

Task, Async, Cache

# Task scheduling & Async I.

- Task scheduling zapnete pomocí Java config tímto způsobem:

```
@Configuration
@EnableAsync
@EnableScheduling

public class AppConfig {
}
```

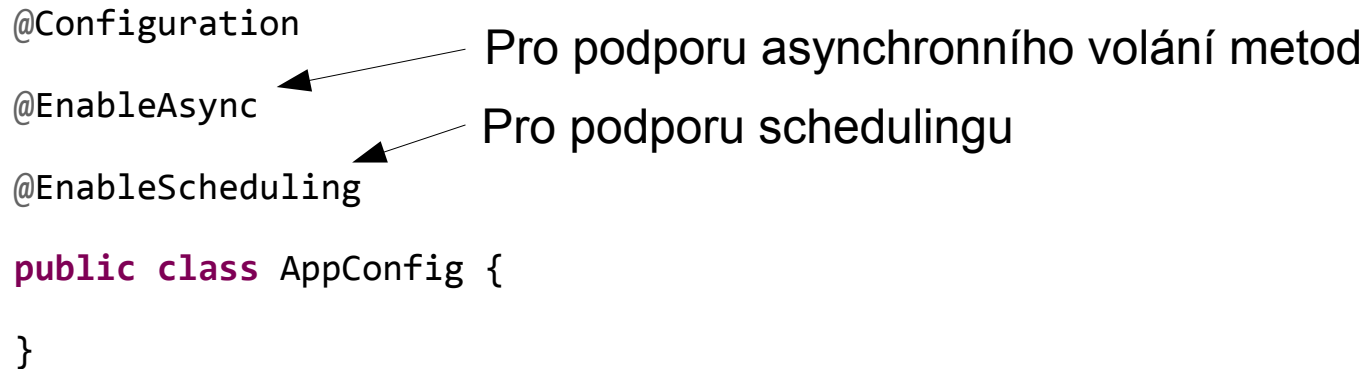


Diagram illustrating the annotations used for task scheduling and async support in Java configuration:

- `@Configuration` (points to `@EnableAsync`): Pro podporu asynchronního volání metod
- `@EnableAsync` (points to `@EnableScheduling`): Pro podporu schedulingu
- `@EnableScheduling`

- Pomocí XML dosáhnete toho samého když přidáte toto do root kontextu:

```
<task:annotation-driven executor="myExecutor" scheduler="myScheduler"/>
<task:executor id="myExecutor" />
<task:scheduler id="myScheduler" />
```

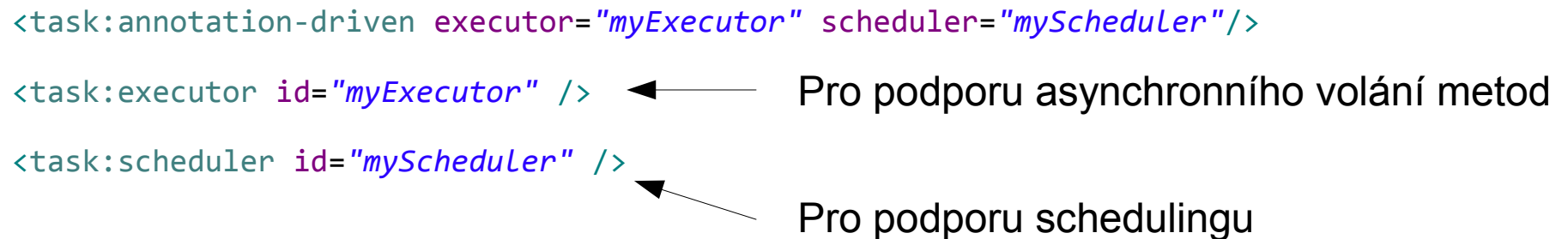


Diagram illustrating the XML configuration for task scheduling and async support:

- `<task:executor id="myExecutor" />` (points to `<task:scheduler id="myScheduler" />`): Pro podporu asynchronního volání metod
- `<task:scheduler id="myScheduler" />` (points to `<task:scheduler id="myScheduler" />`): Pro podporu schedulingu

