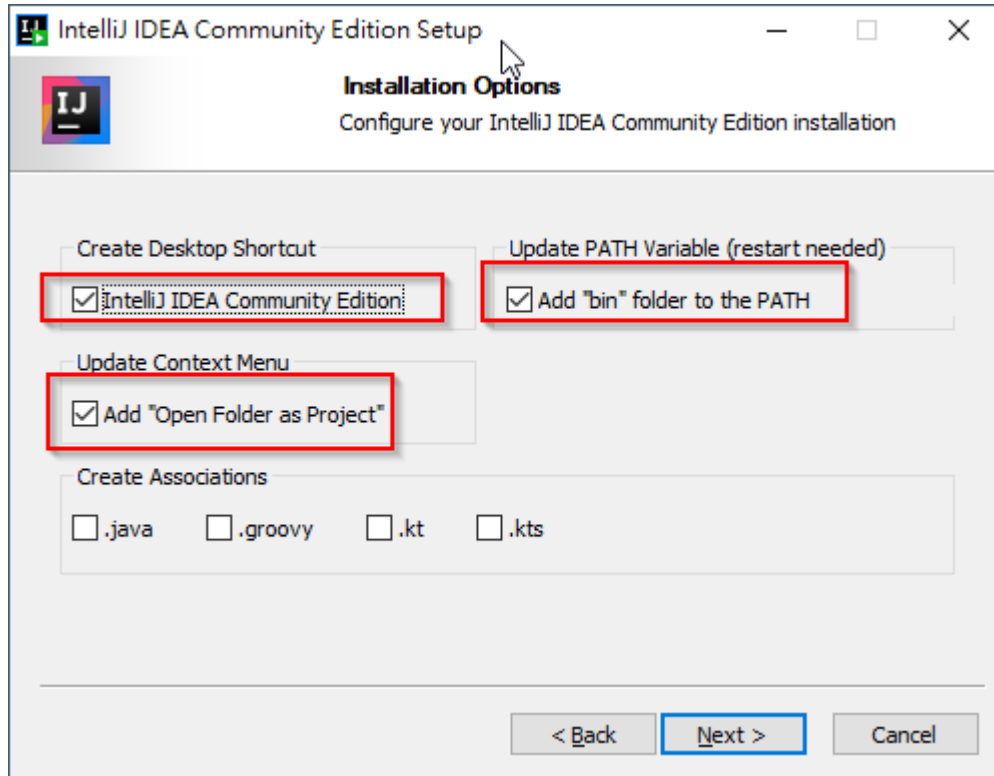


下載 IntelliJ IDEA 並安裝

[Download IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains](#)



下載 JBRSDK 並解壓縮加入環境變數

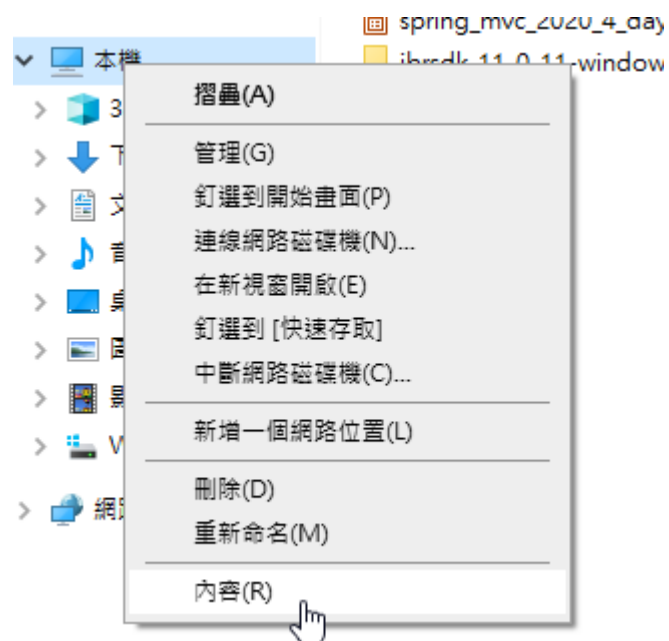
[11.0.11+9-b1504.5 - JetBrains Runtime - Confluence](#)

