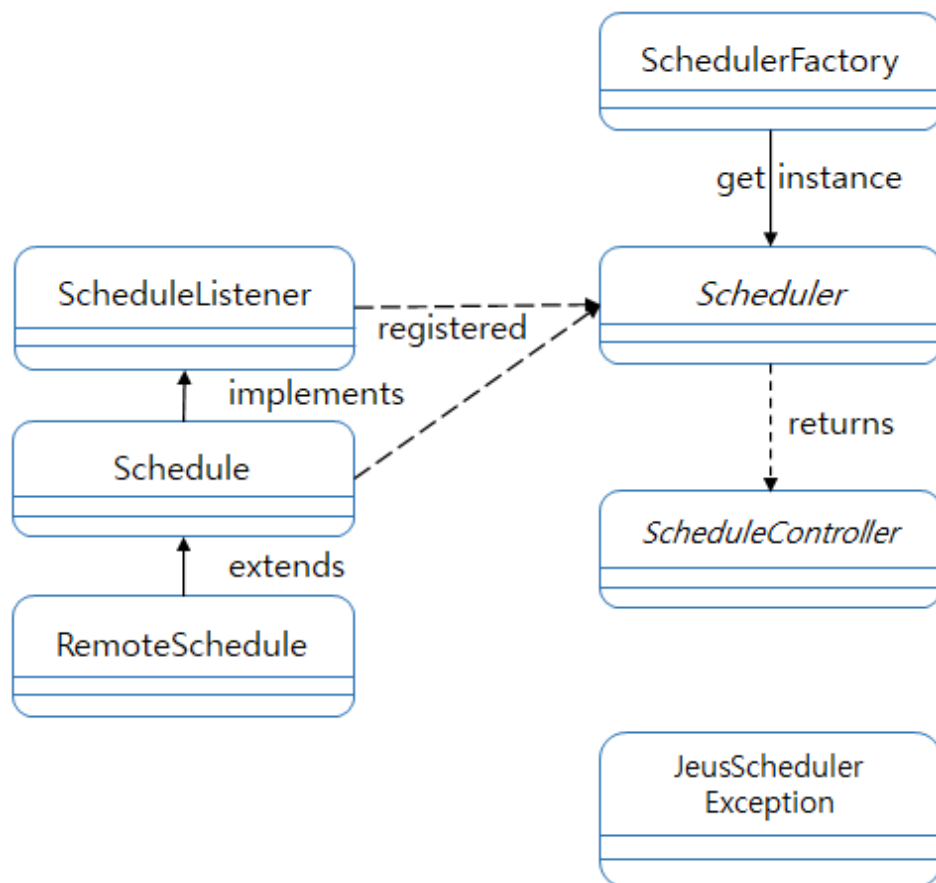
프로그래밍 방식으로 작업을 등록하지 않고 JEUS 서버 설정에 작업을 등록할 수 있는 Job-list 기능이 추가되었다.

JEUS Scheduler 프로그래밍을 이해하기 위해 먼저 JEUS Scheduler를 구성하고 있는 클래스들을 살펴보고, Scheduling 작업을 정의하는 방법, 작업을 등록하는 방법, 등록된 작업을 제어하는 방법에 대해 설명한다. 이후에 JEUS Scheduler를 어떤 경우에 사용할 수 있는지를 알아본다. 본 장에서 설명하는 모든 예제 파일들은 JEUS_HOME/samples/scheduler 디렉터리에서 찾아볼 수 있다.

2.2. JEUS Scheduler 클래스

기본적으로 JEUS Scheduler는 J2SE Timer와 개념적인 부분뿐만 아니라 유사한 인터페이스를 가지고 있다. 작업을 나타내는 `java.util.TimerTask` 클래스는 JEUS Scheduler의 `jeus.schedule.ScheduleListener` 인터페이스와 동일하고, 작업을 등록하는 `java.util.Timer` 클래스는 JEUS Scheduler의 `jeus.schedule.Scheduler` 인터페이스와 동일하다. 따라서 이러한 유사성을 고려하여 JEUS Scheduler를 사용한다면 좀 더 쉽게 익숙해질 수 있을 것이다.

다음은 각 Scheduler API 클래스에 대한 설명이다.



