SpringBoot

【IDE】STS(Stping Tool Suite) 相關

效能調教

STS卡頓（一次STS IDE 優化調優記錄） - 台部落 (twblogs.net)

JVM参□□介：

-Xmx1200m 最大堆□存，一般设置□物理□存的1/4。
-Xms256m 初始堆□存。
-Xmn128m 年□代堆□存。
-XX:PermSize=64m 持久代堆的初始大小
-XX:MaxPermSize=256m 持久代堆的最大大小
年□代堆□存 □象□□建出□□存放在□里
年老代堆□存 □象在被真正会回收之前会存放在□里
持久代堆□存 元□据等存放在□里
堆□存 年□代堆□存 + 年老代堆□存 + 持久代堆□存

原文□接：https://blog.csdn.net/Hello_World_QWP/article/details/83302530

Maven Proxy 設定

setting.xml

```
<proxies>
   <!-- proxy
    | Specification for one proxy, to be used in connecting to the network.
    |
   <proxy>
     <id>optional</id>
     <active>true</active>
     <protocol>http</protocol>
     <username>proxyuser</username>
     <password>proxypass</password>
     <host>proxy.host.net</host>
     <port>80</port>
     <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
   </proxy>
   -->
 <proxy>
     <id>optional</id>
     <active>true</active>
     <protocol>http</protocol>
     <host>192.168.1.1</host>
     <port>3128</port>
   </proxy>
 <proxy>
     <id>optional2</id>
     <active>true</active>
     <protocol>https</protocol>
     <host>192.168.1.1</host>
     <port>3128</port>
   </proxy>
 </proxies>
```

【SpringBoot】 annotation

以下是常用的 Spring Boot Annotation 的說明

- `@SpringBootApplication`：用於標記Spring Boot應用程序的主要類，它結合了`@Configuration`、`@EnableAutoConfiguration`、`@ComponentScan`三個註解的功能。

- @Controller：用來標記一個類，表示這是一個Controller，可以處理HTTP請求。
- `@RestController`：用於標記一個控制器類，並表示該類中所有的方法都會以JSON格式返回結果。
- `@RequestMapping`：使用標記Controller中的方法，並指定該方法的URL路徑。
  - @GetMapping：類似於@RequestMapping，但只是處理HTTP GET請求。
  - @PostMapping：類似於@RequestMapping，但只是處理HTTP POST請求。
  - @PutMapping：類似於@RequestMapping，但只是處理HTTP PUT請求。
  - @DeleteMapping：類似@RequestMapping，但只是處理HTTP DELETE請求。

- @PathVariable：用來獲取URL路徑上的變量值。
- @RequestParam：用來獲取HTTP請求中的參數值。
- @RequestBody：用於獲取HTTP請求中的請求體，一般用於處理POST請求。
- @ResponseBody：使用指定方法返回值的格式，可以是JSON、XML或者其他格式
- `@Autowired`：用於將依賴註冊到Spring容器中，可以在成員變量、構造函數數、Setter方法中使用。
- `@Value`：用於獲取配置文件中的值，例如數據庫連接信息、服務端接口等。
- `@Configuration`：用於標記一個配置類，配置類中通常包含@Bean註解的方法，用於提供Bean的定義。
- `@Bean`：用於將方法返回的對象註冊為Bean，Spring容器管理該Bean的生命週期。
- `@EnableAutoConfiguration`：用於啟動Spring Boot的自動配置功能，Spring Boot會根據當前的classpath和配置文件，自動配置所需的Bean。
- `@ComponentScan`：用於指定需要掃描的包，Spring Boot會掃描該包及其子包下面所有的@Component註解，並將其註冊為Bean。
- `@Transactional`：用於標記一個方法或類需要事務管理，Spring Boot會根據@Transactional註解的配置，自動創建事務，並管理其生命週期。
- `@Profile`：使用指定Bean的配置文件的Profile，不同的Profile可以在不同的環境下使用不同的配置。
- `@Async`：用於標記一個方法為異常方法，Spring Boot會自動將方法放入線程池中執行