Downloads

	JBR with JCEF (bundled by default)	JBR with JCEF (DCEVM)	JBR with JCEF (fastdebug)	JBR (vanilla)	JBRSDK
Linux x64	Download 1504.5	Download 1504.5	Download 1504.5	Download 1504.5	Download 1504.5
Linux aarch64	-	-	-	Download 1504.5	Download 1504.5
Linux x86	-	-	-	Download 1504.5	Download 1504.5
Windows x64	Download 1504.5	Download 1504.5	Download 1504.5	Download 1504.5	Download 1504.5
Windows x86	-	-	-	Download 1504.5	Download 1504.5
macOS x64	Download 1504.5	Download 1504.5	Download 1504.5	Download 1504.5	Download 1504.5
macOS aarch64	Download 1504.5	Download 1504.5	Download 1504.5	Download 1504.5	Download 1504.5

Known major and critical issues

本機→內容



進階設定

相關設定

[BitLocker 設定](#)

[裝置管理員](#)

[遠端桌面](#)

[系統保護](#)

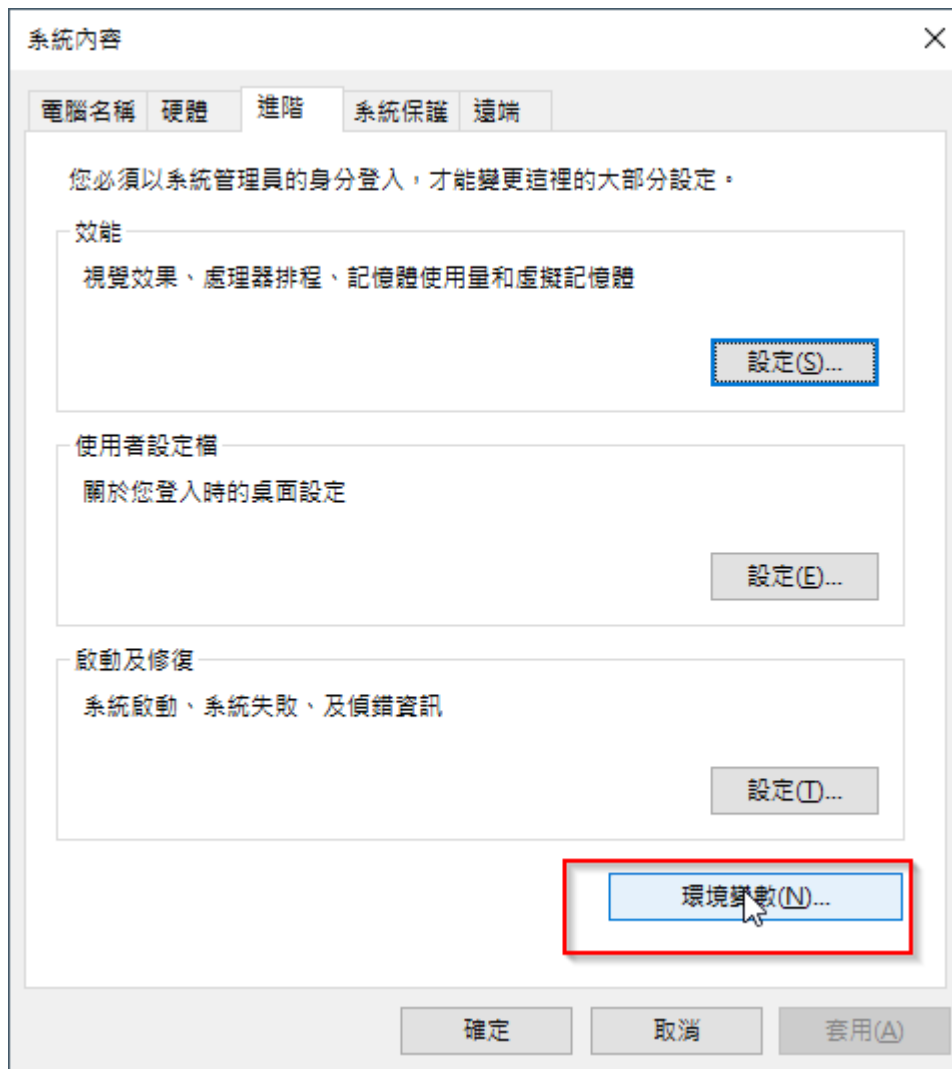
進階系統設定

[重新命名此電腦 \(進階\)](#)

