Spring Framework
Module 9 – Task Execution and Scheduling

Vyacheslav Yakovenko
Last update: March, 2012

Contents

- Spring task and scheduling API;
- Quartz;

Spring :: Task Schedulers

Their purpose is to run tasks at some point in future or once after the specified time.

- There are three core API:
  - Java's Timer is scheduled to run once after the specified time;
  - API Spring Framework:
    - packages:
      - org.springframework.scheduling
      - org.springframework.core.task
  - Quartz scheduler:
    - http://www.quartz-scheduler.org/
    - can be scheduled to run once after the specified time as well;
    - can be scheduled to run at some point in future;

Spring :: Task :: TaskExecutor

- In Spring Framework, an abstraction for task scheduling is based in **`TaskExecutor`** interface.
- The only method provided by **`TaskExecutor`** is `void execute(Runnable task,` which allows to pass a task that implements `Runnable` interface.
- **`TaskExecutor`** is identical to the `java.util.concurrent.Executor` interface. Its primary reason for existence is to abstract away the need for Java 1.4.

Spring :: Task :: TaskExecutor

Implementations of `TaskExecutor`:

- `SimpleAsyncTaskExecutor`: does not reuse any threads, rather it starts up a new thread;
- `SyncTaskExecutor` : each invocation takes place in the calling thread, does not execute invocations asynchronously;
- `ConcurrentTaskExecutor`: a wrapper for `java.util.concurrent.Executor` ;
- `SimpleThreadPoolTaskExecutor`: a subclass of Quartz's `SimpleThreadPool` ;
- `ThreadPoolTaskExecutor`;
- `TimerTaskExecutor`;
- `WorkManagerTaskExecutor`: uses CommonJ WorkManager

Spring :: Task :: TaskScheduler

In addition, Spring 3 introduces TaskScheduler interface:

```
public interface TaskScheduler {
    ScheduledFuture schedule(Runnable task, Trigger trigger);
    ScheduledFuture schedule(Runnable task, Date startTime);
    ScheduledFuture scheduleAtFixedRate(Runnable task, Date
startTime, long period);
    ScheduledFuture scheduleAtFixedRate(Runnable task, long
period);
    ScheduledFuture scheduleWithFixedDelay(Runnable task, Date
startTime, long delay);
    ScheduledFuture scheduleWithFixedDelay(Runnable task, long
delay);
}
```

Spring :: Task :: TaskScheduler

- The primary benefit of this interface is that it doesn't have to be coupled to a particular scheduler implementation.
- This is particularly important when running within application server environment where threads should not be directly created by application.
- For such cases, Spring provides `TimerManagerTaskScheduler` that delegates task execution to CommonJ TimerManager, typically configured with JNDI.

Spring :: Task :: Trigger

- Another interface implemented in Spring 3.
- The basic idea of this interface is that task execution may be determined based on past execution outcomes, that is, should be context-aware.

```
public interface Trigger {
    Date nextExecutionTime(TriggerContext
  triggerContext);
}
public interface TriggerContext {
    Date lastScheduledExecutionTime();
    Date lastActualExecutionTime();
    Date lastCompletionTime();
}
```

Spring :: Task :: Trigger

Example:

```
scheduler.schedule(task,
  new CronTrigger("* 15 9-17 * * MON-FRI"));
```

- Execute:
  - Every 15 minutes;
  - From 9 to 17;
  - From Monday to Friday;

Spring :: Task :: Namespace

- Spring 3 introduces namespace, a task that allows to initialize certain beans in application context:

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

- and to turn on auto detection of components annotated with `@Scheduled`:

```
<task:annotation-driven … />
```

Spring :: Task :: Example

Task specification in application context:

```
<task:scheduled-tasks scheduler="myScheduler">
    <task:scheduled ref="someObject"
method="someMethod" fixed-rate="5000"/>
    <task:scheduled ref="anotherObject"
method="anotherMethod" cron="*/5 * * * * MON-FRI"/>
</task:scheduled-tasks>

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

Spring :: Task :: Example

Specifying tasks with `@Scheduled` annotation:

```
@Scheduled(fixedDelay=5000)
public void doSomething() {
    // something that should execute periodically
}

@Scheduled(fixedDelay=5000)
public void doSomething() {
    // something that should execute periodically
}
```

** Methods to be declared in `@Service` component*

Spring :: Task :: Example

Specifying tasks with `@Scheduled` annotation:

