As heap utilization approaches 100% and the inevitable OutOfMemoryError threatens to crash the production JVM, DevOps professionals who tend real-time Java applications are wont to ask for a mechanism to force Java garbage collection and free up a few megabytes in memory.

Unfortunately, this desire to immediately free up memory must go unrequited, as there is no way to force garbage collection (GC) in Java. However, here are five strategies that will get the Java Virtual Machine ([JVM](https://www.theserverside.com/definition/Java-virtual-machine-JVM)) to prioritize the task.

### 1. Call System.gc()

Developers can call System.gc() anywhere in their code to instruct the JVM to prioritize garbage collection. When a developer calls this method -- and there isn't an extreme load on the JVM -- a Java GC cycle will happen within seconds.

System.gc();

### 2. Call Runtime.getRuntime().gc()

Another option is to use the Runtime.getRuntime().gc() call. This is the second function the [JDK](https://www.theserverside.com/definition/Java-Development-Kit-JDK) provides for developers who want to force Java garbage collection. The Runtime.getRuntime().gc() call actually just invokes the method System.gc() behind the scenes. As a result, the two method calls are exactly the same.

Runtime.getRuntime().gc();

### 3. Use jmap to force GC

The Java Memory Map (JMAP) utility has a method that prints a histogram of the Java heap. One side effect of the jmap command is that when it's called, it forces a garbage collection routine. However, this isn't a foolproof way to force Java garbage collection. If the JVM is busy and a GC cycle cannot be performed, the command will error out.

$ jmap -histo:live 7544

You can try to force Java garbage collection with the JDK's JMAP and JCMD utilities.

### 4. Command line Java GC with jcmd

The Java diagnostic command (JCMD) is another [JDK utility](https://www.theserverside.com/video/Java-command-line-tools-every-JDK-developer-should-know) that will trigger a garbage collection routine if the JVM can safely schedule a stop-the-world pause. If not, this command will error out in the same manner as the jmap utility.

$ jcmd 7544 GC.run

### 5. Use JConsole or Java Mission Control

Both JConsole and Java Mission Control provide a user-friendly interface that interacts with the Java diagnostic command utility, which can be used to force Java garbage collection. The JConsole [monitoring tool](https://openjdk.java.net/tools/svc/jconsole/) provides a button on its memory management page that says Run Garbage Collection. Java Mission Control allows developers to select any jcmd switch -- including GC.run -- and execute the command at the click of a button.

However, these tools don't do anything unique in terms of Java garbage collection. They simply invoke the jcmd utility behind the scenes.

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Spring Boot ViewControllerRegistry

last modified July 6, 2020

Spring Boot ViewControllerRegistry tutorial shows how to use ViewControllerRegistry to create simple routes.

Spring is a popular Java application framework for creating enterprise applications. Spring Boot is an evolution of Spring framework which helps create stand-alone, production-grade Spring based applications with minimal effort.

## ViewControllerRegistry

ViewControllerRegistry allows to create simple automated controllers pre-configured with status code and/or a view.

## Spring Boot ViewControllerRegistry example

In the following example we create a simple route with ViewControllerRegistry.

pom.xml

src

├───main

│ ├───java

│ │ └───com

│ │ └───zetcode

│ │ │ Application.java

│ │ └───config

│ │ AppConfig.java

│ └───resources

│ ├───static

│ │ index.html

│ └───templates

│ hello.html

└───test

└───java

This is the project structure.

pom.xml

<?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-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.zetcode</groupId>

<artifactId>viewregistry</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.1.RELEASE</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Spring Boot starters are a set of convenient dependency descriptors which greatly simplify Maven configuration. The spring-boot-starter-parent has some common configurations for a Spring Boot application. The spring-boot-starter-web is a starter for building web, including RESTful, applications using Spring MVC. It uses Tomcat as the default embedded container. The spring-boot-starter-thymeleaf is a starter for building MVC web applications using Thymeleaf views.