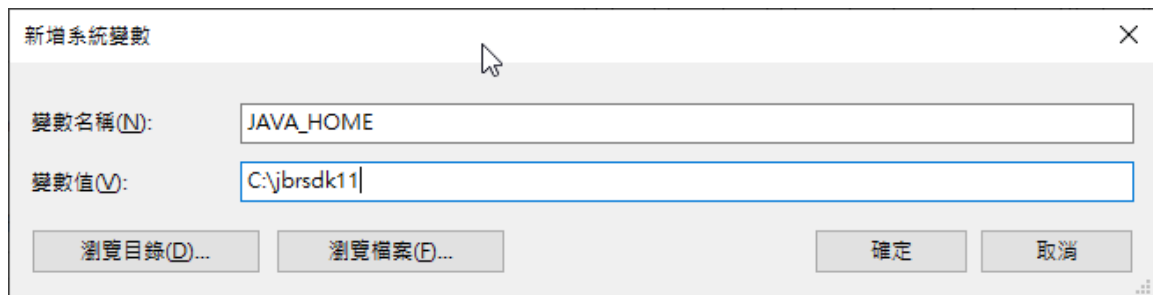
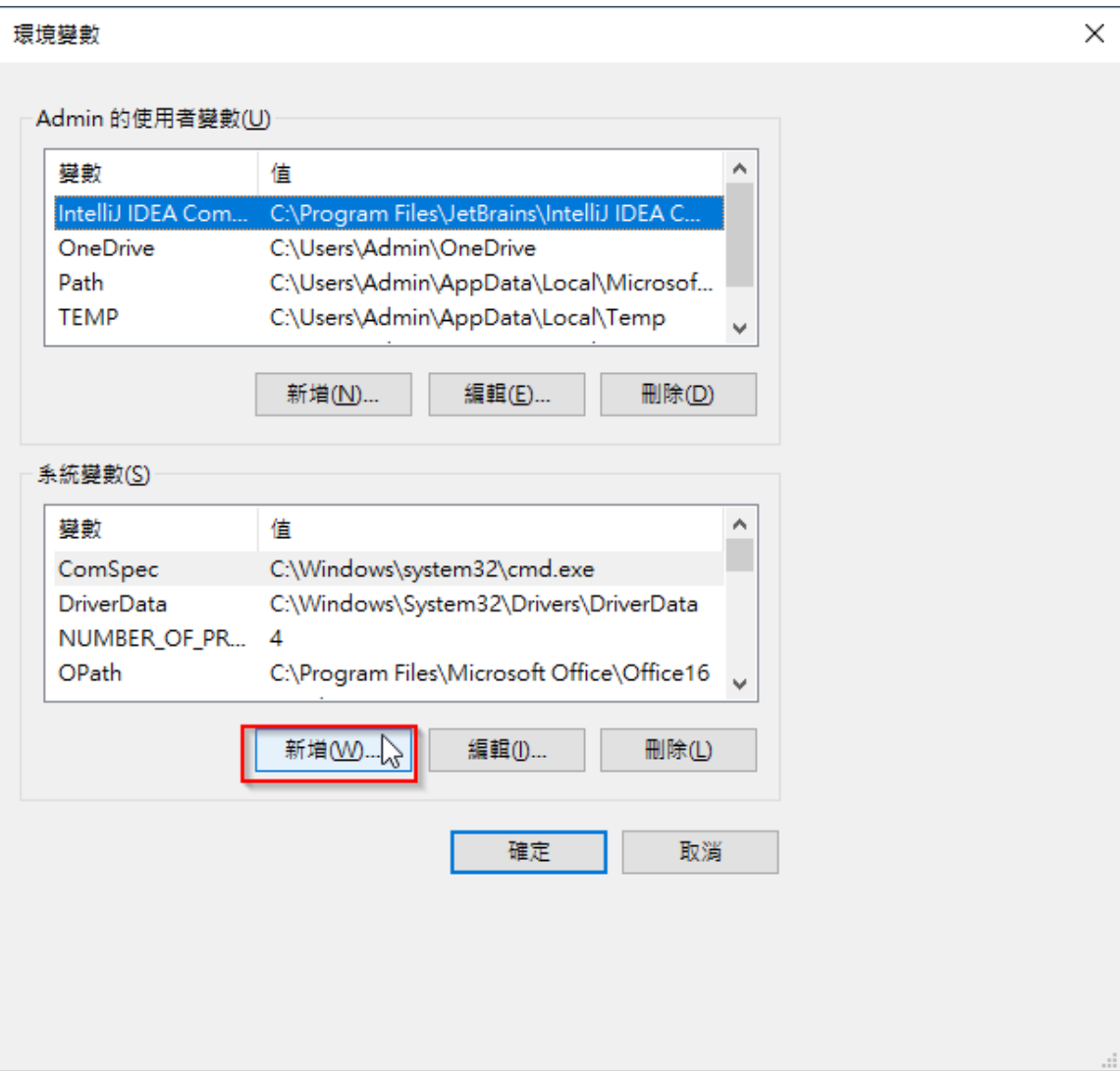
 [取得協助](#)

