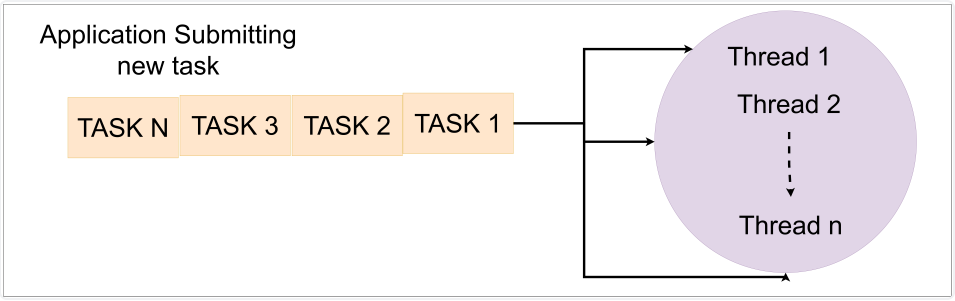
14. Spring boot @Async Annotation - Part1 | ThreadPoolExecutor

Pre-requisite Java Videos:

- Thread Pools & ThreadPoolExecutor
- Future, Callable & CompleatableFuture
- Thread Life Cycle.

What is ThreadPool

- It's collection of threads which are available to perform the submitted tasks.
- Once task completed, worker thread get back to Thread Pool and wait for new task to assigned.
- Means threads can be reused.



- Min Pool size=2
- Max Pool size=4
- Queue Size=3

- Let's we have already define the sizes. now let's come to the flow.
- let's queue currently holding two task task1 and task2 and thread pool also contain two thread t1 and t2,
- t1 assigned with task1 and t2 assigned with task2

- let's task3 has come, there is no thread free in thread pool and queue is also empty so task will go the queue.
- same thing happen for task 4 and task 5 untill queue get's full.
- let's task 6 has come now queue is full and and thread pool has not free thread to pick this task.
- so, what happens, we already know thread pool max size is 4 so we can create up to two more threads. because previous two threads is already in busy state.
- so, task 6 get assigned to thread t4.
- again task 7 has came now we can create one more thread so, task 7 get's assigned to t4.
- now thread pool max size is also reached and queue is also full.
- let's task 8 came it's get's rejected.
- now thread t1 complete their task and now it's free now , so, thread t1 will go back to thread pool and check for queue and it's pick the front task of queue.
- so t1 again assigned with task3 because it is in front.

In java, thread pool is created using ThreadPoolExecutor Object.

```
int minPoolSize=2;
int maxPoolSize=2;
int queueSize=3;
ThreadPoolExecutor poolTaskExecutor= new ThreadPoolExecutor(minPoolSize,maxPoolSize,10,
                TimeUnit.MINUTES,new ArrayBlockingQueue<>(2)) ;
```

In Java, thread pool is created using ***ThreadPoolExecutor*** Object

```
int minPoolSize = 2;
int maxPoolSize = 4;
int queueSize = 3;

ThreadPoolExecutor poolTaskExecutor = new ThreadPoolExecutor(minPoolSize, maxPoolSize, keepAliveTime: 1,
        TimeUnit.HOURS, new ArrayBlockingQueue<>(queueSize));
```

Async Annotations

- Used to mark method that should run asunchronously.
- Runs in new thread, without block the main thread.

@EnableAsync

- Purpose, when spring boot see this annotation it will create bean **async** classes like **async interceptors**, their object onyly be created when we put this annorarions.

```
package com.springProject.SpringCourseProject;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableAsync;


@SpringBootApplication
@EnableAsync
public class SpringCourseProjectApplication {
    public static void main(String[] args) {
       SpringApplication.run(SpringCourseProjectApplication.class, args);
    }
}
```

```
@Async
public void asyncMethodTest(){
    System.out.println("Inside Async Method test current thread is:
"+Thread.currentThread().getName());
}
```

```
@GetMapping(path = "/getUser")
    public String getUserDetails(){
        System.out.println("Inside getUserDetails Current thread name:
"+Thread.currentThread().getName());
        userServices.asyncMethodTest();
        return user.getUserDetails();
    }
```

output

hitting for multiples times API : http://localhost:8080/api/getUser

Inside getUserDetails Current thread name: http-nio-8080-exec-1

Inside Async Method test current thread is: task-1

Inside getUserDetails Current thread name: http-nio-8080-exec-3

Inside Async Method test current thread is: task-2

Inside getUserDetails Current thread name: http-nio-8080-exec-4

Inside Async Method test current thread is: task-3

So, how does this "Async" Annotation, creates a new Thread.

- Many places you will find, which says.
- Spring boot uses by default **"SimpleAsyncTaskExecutor"**, which creates new thread every time.
- it's blindly create new thread because we don't defined maxPoolsize,minPoolsize and queseSize.
- I will say, this not fully correct answer.

So, Whats the correct answer, what's the default Executor Spring boot uses ?

- If we see below spring boot frame work code, it first looks for defaultExecutor, if no defaultExecutor found, only then **SimpleAsyncTaskExecutor** is used.