@SpringBootApplication

掛有@SpringBootApplication

main程式為執行時的進入點。同時也是主要的配置類別。

會掃描所在類別的package及其子package中掛有

@Component
@Controller
@Service
@Repository

等component類別並註冊為spring bean。

包含三種 annotation

@EnableAutoConfiguration
@ComponentScan
@Configuration

【SpringBoot】@Scheduled 定時執行

在Spring Boot中，我□可以使用注□@Scheduled□定□□行方法。□注□可用于□方法□□□需要定期□行的方法，并提供□行□划。

@Scheduled注□支持以下属性：

1. fixedRate：以毫秒□□位的□行速率。
2. fixedDelay：以毫秒□□位的□行延□，即上一次□行□束后的延□□间。
3. initialDelay：第一次□行的延□□间，以毫秒□□位。
4. cron：CRON表达式，用于更高□的定□□划。

下面是一个□□的示例，□示例演示如何使用@Scheduled注□在Spring Boot中定期□行方法：

```
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
public class MyTask {

@Scheduled(fixedRate = 5000)
public void printMessage() {
System.out.println("Hello, world!");
}
}
```

在上述示例中，我□定义了一个名□MyTask的□件，其中包含一个使用@Scheduled注□□□的printMessage方法。此方法□在每隔5秒钟输出一条消息"Hello, world!"。

我□□可以使用fixedDelay属性□延□□行。例如，如果我□□fixedDelay设置□10000，那么□方法□在上一次□行□束后等待10秒才能□始下一次□行。

@Scheduled注□□支持CRON表达式，□使得我□可以更高□地安排任务。例如，如果我□想要在每天早上6点□行任务，我□可以使用以下CRON表达式：

```
0 0 6 * * *
```

在此示例中，我□使用六个星号□指定秒，分，□，日，月和星期几。因此，此表达式□在每天早上6点□行任务。

註解停用

如果你想□□停用@Scheduled注□□□的方法，你可以□□地注□掉它，或者□其□□□禁用，以便稍后重新启用。

要注□掉方法，□□其注□掉：

```
// @Scheduled(fixedRate = 5000)
public void printMessage() {
    System.out.println("Hello, world!");
}
```

要禁用方法，您可以使用@Scheduled注□的enabled属性：

```
// @Scheduled(fixedRate = 5000)
public void printMessage() {
    System.out.println("Hello, world!");
}
```

在□个例子中，我□使用@Scheduled注□的enabled属性□printMessage方法□□□已禁用。要重新启用□方法，只需□enabled属性设置□true即可。

使用命令停用

是的，您可以使用Spring Boot Actuator的端点□动□启用/禁用定□任务。Spring Boot Actuator是Spring Boot的一个附加模块，提供了很多有用的端点，可以在□用程序运行□□视和管理□用程序。

您需要在□目中添加Actuator依□□：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

然后，您可以使用`/actuator/scheduledtasks`端点□□查和管理定□任务。要□□停用定□任务，您可以使用□端点□取消□划的任务。例如：

```
@Scheduled(fixedRate = 5000, name = "myTask")
public void printMessage() {
    System.out.println("Hello, world!");
}
```

```
$ curl -X POST http://localhost:8080/actuator/scheduledtasks/taskName
```

□□取消□划名□taskName的任务。要重新启用□任务，您可以使用□端点□重新启用任务：

```
$ curl -X DELETE http://localhost:8080/actuator/scheduledtasks/taskName
```

假设您的Spring Boot□用程序正在运行，并且您已经添加了Actuator依□□，那么您可以使用以下curl命令向`/actuator/scheduledtasks`端点发出HTTP GET□求□□取□前□用程序中所有定□任务的列表：

```
curl http://localhost:8080/actuator/scheduledtasks
```

□□返回一个JSON格式的响□，其中包含有□每个定□任务的信息，例如任务名□、任务是否启用、任务的CRON表达式等。您可以使用□响□□查看□用程序中所有的定□任务信息。

```
{
  "cron": {
    "myScheduledTask": {
      "scheduled": true,
      "expression": "0/30 * * * * ?"
    }
  },
  "fixedDelay": {},
  "fixedRate": {}
}
```