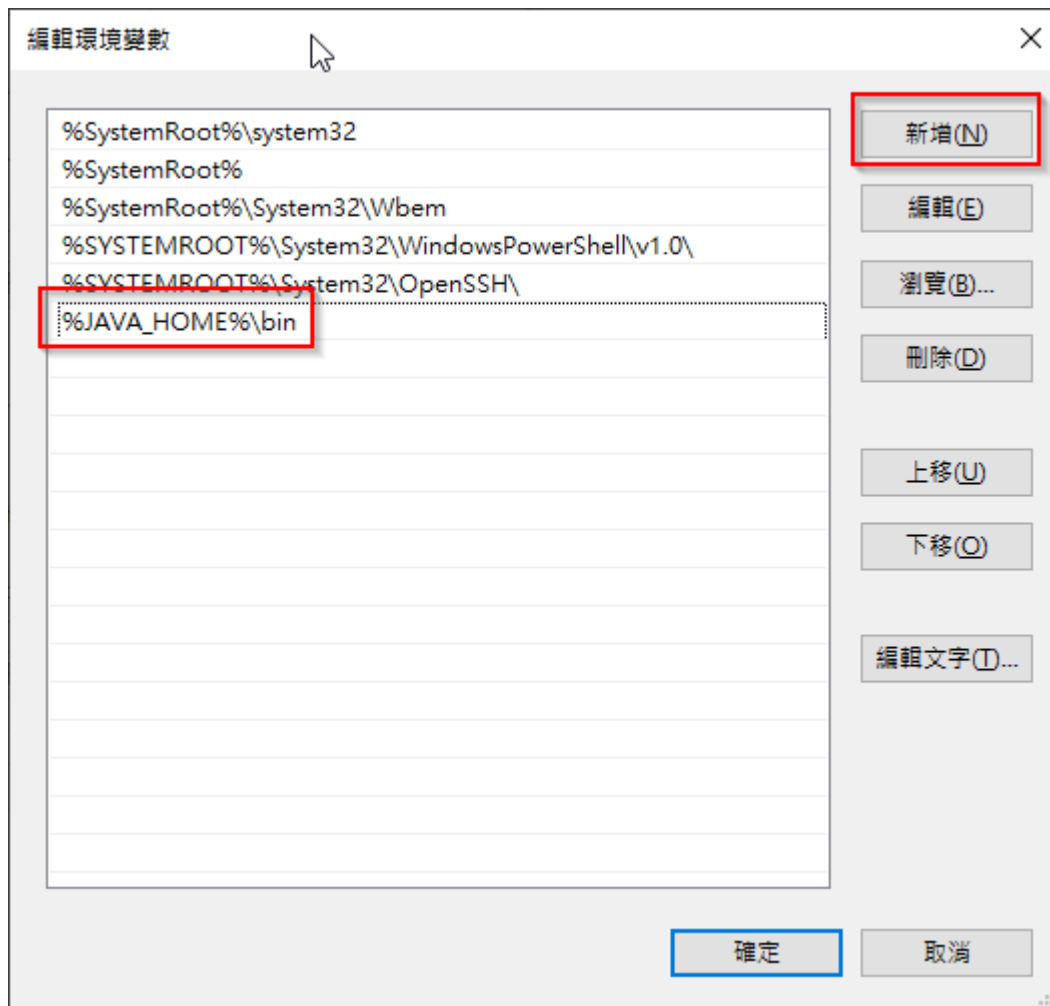
 [提供意見反應](#)

環境變數



新增系統變數





打開 CMD set JAVA

```
C:\Users\bgete>set JAVA
JAVA_HOME=C:\jbrsdk11
```

輸入 javac -version 和 java -version 確認版本

```
C:\Users\bgete>javac -version
javac 11.0.11

C:\Users\bgete>java -version
openjdk version "11.0.11" 2021-04-20
OpenJDK Runtime Environment JBR-11.0.11.9-1504.5-jcef (build 11.0.11+9-b1504.5)
OpenJDK 64-Bit Server VM JBR-11.0.11.9-1504.5-jcef (build 11.0.11+9-b1504.5, mixed mode)
```