AyncExecutionInterceptor

```
@Nullable
protected Executor getDefaultExector(@Nullable BeanFactory beanFactory){
 Executor defaultExecutor = super.getDefaultExecutor(beanFactory);
 return (Executor) (defaultExecutor !=null ? defaultExecutor : new
SimpleAsyncTaskExecutor());
}
```

UseCase 1

```
package com.springProject.SpringCourseProject;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableAsync;


@SpringBootApplication
@EnableAsync
public class SpringCourseProjectApplication {
    public static void main(String[] args) {
       SpringApplication.run(SpringCourseProjectApplication.class, args);
    }
}
```

```
@Confugaration
public class AppConfig{
}
```

```
@Async
public void asyncMethodTest(){
    System.out.println("Inside Async Method test current thread is:
"+Thread.currentThread().getName());
}
```

```
@GetMapping(path = "/getUser")
    public String getUserDetails(){
        System.out.println("Inside getUserDetails Current thread name:
"+Thread.currentThread().getName());
        userServices.asyncMethodTest();
```

```
            return user.getUserDetails();
        }
```

During Application startup, spring boot sees that , no ThreadPoolTaskExecutor Bean present, so it creates its default "ThreadPoolTaskExecutor" with below configurations.

corePoolSize = 8

minPoolSize = Interger.MAX

keepAliveSeconds=60

queueCapacity=Integer.Max

ThreadPoolExecutor - > plane java

ThreadPoolTaskExecutor - > spring wrapper

UseCase1:

```java
@Configuration
public class AppConfig {

}
```

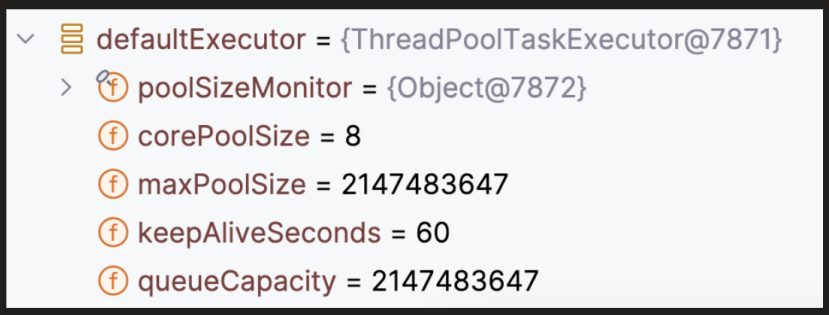
```java
@SpringBootApplication
@EnableAsync
public class SpringbootApplication {

    public static void main(String args[]){
        SpringApplication.run(SpringbootApplication.class, args);
    }
}
```

```java
@Component
public class UserService {

    @Async
    public void asyncMethodTest() {
        System.out.println("inside asyncMethodTest: " + Thread.currentThread().getName());
    }
}
```

During Application startup, Spring boot sees that, no **ThreadPoolTaskExecutor** Bean present, so it creates its default "**ThreadPoolTaskExecutor**" with below configurations.



ThreadPoolTaskExecutor is nothing but a Spring boot object, which is just a wrapper around Java ThreadPoolExecutor.

ThreadPoolTaskExecutor is nothing but a Spring boot Object, which is just a wrapper around Java ThreadPoolExecutor.

ThreadPoolTaskExecutor.java

```
protected ExecutorService initializeExecutor(ThreadFactory threadFactory, RejectedExecutionHandler rejectedExecutionHandler) {
    BlockingQueue<Runnable> queue = this.createQueue(this.queueCapacity);
    ThreadPoolExecutor executor = new ThreadPoolExecutor(this.corePoolSize, this.maxPoolSize, (long) this.keepAliveSeconds, TimeUnit.SECONDS, queue, threadFactory, rejectedExecutionHandler) {
        public void execute(Runnable command) {...}

        protected void beforeExecute(Thread thread, Runnable task) {...}

        protected void afterExecute(Runnable task, Throwable ex) {...}
    };
    if (this.allowCoreThreadTimeOut) {...}

    if (this.prestartAllCoreThreads) {...}

    this.threadPoolExecutor = executor;
    return executor;
}
```

And it's not recommended at all, why ?

- **Underutilization of Threads**: with fixed Min which is 8 in case of default, Pool size and Unbounded Queue [size is too big] , its possible that most of the tasks will sit in the queue rather than creating new thread.
- **High Latency :** Since queue size is too big, tasks will queue up till queue is not fill, high latency might occur during high load.
- **Thread Exhaustion** : Let's say, if queue also get filled up, then Executor will try to create new thread till max pool size is not reached, which is **INTEGER.MAX_VALUE**. This can lead to thread exhaustion. And server might go down because of Overhead of managing so many threads.
- **High Memory Usage**: Each threads need some memory, when we are creating these many threads, which many consume large amount of memory too, which might lead to performance degradation too.

