ActorScheduler By Quartz

.

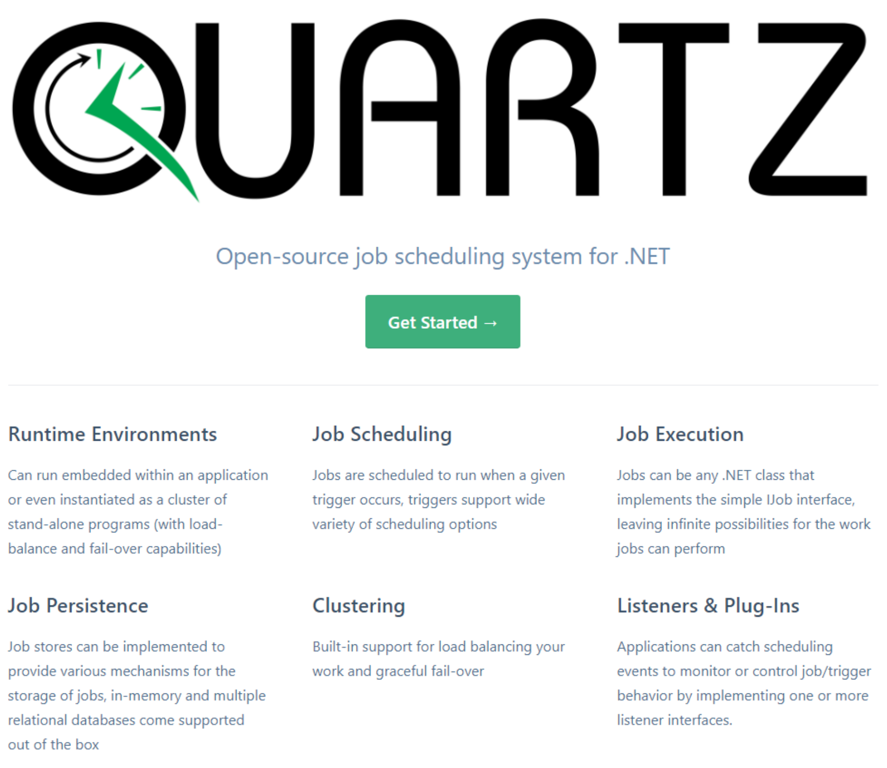
```
 ActorSystem
.Scheduler
.ScheduleTellRepeatedly(
     TimeSpan.FromMinutes(30),   //30
     TimeSpan.FromHours(12),     //12 Full
     indexActor, new ActorCmd()
     {
         CmdType = ActorCmdType.FullIndexing
     }, ActorRefs.NoSender
 );
```

URL : https://getakka.net/articles/utilities/scheduler.html

.

Cron , .

Quartz.NET



QUARTZ

Open-source job scheduling system for .NET

Get Started →

Runtime Environments

Can run embedded within an application or even instantiated as a cluster of stand-alone programs (with load-balance and fail-over capabilities)

Job Scheduling

Jobs are scheduled to run when a given trigger occurs, triggers support wide variety of scheduling options

Job Execution

Jobs can be any .NET class that implements the simple IJob interface, leaving infinite possibilities for the work jobs can perform

Job Persistence

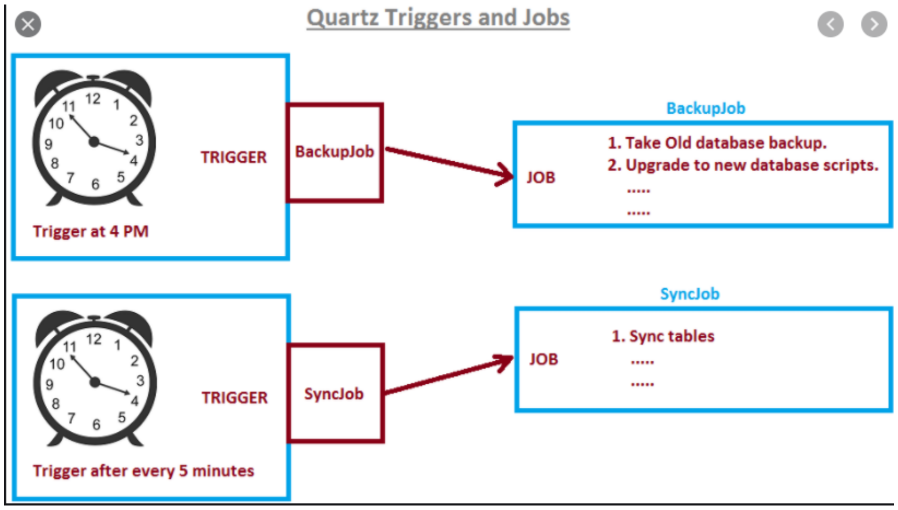
Job stores can be implemented to provide various mechanisms for the storage of jobs, in-memory and multiple relational databases come supported out of the box