下載並安裝 visualvm JAVA記憶體查看軟體

<https://visualvm.github.io/>

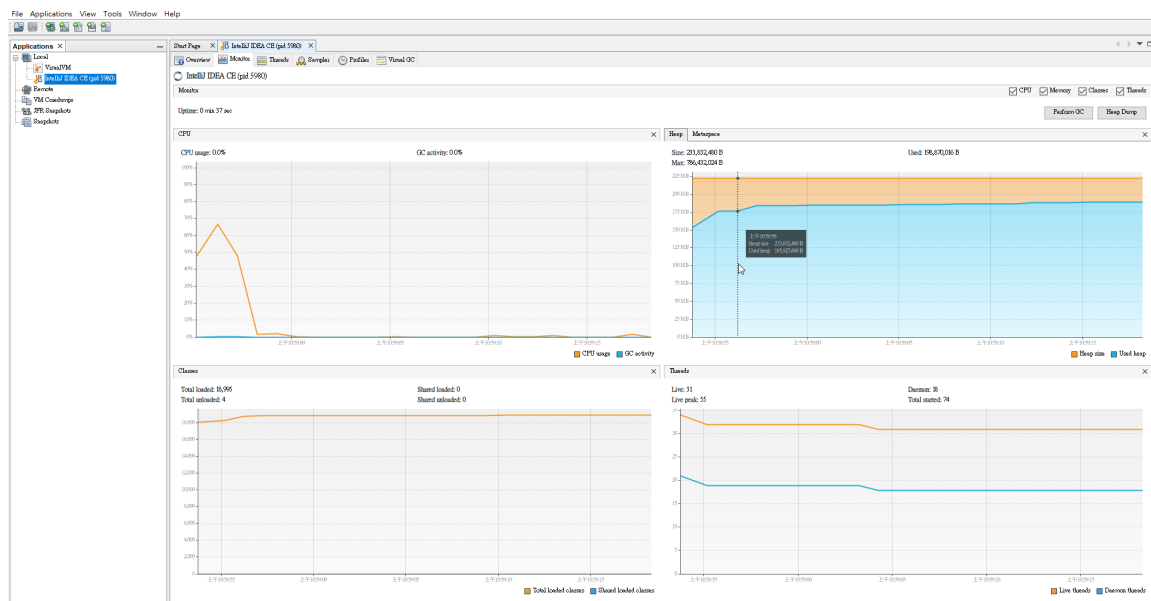
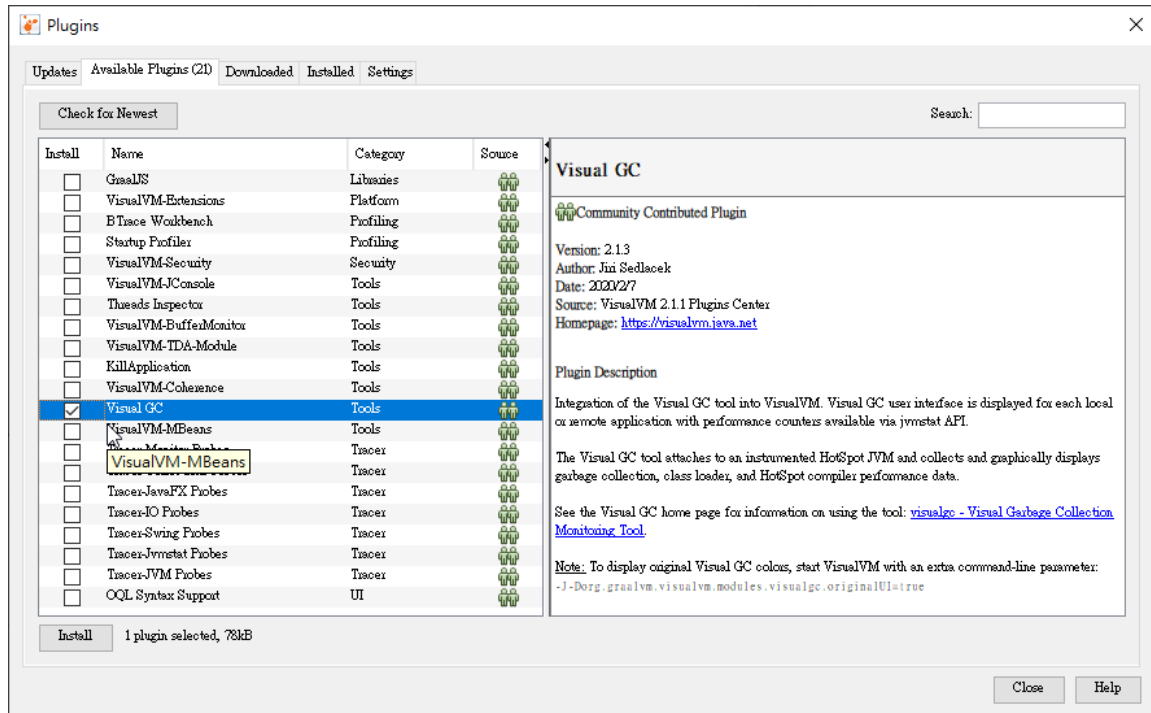
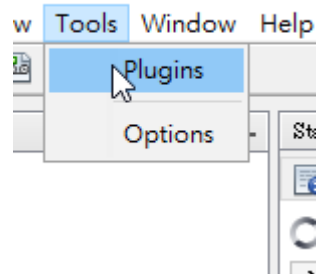
找到安裝目錄底下的設定檔

```
C:\Users\bgete\visualvm_211\etc\visualvm.conf
```