UseCase 2 Creating our own custom, **ThreadPoolTaskExecutor**

```
package com.springProject.SpringCourseProject;

@SpringBootApplication
@EnableAsync
public class SpringCourseProjectApplication {
```

```
    public static void main(String[] args) {
       SpringApplication.run(SpringCourseProjectApplication.class, args);
    }
}
```

```
package com.springProject.SpringCourseProject.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

import java.util.concurrent.Executor;

@Configuration
public class AppConfig {
    @Bean(name = "myThreadPoolExecutor")
    public Executor taskPoolExecutor(){
       int minPoolSize=2;
       int maxPoolSize=4;
       int queueSize=3;

        ThreadPoolTaskExecutor poolTaskExecutor = new ThreadPoolTaskExecutor();
        poolTaskExecutor.setCorePoolSize(minPoolSize);
        poolTaskExecutor.setMaxPoolSize(maxPoolSize);
        poolTaskExecutor.setQueueCapacity(queueSize);
        poolTaskExecutor.setThreadNamePrefix("MyThread-");
        poolTaskExecutor.initialize();
        return poolTaskExecutor;

    }
}
```

```
@Async("myThreadPoolExecutor")
public void asyncMethodTest(){
    System.out.println("Inside Async Method test current thread is:
"+Thread.currentThread().getName());
}
```

```
@GetMapping(path = "/getUser")
public String getUserDetails(){
    System.out.println("Inside getUserDetails Current thread name:
"+Thread.currentThread().getName());
    userServices.asyncMethodTest();
    return user.getUserDetails();
}
```

output

hitting for multiples times API : http://localhost:8080/api/getUser

Inside getUserDetails Current thread name: http-nio-8080-exec-1

Inside Async Method test current thread is: MyThread-1

Inside getUserDetails Current thread name: http-nio-8080-exec-3

Inside Async Method test current thread is: MyThread-2

Inside getUserDetails Current thread name: http-nio-8080-exec-4

Inside Async Method test current thread is: MyThread-3

During Application startup. Spring boot sees that, ThreadPoolTaskExecutor Bean present, so it make it default only.

And even when we use @Async withoutany name, our "myThreadPoolExecutor" will get picked only.

```
@Async("myThreadPoolExecutor")
public void asyncMethodTest(){
    System.out.println("Inside Async Method test current thread is:
"+Thread.currentThread().getName());

    try{
        Thread.sleep(50000);
    }catch (Exception e){

    }
}
```

output

hitting for multiples times API : http://localhost:8080/api/getUser

Inside getUserDetails Current thread name: http-nio-8080-exec-2

Inside Async Method test current thread is: MyThread-1

Inside getUserDetails Current thread name: http-nio-8080-exec-4

Inside Async Method test current thread is: MyThread-2

MIN POOL SIZE REACHED WHICH IS 2

Inside getUserDetails Current thread name: http-nio-8080-exec-5

Inside getUserDetails Current thread name: http-nio-8080-exec-6

Inside getUserDetails Current thread name: http-nio-8080-exec-7

QUEUE SIZE IS ALSO REACHED

If More task will come then it got rejected and exception thrown

2024-09-08T22:59:46.446+05:30 ERROR 5272 --- [SpringCourseProject] [nio-8080-exec-9] o.a.c.c.C.[.[.[/].[dispatcherServlet] : Servlet.service() for servlet [dispatcherServlet] in context with path [] threw **exception [Request processing failed: org.springframework.core.task.TaskRejectedException: ExecutorService in active state did not accept task:** org.springframework.aop.interceptor.AsyncExecutionInterceptor$$Lambda/0x000002219248ea40@1925e4e7] with root cause

java.util.concurrent.RejectedExecutionException: Task java.util.concurrent.FutureTask@1d1a731b[Not completed, task = org.springframework.aop.interceptor.AsyncExecutionInterceptor$$Lambda/0x000002219248ea40@1925e4e7] rejected from org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor$1@2d85fb64[Running, pool size = 4, active threads = 4, queued tasks = 3, completed tasks = 0]

at java.base/java.util.concurrent.ThreadPoolExecutor$AbortPolicy.rejectedExecution(ThreadPoolExecutor.java:2081) ~[na:na]

at java.base/java.util.concurrent.ThreadPoolExecutor.reject(ThreadPoolExecutor.java:841) ~[na:na]

at java.base/java.util.concurrent.ThreadPoolExecutor.execute(ThreadPoolExecutor.java:1376) ~[na:na]

Inside getUserDetails Current thread name: http-nio-8080-exec-8

Inside Async Method test current thread is: MyThread-3

NEW THREAD CREATED MyThread-3

Inside Async Method test current thread is: MyThread-1

MyThread-1 get's free pick task 5 which parent thread is http-nio-8080-exec-5

Inside Async Method test current thread is: MyThread-2

MyThread-2 get's free pick task 6 which parent thread is http-nio-8080-exec-6

Inside Async Method test current thread is: MyThread-3

MyThread-3 get's free pick task 7 which parent thread is http-nio-8080-exec-7

It is recommended, as it's solve the issues existing with the previous case

UseCase 3: Creating our own custom, ThreadPoolExecutor [Java one]

```
package com.springProject.SpringCourseProject.config;

import com.springProject.SpringCourseProject.component.Order;
import com.springProject.SpringCourseProject.imple.OfflineOrder;
import com.springProject.SpringCourseProject.imple.OnlineOrder;
import jakarta.annotation.Nullable;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicInteger;

@Configuration
public class AppConfig {

    @Bean(name = "myThreadPoolExecutor")
    public Executor taskPoolExecutor(){
       int minPoolSize=2;
       int maxPoolSize=4;
       int queueSize=3;
        return new ThreadPoolExecutor(minPoolSize,maxPoolSize,1,
```

```
                    TimeUnit.HOURS,new ArrayBlockingQueue<>(queueSize), new
CustomThreadFactory());

    }

    static class CustomThreadFactory implements ThreadFactory{

        private final AtomicInteger threadNo = new AtomicInteger(1);
        @Override
        public Thread newThread(@Nullable Runnable r) {
            Thread thread =new Thread(r);
            thread.setName("MyThread-"+threadNo.getAndIncrement());
            return thread;
        }
    }
}
```

