DZone

# Spring Boot: Creating Asynchronous Methods Using @Async Annotation

**by Ramesh Fadatare MVB · Nov. 15, 18 · Java Zone · Tutorial**

**How do you break a Monolith into Microservices at Scale? This ebook shows strategies and techniques for building scalable and resilient microservices.**

---

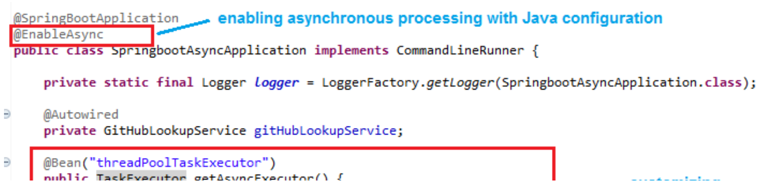
In this article, we'll explore the asynchronous execution support in Spring or Spring Boot using Spring's `@Async` annotation.

We will annotate a bean method; `@Async` will make it execute in a separate thread, i.e. the caller will not wait for the completion of the called method.

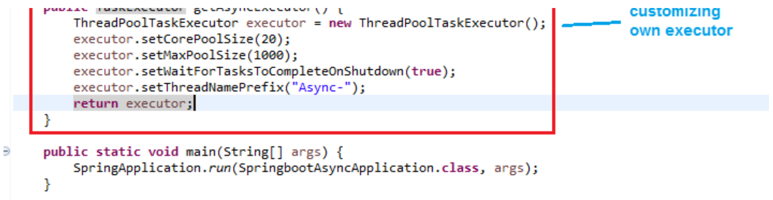
If you have been already working on a Spring or Spring Boot application, and you have a requirement to use as an asynchronous mechanism, then these three quick steps will help you get started.

## Step 1: Enable Async Support

Let's start by enabling asynchronous processing with Java configuration by simply adding the `@EnableAsync` to a configuration class:



```
@SpringBootApplication
@EnableAsync
public class SpringbootAsyncApplication implements CommandLineRunner {

    private static final Logger logger = LoggerFactory.getLogger(SpringbootAsyncApplication.class);

    @Autowired
    private GitHubLookupService gitHubLookupService;

    @Bean("threadPoolTaskExecutor")
```

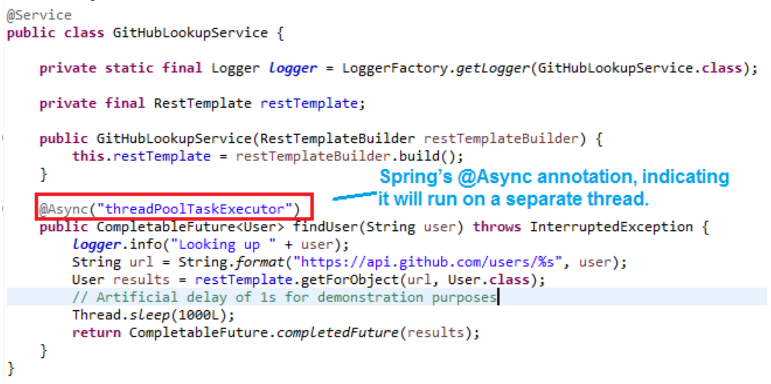


```
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(20);
        executor.setMaxPoolSize(1000);
        executor.setWaitForTasksToCompleteOnShutdown(true);
        executor.setThreadNamePrefix("Async-");
        return executor;
    }

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

The `@EnableAsync` annotation switches Spring's ability to run `@Async` methods in a background thread pool.

## Step 2: Add @Async Annotation to a Method

Make sure that the method we are annotating with `@Async` needs to be public so that it can be proxied. And, self-invocation doesn't work because it bypasses the proxy and calls the underlying method directly.



```
@Service
public class GitHubLookupService {

    private static final Logger logger = LoggerFactory.getLogger(GitHubLookupService.class);

    private final RestTemplate restTemplate;

    public GitHubLookupService(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    @Async("threadPoolTaskExecutor")
    public CompletableFuture<User> findUser(String user) throws InterruptedException {
        logger.info("Looking up " + user);
        String url = String.format("https://api.github.com/users/%s", user);
        User results = restTemplate.getForObject(url, User.class);
        // Artificial delay of 1s for demonstration purposes
        Thread.sleep(1000L);
        return CompletableFuture.completedFuture(results);
    }
}
```

Spring's @Async annotation, indicating
it will run on a separate thread.

## Step 3: Executor (Customize of Default)

Let's customize the `ThreadPoolTaskExecutor`. In our case, we want to limit the number of concurrent threads to two and limit the size of the queue to 500. There are many more things you can tune. By default, a `SimpleAsyncTaskExecutor` is used.