Scheduler API 클래스

API 클래스	설명
package jeus.schedule	JEUS Scheduler의 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. ScheduleListener는 하나의 Callback 메소드인 onTime()을 가지고 있어 해당 시간이 되었을 때 이 메소드가 호출된다.
abstract class Schedule	ScheduleListener를 구현한 추상 클래스로 JEUS 5 이전에 사용하던 작업 클래스이다. 이 클래스는 onTime() 이외에도 nextTime() Callback 메소드가 있어서 작업을 등록할 때 호출될 시간을 예약하는 것이 아니라 작업을 등록한 후에 동적으로 다음 호출될 시간을 결정할 수 있다.
abstract class RemoteSchedule	특수한 Schedule 클래스로 initialize() Callback 메소드를 가지고 있어 객체를 생성할 때 초기화 파라미터 값을 받을 수 있다.
class SchedulerFactory	실제(concrete) Scheduler 객체를 얻어오기 위해 사용된다.

API 클래스	설명
interface Scheduler	JEUS Scheduler에 작업을 등록하기 위한 핵심 인터페이스로 다양한 registerSchedule() 메소드를 정의하고 있다.
interface ScheduleController	Scheduler에 작업을 등록하면 ScheduleController 인터페이스를 구현한 객체를 리턴받는다. 이 핸들 객체는 작업에 대한 정보를 얻거나 작업을 취소할 때 사용된다.
exception JeusSchedulerException	JEUS Scheduler에 작업 등록하거나 취소할 때에 내부적으로 문제가 발생하는 경우 JeusSchedulerException이 발생할 수 있다.



각 인터페이스나 클래스에 대한 자세한 설명은 JEUS Scheduler Javadoc API를 참고한다.

2.3. 작업 정의

작업을 수행하기 위해서는 작업 클래스를 정의하고 작업의 수행시간 및 주기에 따라 알맞은 방법으로 작업을 정의해야 한다.

2.3.1. ScheduleListener 인터페이스 구현

정해진 시간에 수행되어야 할 작업은 ScheduleListener 인터페이스를 구현(implements)하여 클래스로 정의된다.

ScheduleListener는 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;

    // no-arg constructor is required if classname is used for task registration
    public SimpleTask() {
    }

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

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

    ...
}
```

2.3.2. Schedule 클래스 상속

ScheduleListener를 바로 구현하지 않고 Schedule 클래스나 RemoteSchedule 클래스를 상속하여 작업 클래스를 정의할 수 있다. Schedule 클래스나 RemoteSchedule 클래스는 JEUS 5 이전 Scheduler에서 사용되던 작업 클래스이다. JEUS 5와 그 이후부터는 일반적인 작업을 정의할 때 ScheduleListener를 구현하도록 하고 있지만 하위 호환성을 위해 기존 작업 클래스를 그대로 제공하고 있다.

Schedule 추상 클래스는 onTime() 이외에도 nextTime()이라는 Callback 메소드가 있어서 작업을 등록할 때 호출될 시간을 예약하지 않고 작업 클래스 내에서 다음 호출될 시간을 결정하도록 한다. 따라서 고정적인 주기를 갖는 작업보다는 가변적인 주기를 갖는 작업의 경우에 좀 더 효율적으로 사용된다.

JEUS Scheduler는 Schedule 작업 객체의 처음 수행시간을 결정하기 위해 작업 객체를 등록한 후에 먼저 nextTime()을 호출하여 처음 수행시간을 정한다. 그런 다음 해당 시간이 되면 onTime()을 호출하여 작업을 수행하고 onTime()이 종료되면 다시 nextTime()을 호출하여 다음 수행시간을 정하게 된다. nextTime()은 다음에 작업이 수행될 절대시간을 Milli-Second(ms) 값으로 넘겨주어야 한다. 이때 0을 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 프로그래밍하기가 쉽지 않다. 그렇기 때문에 되도록 작업 내에서 nextTime()을 통해 반복 주기를 구현하기 보다는 작업을 등록할 때 반복 주기를 설정하는 것이 좋다.

Schedule object example

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

    // no-arg constructor is required if classname is used for task registration
    public SimpleSchedule() {
    }

    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;
    }

    ...
}
```

2.3.3. RemoteSchedule 클래스 상속

RemoteSchedule 클래스는 원격으로 작업을 등록할 때 초기화 변수를 지정할 수 있는 Schedule 객체이다. 주로 원격에서 클래스 이름을 통해 작업 객체를 등록할 때 사용한다. 이 클래스는 initialize() Callback 메소드를 가지고

있는데 작업 등록 후에 초기화 파라미터로 이 메소드가 한 번 호출된다. 따라서 원격에서 클래스 이름으로 작업을 등록할 때 초기화 값을 설정할 경우 사용할 수 있다.

initialize() Callback은 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;