The spring-boot-maven-plugin provides Spring Boot support in Maven, allowing us to package executable JAR or WAR archives. Its spring-boot:run goal runs the Spring Boot application.

com/zetcode/config/AppConfig.java

package com.zetcode.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class AppConfig implements WebMvcConfigurer {

@Override

public void addViewControllers(ViewControllerRegistry registry) {

registry.addViewController("/hello").setViewName("hello");

}

}

In the AppConfig we register a new route with ViewControllerRegistry's addViewController() method.

resources/templates/hello.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Hello page</title>

</head>

<body>

<p>

Hello there

</p>

</body>

</html>

The hello.html view displays a simple message.

resources/static/index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Home page</title>

</head>

<body>

<p>

This is home page. Go to <a href="hello">hello page</a>

</p>

</body>

</html>

This is a home page.

com/zetcode/Application.java

package com.zetcode;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

The Application sets up the Spring Boot application. The @SpringBootApplication enables auto-configuration and component scanning.

$ mvn spring-boot:run

After the application is run, we can navigate to localhost:8080/.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In my [previous post](http://blog.springsource.org/2013/07/02/spring-security-java-config-preview-introduction/), I introduced Spring Security Java configuration and discussed some of the logistics of the project. In this post, we will start off by walking through a very simple web security configuration. We will then spice things up a bit with configuration that has been customized some.

**Hello Web Security**

In this section we go through the most basic configuration for web based security. It can be broken into four steps:

* Updating your dependencies - [we demonstrated this using Maven](http://blog.springsource.org/2013/07/02/spring-security-java-config-preview-introduction/" \l "availability) in our previous blog post
* Provide the Spring Security configuration - in our example this is done with a [WebSecurityConfigurerAdapter](https://spring.io/blog/2013/07/03/spring-security-java-config-preview-web-security/" \l "wsca)
* Ensure the Spring Security configuration is loaded - in our example this is done with [AbstractAnnotationConfigDispatcherServletInitializer](https://spring.io/blog/2013/07/03/spring-security-java-config-preview-web-security/" \l "aacdsi)
* Configure the springSecurityFilterChain - in our example this is done with [AbstractSecurityWebApplicationInitializer](https://spring.io/blog/2013/07/03/spring-security-java-config-preview-web-security/aswai)

**WebSecurityConfigurerAdapter**

The @EnableWebSecurity annotation and WebSecurityConfigurerAdapter work together to provide web based security. By extending WebSecurityConfigurerAdapter and only a few lines of code we are able to do the following:

* Require the user to be authenticated prior to accessing any URL within our application
* Create a user with the username “user”, password “password”, and role of “ROLE\_USER”
* Enables HTTP Basic and Form based authentication
* Spring Security will automatically render a login page and logout success page for you

@Configuration

@EnableWebSecurity

public class HelloWebSecurityConfiguration

extends WebSecurityConfigurerAdapter {

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) {

auth

.inMemoryAuthentication()

.withUser("user").password("password").roles("USER");

}

}

For your reference, this is similar to the following XML configuration with a few exceptions:

* Spring Security will render the login, authentication failure url, and logout success URLs
* The login-processing-url will only be processed for HTTP POST
* The login-page will only be processed for HTTP GET

<http use-expressions="true">

<intercept-url pattern="/\*\*" access="authenticated"/>

<logout

logout-success-url="/login?logout"

logout-url="/logout"

/>

<form-login

authentication-failure-url="/login?error"

login-page="/login"

login-processing-url="/login"

password-parameter="password"

username-parameter="username"