```
@Scheduled(cron="*/5 * * * * MON-FRI")
public void doSomething() {
    // something that should execute on weekdays only
}
```

Spring :: Task :: Quartz

Quartz is a commonly used external library used for tasks handling. It uses `JobDetail`interface when specifying the task.

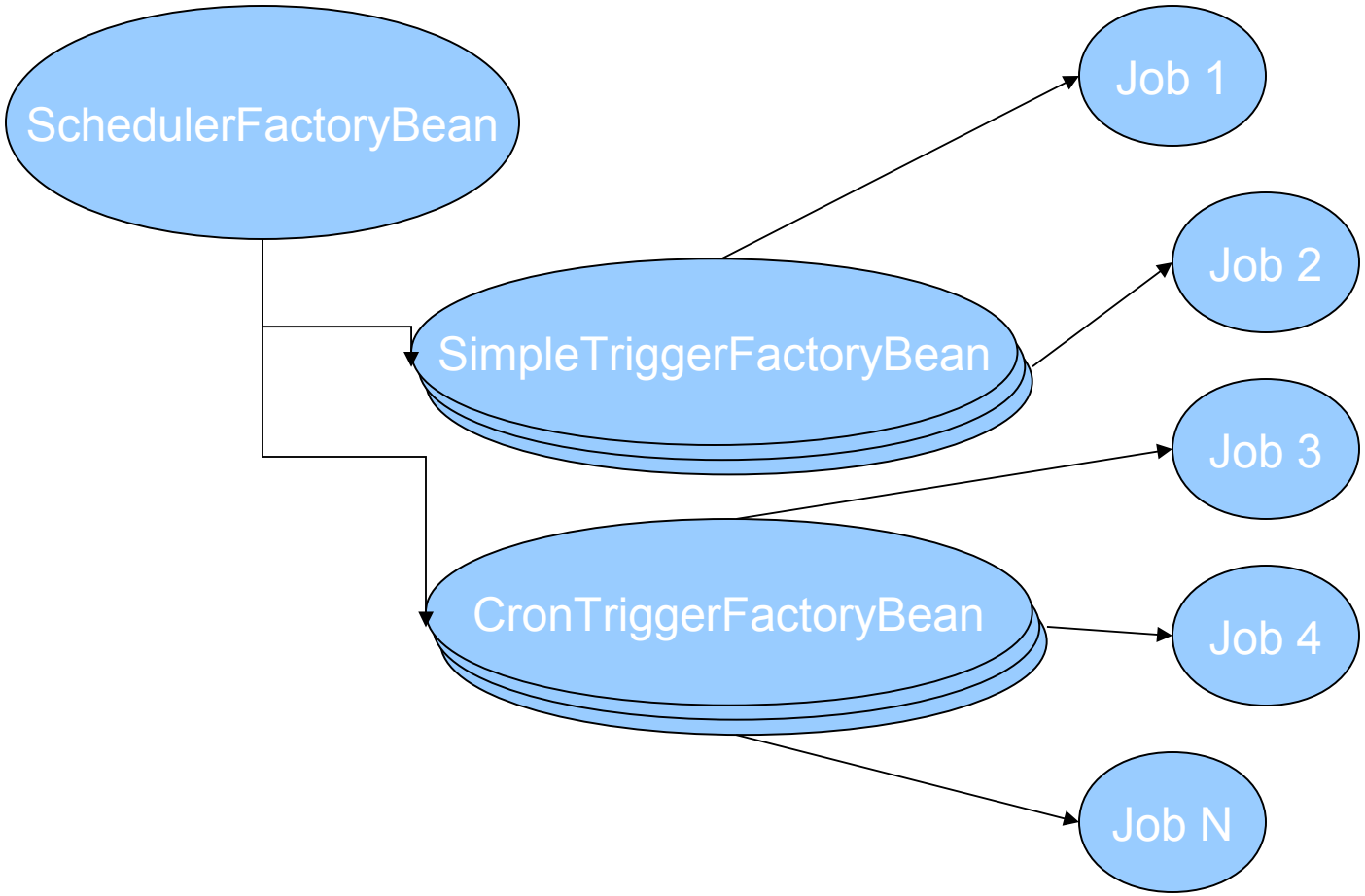
For such cases, Spring 3.1 provides `JobDetailFactoryBean` that supports both Quartz v.1 and Quartz v.2:

```
<bean id="reportJob"
    class="org.springframework.scheduling.quartz.JobDetailFactoryBean">
    <property name="jobClass" value="lab.schedule.PrintingJob" />
</bean>
```

Spring :: Task :: Quartz

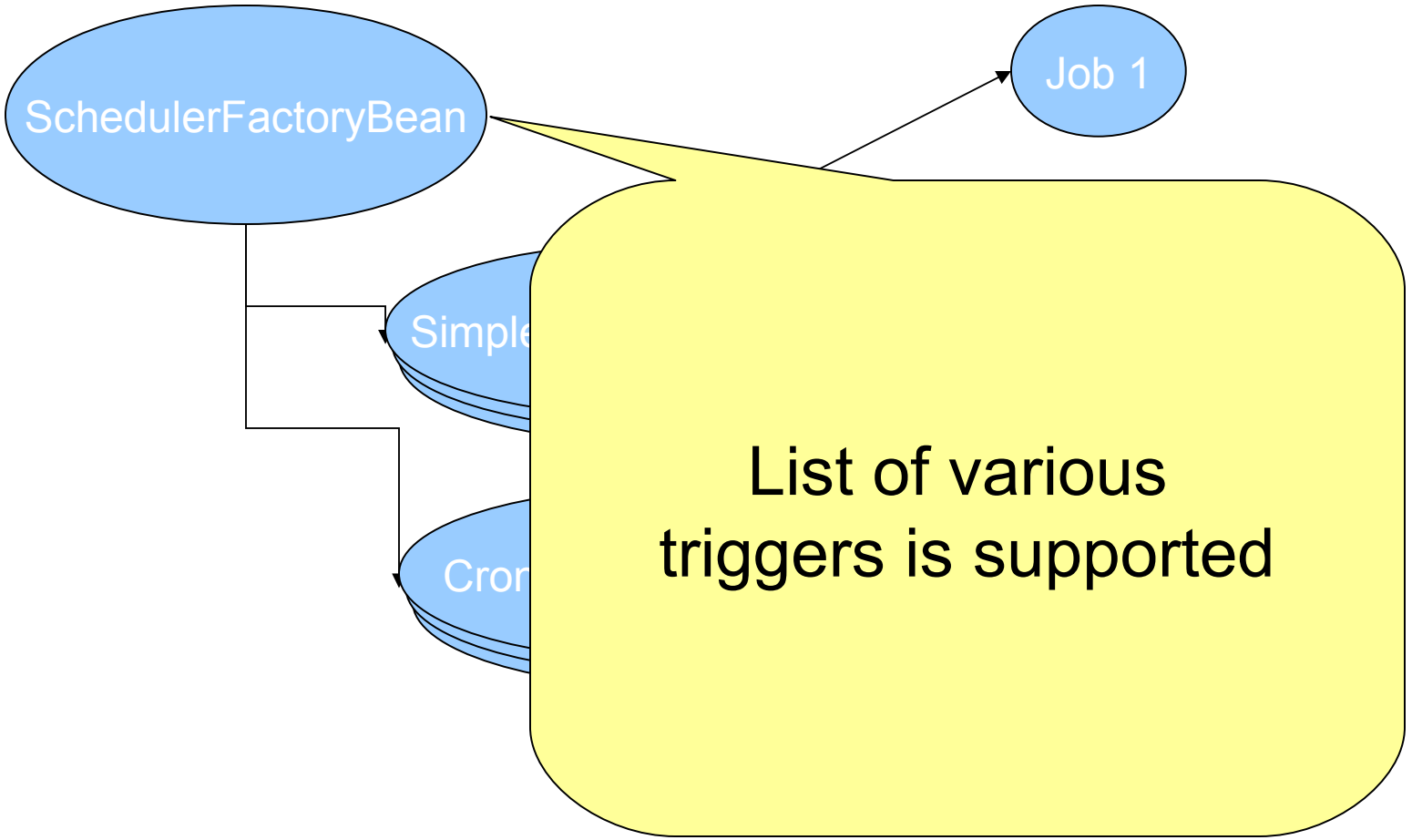
LUXOFT TRAINING

General bean interaction diagram:



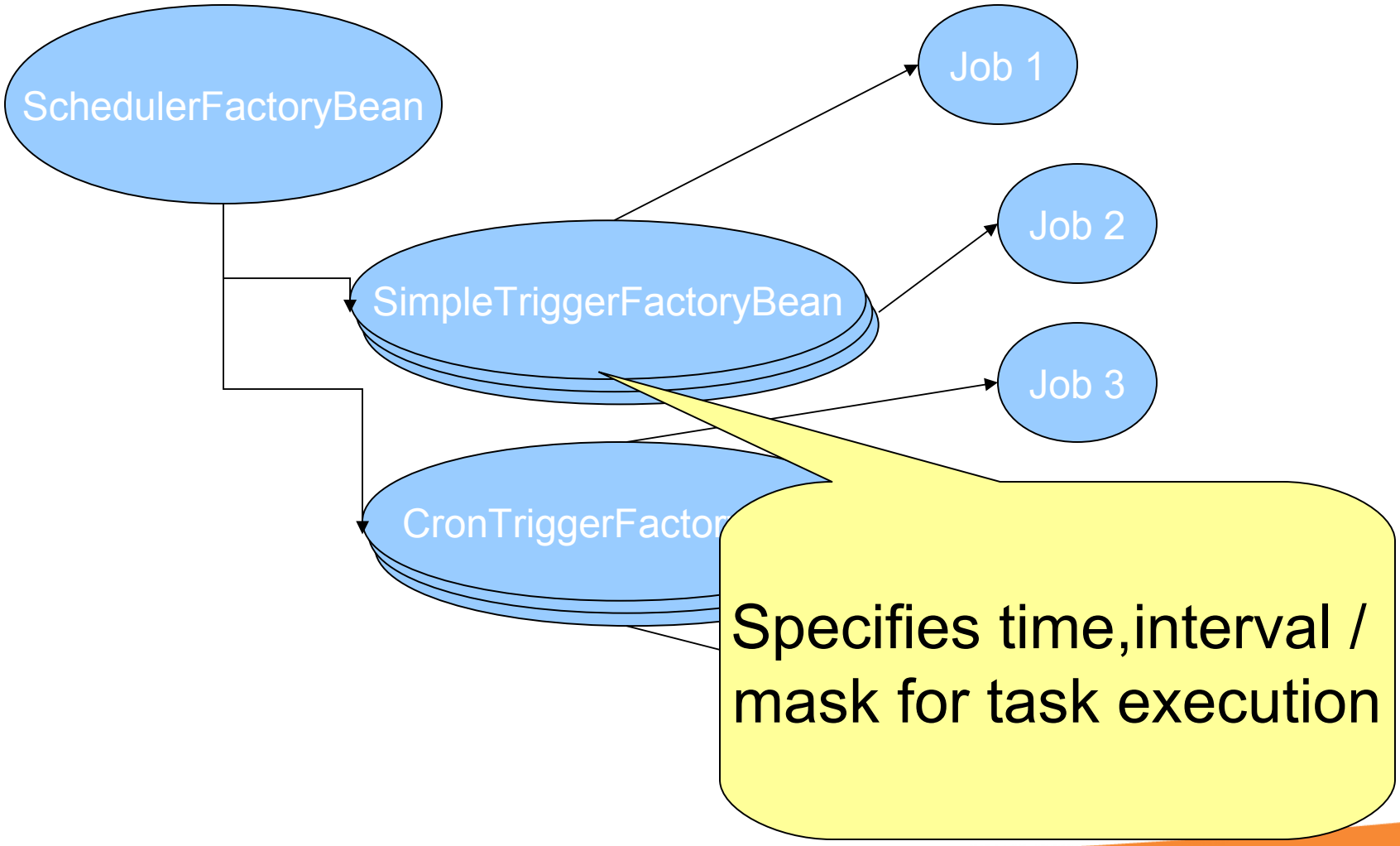
Spring :: Task :: Quartz

Bean interaction diagram:



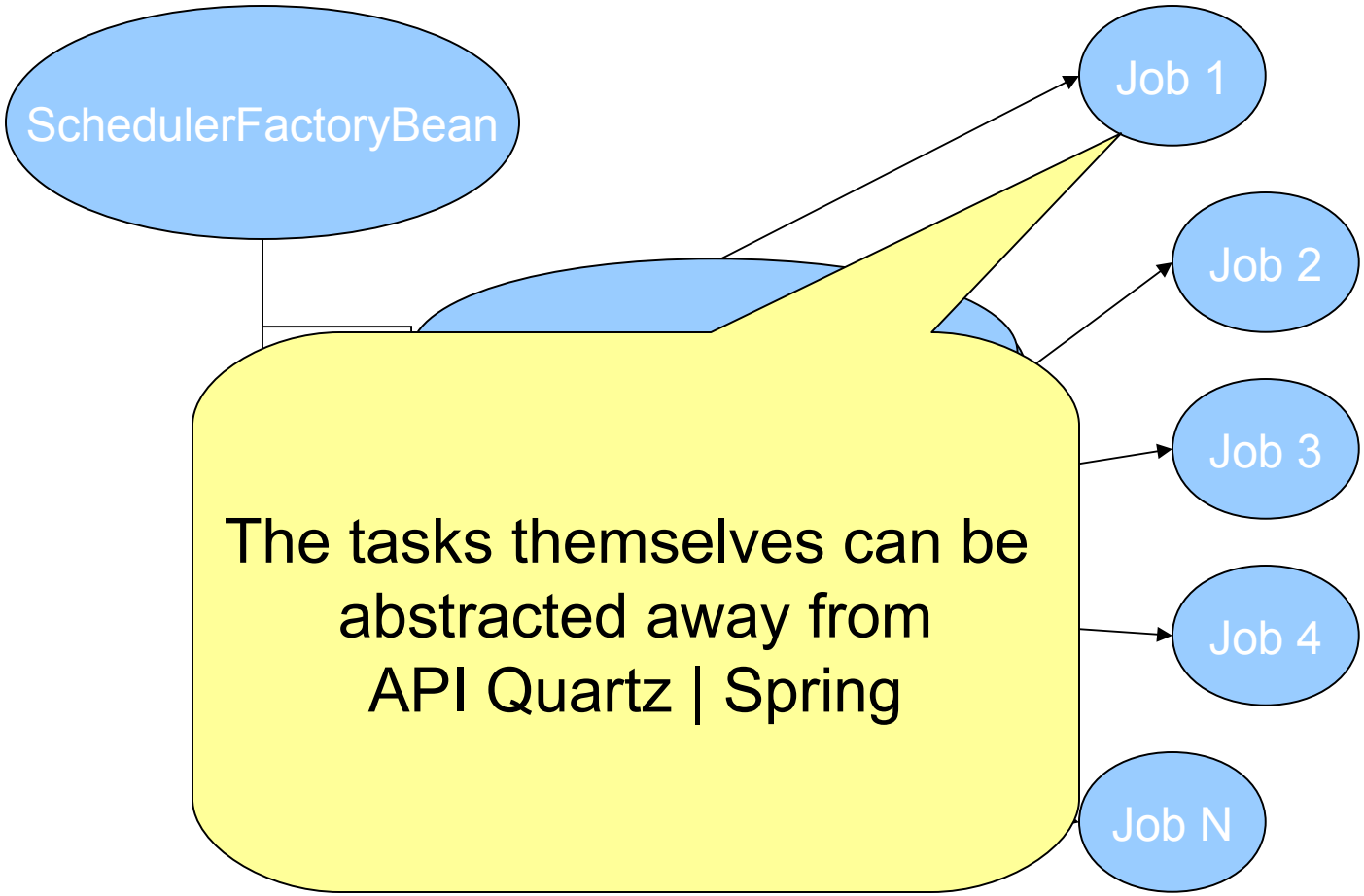
Spring :: Task :: Quartz

Bean interaction diagram :



Spring :: Task :: Quartz

Bean interaction diagram :



Spring :: Task :: Example

One task

```
<bean id="schedulerFactoryBean"
    class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <property name="triggers">
            <list>
                <ref bean="reportTrigger" />
            </list>
        </property>
</bean>

<bean id="reportTrigger"
class="org.springframework.scheduling.quartz.SimpleTriggerFactoryBean">
    <property name="jobDetail" ref="reportJob" />
    <property name="repeatInterval" value="1000" />
    <property name="startDelay" value="5000" />
</bean>
```

Exercises

- №: 10 : Using task scheduling.
  - 30 min for practice;

Any questions!?