    // no-arg constructor is required if classname is used for task registration
    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;
    }
    ...
}
```



작업을 Job-list를 사용하여 등록하거나 classname을 사용하는 API를 통해 등록하는 경우에는 컨터이너가 해당 클래스를 초기화하기 때문에 작업 클래스에는 no-arg(default) 생성자(constructor)가 반드시 필요하다.

2.4. Scheduler 객체 얻기

본 절에서는 Scheduler 객체를 얻는 방법에 대해서 알아본다. JEUS Scheduler는 로컬 환경과 원격 환경에서 모두 구동된다.

2.4.1. 로컬 환경에서 Scheduler 객체 얻기

로컬 환경에서 구동된다는 것은 프로그램이 구동되고 있는 로컬 JVM 내에 Scheduler 인스턴스가 생성되며, 등록된

모든 작업이 같은 JVM 내에서 구동된다는 것을 의미한다.

JEUS Scheduler는 JVM 내에서 현재 하나의 인스턴스만 생성되며 이 인스턴스를 Default Scheduler라고 한다. 따라서 현재는 JVM 내에 모든 클라이언트들은 Default Scheduler를 공유하게 된다. 로컬 환경의 JEUS Scheduler는 일반 J2SE 애플리케이션이나 Jakarta EE 애플리케이션 클라이언트, Jakarta EE 컴포넌트 등에서(Servlet, JSP, EJB 등) 사용된다. 로컬 환경의 JEUS Scheduler를 사용하기 위해서 SchedulerFactory를 이용한다.

다음과 같이 간단하게 Default Scheduler 인스턴스를 얻을 수 있다.

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

2.4.2. 원격 환경에서 Scheduler 객체 얻기

원격 환경에서 구동된다는 것은 프로그램이 구동되고 있는 JVM이 아닌 원격의 다른 JVM에서 Scheduler 인스턴스가 생성되고, 등록된 모든 작업이 원격 JVM 내에서 구동된다는 것을 의미한다.

원격 Scheduler는 RMI 객체 형태로 나타나기 때문에 클라이언트는 RMI Call을 통해 JEUS Scheduler를 사용하게 된다. JEUS 환경에서는 JEUS 서버에 원격 Scheduler Service가 기동된다. 원격 환경의 JEUS Scheduler는 원격 클라이언트가 JEUS 서버에 작업을 등록할 때 사용된다.

JEUS 서버에 있는 원격 JEUS Scheduler를 사용하기 위해서 JNDI Lookup을 이용한다.

다음과 같이 JEUS Server Scheduler Instance(Stub)를 얻을 수 있다.

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



JEUS 5 이전에 사용되던 `jeus.schedule.server.SchedulerManager`와 `jeus.schedule.client.SchedulerManager`는 더 이상 사용되지 않는다(deprecated). 대신 `SchedulerFactory`를 통해 Scheduler 객체를 얻어서 사용하기를 권장한다. 하지만 하위 호환성을 유지하기 위해 위 클래스들은 그대로 제공된다.

2.5. 작업 등록

본 절에서는 Scheduler Interface를 이용하여 작업을 등록하는 방법에 대해서 알아본다. 로컬 환경이나 원격 환경에서 Scheduler Instance를 얻어왔다면 작업을 등록하는 방법은 동일하다. 단, 원격 JEUS Scheduler는 작업 객체가 원격으로 전송(serialization)되어 원격으로 운용된다.

2.5.1. 한 번 수행되는 작업 등록

단지 특정 시간에 한 번만 수행되어야 할 작업의 경우에는 하나의 수행시간만 설정하여 작업을 등록할 수 있다. 이때 수행시간은 `java.util.Date` 객체로 절대시간을 설정하거나, `Milli-Second(ms)` 값으로 현재 시간을 기준으로 얼마의 시간이 지난 후에 수행되어야 하는지 설정할 수 있다.

다음의 메소드를 이용하여 작업을 등록한다.

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

다음은 메소드 사용에 대한 예이다. 메소드의 파라미터에 대한 설명은 [반복되는 작업 등록](#)을 참고한다.

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

2.5.2. 반복되는 작업 등록

반복되는 작업의 경우 첫 수행시간, 주기, 종료시간, 최대 수행 횟수 등을 주어 작업을 등록할 수 있다. 반복되는 작업의 특성에 따라 반복 주기를 `Fixed-delay` 방식이나 `Fixed-rate` 방식으로 결정해야 한다.

• Fixed-delay 방식

작업이 수행되는 간격이 일정하게 유지된다. 작업의 다음 수행시간은 이전 수행시간과 주기에 의해서 결정된다. 만약 작업의 수행이 지체(작업 수행시간이 오래 걸리거나 `Garbage Collection`과 같은 외부 이유에 의해서 지체되는 경우)되어 다음 작업이 수행되어야 할 시기가 지난 경우에 다음 작업은 바로 수행되며, 그 이후에 수행되는 작업들은 그만큼 지체된다. 따라서 장기적으로는 작업의 수행시간이 조금씩 뒤쳐질 수 있다.

• Fixed-rate 방식

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 Scheduler는 `Fixed-rate` 방식으로 작업을 등록하면 비교적 정확한 호출시간을 보장해주기 위해 작업 수행이 지체되더라도 시간이 되면 다른 스레드에 의해 작업을 호출한다. 따라서 작업 수행이 지체되는 경우에 같은 작업이 동시에(`concurrently`) 수행된다. 따라서 이러한 경우에는 작업 객체가 `Thread-safe`한지 고려해야 한다.

다음의 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)
```