During Application startup, Spring boot seet that, ThreadPoolExecutor (java one) Bean is present, do not create its own default ThreadPoolTaskExecutor [spring wrapper one] insteadit set default executor is "SimpleAsyncTaskExecutor"

Now we run above code what we can see.

output

hitting for multiples times API : http://localhost:8080/api/getUser

Inside getUserDetails Current thread name: http-nio-8080-exec-2

2024-09-09T00:00:42.165+05:30 INFO 17368 --- [SpringCourseProject] [nio-8080-exec-2] .s.a.AnnotationAsyncExecutionInterceptor : No task executor bean found for async processing: no bean of type TaskExecutor and no bean named 'taskExecutor' either

Inside Async Method test current thread is: SimpleAsyncTaskExecutor-1

Inside getUserDetails Current thread name: http-nio-8080-exec-3

Inside Async Method test current thread is: SimpleAsyncTaskExecutor-2

Inside getUserDetails Current thread name: http-nio-8080-exec-4
Inside Async Method test current thread is: SimpleAsyncTaskExecutor-3

What if we want plane java ThreadPoolExecutor we need to provide proper name.

```
@Async("myThreadPoolExecutor")
public void asyncMethodTest(){
    System.out.println("Inside Async Method test current thread is:
"+Thread.currentThread().getName());

    try{
        Thread.sleep(50000);
    }catch (Exception e){

    }
}
```

output
hitting for multiples times API : http://localhost:8080/api/getUser

Inside getUserDetails Current thread name: http-nio-8080-exec-1
Inside Async Method test current thread is: MyThread-1

Inside getUserDetails Current thread name: http-nio-8080-exec-3
Inside Async Method test current thread is: MyThread-2

Inside getUserDetails Current thread name: http-nio-8080-exec-4
Inside Async Method test current thread is: MyThread-3

And it's not recommended at all to use "SimpleAsyncTaskExecutor", why?

It just creates new Thread every time, So it may lead to.

- **Thread Exhaustion:** Just blindly creating new thread with every Async request, might lead up to thread exhaustion.
- **Thread Creation Overhead**: Since Threads are not reused, so thread management (creation, destroying) is an additional overhead.
- **High Memory Usage:** Each threads need some memory, when we are creating these many threads, which may consume large amount of memory too, which are creating these many threads, which may consume large amount of memory too, which might lead to performance degradation too.

Hey, I don't want all this cofusion, UseCase1, Usecase2, or Usecase3.

I always want to set my executor as default one even if anyone use @Async, my executor only should be picked.

Industry Standard

```
package com.springProject.SpringCourseProject.config;


import jakarta.annotation.Nullable;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;

import org.springframework.scheduling.annotation.AsyncConfigurer;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;


import javax.sql.DataSource;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicInteger;

@Configuration
public class AppConfig implements AsyncConfigurer {

    private ThreadPoolExecutor poolExecutor;

    @Override
    public synchronized Executor getAsyncExecutor(){
        if(poolExecutor==null){
```

```
            int minPoolSize=2;
            int maxPoolSize=4;
            int queueSize=3;

            //Here We can also use ThreadPoolTaskExecutor
            poolExecutor= new ThreadPoolExecutor(minPoolSize,maxPoolSize,1,
                    TimeUnit.HOURS,new ArrayBlockingQueue<>(queueSize), new
CustomThreadFactory());

        }
        return poolExecutor;
    }




    static class CustomThreadFactory implements ThreadFactory{

        private final AtomicInteger threadNo = new AtomicInteger(1);
        @Override
        public Thread newThread(@Nullable Runnable r) {
            Thread thread =new Thread(r);
            thread.setName("MyThread-"+threadNo.getAndIncrement());
            return thread;
        }
    }
}
```

```
@Async
public void asyncMethodTest(){
    System.out.println("Inside Async Method test current thread is:
"+Thread.currentThread().getName());

    try{
        Thread.sleep(50000);
    }catch (Exception e){

    }
}
```

```
@GetMapping(path = "/getUser")
public String getUserDetails(){
    System.out.println("Inside getUserDetails Current thread name:
"+Thread.currentThread().getName());
    userServices.asyncMethodTest();
    return user.getUserDetails();
}
```

output
hitting for multiples times API : http://localhost:8080/api/getUser

Inside getUserDetails Current thread name: http-nio-8080-exec-1

Inside Async Method test current thread is: MyThread-1

Inside getUserDetails Current thread name: http-nio-8080-exec-3
Inside Async Method test current thread is: MyThread-2

Inside getUserDetails Current thread name: http-nio-8080-exec-4

Inside getUserDetails Current thread name: http-nio-8080-exec-5

Inside getUserDetails Current thread name: http-nio-8080-exec-6

Inside Async Method test current thread is: MyThread-1

Inside Async Method test current thread is: MyThread-2

Inside Async Method test current thread is: MyThread-1

Still, default executor configuration picked is mine one, not SimpleAsyncTaskExecutor.

Spring boot @Async Annotation - Part2 | Async Annotation Important Interview

questions

Conditions for @Async Annotation to work properly.