Clustering

Built-in support for load balancing your work and graceful fail-over

Listeners & Plug-Ins

Applications can catch scheduling events to monitor or control job/trigger behavior by implementing one or more listener interfaces.



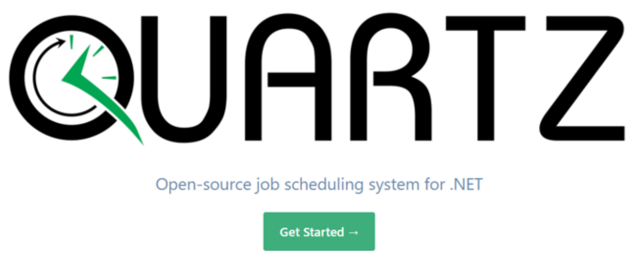
JAVA Spring Boot Schedule .

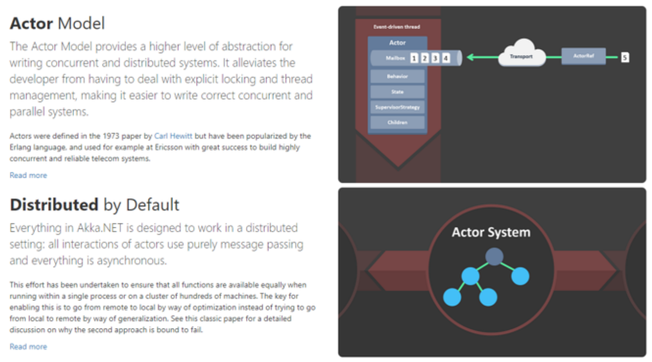
Persitence,Cluster

Quartz .

:https://www.quartz-scheduler.net/documentation/

Quartz.Actor





Actor Model

The Actor Model provides a higher level of abstraction for writing concurrent and distributed systems. It alleviates the developer from having to deal with explicit locking and thread management, making it easier to write correct concurrent and parallel systems.

Actors were defined in the 1973 paper by Carl Hewitt but have been popularized by the Erlang language, and used for example at Ericsson with great success to build highly concurrent and reliable telecom systems.

Read more

Distributed by Default

Everything in Akka.NET is designed to work in a distributed setting: all interactions of actors use purely message passing and everything is asynchronous.

This effort has been undertaken to ensure that all functions are available equally when running within a single process or on a cluster of hundreds of machines. The key for enabling this is to go from remote to local by way of optimization instead of trying to go from local to remote by way of generalization. See this classic paper for a detailed discussion on why the second approach is bound to fail.

Read more

Akka.Quartz.Actor Quartz.net , .

Quartz , Job .( )

+ .

QuartzActor

(quartzActor) , (Receiver) .

```
PM>Install-Package Akka.Quartz.Actor

class Receiver: ActorBase
{
    public Receiver()
    {
    }

    protected override bool Receive(object message)
    {
            //handle scheduled message here
    }
 }
var receiver = Sys.ActorOf(Props.Create(() => new Receiver()), "Receiver");

var quartzActor = Sys.ActorOf(Props.Create(() => new QuartzActor()), "QuartzActor");

quartzActor.Tell(new CreateJob(receiver, "Hello", TriggerBuilder.Create().WithCronSchedule( " * * * * * ?").
Build()));
```

- , .
- , .
- , .( )

,

. StartAt .

```
quartzActor.Tell(new CreateJob(indexActor, cmdUpdate_Index, TriggerBuilder.Create()
    .StartAt(new DateTimeOffset(DateTime.Now).AddMinutes(1))
    .WithCronSchedule(" * * * * * ?").Build()));
```

: https://github.com/akkadotnet/Akka.Quartz.Actor