【SpringBoot】 監控工具 Actuator

參考: https://kucw.github.io/blog/2020/7/spring-actuator/
官方文件: https://docs.spring.io/spring-boot/docs/current/actuator-api/htmlsingle/

Actuator 提供的所有 endpoint

> 此處使用的是 SpringBoot 2.2.8 版本，Spring 官方文件

HTTP方法	Endpoint	描述
GET	/actuator	查看有哪些 Actuator endpoint 是開放的
GET	/actuator/auditevent	查看 audit 的事件，例如認證進入、訂單失敗，需要搭配 Spring security 使用，sample code
GET	/actuator/beans	查看運行當下裡面全部的 bean，以及他們的關係
GET	/actuator/conditions	查看自動配置的結果，記錄哪些自動配置條件通過了，哪些沒通過
GET	/actuator/configprops	查看注入帶有 @ConfigurationProperties 類的 properties 值為何（包含默認值）
GET	**/actuator/env**（常用）	查看全部環境屬性，可以看到 SpringBoot 載入了哪些 properties，以及這些 properties 的值（但是會自動`*`掉帶有 key、password、secret 等關鍵字的 properties 的值，保護安全資訊，超聰明！）
GET	/actuator/flyway	查看 flyway DB 的 migration 資訊
GET	**/actuator/health**（常用）	查看當前 SpringBoot 運行的健康指標，值由 HealthIndicator 的實現類提供（所以可以自定義一些健康指標資訊，加到這裡面）
GET	/actuator/heapdump	取得 JVM 當下的 heap dump，會下載一個檔案
GET	/actuator/info	查看 properties 中 info 開頭的屬性的值，沒啥用
GET	/actuator/mappings	查看全部的 endpoint（包含 Actuator 的），以及他們和 Controller 的關係
GET	/actuator/metrics	查看有哪些指標可以看（ex: jvm.memory.max、system.cpu.usage），要再使用`/actuator/metrics/{metric.name}`分別查看各指標的詳細資訊
GET	/actuator/scheduledtasks	查看定時任務的資訊
POST	/actuator/shutdown	唯一一個需要 POST 請求的 endpoint，關閉這個 SpringBoot 程式

開啟受保護的 endpoint 的方法

因為安全的因素，所以 Actuator 默認只會開放`/actuator/health`和`/actuator/info`這兩個 endpoint，如果要開放其他 endpoint 的話，需要額外在 application.properties 中做設置

```
# 可以這樣寫，就會開啟所有endpoints(不包含shutdown)
management.endpoints.web.exposure.include=*

# 也可以這樣寫，就只會開啟指定的endpoint，因此此處只會再額外開啟/actuator/beans和/actuator/mappings
management.endpoints.web.exposure.include=beans,mappings

# exclude可以用來關閉某些endpoints
# exclude通常會跟include一起用，就是先include了全部，然後再exclude /actuator/beans這個endpoint
management.endpoints.web.exposure.exclude=beans
management.endpoints.web.exposure.include=*

# 如果要開啟/actuator/shutdown，要額外再加這一行
management.endpoint.shutdown.enabled=true
```

除此之外，也可以改變`/actuator`的路徑，可以自定義成自己想要的路徑

```
#這樣寫的話，原本內建的/actuator/xxx路徑，都會變成/manage/xxx，可以用來防止被其他人猜到
management.endpoints.web.base-path=/manage
```

常用命令

curl -X GET 'http://192.168.77.185:8080/actuator/mappings'