```
@Bean("threadPoolTaskExecutor")
    public TaskExecutor getAsyncExecutor() {
        ThreadPoolTaskExecutor executor = new T
        executor.setCorePoolSize(20);
        executor.setMaxPoolSize(1000);
        executor.setWaitForTasksToCompleteOnShu
        executor.setThreadNamePrefix("Async-");
        return executor;
    }
```

That's all, these are three quick steps that help you create asynchronous services using Spring or Spring Boot. Let's develop a complete example to demonstrate how we can create asynchronous services using Spring or Spring Boot.

---

**Learn and master in Spring Boot at Spring Boot Tutorial**

---

## What We’ll Build

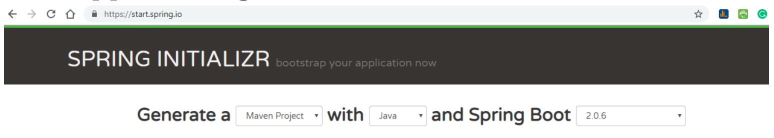
We’ll build a lookup service that queries GitHub user information and retrieves data through GitHub’s API. One approach to scaling services is to run expensive jobs in the background and wait for the results using Java’s `CompletableFuture` interface.
Java’s `CompletableFuture` is an evolution of the regular `Future`. It makes it easy to pipeline multiple asynchronous operations, merging them into a single asynchronous computation.

## Tools and Technologies Used

- Spring Boot - 2.0.6.RELEASE
- JDK - 1.8 or later
- Spring Framework - 5.0.9 RELEASE
- Maven - 3.2+
- IDE - Eclipse or Spring Tool Suite (STS)

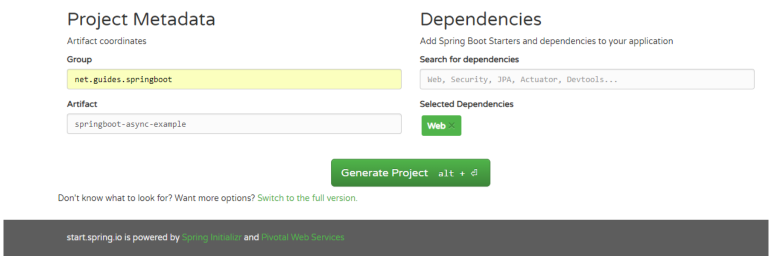
## Create and Import Spring Boot Project

There are many ways to create a Spring Boot application. The simplest way is to use Spring Initializr at http://start.spring.io/, which is an online Spring Boot application generator.


https://start.spring.io

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.0.6



Project Metadata

Artifact coordinates

Group

net.guides.springboot

Artifact

springboot-async-example

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Web ×

Generate Project alt + ⏎

Don't know what to look for? Want more options? Switch to the full version.

start.spring.io is powered by Spring Initializr and Pivotal Web Services

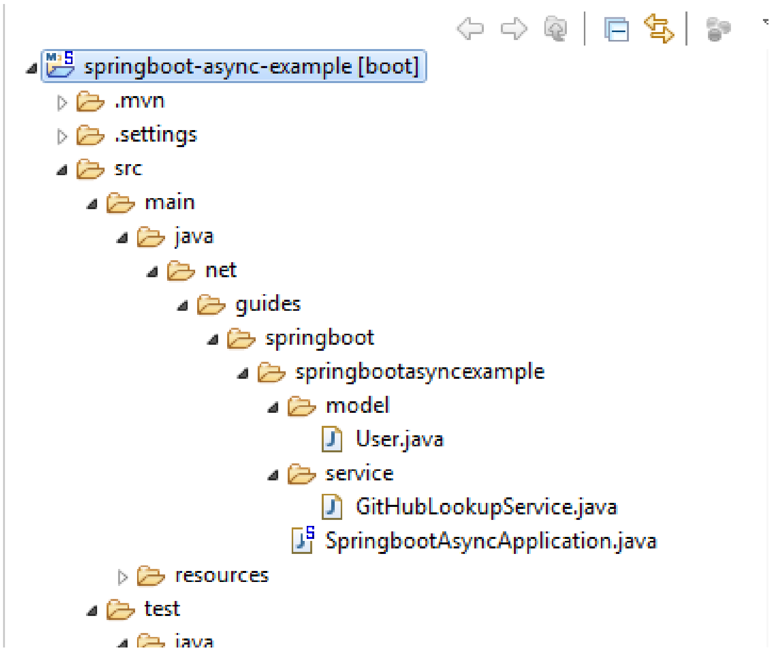
Look at the above diagram, we have specified the following details:

- Generate: Maven Project
- Java Version: 1.8 (Default)
- Spring Boot:2.0.4
- Group: net.javaguides.springboot
- Artifact: springboot-async-example
- Name: springboot-async-example
- Description: Demo project for Spring Boot
- Package Name : net.guides.springboot.springbootasyncexample
- Packaging: jar (This is the default value)
- Dependencies: Web

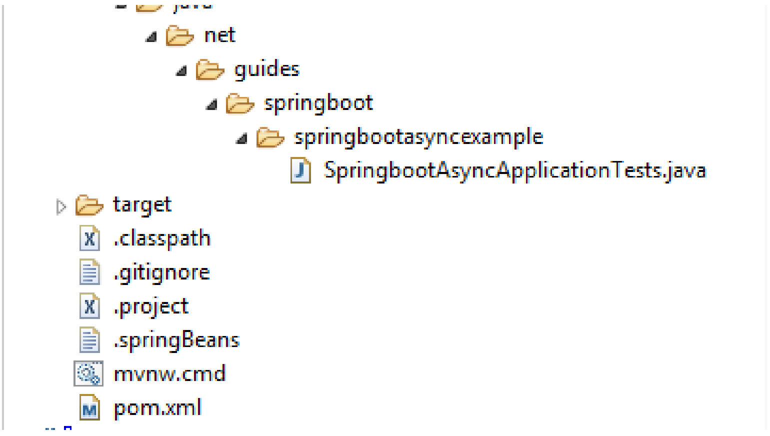
Once all the details are entered, click on the Generate Project button. It will generate a Spring Boot project and download it. Next, unzip the downloaded zip file and import it into your favorite IDE.

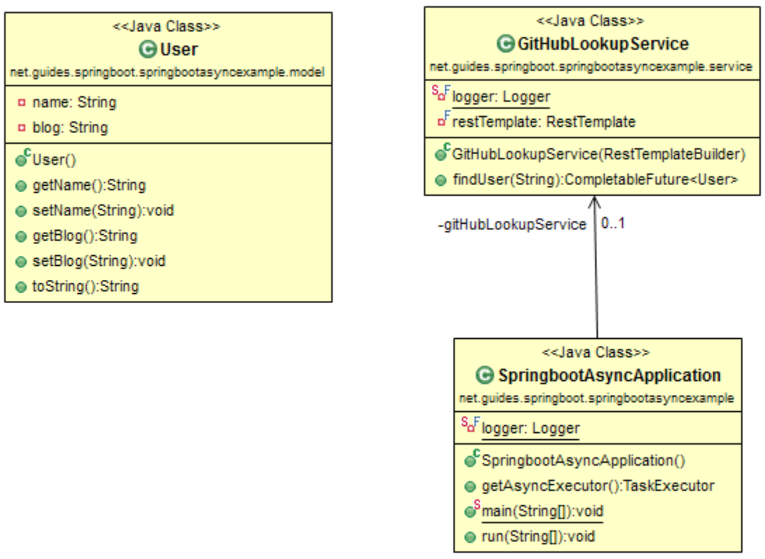
## Project Directory Structure

Below, the diagram shows a project structure for reference:



```
springboot-async-example [boot]
  .mvn
  .settings
  src
    main
      java
        net
          guides
            springboot
              springbootasyncexample
                model
                  User.java
                service
                  GitHubLookupService.java
                SpringbootAsyncApplication.java
      resources
    test
      java
```




<<Java Class>>
User
net.guides.springboot.springbootasyncexample.model
name: String
blog: String
User()
getName():String
setName(String):void
getBlog():String
setBlog(String):void
toString():String
<<Java Class>>
GitHubLookupService
net.guides.springboot.springbootasyncexample.service
logger: Logger
restTemplate: RestTemplate
GitHubLookupService(RestTemplateBuilder)
findUser(String):CompletableFuture<User>
-gitHubLookupService
0..1
<<Java Class>>
SpringbootAsyncApplication
net.guides.springboot.springbootasyncexample
logger: Logger
SpringbootAsyncApplication()
getAsyncExecutor():TaskExecutor
main(String[]):void
run(String[]):void

## The pom.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema
 xsi:schemaLocation="http://maven.apache.org/PC
    <modelVersion>4.0.0</modelVersion>
    <groupId>net.guides.springboot</groupId>
    <artifactId>springboot-async-example</artif
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>springboot-async-example</name>
    <description>Demo project for Spring Boot</
    <parent>
        <groupId>org.springframework.boot</grou
        <artifactId>spring-boot-starter-parent<
        <version>2.0.6.RELEASE</version>
        <relativePath/>
        <!-- lookup parent from repository -->
    </parent>
    <properties>
        <project.build.sourceEncoding>UTF-8</pr
        <project.reporting.outputEncoding>UTF-8
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</
            <artifactId>spring-boot-starter-web
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</
            <artifactId>spring-boot-starter-tes
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.bo
                <artifactId>spring-boot-maven-p
            </plugin>
        </plugins>
    </build>
</project>
```