1. **Different Class:**

- The method which is being called and annotated with @Async should be in other class.
- If @Async annotation is applied to the method within the same class from which it is being called, then mechanism is skipped because internal method calls are not INTERCEPTED.

2. **Public method.**

- Method annotated with @Async must be public. and again, AOP interception works only on public method

>Both in same class , proxy will get bypassed

>This is wrong way of doing @Async method should be public

```
package com.springProject.SpringCourseProject.controller;

@RestController
@RequestMapping(value = "/api/")
public class UserController {
    @GetMapping(path = "/getUser")
    public String getUserDetails(){
        System.out.println("Inside getUserDetails Current thread name:
"+Thread.currentThread().getName());

        asyncMethodTest();
        return user.getUserDetails();
    }

    @Async
    public void asyncMethodTest() {
        System.out.println("Inside asyncMethodTest() of usercontroller Current thread
name: "+Thread.currentThread().getName());
    }

}
```

output

Inside getUserDetails Current thread name: http-nio-8080-exec-1

Inside asyncMethodTest() of usercontroller Current thread name: http-nio-8080-exec-1

Here async is always create new thread for running it's task but in this case it's uses parent thread.

Why this is happening?

- because @Async is uses AOP internally , so in AOP first intercept the annotation and apply proxy. and that proxy internally called these method like business logic, before and after method.
- Interceptions only worked on public method and when class1 invoked other class 2.

How @Async and Transaction management worked together.

Usecase 1:

- Transaction Context do not transfer from caller to new thread and which got created by @Async
- When transactional method calls some other class public @Async method then transaction is not transferred or carry forwarded to that method.
- So any exception occur in transactional method the rollback is going to happen only transactional method context, only that changes will be roll back, but changes in other class public @Async method has no affect.

```
package com.springProject.SpringCourseProject.utility;

import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Component;

@Component
public class UserUtility {

    @Async
    public void updateUserBalanceWithTransactionAndAsyncUseCase1() {
        System.out.println("Inside updateUserBalanceWithTransactionAndAsyncUseCase1() of
UtilityUserClass Current thread name: "+Thread.currentThread().getName());
    }
}
```

```
@Service
public class UserServices {

    @Autowired
    UserUtility userUtility;

    @Transactional
    public void updateUserWithTransactionAndAsyncUseCase1(){
        System.out.println("Inside UserServices class method name
updateUserWithTransactionAndAsyncUseCase1()");
        System.out.println("Updating UserDetails");
        userUtility.updateUserBalanceWithTransactionAndAsyncUseCase1();
    }
}
```

```
@RestController
@RequestMapping(value = "/api/")
public class UserController {


    @Autowired
    UserServices userServices;

    @GetMapping(path = "/updateUser/updateUserWithTransactionAndAsyncUseCase1")
    public String updateUserWithTransactionAndAsyncUseCase1(){
        System.out.println("Inside updateUserWithTransactionAndAsyncUseCase1() method of
UserController.");
        userServices.updateUserWithTransactionAndAsyncUseCase1();

        return "updateUserWithTransactionAndAsyncUseCase1() method completed";
    }

}
```

OUTPUT

Hiiting API: http://localhost:8080/api/updateUser/updateUserWithTransactionAndAsyncUseCase1

Inside updateUserWithTransactionAndAsyncUseCase1() method of UserController.

Inside UserServices class method name updateUserWithTransactionAndAsyncUseCase1()

Updating UserDetails

Inside updateUserBalanceWithTransactionAndAsyncUseCase1() of UtilityUserClass Current thread name: MyThread-1

UseCase 2: **Use with Precaution**,

- as new thread will be created and have transaction management too but context is not same as parent thread. So, Propagation will not work expected.
- If we applying @Transactional on controller method as well, then what happens the controller transaction context with different propagation like **[@Transactional(propagation = Propagation.REQUIRES_NEW)]** will not carry forwarded to services method which is annotated with @Transaction and @Async
- It's only because of @Async because as it's created new Thread context.

```
@Service
public class UserServices {



    @Autowired
    UserUtility userUtility;



    @Transactional
    @Async
    public void updateUserWithTransactionAndAsyncUseCase2(){
        System.out.println("Inside UserServices class method name
updateUserWithTransactionAndAsyncUseCase2(): "+Thread.currentThread().getName());
        System.out.println("Updating UserDetails with transaction and @Async");

    }
}
```

```
@RestController
@RequestMapping(value = "/api/")
public class UserController {


    @Autowired
    UserServices userServices;
```

```
    //@Transactional(propagation = Propagation.REQUIRES_NEW)
    @GetMapping(path = "/updateUser/updateUserWithTransactionAndAsyncUseCase2")
    public String updateUserWithTransactionAndAsyncUseCase2(){
        System.out.println("Inside updateUserWithTransactionAndAsyncUseCase2() method of
UserController. current threadname: "+Thread.currentThread().getName());
        userServices.updateUserWithTransactionAndAsyncUseCase2();

        return "updateUserWithTransactionAndAsyncUseCase2() method completed";
    }

}
```

OUTPUT

Hiiting API: http://localhost:8080/api/updateUser/updateUserWithTransactionAndAsyncUseCase2

Inside updateUserWithTransactionAndAsyncUseCase2() method of UserController. current threadname: http-nio-8080-exec-2

Inside UserServices class method name updateUserWithTransactionAndAsyncUseCase2(): MyThread-1

Updating UserDetails with transaction and @Async

Usecase 3 : Industry Standard Most popular.