【SpringBoot】 開啟tomcat log

application.properties

https://docs.spring.io/spring-boot/docs/3.1.6/reference/htmlsingle/#appendix.application-properties.server

```
server.tomcat.accesslog.buffered=true
server.tomcat.accesslog.enabled=true
server.tomcat.accesslog.file-date-format=.yyyy-MM-dd
server.tomcat.accesslog.pattern=%{yyyy-MM-dd HH:mm:ss}t %a %A %p %m %U %q %s %T %H  %{referer}i %v %l %b %S %u
server.tomcat.accesslog.prefix=access_log
server.tomcat.accesslog.rename-on-rotate=false
server.tomcat.accesslog.request-attributes-enabled=false
server.tomcat.accesslog.rotate=true
server.tomcat.accesslog.suffix=.log
server.tomcat.accesslog.directory=logs
server.tomcat.basedir=${workspace.path}/docker/log/liveportal
```

tomcat log pattern 文件 https://tomcat.apache.org/tomcat-8.0-doc/config/valve.html

- **%a** - Remote IP address
- **%A** - Local IP address
- **%b** - Bytes sent, excluding HTTP headers, or '-' if zero
- **%B** - Bytes sent, excluding HTTP headers
- **%h** - Remote host name (or IP address if `enableLookups` for the connector is false)
- **%H** - Request protocol
- **%l** - Remote logical username from identd (always returns '-')
- **%m** - Request method (GET, POST, etc.)
- **%p** - Local port on which this request was received. See also `%{xxx}p` below.
- **%q** - Query string (prepended with a '?' if it exists)
- **%r** - First line of the request (method and request URI)
- **%s** - HTTP status code of the response
- **%S** - User session ID
- **%t** - Date and time, in Common Log Format
- **%u** - Remote user that was authenticated (if any), else '-'
- **%U** - Requested URL path
- **%v** - Local server name
- **%D** - Time taken to process the request, in millis
- **%T** - Time taken to process the request, in seconds
- **%F** - Time taken to commit the response, in millis
- **%I** - Current request thread name (can compare later with stacktraces)

There is also support to write information incoming or outgoing headers, cookies, session or request attributes and special timestamp formats. It is modeled after the Apache HTTP Server log configuration syntax. Each of them can be used multiple times with different `xxx` keys:

- `%{xxx}i` write value of incoming header with name `xxx`
- `%{xxx}o` write value of outgoing header with name `xxx`
- `%{xxx}c` write value of cookie with name `xxx`
- `%{xxx}r` write value of ServletRequest attribute with name `xxx`
- `%{xxx}s` write value of HttpSession attribute with name `xxx`
- `%{xxx}p` write local (server) port (`xxx==local`) or remote (client) port (`xxx=remote`)
- `%{xxx}t` write timestamp at the end of the request formatted using the enhanced SimpleDateFormat pattern `xxx`

All formats supported by SimpleDateFormat are allowed in `%{xxx}t`. In addition the following extensions have been added:

- `sec` - number of seconds since the epoch
- `msec` - number of milliseconds since the epoch
- `msec_frac` - millisecond fraction

【SpringBoot】相關資源

- [尚圭谷]SpringBoot3全指南
  https://www.youtube.com/playlist?list=PLmOn9nNkQxJEeIH75s5pdTUnCo9-xOc7c
  https://www.yuque.com/leifengyang/springboot3
  https://gitee.com/leifengyang/spring-boot-3
- openapi (swagger)
  https://springdoc.org/

【SpringBoot】Redis

出處 https://www.cnblogs.com/rb2010/p/12905470.html

pom.xml

```
< dependency >
   < groupId > org.springframework.boot </ groupId >
   < artifactId > spring-boot-starter-data-redis </ artifactId >
</ dependency >
```

二、設定YML檔（二選一）

1. **sentinel模式**

```
spring:
  redis:
    sentinel:
      nodes:
         - 192.168.0.106:26379
         - 192.168.0.106:26380
         - 192.168.0.106:26381 //哨兵的ip和端口
      master: mymaster //這就是哨兵設定檔中sentinel monitor mymaster 192.168.0.103 6379 2設定的mymaster
```

2.Cluster模式

```
spring:
  redis:
    cluster:
      nodes: 192.168.0.106:7000,192.168.0.106:7001,192.168.0.106:7002,192.168.0.106:7003
```