/>

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name="user"

password="password"

authorities="ROLE\_USER"/>

</user-service>

</authentication-provider>

</authentication-manager>

**AbstractAnnotationConfigDispatcherServletInitializer**

The next step is to ensure that the root ApplicationContext includes the [HelloWebSecurityConfiguration we just specified](https://spring.io/blog/2013/07/03/spring-security-java-config-preview-web-security/" \l "web-security-hellowebsecurityconfiguration). There are [many different ways](http://static.springsource.org/spring/docs/current/javadoc-api/org/springframework/web/WebApplicationInitializer.html) we could do this, but if you are using Spring’s [AbstractAnnotationConfigDispatcherServletInitializer](http://static.springsource.org/spring/docs/current/spring-framework-reference/htmlsingle/" \l "mvc-container-config) it might look something like this:

public class SpringWebMvcInitializer extends

AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class<?>[] getRootConfigClasses() {

return new Class[] { HelloWebSecurityConfiguration.class };

}

...

}

To put this in perspective, Spring Security was traditionally initialized using something similar to the following lines within the web.xml:

<!-- Creates the Spring Container shared by all Servlets and Filters -->

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<!-- Load all Spring XML configuration including our security.xml file -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/\*.xml</param-value>

</context-param>

“Ordering of WebApplicationInitializer”

If any servlet Filter mappings are added after AbstractSecurityWebApplicationInitializer is invoked, they might be accidentally added before springSecurityFilterChain. Unless an application contains Filter instances that do not need to be secured, springSecurityFilterChain should be before any other Filter mappings. The @Order annotation can be used to help ensure that any WebApplicationInitializer is loaded in a deterministic order.

**AbstractSecurity WebApplicationInitializer**

The last step is we need to map the springSecurityFilterChain. We can easily do this by extending [AbstractSecurityWebApplicationInitializer](http://static.springsource.org/spring-security/site/docs/3.2.x/apidocs/org/springframework/security/web/context/AbstractSecurityWebApplicationInitializer.html) and optionally overriding methods to customize the mapping.

The most basic example below accepts the default mapping and adds springSecurityFilterChain with the following characteristics:

* springSecurityFilterChain is mapped to “/\*”
* springSecurityFilterChain uses the dispatch types of ERROR and REQUEST
* The springSecurityFilterChain mapping is inserted before any servlet Filter mappings that have already been configured

public class SecurityWebApplicationInitializer

extends AbstractSecurityWebApplicationInitializer {

}

The above code is the equivalent of the following lines within the web.xml:

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>

org.springframework.web.filter.DelegatingFilterProxy

</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

<dispatcher>ERROR</dispatcher>

<dispatcher>REQUEST</dispatcher>

</filter-mapping>

**CustomWebSecurityConfigurerAdapter**

Our [HelloWebSecurityConfiguration](https://spring.io/blog/2013/07/03/spring-security-java-config-preview-web-security/#web-security-hellowebsecurityconfiguration) sample, demonstrates that Spring Security Java configuration can provide some very nice defaults for us. Let’s take a look at some basic customization.

@EnableWebSecurity

@Configuration

public class CustomWebSecurityConfigurerAdapter extends

WebSecurityConfigurerAdapter {

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) {

auth

.inMemoryAuthentication()

.withUser("user") // #1

.password("password")

.roles("USER")

.and()

.withUser("admin") // #2

.password("password")

.roles("ADMIN","USER");

}

@Override

public void configure(WebSecurity web) throws Exception {

web

.ignoring()

.antMatchers("/resources/\*\*"); // #3

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeUrls()

.antMatchers("/signup","/about").permitAll() // #4

.antMatchers("/admin/\*\*").hasRole("ADMIN") // #6

.anyRequest().authenticated() // 7

.and()

.formLogin() // #8

.loginUrl("/login") // #9

.permitAll(); // #5

}

}

Assuming that we adjust [AbstractAnnotationConfigDispatcherServletInitializer](https://spring.io/blog/2013/07/03/spring-security-java-config-preview-web-security/" \l "aacdsi) to load our new configuration, our CustomWebSecurityConfigurerAdapter will do the following:

* Allow in memory authentication with a user named “user”
* Allow in memory authentication with an administrative user named “admin”
* Ignore any request that starts with “/resources/”. This is similar to configuring [http@security=none](mailto:http@security=none) when using the XML namespace configuration.
* Allow anyone (including unauthenticated users) to access to the URLs “/signup” and “/about”
* Allow anyone (including unauthenticated users) to access to the URLs “/login” and “/login?error”. The permitAll() in this case means, allow access to any URL that formLogin() uses.
* Any URL that starts with “/admin/” must be an administrative user. For our example, that would be the user “admin”.
* All remaining URLs require that the user be successfully authenticated
* Setup form based authentication using the Java configuration defaults. Authentication is performed when a POST is submitted to the URL “/login” with the parameters “username” and “password”.
* Explicitly state the login page, which means the developer is required to render the login page when **GET /login** is requested.

For those that are familiar with the XML based configuration, the configuration above is very similar to the following XML configuration:

<http security="none" pattern="/resources/\*\*"/>

<http use-expressions="true">

<intercept-url pattern="/logout" access="permitAll"/>

<intercept-url pattern="/login" access="permitAll"/>

<intercept-url pattern="/signup" access="permitAll"/>

<intercept-url pattern="/about" access="permitAll"/>

<intercept-url pattern="/\*\*" access="hasRole('ROLE\_USER')"/>

<logout

logout-success-url="/login?logout"

logout-url="/logout"

/>

<form-login

authentication-failure-url="/login?error"

login-page="/login"

login-processing-url="/login"

password-parameter="password"

username-parameter="username"

/>

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name="user"

password="password"

authorities="ROLE\_USER"/>

<user name="admin"

password="password"

authorities="ROLE\_USER,ROLE\_ADMIN"/>

</user-service>

</authentication-provider>

</authentication-manager>

**Similarities to the XML Namespace**

After looking at our slightly more complicated example, you might be able to find some similarities between the XML namespace and the Java configuration. Here are some of the more useful points:

* HttpSecurity is quite similar to the http namespace element. It allows configuring web based security for a certain selection (in this case all) requests.
* WebSecurity is quite similar to any Security namespace elements that are for the web and that do not require a parent (i.e. security=none, debug, etc). It allows configuring things that impact all of web security.
* WebSecurityConfigurerAdapter is a convenience class that allows customization to both WebSecurity and HttpSecurity. We can extend WebSecurityConfigurerAdapter multiple times (in distinct objects) to replicate the behavior of having multiple http elements.
* By formatting our Java configuration code it is much easier to read. It can be read similar to the XML namespace equivalent where “and()” represents optionally closing an XML element.

**Differences to the XML Namespace**

You will notice that there are some important differences between the XML and Java configuration too.

* When creating our users in [#1 and #2](https://spring.io/blog/2013/07/03/spring-security-java-config-preview-web-security/#cwsca), we do not specify “ROLE\_” as we would with the XML configuration. Since this convention is so common, the “roles” method automatically adds “ROLE\_” for you. If you did not want “ROLE\_” added you could use the authorities method instead.
* Java configuration has different defaults URLs and parameters. Keep this in mind when creating custom login pages. The result is that our URLs are more RESTful. Additionally, it is not quite so obvious we are using Spring Security which helps to prevent [information leaks](https://www.owasp.org/index.php/Information_Leak_(information_disclosure)). For example:
* **GET /login** renders the login page instead of **/spring\_security\_login**
* **POST /login** authenticates the user instead of **/j\_spring\_security\_check**
* The username parameter defaults to **username** instead of **j\_username**
* The password parameter defaults to **password** instead of **j\_password**
* Java configuration can easily map multiple request matchers to the same roles. This is apparent in [#4](https://spring.io/blog/2013/07/03/spring-security-java-config-preview-web-security/#cwsca) where we map two URLs to be accessible to anyone
* Java configuration tries to remove redundant code. For example, instead of repeating our /login URL in the form-login element and the intercept-url element as we did with the XML, we can simply declare that users should have access to any URL related to formLogin() as shown with [#5](https://spring.io/blog/2013/07/03/spring-security-java-config-preview-web-security/#cwsca)
* When mapping HTTP requests using the hasRole method as we did in [#6](https://spring.io/blog/2013/07/03/spring-security-java-config-preview-web-security/#cwsca), we do not need to specify “ROLE\_” as we would in XML. Again, this is so common of a convention that the hasRole method automatically adds “ROLE\_” for you. If you did not want to automatically prefix with “ROLE\_”, you could use the “access” method.