- Here in userservices we are annotating method with @Async and it's create new thread.
- and we are calling Userutility method from userservices method which is annotated with @Transactional
- So, newly created thread context os userservices is carry forwared to userutility method.

```
@Component
public class UserUtility {

    @Transactional
    public void updateUserTransactionAndAsyncUseCase3() {
        System.out.println("Inside updateUserTransactionAndAsyncUseCase3() of
UtilityUserClass Current thread name: "+Thread.currentThread().getName());
    }
}
```

```
@Service
public class UserServices {


    @Autowired
    UserUtility userUtility;



    @Async
    public void updateUserWithTransactionAndAsyncUseCase3(){
        System.out.println("Inside UserServices class method name
updateUserWithTransactionAndAsyncUseCase3(): "+Thread.currentThread().getName());
        System.out.println("Updating UserDetails with transaction and @Async");
        userUtility.updateUserTransactionAndAsyncUseCase3();

    }
}
```

```
@RestController
@RequestMapping(value = "/api/")
public class UserController {


    @Autowired
    UserServices userServices;



    @GetMapping(path = "/updateUser/updateUserWithTransactionAndAsyncUseCase3")
    public String updateUserWithTransactionAndAsyncUseCase3(){
        System.out.println("Inside updateUserWithTransactionAndAsyncUseCase3() method of
UserController. current threadname: "+Thread.currentThread().getName());
        userServices.updateUserWithTransactionAndAsyncUseCase3();

        return "updateUserWithTransactionAndAsyncUseCase3() method completed";
    }

}
```

OUTPUT

Hiiting API: http://localhost:8080/api/updateUser/updateUserWithTransactionAndAsyncUseCase3

Inside updateUserWithTransactionAndAsyncUseCase3() method of UserController. current threadname: http-nio-8080-exec-1

Inside UserServices class method name updateUserWithTransactionAndAsyncUseCase3(): MyThread-1

Updating UserDetails with transaction and @Async

Inside updateUserTransactionAndAsyncUseCase3() of UtilityUserClass Current thread name: MyThread-1

@Async Method Return type

Both Future and Completable Future can be the return type of Async method.

Checkout Java Multithreading notes to learn more in depths of Future and Completable.

Using Future

- Now Deprecated.

```
import java.util.concurrent.Future;

@Service
public class UserServices {



    @Async
    public Future<String> performTaskAsync(){
        System.out.println("Inside UserServices class method name performTaskAsync():
"+Thread.currentThread().getName());
        return new AsyncResult<>("Async Task result");
    }
}
```

```
import java.util.concurrent.Future;

@RestController
@RequestMapping(value = "/api/")
```

```
public class UserController {


    @Autowired
    UserServices userServices;


    @GetMapping(path = "/getUserAsyncReturnTypeUsingFuture")
    public String getUserAsyncReturnTypeUsingFuture(){
        System.out.println("Inside getUserAsyncReturnTypeUsingFuture Current thread
name: "+Thread.currentThread().getName());

        Future<String> result = userServices.performTaskAsync();
        String output=null;
        try {
//get() a lot like await so here what happens it wait for the response from async
function
            output=result.get();
            System.out.println("Inside getUserAsyncReturnTypeUsingFuture of
userController output: "+output);
        }catch (Exception e){
            System.out.println(e.toString());
        }
        return output;
    }



}
```

OUTPUT

Hiiting API:

http://localhost:8080/api/getUserAsyncReturnTypeUsingFuture

Inside getUserAsyncReturnTypeUsingFuture Current thread name: http-nio-8080-exec-1

Inside UserServices class method name performTaskAsync(): MyThread-1

Inside getUserAsyncReturnTypeUsingFuture of userController output: Async Task result

S. NO.	Method Available in Future Interface.	purpose
1.	boolean cancel(boolean mayInterruptIfRunning)	• Attempts to cancel the the execution of yhe task. • Return false, if task can notnbe canceled. • [Typically bcoz task already completed]; returns true otherwise.
2,	boolean isCanceled()	• Returns true, if task was cancelled befor it was cancelled before it got completed.
3.	boolean isDone()	• Return true if this task completed. • Completion may due to normal termination, an exception, or cancellation In all of these cases, this method will return true.
4.	V get()	• Wait if required, for the completion of the task. • Locking the caller • Waiting indefinately • Main thread will be block untill tsk execution. • After task completed, retrieve the result if available.
5.	V get(long timeout, TimeUnit unit)	• wait if required, for most of the given timeperiod. • Max it can wait depnds on timeunit. • Throws "TimeoutException" if timeout period finished and task is not yet completed.