작업을 등록할 때 사용하는 파라미터들은 다음과 같다.

파라미터	설명
Date firstTime	시작시간으로 처음 수행될 시간을 지정한다.
long delay	시작시간이다. 현재 이후에 처음 수행될 시간을 지정한다. (단위: ms)
long period	반복 수행 주기를 지정한다. (단위: ms)
Date endTime	종료시간이다. 이 시간 이후에는 작업이 더 이상 수행되지 않고 null인 경우에는 종료시간의 제약이 없다.
long maxcount	최대 수행 횟수이다. Scheduler.UNLIMITED인 경우에는 제한이 없다.
boolean isDaemon	원격으로 Schedule을 등록하는 경우에만 의미가 있으며 true 값으로 설정하면 클라이언트와의 연결이 종료되었을 때 작업이 종료된다. 현재 RMI Runtime의 DGC(Distributed Garbage Collection) 정책에 의해 클라이언트가 연결이 종료되었음을 판단하기 때문에 실제로 클라이언트의 연결이 종료되고 15분 정도가 지나야 종료되었음을 탐지하게 되어 Scheduling이 취소된다.
boolean isThreaded	더 이상 사용되지 않는다(deprecated).

다음은 작업을 등록하는 예제이다.

반복되는 작업 등록

```
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);
```

2.5.3. Schedule 작업 객체 등록

Schedule이나 RemoteSchedule 작업 객체를 등록하는 것은 하위 호환성을 유지하기 위해 제공된다.

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

```
registerSchedule(Schedule task, boolean isDaemon)
registerSchedule(String classname, Hashtable params, boolean isDaemon)
```

다음은 작업을 등록하는 예제이다.

Schedule 작업 객체 등록

```
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);
```



Scheduler는 Managed Server(MS)의 기동 단계에서 제공하는 서비스이므로 설정에 오류가 없도록 주의해야 한다. 만약 설정이 잘못되어 Scheduler를 생성할 수 없다면, MS는 에러 메시지를 표시하고 기동 자체에 실패하게 된다.

2.6. 작업 제어

JEUS Scheduler에 작업을 등록하면 핸들(handle)인 ScheduleController 객체를 리턴한다. 이 객체는 등록된 작업 하나당 만들어지는데 등록된 작업을 제어하기 위해 사용한다. 이 핸들을 이용하여 작업에 대한 정보를 얻어오거나 작업을 취소할 수 있다.

다음은 ScheduleController.cancel() 메소드를 호출하여 작업을 취소하는 예제이다.

ScheduleController.cancel 메소드 사용

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

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

2.7. Scheduler 사용

작업 정의, Scheduler 객체 얻기, 작업 등록, 작업 제어까지 완료하였다면 JEUS Scheduler를 사용할 준비가 모두 되었다. 본 절에서는 환경에 따른 Scheduler의 사용 방법에 대해 설명한다.

2.7.1. Standalone 환경에서 사용

일반 J2SE 애플리케이션이나 Jakarta EE 애플리케이션 클라이언트에서 JEUS Scheduler를 사용할 수 있다. 이

경우에는 JEUS 서버와 별개로 JEUS Scheduler를 J2SE Timer와 같이 라이브러리처럼 사용할 수 있다. Scheduler 객체를 얻기 위해서는 SchedulerFactory 클래스를 이용한다. 또한 로컬 환경에서 작업을 등록할 때 daemon flag는 사용되지 않으므로 어떤 값을 넣어도 무방하다.

다음은 Standalone 클라이언트 예제이다.