# Task scheduling & Async II.

- Poté můžete použít na metodě anotace:
  - `@Scheduled` – můžete nastavovat dobu v milisekundách mezi voláním této metody nebo pomocí cronu, který má syntaxi jako v Linuxu.
    - Tato metoda nesmí mít žádné parametry a musí vracet `void`.
    - Pokud metoda interaguje s objekty ze Spring kontextu, pak je nutné je přidat do třídy pomocí dependency injection (`@Autowired`).
  - `@Async` – taková metoda se zavolá asynchronně (Spring vytvoří vlákno, ve kterém obslouží její volání a ihned vrátí výsledek této metody)
    - Tato metoda může mít parametry (protože je volána "normálním" způsobem).
    - Vracet může buď `void`, nebo objekt typu `Future<Typ>`

# Task scheduling & Async – konfigurace

- Pokročilejší konfigurace schedulingu:
  - <http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/scheduling/annotation/EnableScheduling.html>
- Pokročilejší konfigurace async:
  - <http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/cache/annotation/EnableCaching.html>

# Task scheduling & Async III.

- Pokud chcete mít metody, které se volají v scheduleru na jednom místě, pak zvolte tento způsob XML konfigurace (v root kontextu):

```
<task:scheduled-tasks scheduler="myScheduler">  
    <task:scheduled ref="beanA" method="methodA" fixed-rate="5000"/>  
    <task:scheduled ref="beanB" method="methodB" cron="*/5 * * * * MON-FRI"/>  
</task:scheduled-tasks>
```

Syntaxe je stejná jako u Linux CRONu,  
ale oproti němu navíc podporuje vteřiny

# Task scheduling & Async IV.

- Popsaný task scheduling není možné provádět když aplikace běží v clusteru! K tomu je buď nutné funkcionalitu task schedulingu vyčlenit do aplikace, která běží zvlášť a neběží v clusteru, nebo použít projekt Quartz:
  - <http://quartz-scheduler.org/>
  - <http://docs.spring.io/spring/docs/3.2.4.RELEASE/spring-framework-reference/html/scheduling.html>

# Task Executor – pokročilejší témata I.

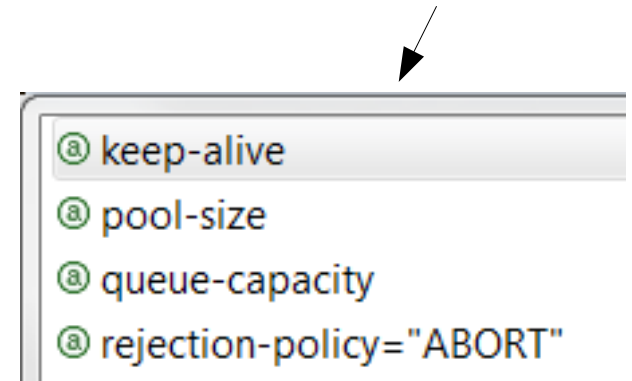
- Tag `<task:annotation-driven />` není žádná magie, fyzicky tento tag přidá do Spring kontextu beanu, která je implementací interface `TaskExecutor`, který je identický s interface `java.util.concurrent.Executor`, který je od Java 5 a slouží k práci s pooly vláken.
- Těchto bean je několik, každá funguje jiným způsobem a každá má řadu nastavení:
- `SimpleAsyncTaskExecutor`: výchozí bean, která pro každý task vytvoří nové vlákno

```
<task:annotation-driven />
```

# Task Executor – pokročilejší témata II.

- ThreadPoolTaskExecutor: nejčastěji používaná implementace, od Java 5, použije se pool vláken, kolik jich bude závisí na nastavení:

```
<task:annotation-driven executor="executor" />  
<task:executor id="executor"/>
```



- Toho samého (ale s většími možnostmi nastavení) dosáhnete tímto způsobem:

```
<task:annotation-driven executor="executor" />  
<bean class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor"  
      id="executor">  
    Tady je možné nastavit property  
</bean>
```



# Task Executor – pokročilejší témata III.

- Existují další beany, které se definují pomocí tagu <bean> (jako v posledním příkladu). Celý jejich seznam naleznete v dokumentaci:
  - <http://docs.spring.io/spring/docs/3.2.4.RELEASE/spring-framework-reference/html/scheduling.html>
- Pokud chcete pracovat s TaskExecutor přímo, pak můžete ve Vaší Service třídě jednoduše udělat následující:

```
@Autowired
```

```
private TaskExecutor taskExecutor;
```

- Nebo můžete injectnout konkrétní instanci implementace TaskExecutor kterou používáte.