Using CompletableFuture

- Introduced In Java 8

```
import java.util.concurrent.Future;

@Service
public class UserServices {



    @Async
    public CompleatableFuture<String> performTaskAsync(){
        System.out.println("Inside UserServices class method name performTaskAsync():
"+Thread.currentThread().getName());
        return CompleatableFuture.completedFuture("Async Task result");
    }
}
```

```
import java.util.concurrent.Future;

@RestController
@RequestMapping(value = "/api/")
public class UserController {



    @Autowired
    UserServices userServices;



    @GetMapping(path = "/getUserAsyncReturnTypeUsingFuture")
    public String getUserAsyncReturnTypeUsingFuture(){
        System.out.println("Inside getUserAsyncReturnTypeUsingFuture Current thread
name: "+Thread.currentThread().getName());
        CompletableFuture<String> result = userServices.performTaskAsync();
        String output=null;
        try {
            //get() a lot like await so here what happens it wait for the response from
async function
            output=result.get();
            System.out.println("Inside getUserAsyncReturnTypeUsingFuture of
userController output: "+output);
        }catch (Exception e){
```

```
            System.out.println(e.toString());
        }
        return output;
    }



}
```

OUTPUT

Hiiting API:

http://localhost:8080/api/getUserAsyncReturnTypeUsingFuture

Inside getUserAsyncReturnTypeUsingFuture Current thread name: http-nio-8080-exec-1

Inside UserServices class method name performTaskAsync(): MyThread-1

Inside getUserAsyncReturnTypeUsingFuture of userController output: Async Task result

Exception Handling

- If you have method of return type void and also annotated with @Async so if any operation which is running in main function thread it will proceed further without waiting for async function if any exception occurs in async function how gonna be handle this.

1. **Method Which has Return Type.**

```
import java.util.concurrent.Future;

@Service
public class UserServices {



    @Async
    public CompleatableFuture<String> performTaskAsync(){
        System.out.println("Inside UserServices class method name performTaskAsync():
"+Thread.currentThread().getName());
        return CompleatableFuture.completedFuture("Async Task result");
    }
}
```

```
import java.util.concurrent.Future;

@RestController
```

```
@RequestMapping(value = "/api/")
public class UserController {


    @Autowired
    UserServices userServices;


    @GetMapping(path = "/getUserAsyncReturnTypeUsingFuture")
    public String getUserAsyncReturnTypeUsingFuture(){
        System.out.println("Inside getUserAsyncReturnTypeUsingFuture Current thread
name: "+Thread.currentThread().getName());
        CompletableFuture<String> result = userServices.performTaskAsync();
        String output=null;
        try {
            //get() a lot like await so here what happens it wait for the response from
async function
            output=result.get();
            System.out.println("Inside getUserAsyncReturnTypeUsingFuture of
userController output: "+output);
        }catch (Exception e){
            System.out.println(e.toString());
        }
        return output;
    }



}
```

2. **Method which do not return any thing.**

```
import java.util.concurrent.Future;

@Service
public class UserServices {



    @Async
    public void performTaskAsync(){
        System.out.println("Inside UserServices class method name performTaskAsync():
"+Thread.currentThread().getName());
   );
int t=1/0;
    }
}
```

```
import java.util.concurrent.Future;

@RestController
@RequestMapping(value = "/api/")
public class UserController {


    @Autowired
    UserServices userServices;


    @GetMapping(path = "/getUserAsyncReturnTypeUsingFuture")
    public String getUserAsyncReturnTypeUsingFuture(){
        System.out.println("Inside getUserAsyncReturnTypeUsingFuture Current thread
name: "+Thread.currentThread().getName());
       userServices.performTaskAsync();

    }



}
```

Two ways to handle it.