Standalone 클라이언트에서 Scheduler 사용

```
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 class extends jeus.schedule.Schedule
            // and has 2-seconds delayed scheduling logic in "nextTime" method.
            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);
}
}

```

Jakarta EE 애플리케이션 클라이언트에서 JEUS Scheduler를 사용하는 경우에는 DD(jeus-client-dd.xml)에 JEUS Scheduler의 Thread Pool에 관련된 설정을 할 수 있다.



1. Scheduler의 Thread Pool과 관련된 설정은 JEUS Reference 안내서의 "Thread Management 관련 명령어"를 참고한다.
2. Scheduler를 사용한 코드를 컴파일하거나 기동시키기 위해서는 JEUS 관련 클래스(jeus.jar 등)가 클래스 패스에 지정되어 있어야 한다.

2.7.2. JEUS 서버에서 사용

JEUS 서버에 Scheduler Service가 기동되어 있다면 원격 클라이언트가 이것을 사용할 수 있다. 그 전에 먼저 JEUS 서버에 Scheduler Service가 기동되도록 설정되어 있어야 한다.

JEUS 서버 Scheduler Service는 RMI Scheduler 객체를 JNDI에 등록한다. 따라서 클라이언트에서는 JNDI Lookup을 통해서 원격 Scheduler 객체(실제로는 Stub 객체)를 얻을 수 있다. 이 객체의 JNDI 이름은 "jeus_service/Scheduler"로 Scheduler.SERVER_SCHEDULER_NAME 상수를 사용한다.

일단 Scheduler 객체를 얻으면 작업을 등록하는 방법은 동일하다. 단, 등록된 작업 객체는 전송(Serialization)되어 원격 Scheduler에서 실제로 운용된다. 즉, 원격 JEUS 서버에서 수행된다. 이 경우 등록할 때 daemon flag는 의미가 있으며, daemon flag를 true로 등록하면 원격 클라이언트가 종료되었을 때 원격 작업도 종료된다.



JEUS 서버에 Scheduler Service가 기동되도록 설정하는 방법은 JEUS Reference 안내서의 "Thread Management 관련 명령어"를 참고한다.

다음은 리모트 클라이언트 예제이다.

리모트 클라이언트에서 Scheduler 사용

```

public class RemoteClient {
    public static void main(String args[]) {
        try {
            // Get the remote scheduler

```

```

    InitialContext ic = new InitialContext();
    Scheduler scheduler = (Scheduler)ic.lookup(
        Scheduler.SERVER_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));
    // SimpleRemoteSchedule class extends
    // jeus.schedule.RemoteSchedule
    // and has 3-seconds delayed scheduling logic in "nextTime" method.
    ScheduleController handle3 = scheduler.registerSchedule(
        "samples.scheduler.SimpleRemoteSchedule", params, true);

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

    // Cancel all tasks
    echo("Cancel all tasks registered 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);
}
}

```



위 예제를 실행하려면 해당 작업 클래스 파일을 JAR 파일로 묶어서

DOMAIN_HOME/lib/application 또는 SERVER_HOME/lib/application에 복사하여 위치시켜야 한다. JEUS 서버가 해당 작업을 수행하기 위해서는 해당 클래스를 로딩해야 하기 때문에 JEUS가 이미 기동되어 있는 상태라면 반드시 재기동해야 한다.

2.7.3. Jakarta EE 컴포넌트에서 사용

EJB나 서블릿과 같은 Jakarta EE 컴포넌트에서 JEUS Scheduler를 사용할 수 있다. 이때 JEUS Scheduler는 JEUS 서버에서 기동된다.

EJB 2.1 표준에는 EJB Timer Service를 명시하고 있으며, EJB Timer Service를 제공하고 있다. 따라서 EJB 컴포넌트의 경우 Jakarta EE 표준을 준수하려면 JEUS Scheduler 보다는 EJB Timer Service를 사용할 것을 권장한다. 하지만 EJB 외의 Jakarta EE 컴포넌트에서는 EJB Timer Service를 사용할 수 없으므로 JEUS Scheduler를 사용해야 한다.

Jakarta EE 컴포넌트에서 JEUS Scheduler를 사용하는 것은 Standalone JEUS Scheduler를 사용하는 방식과 동일하다. SchedulerFactory 클래스를 이용하여 서버에서 기동되는 Scheduler 객체를 얻어온 후에 필요한 등록 메소드를 호출하여 작업을 등록한다.



JEUS 서버에서 동작하는 JEUS Scheduler에 대해 Thread Pool을 설정할 수 있다. 설정하는 방법에 대해서는 [Thread Pool 설정](#)을 참고한다.

다음은 EJB에서 JEUS Scheduler 사용 예제이다.