**Additional Web Samples**

**Sample Compatibility** Since the code was merged into Spring Security 3.2 M2 with no changes, the samples will be compatible with either the stand alone module or spring-security-config-3.2.0.M2+

We have given a few examples of how the Spring Security Java configuration can be used to secure your web application in order to wet your appetite. Below you can find a number of resources with additional samples.

* There are plenty of samples in the [HttpSecurity Javadoc](http://static.springsource.org/spring-security/site/docs/3.2.x/apidocs/org/springframework/security/config/annotation/web/builders/HttpSecurity.html). Be sure to check out the Javadoc on individual methods which gives examples for how to do things like [openid](http://static.springsource.org/spring-security/site/docs/3.2.x/apidocs/org/springframework/security/config/annotation/web/builders/HttpSecurity.html#openidLogin()), [remember me](http://static.springsource.org/spring-security/site/docs/3.2.x/apidocs/org/springframework/security/config/annotation/web/builders/HttpSecurity.html#rememberMe()), etc.
* [Web Samples](https://github.com/spring-projects/spring-security/tree/master/samples)

**XML namespace to Java Config**

If you are having trouble converting from the XML namespace to the Java configuration, you can refer to the tests. The convention is that the test for a given XML element will start with “Namespace”, contain the XML element name, and end with “Tests”. For example, to learn how the http element maps to Java configuration, you would refer to the [NamespaceHttpTests](https://github.com/SpringSource/spring-security-javaconfig/blob/master/spring-security-javaconfig/src/test/groovy/org/springframework/security/config/annotation/web/builders/NamespaceHttpTests.groovy). Another example, is that you can figure out how the remember-me namespace maps to Java configuration in the [NamespaceRememberMeTests](https://github.com/SpringSource/spring-security-javaconfig/blob/master/spring-security-javaconfig/src/test/groovy/org/springframework/security/config/annotation/web/configurers/NamespaceRememberMeTests.groovy).

**Feedback Please**

If you encounter a bug, have an idea for improvement, etc please do not hesitate to bring it up! We want to hear your thoughts so we can ensure we get it right before the code is generally available. Trying out new features early is a good and simple way to give back to the community. This also ensures that the features you want are present and working as you think they should

Please log any issues or feature requests to the [Spring Security JIRA](https://jira.springsource.org/browse/SEC) under the category “Java Config”. After logging a JIRA, we encourage (but do not require) you to submit your changes in a pull request. You can read more about how to do this in the [Contributor Guidelines](https://github.com/SpringSource/spring-security/blob/master/CONTRIBUTING.md)

If you have questions on how to do something, please use the [Spring Security forums](http://forum.springsource.org/forumdisplay.php?33-Security) or [Stack Overflow with the tag spring-security](https://stackoverflow.com/questions/tagged/spring-security) (I will be monitoring them closely). If you have specific comments questions about this blog, feel free to leave a comment. Using the appropriate tools will help make it easier for everyone.

**Conclusion**

You should have a fairly good idea of how to use Spring Security Java configuration for web based security. In the [next post](http://blog.springsource.org/2013/07/04/spring-security-java-config-preview-method-security/), we will take a look at how to setup method based security with Java configuration.

Get the Spring newsletter

Top of Form

Yes, I would like to be contacted by The Spring Team and VMware for newsletters, promotions and events

Bottom of Form