Task Execution & Scheduling Kabiliyetleri

harezmi bilişim çözümleri

Task Execution ve Scheduling

- Spring, kullanıcı etkileşimi olmadan arka planda ve periyodik çalışacak işler için **asenkron task execution** ve **task scheduling** kabiliyetleri sunmaktadır
- **TaskExecutor** ve **TaskScheduler** temel arayüzlerdir
- Bu arayüzler sayesinde Java SE 5, Java SE 6 ve Java EE ortamlarındaki **farklı kullanım biçimleri** ortaklanmış ve uygulamadan izole edilmiş olmaktadır

Task Execution

- **TaskExecutor**, Java SE 5'de **thread pool kavramı**na karşılık gelmektedir
- Tam olarak **java.util.concurrent.Executor** arayüzüne karşılık gelir
- Avantajı **Java 5'e ihtiyaç duymadan** thread pool ile task execution gerçekleştirebilmektir

TaskExecutor Tipleri

- Farklı **built-in TaskExecutor gerçekleştirimleri** mevcuttur
- Sıfırdan yeni bir TaskExecutor yazmak kolay kolay söz konusu olmaz
- Yaygın kullanılan **TaskExecutor tipleri**
- **SyncTaskExecutor**
  - Asenkron değildir
  - Task'ları caller thread içinde çalıştırır
  - Testler için uygundur

TaskExecutor Tipleri

- **SimpleAsyncTaskExecutor**
  - Asenkrondur
  - Ancak thread pool özelliği yoktur
  - Her task execution'ı için yeni bir thread oluşturulur
- **ThreadPoolTaskExecutor**
  - Java5 ortamına özeldir
  - Arka tarafta java.util.concurrent.ThreadPoolExecutor'ı kullanır

TaskExecutor Tipleri

- **ConcurrentTaskExecutor**
  - java.util.concurrent.TaskExecutor nesnesini **Spring bean olarak** expose etmeyi sağlar
  - java.util.concurrent.ScheduledThreadPoolExecutor gibi bir executor'a ihtiyaç varsa bunun vasıtası ile kullanılabilir

TaskExecutor Tanımı

- **TaskExecutor**, Spring container içerisinde bir **bean olarak tanımlanır** ve ayarlanır ve **kullanılacağı yere enjekte edilir**

```
<bean id="taskExecutor"
class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">
    <property name="corePoolSize" value="5" />
    <property name="maxPoolSize" value="10" />
    <property name="queueCapacity" value="25" />
</bean>
```

TaskExecutor Kullanımı

- TaskExecutor, **java.lang.Runnable** arayüzünü implement eden **task nesneleri**ni kabul eder

```
taskExecutor.execute(new Runnable() {

    @Override
    public void run() {
        System.out.println("my task...");
    }

});
```

Task Scheduling

- Runnable task'ların hemen o anda değil de **gelecekte bir zamanda veya periyodik olarak** çalıştırılmaları da gerekebilir
- Spring bunun için de **TaskScheduler** arayüzünü sunmaktadır
- **TaskScheduler,** bir **Runnable** task ve **Date** nesnesi kabul eder. Böylece task'ı belirtilen tarihte bir kereliğine çalıştırmak mümkün olur

```
taskScheduler.schedule(new Runnable() {

     @Override
     public void run() {
          System.out.println("my task!");
     }
}, new Date());
```

Trigger & TriggerContext Arayüzleri

- **Trigger** arayüzü ile task'ların çok daha farklı zaman ve durumda, tekrarlanarak çalıştırılmaları da sağlanabilir
- **JSR-236**'dan esinlenerek oluşturulmuştur
- Task'ların bir sonraki çalışma zamanını **farklı parametrelerle belirleme**ye olanak sağlar
- Hatta bu parametrelerden birisi **bir önceki execution'ın sonucu** da olabilir

```
public interface Trigger {

    Date nextExecutionTime(TriggerContext triggerContext);

}
```

Trigger & TriggerContext Arayüzleri

- Bir sonraki task execution **zamanını tespit edebilmek için** her türlü veriyi sunar
- Trigger içerisinde bu veriler kullanılarak bir sonraki task execution zamanı tespit edilir
- Default gerçekleştirim **SimpleTriggerContext** sınıfıdır

```
public interface TriggerContext {

    Date lastScheduledExecutionTime();

    Date lastActualExecutionTime();

    Date lastCompletionTime();

}
```