EJB에서 JEUS Scheduler 사용

```
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);
            isStarted = true;
        }
    }

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

```

    }
}

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();
        }
    }
}

```

2.8. Job-list 사용

Job-list는 JEUS 서버에 프로그래밍 방식으로 작업을 등록하지 않고 설정 파일을 이용하여 작업을 등록하는 것이다. Job-list는 JEUS 서버 Scheduler에 등록할 수 있다. Job-list로 작업을 등록하면 작업은 Fixed-rate 방식으로 반복 수행된다.



JEUS 서버에서 동작하는 JEUS Scheduler에 대해 Job-list를 설정할 수 있다. 설정하는 방법에 대해서는 [Job-list 설정](#)을 참고한다.

3. Scheduler 설정

본 장에서는 JEUS 설정 파일이나 Deployment Descriptor(DD)에 JEUS Scheduler를 설정하는 방법에 대해서 설명한다.

3.1. 개요

JEUS Scheduler Service를 사용하기 위해서는 각 서비스별로 필요한 내용을 설정해야 한다.

• JEUS 서버에 Scheduler Service 설정

원격에서 JEUS 서버 Scheduler Service에 접근하거나, Job-list를 이용하여 JEUS 서버에서 주기적인 작업을 수행하려면 Scheduler Service를 활성화시켜야 한다. Scheduler Service를 활성화하려면 domain.xml 설정 파일을 직접 편집하여 Scheduler Service를 사용하겠다는 설정을 해야 한다. 이 설정이 되어 있어야 서버가 부팅할 때 Scheduler Service를 시작한다. 설정하여야 하는 내용을 간단히 요약하면 아래와 같다.

- Scheduler Service를 사용하도록 설정한다.
- Scheduler Service에서 사용할 Thread Pool을 설정한다.
- 서버에서 scheduling되어 수행할 Job-list에 대해서 설정한다.

• 클라이언트 컨테이너에 Scheduler Service 설정

- Deployment Descriptor(DD)에서 위의 내용을 설정한다.

즉, Scheduler를 사용하려면 서비스를 활성화시키는 설정과 Thread Pool 설정 그리고 실제 작업을 수행할 Job-list 설정을 해야 한다.

3.2. 서버 스케줄러 설정

본 절에서는 domain.xml 직접 편집을 통해 실제 서버에 Scheduler를 설정하는 방법에 대해서 설명한다.

원격에서 JEUS 서버 Scheduler Service에 접근하거나 Job-list를 이용하여 JEUS 서버에서 주기적인 작업을 수행하려면 서버에서 Scheduler Service를 활성화해야 한다. 이를 위해 domain.xml에 Scheduler Service 사용 설정을 추가해야 하며, 해당 설정이 되어 있어야 서버가 부팅할 때 Scheduler Service를 시작된다.

다음은 [Schedule 클래스 상속](#)에서 작성한 예제를 domain.xml 설정을 통해 서버에 Job으로 추가하는 방법이다.

JEUS domain 설정

```
...
<server>
  ...
  <scheduler>
    <enabled>true</enabled>
    <pooling>
      <dedicated>
        <min>0</min>
        <max>10</max>
        <keep-alive-time>60000</keep-alive-time>
      </dedicated>
    </pooling>
  </scheduler>
</server>
```

```

        <queue-size>4096</queue-size>
        <stuck-thread-handling>
            <max-stuck-thread-time>3600000</max-stuck-thread-time>
            <action-on-stuck-thread>None</action-on-stuck-thread>
            <stuck-thread-check-period>300000</stuck-thread-check-period>
        </stuck-thread-handling>
    </dedicated>
</pooling>
<job-list>
    <job>
        <name>SimpleTask</name>
        <class-name>java.sample.scheduler.SimpleTask</class-name>
        <begin-time>2024-08-26T10:30:00</begin-time>
        <end-time>2024-08-27T10:30:00</end-time>
        <interval>
            <hourly>1</hourly>
        </interval>
        <count>10</count>
    </job>
</job-list>
</scheduler>
...
</server>
...

```

위 domain.xml 설정 항목 중, <pooling> 항목에 대한 자세한 내용은 JEUS Server 안내서의 "Thread Pool 설정"을 참고한다.

또한 <job-list> 항목에 대한 설명은 아래 [Job-list 설정](#)을 참고한다.



운영 중인 서버에 Scheduler를 추가하거나 변경하는 작업은 동적으로 반영되지 않는다. 설정만 저장해 두었다가 서버가 재기동될 때 반영된다.

서버를 재기동하면 추가한 Job이 수행되면서 서버 로그에 로깅된 결과를 조회할 수 있다.