## Create a Representation of a GitHub User

Let's create a GitHub User model class with name and blog fields.

```
package net.guides.springboot.springbootasyncex
```

```

import com.fasterxml.jackson.annotation.JsonIgr

@JsonIgnoreProperties(ignoreUnknown = true)
public class User {

    private String name;
    private String blog;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getBlog() {
        return blog;
    }

    public void setBlog(String blog) {
        this.blog = blog;
    }

    @Override
    public String toString() {
        return "User [name=" + name + ", blog="
    }
}
```

Note that Spring uses the Jackson JSON library to convert GitHub's JSON response into a User object. The `@JsonIgnoreProperties` annotation signals Spring to ignore any attributes not listed in the class. This makes it easy to make REST calls and produce domain objects. In this article, we are only grabbing the name and the blog URL for demonstration purposes.

## Create a GitHub Lookup Service

Next, we need to create a service that queries GitHub to find user information.

```
package net.guides.springboot.springbootasyncex

import java.util.concurrent.CompletableFuture;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.web.client.Rest
import org.springframework.scheduling.annotatic
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTempl

import net.guides.springboot.springbootasyncexa

@Service
public class GitHubLookupService {

    private static final Logger logger = Logger

    private final RestTemplate restTemplate;

    public GitHubLookupService(RestTemplateBuil
        this.restTemplate = restTemplateBuilder
    }

    @Async("threadPoolTaskExecutor")
    public CompletableFuture < User > findUser(
        logger.info("Looking up " + user);
        String url = String.format("https://api
        User results = restTemplate.getForObjec
        // Artificial delay of 1s for demonstra
        Thread.sleep(1000 L);
        return CompletableFuture.completedFutur
    }
}
```

The `GitHubLookupService` class uses Spring's `RestTemplate` to invoke a remote REST point (api.github.com/users/) and then convert the answer into a User object. Spring Boot automatically provides a `RestTemplateBuilder` that customizes the defaults with any auto-configuration bits (i.e. `MessageConverter` ). The `findUser` method is flagged with Spring's `@Async` annotation, indicating it will run on a separate thread. The method's return type is `CompletableFuture` , instead of `User` , a requirement for any asynchronous service. This code uses the `completedFuture` method to return a `CompletableFuture` instance, which is already complete with a result of the GitHub query.

## Make the Application Executable

To run a sample, you can create an executable jar. Let's use `CommandLineRunner` that injects the `GitHubLookupService` and calls that service four times to demonstrate the method is executed asynchronously.

```
package net.guides.springboot.springbootasyncex

import java.util.concurrent.CompletableFuture;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annota
import org.springframework.boot.CommandLineRunr
import org.springframework.boot.SpringApplicati
import org.springframework.boot.autoconfigure.S
import org.springframework.context.annotation.E
import org.springframework.core.task.TaskExecut
import org.springframework.core.task.TaskExecut
import org.springframework.scheduling.annotatic
import org.springframework.scheduling.concurrer

import net.guides.springboot.springbootasyncexa
import net.guides.springboot.springbootasyncexa

@SpringBootApplication
@EnableAsync
public class SpringbootAsyncApplication impleme

    private static final Logger logger = Logger

    @Autowired
    private GitHubLookupService gitHubLookupSer

    @Bean("threadPoolTaskExecutor")
    public TaskExecutor getAsyncExecutor() {
        ThreadPoolTaskExecutor executor = new T
        executor.setCorePoolSize(20);
        executor.setMaxPoolSize(1000);
        executor.setWaitForTasksToCompleteOnShu
        executor.setThreadNamePrefix("Async-");
        return executor;
    }

    public static void main(String[] args) {
        SpringApplication.run(SpringbootAsyncAp
    }

    @Override
    public void run(String...args) throws Excep
        // Start the clock
        long start = System.currentTimeMillis()
        // Kick of multiple, asynchronous looku
        CompletableFuture < User > page1 = gitH
        CompletableFuture < User > page2 = gitH
        CompletableFuture < User > page3 = gitH
        CompletableFuture < User > page4 = gitH
        // Wait until they are all done
        CompletableFuture.allOf(page1, page2, p

        // Print results, including elapsed tim
        logger.info("Elapsed time: " + (System.
        logger.info("--> " + page1.get());
        logger.info("--> " + page2.get());
        logger.info("--> " + page3.get());
        logger.info("--> " + page4.get());
    }
}
```