設定JDK路徑

```
visualvm_jdkhome="C:\\jbrsdk11"
```

打開軟體並安裝擴充插件



> 去spring網站按照以下設定下載初始包

[Spring Initializr](#)

spring initializr

Project

Maven Project

Gradle Project

Language

Java

Kotlin

Groovy

Spring Boot

2.7.0 (SNAPSHOT)

2.6.3 (SNAPSHOT)

2.6.2

2.5.9 (SNAPSHOT)

2.5.8

Project Metadata

Group

com.systemx

Artifact

demo1

Name

demo1

Description

Demo project for Spring Boot.

Package name

com.systemx.demo1

Packaging

Jar

War

Java

17

11

8

Dependencies

ADD DEPENDENCIES...

CTRL + B

Spring Web

WEB

Build web applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Boot Actuator

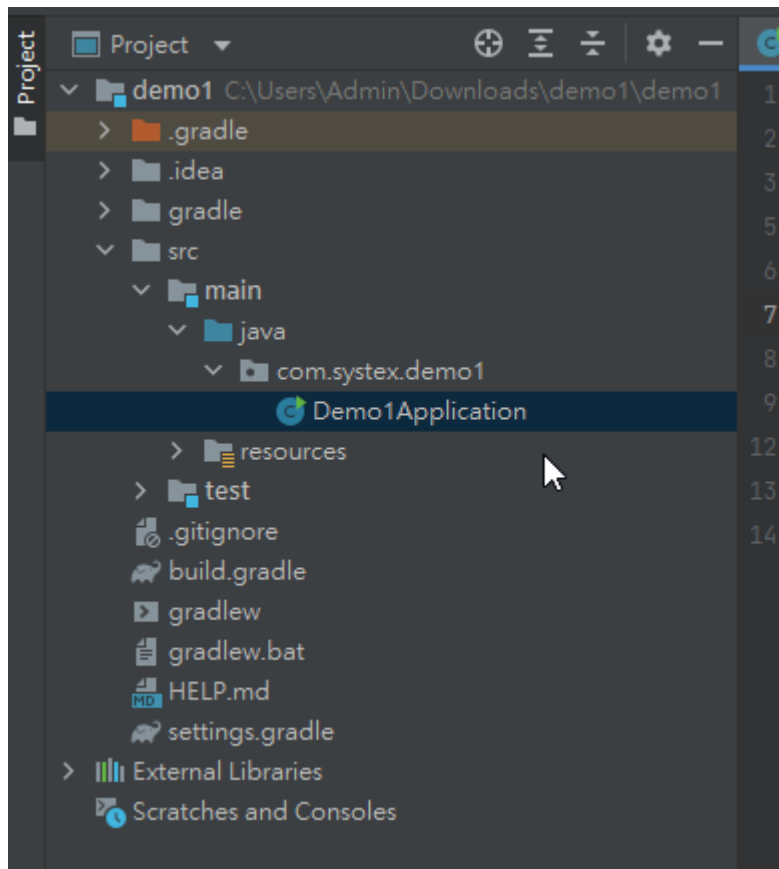
OPS

Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

下載完後按照以下步驟打開初始包

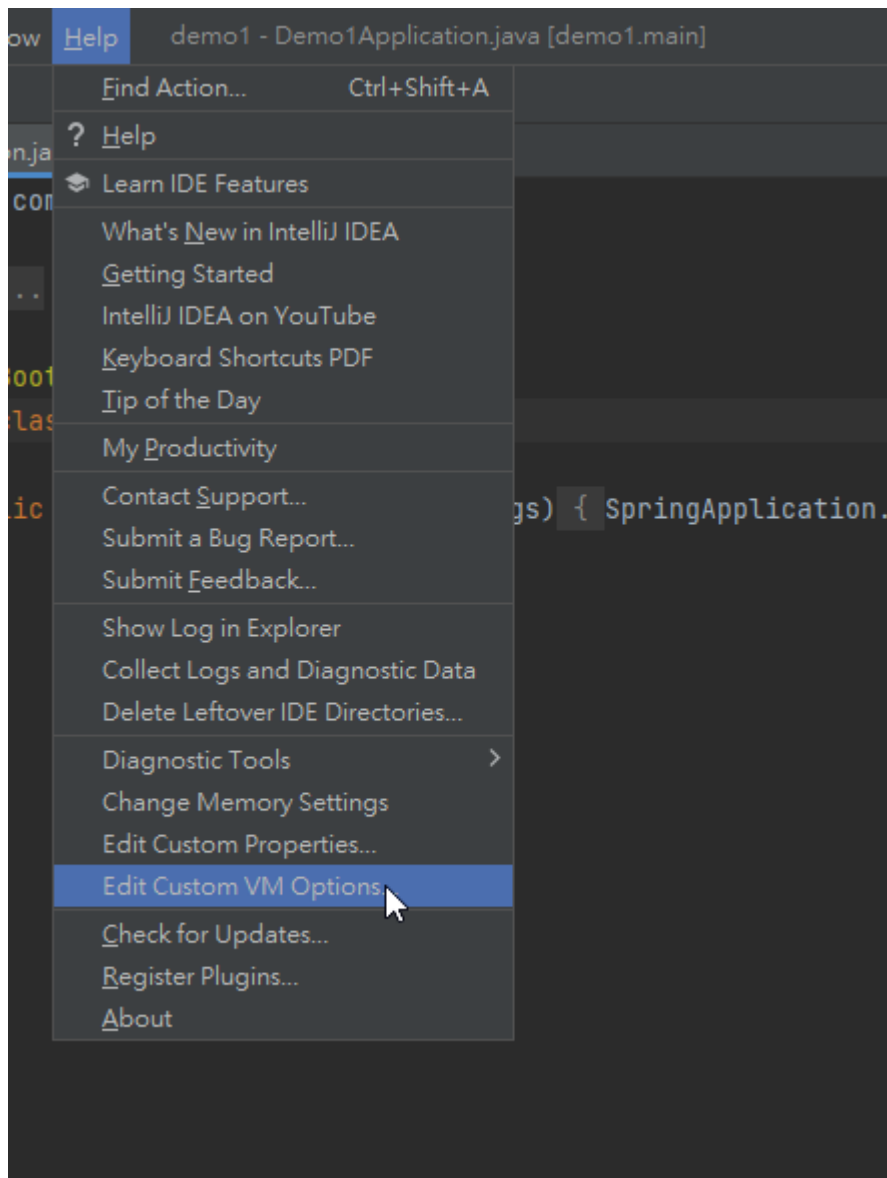
The image shows the IntelliJ IDEA interface. In the background, the 'Welcome to IntelliJ' screen has the 'Open' button highlighted with a red box. Overlaid on this is a file explorer window titled 'Open File or Project'. The address bar shows 'C:\Users\Admin\Downloads\demo1\demo1'. The file tree shows the 'demo1' folder selected, with its contents listed below: 'gradle', 'src', '.gitignore', 'build.gradle' (highlighted with a red box), 'gradlew', 'gradlew.bat' (highlighted with a red box), 'HELP.md', and 'settings.gradle'. Below the file explorer is a dialog box titled 'Trust and Open Gradle Project?'. It contains a warning icon and text: 'If you don't trust the source, preview the project in the safe mode to only browse its code.' and 'Loading, running, or building a Gradle project may execute potentially malicious code from its build scripts.' There is a checkbox labeled 'Trust projects in ~\Downloads\demo1' which is checked. At the bottom, there are three buttons: 'Trust Project' (highlighted with a red box and a mouse cursor), 'Preview in Safe Mode', and 'Don't Open'.