```

##### waked on Mon Aug 26 10:30:00 KST 2024
##### waked on Mon Aug 26 11:30:00 KST 2024

```

3.3. Job-list 설정

본 절에서는 실제 작업을 수행할 Job을 등록하는 방법에 대해 설명한다.

프로그램 코드에서 작업을 등록하는 것 외에 JEUS 설정 파일에 작업을 등록하면 JEUS 서버가 구동될 때 작업이 자동으로 scheduling된다. Job은 scheduling될 하나의 작업 단위를 의미한다. Job에 해당하는 클래스는 반드시 jeus.schedule.ScheduleListener를 구현해야 하며 해당 클래스와 관련 클래스를 JAR 파일로 묶어 다음의 경로에 위치시켜야 한다.

- 아래 디렉터리는 Master Server와 원격에 위치한 Managed Server(이하 MS) 사이에 동기화되지 않기 때문에 사용자가 Scheduler Service를 사용할 서버 장비마다 수동으로 동기화해야 한다.

DOMAIN_HOME/lib/application

- 아래 디렉터리는 JEUS를 설치했을 때 생성되는 디렉터리가 아니므로 사용자가 직접 생성해서 위치시켜야 한다.

SERVER_HOME/lib/application

Job은 서버에 설정하면 JEUS Scheduler Service에서 작업이 수행된다.

다음은 Job을 설정할 때 작성해야 할 항목에 대한 설명이다.

항목	설명
Name	Job의 이름을 지정한다.
Class Name	Job을 수행하는 클래스의 Fully Qualified Name을 설정한다.
Description	Job의 설명을 입력한다.
Begin Time	작업이 최초로 수행될 시간을 지정한다. 설정하지 않을 경우 JEUS 서버가 기동할 때 시작한다. ◦ 타입: XML dateTime type ◦ 형식: yyyy-mm-ddThh:mm:ss (예: 2024-08-26T10:30:00) 만약 등록한 작업의 Begin Time이 과거이면 주기적으로 현재 시간 이후에 최초의 수행되어야 할 시간에 최초 수행되도록 조정된다.
End Time	작업이 끝날 시간을 지정한다. 설정되지 않을 경우 종료하지 않는다. ◦ 타입: XML dateTime type ◦ 형식: yyyy-mm-ddThh:mm:ss (예: 2024-08-27T10:30:00) End Time이 과거인 경우에 작업은 한 번도 수행되지 않는다.
Interval	Job이 수행되는 주기를 지정한다. 아래 중에서 선택하여 지정할 수 있다. ◦ millisecond ◦ minutely ◦ hourly ◦ daily
Count	작업의 최대 수행 횟수를 지정한다. 설정되지 않거나 -1인 경우 최대 수행 횟수에 제한이 없다.



Job-list 방식으로 등록된 작업은 Fixed-rate 방식으로 반복된다. 따라서 비교적 정확한 시간에 호출되지만 작업 수행시간이 오래 걸릴 때는 작업이 동시에 진행될 수 있기 때문에 작업이 Thread-safe하도록 고려해야 한다.

3.4. Thread Pool 설정

Scheduler Service는 Thread Pool을 설정하여 Scheduler Service를 수행하는 데 필요한 Thread의 개수를 조절한다. Thread Pool은 System Thread Pool을 공유해서 사용하는 **공용 Thread Pool**과 별도의 Thread Pool을 설정하는 **전용 Thread Pool**로 나뉜다.

본 절에서는 콘솔 툴을 사용해서 Thread Pool을 설정하는 방법에 대해서 설명한다. 항목에 대한 자세한 내용은 JEUS Server 안내서의 "Thread Pool 설정"을 참고한다.

3.4.1. 공용 Thread Pool

Scheduler Service에서 공용 Thread Pool을 사용할 경우는 Thread 개수만 미리 할당해 놓으면 된다.

콘솔 툴 사용

다음은 콘솔 툴(jeusadmin)을 통해 Scheduler의 Thread Pool을 설정하는 방법이다.