The `@EnableAsync` annotation switches on Spring's ability to run `@Async` methods in a background thread pool. This class also customizes the used `Executor`. In our case, we want to limit the number of concurrent threads to two and limit the size of the queue to 500. There are many more things you can tune. By default, a `SimpleAsyncTaskExecutor` is used.

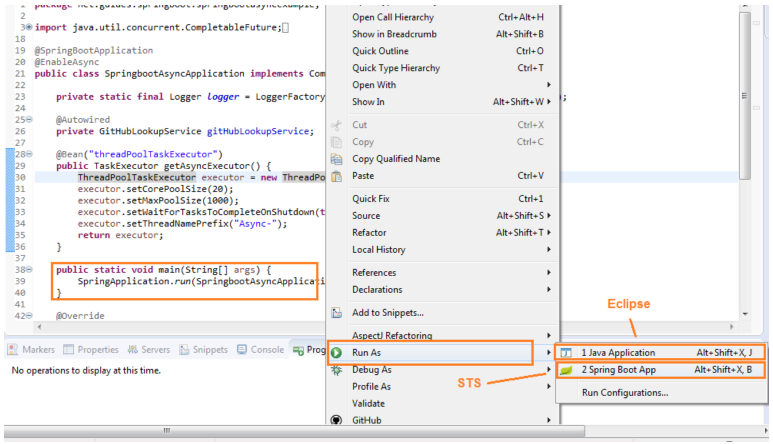
## Running Application

There are two ways we can start the standalone Spring Boot application.

- We are using Maven to run the application using ./mvnw spring-boot:run. Or, you can build the JAR file with ./mvnw clean package. Then, you can run the JAR file:

```
java -jar target/springboot-async-example.jar
```

- Below diagram shows how to run your Spring Boot application from an IDE — right click, run the `SpringbootAsyncApplication.main()` method as a standalone Java class.

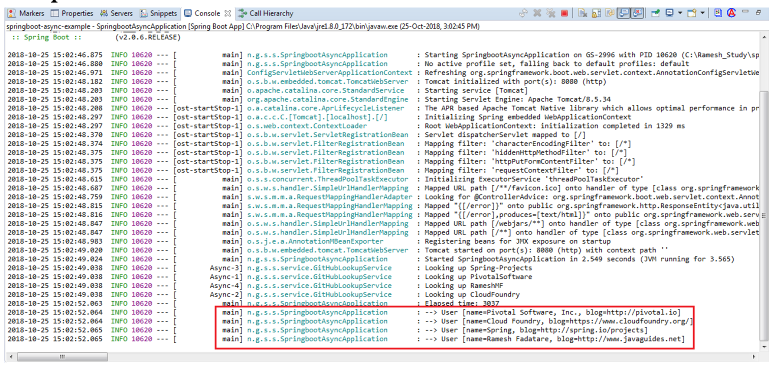


```
package net.guides.springboot.springbootasyncexample;

import java.util.concurrent.CompletableFuture;

@SpringBootApplication
@EnableAsync
public class SpringbootAsyncApplication implements Com

    private static final Logger logger = LoggerFactory

    @Autowired
    private GitHubLookupService gitHubLookupService;

    @Bean("threadPoolTaskExecutor")
    public TaskExecutor getAsyncExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPo
        executor.setCorePoolSize(20);
        executor.setMaxPoolSize(1000);
        executor.setWaitForTasksToCompleteOnShutdown(t
        executor.setThreadNamePrefix("Async-");
        return executor;
    }

    public static void main(String[] args) {
        SpringApplication.run(SpringbootAsyncApplicati
    }

    @Override
```

No operations to display at this time.