1. **within Async Method Itself**

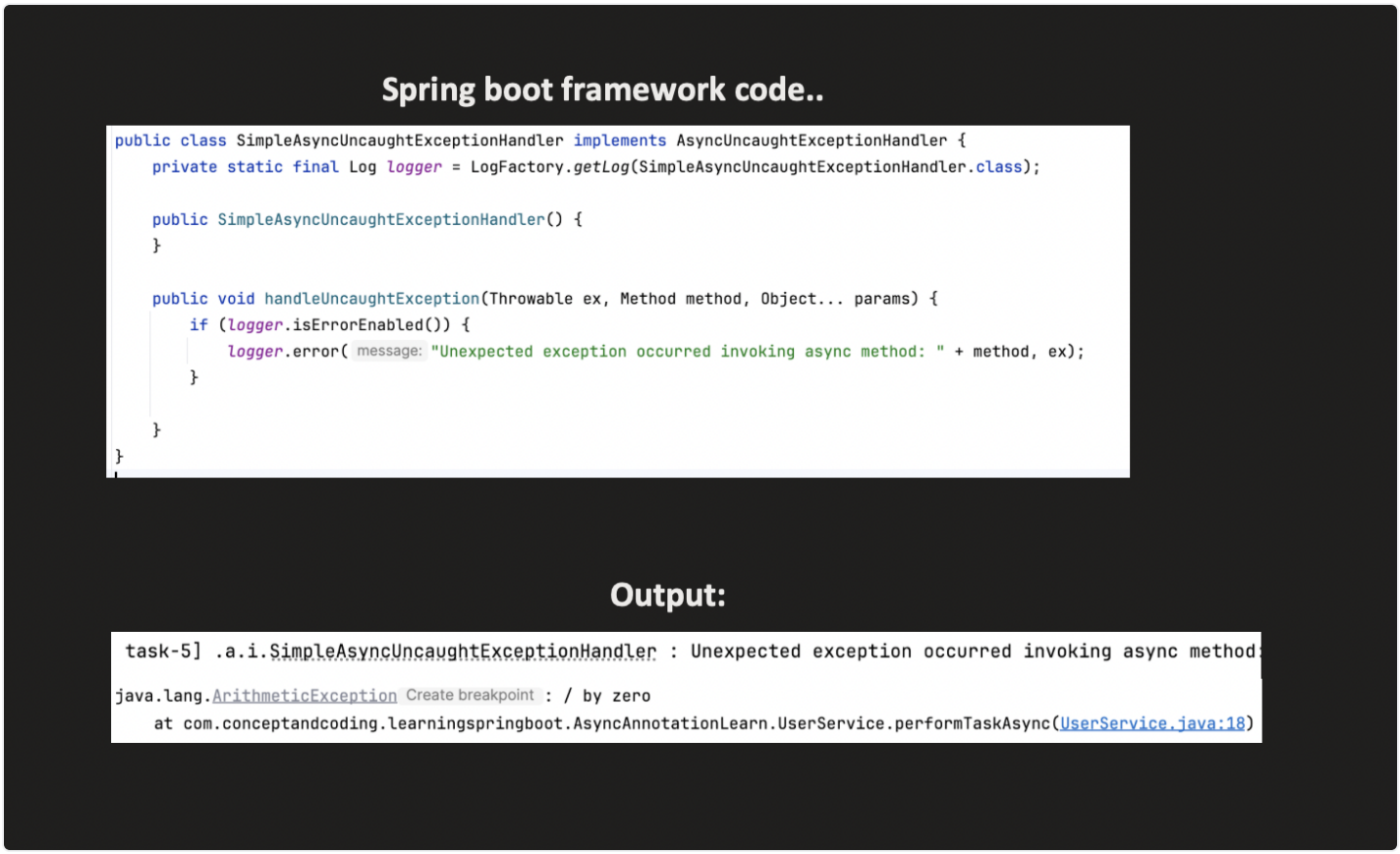
```
import java.util.concurrent.Future;

@Service
public class UserServices {
    @Async
    public void performTaskAsync(){
        System.out.println("Inside UserServices class method name performTaskAsync():
"+Thread.currentThread().getName());
   );

try{
   int t=1/0;
}Catch(Exception ex){
  //handle exception
}
    }
}
```

2. 2nd Way

- If we are not applying try catch then Spring boot called Default exception class which is **SimpleAsyncUncaugtExceptionHandler** class



Custom Handling of Async Exception

```
package com.springProject.SpringCourseProject.config;

import org.springframework.aop.interceptor.AsyncUncaughtExceptionHandler;



@Configuration
public class AppConfig implements AsyncConfigurer {


    @Autowired
    private AsyncUncaughtExceptionHandler asyncUncaughtExceptionHandler;
```

```
    @Override
    public AsyncUncaughtExceptionHandler getAsyncUncaughtExceptionHandler(){
        return this.asyncUncaughtExceptionHandler;
    }


    }


}
```

```
package com.springProject.SpringCourseProject.ExceptionHandaling;

import org.springframework.aop.interceptor.AsyncUncaughtExceptionHandler;
import org.springframework.stereotype.Component;

import java.lang.reflect.Method;

@Component
public class DefaultCustomAsyncUncaughtExceptionHandler implements
AsyncUncaughtExceptionHandler {
    @Override
    public void handleUncaughtException(Throwable ex, Method method, Object... params) {
        System.out.println("In default Uncaught Exception method. of
DefaultCustomAsyncUncaughtExceptionHandler");
        //logging can be done here
    }
}
```

```
@Async
public void performTaskAsyncForCustomExceptionHandler(){
    System.out.println("Inside UserServices class method name
performTaskAsyncForCustomExceptionHandler(): "+Thread.currentThread().getName());
    int i=0;
    System.out.println("Inside UserServices class method name
performTaskAsyncForCustomExceptionHandler(): divided by 0");
    i=5/i;
    System.out.println("Inside UserServices class method name
performTaskAsyncForCustomExceptionHandler(): i="+i);
}
```

```
@GetMapping(path = "/performTaskAsyncForCustomExceptionHandler")
public String performTaskAsyncForCustomExceptionHandler(){
    System.out.println("Inside performTaskAsyncForCustomExceptionHandler Current thread
name: "+Thread.currentThread().getName());
```

```
        userServices.performTaskAsyncForCustomExceptionHandler();
        System.out.println("performTaskAsyncForCustomExceptionHandler completed");
        return "performTaskAsyncForCustomExceptionHandler completed";
    }
```

OUTPUT

Hiiting API:

http://localhost:8080/api/performTaskAsyncForCustomExceptionHandler

Inside performTaskAsyncForCustomExceptionHandler Current thread name: http-nio-8080-exec-2

performTaskAsyncForCustomExceptionHandler completed

Inside UserServices class method name performTaskAsyncForCustomExceptionHandler(): MyThread-1

Inside UserServices class method name performTaskAsyncForCustomExceptionHandler(): divided by 0

In default Uncaught Exception method. of DefaultCustomAsyncUncaughtExceptionHandler

15. Spring Boot Async Annotation Part1:

https://notebook.zohopublic.in/public/notes/dcr5zd72620f1d2574b40a39fe1b017931423

16. Spring Boot Async Part2:

https://notebook.zohopublic.in/public/notes/dcr5z0896f8dfa8ad4dcfafa5d737b2b9c26d