```
[MASTER]domain1.adminServer>modify-system-thread-pool server1 -service scheduler -r 10
Successfully performed the MODIFY operation for The scheduler thread pool of the server (server1).,
but all changes were non-dynamic. They will be applied after restarting.
Check the results using "show-system-thread-pool server1 -service scheduler or modify-system-thread-
pool server1 -service scheduler".
```

```
[MASTER]domain1.adminServer>show-system-thread-pool server1 -service scheduler
Shows the current configuration.
the system thread pool of the server (server1)
```

```
=====
+-----+-----+
| Min                | 0                |
| Max                | 100              |
| Keep-Alive Time    | 300000           |
| Queue Size         | 4096             |
| Max Stuck Thread Time | 3600000          |
| Action On Stuck Thread | IGNORE_AND_REPLACE |
| Stuck Thread Check Period | 300000          |
| Reserved Threads for the Service transaction | 0                |
| Reserved Threads for the Service scheduler   | 10               |
| Reserved Threads for the Service namingserver | 0                |
+-----+-----+
=====
```

3.4.2. 전용 Thread Pool

Scheduler Service에서 전용 Thread Pool을 사용할 경우 콘솔 툴(jeusadmin)을 통해 설정할 수 있다.



Scheduler Service의 Thread Pool 설정을 수정하는 것은 동적 반영 가능하기 때문에 서버를 재기동하지 않아도 된다. 하지만 공용 Thread Pool을 사용하다가 전용 Thread Pool을 사용하도록 Thread Pool의 타입을 변경하는 경우 동적 반영되지 않기 때문에 서버를

재기동해야 한다.

콘솔 툴 사용

다음은 콘솔 툴(jeusadmin)을 통해 Scheduler의 Thread Pool을 설정하는 방법이다.

```
[MASTER]domain1.adminServer>show-service-thread-pool server1 -service scheduler
Shows the current configuration.
=====
+-----+-----+
(No data available)
=====

[MASTER]domain1.adminServer>modify-service-thread-pool server1 -service scheduler -min 0 -max 20
Successfully performed the MODIFY operation for The scheduler thread pool of the server (server1).,
but all changes were non-dynamic. They will be applied after restarting.
Check the results using "show-service-thread-pool server1 -service scheduler or modify-service-
thread-pool server1 -service scheduler".

[MASTER]domain1.adminServer>show-service-thread-pool server1 -service scheduler
Shows the current configuration.
=====
+-----+-----+
| Min                | 0      |
| Max                | 20     |
| Keep-Alive Time    | 60000  |
| Queue Size         | 4096   |
| Max Stuck Thread Time | 3600000 |
| Action On Stuck Thread | NONE   |
| Stuck Thread Check Period | 300000 |
+-----+-----+
```

3.5. 클라이언트 컨테이너 설정

Jakarta EE 애플리케이션을 사용하는 경우 클라이언트 컨테이너에서 구동되는 JEUS Scheduler에 대해 설정한다. 애플리케이션 클라이언트를 위한 JEUS DD인 jeus-client-dd.xml 파일에 다음과 같이 **<scheduler>** 설정을 추가한다.

클라이언트 컨테이너 설정 : <jeus-client-dd.xml>

```
<?xml version="1.0"?>
<jeus-client-dd>
  <module-info>
    ...
  </module-info>
  ...
  <scheduler>
    <enabled>true</enabled>
    <!-- Scheduler Thread-pool settings -->
    <pooling>
      <dedicated>
```

```

        <min>2</min>
        <max>10</max>
        <keep-alive-time>60000</keep-alive-time>
        <queue-size>4096</queue-size>
        <stuck-thread-handler>
            <max-stuck-thread-time>3600000</max-stuck-thread-time>
            <action-on-stuck-thread>None</action-on-stuck-thread>
        </stuck-thread-handler>
    </dedicated>
</pooling>
</scheduler>
...
</jeus-client-dd>

```

위의 설정 파일을 수정한 후에 JEUS 서버를 재기동할 필요는 없지만, 클라이언트 모듈과 클라이언트 컨테이너는 재기동해야 수정한 설정 내용이 반영된다.



1. 클라이언트 컨테이너에서 Scheduler Service를 사용할 때는 공용 Thread Pool은 사용할 수 없다.
2. 클라이언트 컨테이너와 애플리케이션 클라이언트에 대해서는 "JEUS Application Client 안내서"를 참고한다.

다음은 클라이언트 컨테이너에서 수행할 Job 설정 예제이다.

Job-list 설정 : <jeus-client-dd.xml>

```

<scheduler>
    ...
    <job-list>
        <job>
            <class-name>samples.ScheduleJob</class-name>
            <name>ScheduleJob</name>
            <description>This is a sample for scheduler service</description>
            <begin-time>2024-08-26T00:00:00</begin-time>
            <end-time>2024-08-27T00:00:00</end-time>
            <interval>
                <minutely>30</minutely>
            </interval>
            <count>-1</count>
        </job>
    </job-list>
</scheduler>

```