## Output

When we run the application, we will see the following output:



```
springboot-async-example - SpringbootAsyncApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_172\bin\javaw.exe (25-Oct-2018, 3:02:45 PM)
 :: Spring Boot ::        (v2.0.6.RELEASE)

2018-10-25 15:02:46.875  INFO 10620 --- [           main] n.g.s.s.SpringbootAsyncApplication       : Starting SpringbootAsyncApplication on GS-2996 with PID 10620 (C:\Ramesh_Study\sp
2018-10-25 15:02:46.880  INFO 10620 --- [           main] n.g.s.s.SpringbootAsyncApplication       : No active profile set, falling back to default profiles: default
2018-10-25 15:02:46.971  INFO 10620 --- [           main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationConfigServletWe
2018-10-25 15:02:48.182  INFO 10620 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2018-10-25 15:02:48.203  INFO 10620 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2018-10-25 15:02:48.203  INFO 10620 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.34
2018-10-25 15:02:48.208  INFO 10620 --- [ost-startStop-1] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native library which allows optimal performance in pr
2018-10-25 15:02:48.297  INFO 10620 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2018-10-25 15:02:48.297  INFO 10620 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 1329 ms
2018-10-25 15:02:48.370  INFO 10620 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean  : Servlet dispatcherServlet mapped to [/]
2018-10-25 15:02:48.374  INFO 10620 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
2018-10-25 15:02:48.375  INFO 10620 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2018-10-25 15:02:48.375  INFO 10620 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2018-10-25 15:02:48.375  INFO 10620 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2018-10-25 15:02:48.615  INFO 10620 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'threadPoolTaskExecutor'
2018-10-25 15:02:48.687  INFO 10620 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework
2018-10-25 15:02:48.759  INFO 10620 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet.context.Annot
2018-10-25 15:02:48.815  INFO 10620 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util
2018-10-25 15:02:48.816  INFO 10620 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.serv
2018-10-25 15:02:48.847  INFO 10620 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web
2018-10-25 15:02:48.847  INFO 10620 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet
2018-10-25 15:02:48.983  INFO 10620 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2018-10-25 15:02:49.020  INFO 10620 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2018-10-25 15:02:49.024  INFO 10620 --- [           main] n.g.s.s.SpringbootAsyncApplication       : Started SpringbootAsyncApplication in 2.549 seconds (JVM running for 3.565)
2018-10-25 15:02:49.038  INFO 10620 --- [        Async-3] n.g.s.s.service.GitHubLookupService      : Looking up Spring-Projects
2018-10-25 15:02:49.038  INFO 10620 --- [        Async-1] n.g.s.s.service.GitHubLookupService      : Looking up PivotalSoftware
2018-10-25 15:02:49.038  INFO 10620 --- [        Async-4] n.g.s.s.service.GitHubLookupService      : Looking up RameshMF
2018-10-25 15:02:49.038  INFO 10620 --- [        Async-2] n.g.s.s.service.GitHubLookupService      : Looking up CloudFoundry
2018-10-25 15:02:52.063  INFO 10620 --- [           main] n.g.s.s.SpringbootAsyncApplication       : Elapsed time: 3037
2018-10-25 15:02:52.064  INFO 10620 --- [           main] n.g.s.s.SpringbootAsyncApplication       : --> User [name=Pivotal Software, Inc., blog=http://pivotal.io]
2018-10-25 15:02:52.064  INFO 10620 --- [           main] n.g.s.s.SpringbootAsyncApplication       : --> User [name=Cloud Foundry, blog=https://www.cloudfoundry.org/]
2018-10-25 15:02:52.065  INFO 10620 --- [           main] n.g.s.s.SpringbootAsyncApplication       : --> User [name=Spring, blog=http://spring.io/projects]
2018-10-25 15:02:52.065  INFO 10620 --- [           main] n.g.s.s.SpringbootAsyncApplication       : --> User [name=Ramesh Fadatare, blog=http://www.javaguides.net]
```

---

**Learn and master in Spring Boot at Spring Boot Tutorial**

---

The source code of this article available on my GitHub repository.

## References

- https://spring.io/guides/gs/async-method/
- https://www.baeldung.com/spring-async
- Spring Boot Tutorial

---

### How do you break a Monolith into Microservices at Scale? This ebook shows strategies and techniques for building scalable and resilient

**techniques for building scalable and resilient microservices.**

Topics: JAVA , SPRING BOOT , TUTORIAL , MAVEN , XML , IDE , ECLIPSE , SPRING TOOL SUITE , ASYNC

## Java Partner Resources