# Task Scheduler – pokročilejší témata I.

- U scheduleru existuje obdobná, i když výrazně menší možnost flexibility:
- Pokud neuvedete scheduler, pak se ve výchozím nastavení použije executor, který běží v jednom vlákně:

```
<task:annotation-driven />
```

- Pokud uvedete scheduler, pak se vytvoří instance bean `ThreadPoolTaskScheduler`:

```
<task:annotation-driven scheduler="scheduler" />
```

```
<task:scheduler id="scheduler" />
```

- Standardně máte tímto způsobem pouze možnost nastavit počet vláken v poolu. Pro pokročilejší nastavení je opět nutné místo `<task:scheduler>` nadefinovat `<bean>`

# Task Scheduler – pokročilejší témata II.

- Pokud potřebujete spravovat scheduling na serveru, pak je možné místo třídy `ThreadPoolTaskScheduler` použít třídu `TimerManagerTaskScheduler`, která umožňuje přes JNDI injectnout instanci `CommonJ TimerManager`.
  - [http://docs.oracle.com/cd/E11035\\_01/wls100/javadocs/commonj/timers/TimerManager.html](http://docs.oracle.com/cd/E11035_01/wls100/javadocs/commonj/timers/TimerManager.html)
- Pokud Vám nestačí definování kdy se bude spouštět aplikace pomocí Cronu nebo dalších standardních způsobů, pak můžete vytvořit instanci třídy `Trigger`, kterou poté zapojíte do `TaskScheduleru` a implementujete v ní metodu `nextExecutionTime()`, která vrací datum kdy příště se má metoda zavolat.

# Cache I.

- Spring umožňuje jednoduchým způsobem cachování volání metod (výsledek se uloží do cache a při příštím zavolání metody se vrátí z cache).
- Ve výchozím nastavení používá `java.util.concurrent.ConcurrentMap`
- Nastavení na metodě:

```
@Cacheable("books")
```

```
public Book findBook(ISBN isbn) {...}
```

- Data budou uložena v cache do jejího smazání. K tomu je nutné vytvořit takovouto metodu:

```
@CacheEvict(value = "books", allEntries = true)
```

```
public void reloadBooks() { ... }
```

- Před tuto metodu můžete nastavit anotaci `@Scheduled` a volat ji tak periodicky. Také je užitečné nastavit `@CacheEvict` na `save()` a `delete()` operace.

# Cache II. - xml

- Dále je nutná konfigurace:

```
<cache:annotation-driven />
```

```
<bean id="cacheManager" class="org.springframework.cache.support.SimpleCacheManager">
```

```
  <property name="caches">
```

```
    <set>
```

```
      <bean
```

```
        class="org.springframework.cache.concurrent.ConcurrentMapCacheFactoryBean"
```

```
        p:name="ebooks" />
```

```
    </set>
```

```
  </property>
```

```
</bean>
```

# Cache II. - anotace

- Nebo pomocí anotací:

`@EnableCaching` u třídy s anotací `@Configuration`

- Dále přidejte tuto bean:

`@Bean`

```
public CacheManager cacheManager() {  
    SimpleCacheManager cacheManager = new SimpleCacheManager();  
    cacheManager.setCaches(Arrays.asList(new ConcurrentMapCache("books")));  
    return cacheManager;  
}
```

# Cache III.

- Pro složitější použití má Spring integraci na EhCache:
  - <http://ehcache.org/>
  - <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/cache.html>
  - <http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/cache/annotation/EnableCaching.html>