JavaCro'15

PRIVREDNA BANKA ZAGREB

Spring @Async

Dragan Juričić, PBZ
May 2015

INTESA SANPAOLO

PBZ je član grupe INTESA SANPAOLO

Topics

- ❑ Concept of thread pools
- ❑ Servlet 3 async configuration
- ❑ Task Execution and Scheduling
- ❑ Servlet 3 - asynchronous request processing
- ❑ Benefits and downsides

Concept of thread pools

Concept of thread pools

- thread per request – server model (Tomcat, Jetty, WAS...)
- simplistic model - create a new thread for each task
- disadvantages of the thread-per-task approach:
  - overhead of creating creating and destroying threads
  - too many threads cause the system to run out of memory
- thread pools based on work queue offers a solution
- Spring **TaskExecutor** - abstraction for thread pooling

TaskExecutor types

- pre-built implementations included with the **Spring**
  - **SimpleAsyncTaskExecutor** - starts up a new thread for each invocation, support a concurrency limit
  - **SyncTaskExecutor** - implementation doesn't execute invocations asynchronously, takes place in the calling thread
  - **ConcurrentTaskExecutor** - wrapper for a **Java** 5 java.util.concurrent.Executor
  - **ThreadPoolTaskExecutor** - exposes the **Executor** configuration parameters as bean properties
  - **WorkManagerTaskExecutor** - implements the **CommonJ WorkManager** interface - standard across **IBM's**

Servlet 3 async configuration

Servlet 3 async configuration

- **Spring** web application configuration:
  - **XML** config - update web.xml to version 3.0
  - **JavaConfig** - via **WebApplicationInitializer** interface
- **DispatcherServlet** need to have:
  - „asyncSupported” flag
- **Filter** involved in async dispatches:
  - „asyncSupported” flag
  - ASYNC dispatcher type

Spring MVC async configuration

- **WebMvcConfigurationSupport** – the main class providing the configuration behind the MVC **JavaConfig**:
  - the default timeout value for async requests
  - **TaskAsyncExecutor** (default is **SimpleAsyncTaskExecutor)**

```
protected  void configureAsyncSupport(AsyncSupportConfigurer configurer) {
      configurer.setDefaultTimeout(30*1000L);
      configurer.setTaskExecutor(mvcTaskExecutor());
  }

 protected ThreadPoolTaskExecutor mvcTaskExecutor() {
     ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
     executor.setCorePoolSize(10);
     executor.setQueueCapacity(100);
     executor.setMaxPoolSize(25);
     return executor;
}
```

Task Execution and Scheduling

Asynchronous invocation in Spring 3.0

- **@Async** annotation - executing tasks asynchronously (annotation on a method)
- the caller will return immediately and the actual execution of the method will occur in a task submitted to **TaskExecutor**
- methods are required to have a **Future<T>** return value

```
@Async
 Future<Task> returnSomething(int i) {
     // this will be executed asynchronously
   return new AsyncResult<Task>(results);
 }
```

- Spring wraps call to this method in a Runnable instance and schedule this Runnable on a task executor

Async method return value

- **Future<T>** is a proxy or a wrapper around an object - container that holds the potential result

- asynchronous task done - extract result

- **Future<T>** methods:
  - `get()` - blocks and waits until promised result is available
  - `isDone()` - poll if the result has arrived
  - `cancel()` - attempts to cancel execution of this task
  - `isCanceled()` - returns true if this task was cancelled before it completed normally.

- Concrete implementation **AsyncResult** **-** wrap result in **AsyncResult** implementing **Future<T>** interface

Exceptions with @Async

- **Exception** that was thrown during the method execution
  - **@Async** method has a Future typed return value - exception will be thrown when calling **get()** method on the Future result
  - **@Async** method has **void** return type - the exception is uncaught and cannot be transmitted
- **void return type - AsyncUncaughtExceptionHandler** can be provided to handle such exceptions

The @Scheduled Annotation

- **TaskScheduler** abstraction for scheduling tasks:
  - **TimerManagerTaskScheduler** - delegates to a **CommonJ TimerManager** instance
  - **ThreadPoolTaskScheduler** external thread management is not a requirement (implements **Spring's TaskExecutor**)
- **@Scheduled** annotation – add to a method along with trigger metadata

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

@Scheduled(cron="* 15 9-17 * * MON-FRI")
public void doSomething() {
 // something that should execute on weekdays only
}
```

Servlet 3 - asynchronous request processing

Asynchronous request handling

- **Spring 3.2** introduced **Servlet 3** based asynchronous request processing
- controller method can now return **Callable** or **DeferredResult** instance
- Servlet container thread is released and allowed to process other request:
    - **Callable** uses **TaskExecutor** thread
    - **DeferredResult** uses thread not known to **Spring**

- *Asynchronous* request processing:
    - Controller returns and **Spring MVC** starts async processing
    - Servlet and all filters exit the request thread, but response remains open
    - Other thread will complete processing and „dispetch" request back to **Servlet**
    - Servlet is invocked again and processing resumes with async result

Callable – an example controller method

```
@RequestMapping(value = {"callable.html"}, method = RequestMethod.GET)
public Callable<String> callableView(final ModelMap p_model) {
 return new Callable<String>() {
     @Override
      public String call() throws Exception {
          //... processing
       return „someView";
     }
  };
}
```

- **WebAsyncTask –** wrap **Callable** for customization:
  - timeout
  - **TaskExecutor**

PRIVREDNA BANKA ZAGREB

INTESA SANPAOLO

DeferredResult – an example controller method

```
@RequestMapping("/response-body")
@ResponseBody
public DeferredResult<String> quotes() {

    DeferredResult<String> deferredResult = new
DeferredResult<String>();
    // Save the deferredResult in in-memory queue ...

    return deferredResult;
}

// In some other thread...
deferredResult.setResult(data);
```

Exception handling for async requests

- What happens if a **Callable** or **DeferredResult** returned from a controller method raises an **Exception**?

- **Callable**
  - **@ExeceptionHandler** method in the same controller
  - one of the configured **HandlerExceptionResolver** instances

- **DeferredResult**
  - calling **setErrorResult()** method and provide an **Exception** or any other Object as result
  - **@ExeceptionHandler** method in the same controller
  - one of the configured **HandlerExceptionResolver** instances

Benefits and downsides

Benefits

- @Async method:
  - asynchronous method calls solves a critical scaling issue
  - the longer the task takes and the more tasks are invoked - the more benefit with making things asynchronous
- Async request:
  - decouple processing from Servlet container thread - longer request can exhaust container thread pool quickly
  - processing of AJAX applications efficiently
  - browser real-time update – server push (alternative to standard HTTP request-response approaches: polling, long polling, HTTP streaming)
- Servlet 3 specification:
  - asynchronous support
  - JavaConfig without need for web.xml and enhancements to servlet API

Downsides

- threading risks
- additional configuration (servlet, filter, thread pool...)
- asynchronous method calls adds a layer of indirection - no longer dealing directly with the results, but must instead poll for them
- converting request or method calls to an asynchronous approach may require extra work

References

- http://spring.io/
- http://oracle.com/
- http://docs.spring.io/spring/docs/current/spring-framework-reference/html/scheduling.html
- http://www.slideshare.net/bruce.snyder/beyond-horizontal-scalability-concurrency-and-messaging-using-spring
- http://www.slideshare.net/chintal75/asynchronous-programmingtechniques
- http://www.ibm.com/developerworks/library/j-jtp0730.html