三、配置RedisTemplate模版

```
@Configuration
 public  class RedisConf {
    @Bean
    public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory) {
        Jackson2JsonRedisSerializer serializer = new Jackson2JsonRedisSerializer(Object. class );
        RedisTemplate <Object, Object> template = new RedisTemplate<> ();
        template.setConnectionFactory(redisConnectionFactory);
        template.setKeySerializer(serializer); //設定key序列化(不一定要)
        template.setValueSerializer(serializer);//設定value序列化(不一定要)
return template; }

   // 讀寫分離
   @Bean
 public LettuceClientConfigurationBuilderCustomizer clientConfigurationBuilderCustomizer() {
   return clientConfigurationBuilder -> {
        clientConfigurationBuilder.readFrom(ReadFrom.REPLICA_PREFERRED);
    };
 }
 }
```

四、測試（簡單的model就省略了）

```
@RestController
 public  class RedisTestController {
    @Autowired
    RedisTemplate redisTemplate;

    @GetMapping( "set" )
     public  void set(){
        redisTemplate.opsForValue().set( "key1","123" );
        User u = new User();
        u.setId( 1 );
```

```
        u.setName( "name姓名" );
        redisTemplate.opsForValue().set( "user" ,u);
    }
    @GetMapping( "get" )
     public Map get(){
        Map map = new HashMap();
        map.put( "v1",redisTemplate.opsForValue().get("key1" ));
        map.put( "v2",redisTemplate.opsForValue().get("user" ));
         return map;
    }
}
```

相關連結

- redis分佈鎖
- Springboot集成Redis实□分布式□
- 阿里華為等大廠的Redis分散式鎖是如何設計的?
- spring-boot-redisson 与 spring-data-redis 共存□□

【SpringBoot】cors

Controll 新增 @CrossOrigin

```
@CrossOrigin(origins = "*", exposedHeaders = {"X-Total-Count"})
```

```
@RestController
@RequestMapping("/api")
@CrossOrigin(origins = "*", exposedHeaders = {"X-Total-Count"})
public class ActionController {
    private static final Logger logger = LoggerFactory.getLogger(ActionController.class);
    @Autowired
    private ShowRepository showRepository;
}
```

```
// http.cors().and().csrf().disable()
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().disable()
                .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
                .authorizeRequests()
                .antMatchers("/build/**").permitAll() // build-info
                .anyRequest().authenticated();
        http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);
    }
}
```

【SpringBoot】java 啟動參數

```
nohup java -server
-Xms4096m -Xmx4096m -XX:+UseG1GC
-XX:G1HeapRegionSize=1m
-XX:+UseStringDeduplication
-XX:MaxDirectMemorySize=1024m
-XX:+UnlockDiagnosticVMOptions
javaagent:/usr/AP/pinpoint/pinpoint-bootstrap-2.3.3.jar
-Dpinpoint.agentId=$agentId -Dpinpoint.applicationName=member
-jar test.jar
```

啟動參數

-server:
啟用Java虛擬機的伺服器模式。伺服器模式通常用於長時間運行的應用程序，以提供更好的性能。

-Xms4096m:

設定Java堆的初始內存大小為4 GB。這是Java應用程序啟動時分配的堆內存。
-Xmx4096m:

設定Java堆的最大內存大小為4 GB。這是Java應用程序在運行時能夠使用的最大堆內存。
-XX:+UseG1GC:

啟用G1垃圾回收器。G1（Garbage First）是一種面向服務端應用的垃圾回收器，旨在提供更穩定的性能和可預測的停頓時間。
-XX:G1HeapRegionSize=1m:

設定G1垃圾回收器的堆區域大小為1 MB。G1將Java堆劃分為多個相同大小的區域，這個參數設定每個區域的大小。
-XX:+UseStringDeduplication:

啟用字符串去重。這個選項將嘗試減少堆上相同字符串對象的重複，以節省內存。
-XX:MaxDirectMemorySize=1024m:

設定最大直接內存大小為1 GB。直接內存是一種不受Java堆管理的內存，通常由NIO庫使用。
-XX:+UnlockDiagnosticVMOptions:

解鎖診斷性虛擬機選項。這個選項允許使用一些診斷性工具和功能，通常在調試和性能分析中使用。
-Dpinpoint.agentId=$agentId:

設定系統屬性 pinpoint.agentId 為 $agentId。這可能是應用程序使用的某種標識符，通常用於分佈式追蹤或性能監控。
-Dpinpoint.applicationName=member:

設定系統屬性 pinpoint.applicationName 為 member。這可能是應用程序的名稱，也用於分佈式追蹤或性能監控。
-jar:
指定後面的參數為可執行的JAR文件。在這個命令中，應用程序的主體代碼存儲在一個JAR文件中。
這些參數一般用於調整Java應用程序的性能和行為。具體的值可能需要根據應用程序的需求和硬體配置進行調整。

property use list

application.yml

```
redis:
  app:
    enable: true
    disableList: >
      kafkaController.test,
      RedisTestController.testUseRedis
```

test.kt

```
    @Value("\${redis.app.disableList:null}")
    private var redisDisableList: List<String>? = null
```

【SpingBoot】取得git branch

出處：https://blog.elliot.tw/?p=658

Spring Boot可以提供的Application資訊，參考以下連結

https://docs.spring.io/spring-boot/reference/actuator/endpoints.html#actuator.endpoints.info

ID	Name	Description	Prerequisites
`build`	`BuildInfoContributor`	Exposes build information.	A `META-INF/build-info.properties` resource.
`env`	`EnvironmentInfoContributor`	Exposes any property from the `Environment` whose name starts with `info.`.	None.
`git`	`GitInfoContributor`	Exposes git information.	A `git.properties` resource.
`java`	`JavaInfoContributor`	Exposes Java runtime information.	None.
`os`	`OsInfoContributor`	Exposes Operating System information.	None.

裡面提到了`git`，主要就是讀取`git.properties`

所以只要能產生`git.properties`

在此有兩種方式，一個是由Maven Plugin產生，另一個則由Gradle Plugin產生

Generate Git Information

Maven – git-commit-id-maven-plugin

```
    <plugin>
      <groupId>io.github.git-commit-id</groupId>
      <artifactId>git-commit-id-maven-plugin</artifactId>
      <version>6.0.0</version>
      <executions>
        <execution>
          <id>get-the-git-infos</id>
          <goals>
            <goal>revision</goal>
          </goals>
          <phase>initialize</phase>
        </execution>
      </executions>
      <configuration>
        <generateGitPropertiesFile>true</generateGitPropertiesFile>
<generateGitPropertiesFilename>${project.build.outputDirectory}/git.properties</generateGitPropertiesFilename>
        <commitIdGenerationMode>full</commitIdGenerationMode>
      </configuration>
    </plugin>
```

Gradle – gradle-git-properties

```
    id("com.gorylenko.gradle-git-properties") version "2.4.1"
```

Spring Boot – Application Yaml

再來只要在`managment`裡加入`info.git.enabled=true` 即可

```
management:
  endpoints:
    web:
      exposure:
        include: '*'
  info:
    git:
      enabled: true
    java:
      enabled: true
```

Actuator Info

簡單透過actuator 提供的endpoint
http://localhost:8080/actuator/info


```
{
    "git": {
        "branch": "test/performance",
        "commit": {
            "id": "f95f360",
            "time": "2024-10-25T02:09:37Z"
        }
    },
    "build": {
        "artifact": "live-show",
        "name": "live-show",
        "time": "2024-10-25T02:54:21.087Z",
        "version": "2411.2.0",
        "group": "com.momo.app"
    },
    "java": {
        "version": "17.0.11",
        "vendor": {
            "name": "Amazon.com Inc.",
            "version": "Corretto-17.0.11.9.1"
        },
        "runtime": {
            "name": "OpenJDK Runtime Environment",
            "version": "17.0.11+9-LTS"
        },
        "jvm": {
            "name": "OpenJDK 64-Bit Server VM",
            "vendor": "Amazon.com Inc.",
            "version": "17.0.11+9-LTS"
        }
    }
}
```