Trigger Tipleri

- Spring'in sunduğu iki farklı Trigger gerçekleştirimi vardır
- **CronTrigger**
  - Taskların zamanlaması **cron ifadeleri** şeklinde belirtilebilir
  - ```
    scheduler.schedule(task, new CronTrigger("* 15 9-17 * * MON-FRI"));
    ```
- **PeriodicTrigger**
  - Taskların zamanlaması **belirli bir periyot içinde tekrarlanma**sı şeklinde olur
    ```

TaskScheduler Tanımı ve Kullanımı

- Çoğunlukla **ThreadPoolTaskScheduler** sınıfından bir taskScheduler bean tanımı yapılır
- ThreadPoolTaskScheduler sınıfı **TaskScheduler** arayüzünün yanında **TaskExecutor** arayüzünü de implement etmektedir
- Bu sayede **hem asenkron task execution**, **hem de task scheduling** yapılabilmektedir

TaskScheduler Tanımı ve Kullanımı

```
<bean id="taskScheduler"
class="org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler">

    <property name="poolSize" value="5" />

</bean>
```

```
taskScheduler.schedule(
    new Runnable() {

        @Override
        public void run() {
            System.out.println("my scheduled task");
        }
    },
    new CronTrigger("* 15 9-17 * * MON-FRI"));
```

Task Namespace Desteği

- Spring task namespace'i scheduler ve executor bean'lerini tanımlamayı kolaylaştırır
- Böylece **ThreadPoolTaskScheduler** tipinde bir **scheduler** ve **ThreadPoolTaskExecutor** tipinde **executor** bean'lerini tanımlama ve ayarlama işlerini ortadan kaldırır
- Bunun yanında **scheduled task**'ları da Spring container içerisinden tanımlayıp yönetmek mümkün hale gelir

Task Namespace Desteği

```
<task:executor
        id="taskExecutor"
        pool-size="5-25"
        queue-capacity="100"
        rejection-policy="CALLER_RUNS"/>


<task:scheduler id="taskScheduler" pool-size="10"/>


<task:scheduled-tasks scheduler="taskScheduler">
    <task:scheduled ref="beanA" method="methodA" fixed-delay="5000"
initial-delay="1000"/>
    <task:scheduled ref="beanB" method="methodB" fixed-rate="5000"/>
    <task:scheduled ref="beanC" method="methodC" cron="*/5 * * * *
MON-FRI"/>
</task:scheduled-tasks>
```

@Scheduled & @Async Annotasyonları

- XML veya Java based konfigürasyon ile **@Async** ve **@Scheduled** annotasyonlarına sahip metotlar asenkron ve zamanlanmış biçimde çalıştırılabilir

```
<beans...>
    <task:executor id="taskExecutor"
        pool-size="5"/>

    <task:scheduler id="taskScheduler"
        pool-size="10" />

    <task:annotation-driven
        executor="taskExecutor"
        scheduler="taskScheduler" />
</beans>
```

```
@Configuration
@EnableAsync
@EnableScheduling
public class AppConfig {

}
```

@Scheduled & @Async Annotasyonları

```
@Scheduled(cron="*/5 * * * * MON-FRI")
public void doSomething() {
    //sadece hafta ici calisan bir task
}
```

```
@Scheduled(initialDelay=1000, fixedRate=5000)
public void doSomething() {
    //periyodik calisan bir task
}
```

Explicit biçimde çalıştırılmadıklarından input argüman alamazlar ve herhangi bir değer de dönemezler

Crob veya fixed rate trigger'lar tanımlanabilir

@Scheduled & @Async Annotasyonları

```
@Async
void doSomething(String s) {
    //asenkron calistirilacak bir task
}
```

Scheduled metotların aksine input argüman alabilirler, çünkü explicit biçimde çalıştırılacaklardır

```
@Async
Future<String> doSomething(int i) {
    //asenkron calistirilacak ve deger donen bir task
    return new AsyncResult<String>("test");
}
```

Asenkron tasklar değer de dönebilirler, ancak bu değer Future nesnesi üzerinden erişilebilir

```
@Async("myTaskExecutor")
void doSomething(String s) {
    //asenkron calistirilacak bir task
}
```

Default taskExecutor dışında başka bir TaskExecutor kullanılması da mümkündür

AsyncConfigurer ve Java Tabanlı Konfigürasyon

Default olarak Spring Executor interface'inde bir bean tanımı arar, yoksa taskExecutor isimli bir bean tanımına bakar, o da yoksa SimpleAsyncTaskExecutor'dan bir bean oluşturur

Yukarıdaki süreci özelleştirmek için AsyncConfigurer arayüzü implement edilerek custom Executor ve ExceptionHandler tanımlanabilir

```
@Configuration
@EnableAsync
public class AppConfig implements AsyncConfigurer{

    @Override
    public Executor getAsyncExecutor() {
        ThreadPoolTaskScheduler bean = new ThreadPoolTaskScheduler();
        bean.setPoolSize(10);
        bean.setDaemon(false);
        return bean;
    }

    @Override
    public AsyncUncaughtExceptionHandler getAsyncUncaughtExceptionHandler() {
        return new SimpleAsyncUncaughtExceptionHandler();
    }
}
```

Future dönmeyen async metot'larda meydana gelen hatalarda devreye girer

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com