執行預設路徑index



預設為<http://localhost:8080/>

按照以下方式修改可用記憶體大小



點選並輸入

```
-Xmx2048m
```

```
-Xms2048m
```

去以下路徑 `C:\Users\Admin\.gradle` 新增並修改gradle.properties
將檔案內改成以下

```
org.gradle.daemon=True  
org.gradle.parallel=True  
org.gradle.jvmargs=-Xms2048m -Xmx2048m
```

回到專案內點選build.gradle並將以下依賴項及套件修改為

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-actuator'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'  
    runtimeOnly 'com.h2database:h2'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

```
plugins {  
    id 'org.springframework.boot' version '2.6.2'  
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'  
    id 'java'  
    id "io.freefair.lombok" version "6.3.0"  
}
```

創建一個新的html路徑如下src/main/resources/templates 名稱為home.html

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>Hello World</title>  
</head>  
<body>  
    <h1>My First spring boot</h1>  
</body>  
</html>
```

在以下com.systex.demo1路徑創建java class 名稱為RootController

```
package com.systex.demo1;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.Mapping;  
  
@Controller
```

```
public class RootController {  
    @GetMapping("/")  
    public String myHome() {  
        return "home";  
    }  
}
```

@Controller標示為一個控制器 @GetMapping("/")接受到GET 為"/"路徑為回傳 home

修改home.html 新增一個圖片路徑 注意此寫法是固定路徑假設初始網頁有更改圖片會找不到

自動換行 ☐

```
1 <!DOCTYPE html>  
2 <html lang="en">  
3 <head>  
4     <meta charset="UTF-8">  
5     <title>Hello World</title>  
6 </head>  
7 <body>  
8     <h1>My First spring boot</h1>  
9       
10 </body>  
11 </html>
```

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>Hello World</title>  
</head>  
<body>  
    <h1>My First spring boot</h1>  
  
</body>  
</html>
```

改成以下方式則可以自動帶入初始網頁路徑 這有初始網頁路徑有改也沒關西

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Hello World</title>
</head>
<body>
    <h1>My First spring boot</h1>

</body>
</html>
```

在src\main\resources 的 application.properties可以修改網頁顯示初始路徑
新增以下

```
server.servlet.context-path=/demo1
```

則初始路徑變為

http://localhost:8080/demo1

在com.system.demo1 新增Test1.java

@SpringBootTest 進行單元測試

@Test測試得單元

@Autowired預設會依注入對象的類別型態來選擇容器中相符的物件來注入

也就是省去寫宣告物件的get 和 set

```
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest
public class Test1 {
    @Autowired
    private RootController rootController;
```

```
@Test
void contextLoad() {
    assertThat(rootController).isNotNull();
}
}
```

建立test2

```
package com.systex.demo1;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.web.servlet.MockMvc;

import static org.hamcrest.core.StringContains.containsString;

import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@SpringBootTest
@AutoConfigureMockMvc
public class Test2 {
    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testRootPage() throws Exception {
        mockMvc.perform(get("/"))
            .andExpect(status().isOk())
            .andExpect(content().string(containsString("spring boot2")));
    }
}
```

將test2修改為下面 使用@WebMvcTest是指啟用WEB層也就是展示層

```
package com.systex.demo1;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.web.servlet.MockMvc;

import static org.hamcrest.core.StringContains.containsString;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

//1
//@SpringBootTest
//@AutoConfigureMockMvc
//2
@WebMvcTest
public class Test2 {
    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testRootPage() throws Exception {
        mockMvc.perform(get("/"))
            .andExpect(status().isOk())
            .andExpect(content().string(containsString("spring boot")));
    }
}
```

將test2修改為下列 使用@WebMvcTest 可以測試單一Controller項

```

package com.systex.demo1;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.web.servlet.MockMvc;

import static org.hamcrest.core.StringContains.containsString;

import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
//1
//@SpringBootTest
//@AutoConfigureMockMvc
//2
//@WebMvcTest
//3
@WebMvcTest(controllers = RootController.class)
public class Test2 {
    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testRootPage() throws Exception {
        mockMvc.perform(get("/"))
            .andExpect(status().isOk())
            .andExpect(content().string(containsString("spring boot")));
    }
}

```

在com.systex.demo1 新增一個java calss 名稱為GreetingService

GreetingService.java

```
package com.systex.demo1;

import org.springframework.stereotype.Service;

@Service
public class GreetingService {
    public String greet() {
        return "hello my service, send someting....";
    }
}
```

在RootController 新增為

```
package com.systex.demo1;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class RootController {
    @Autowired
    private GreetingService service;
    @GetMapping("/")
    public String myHome() {
        return "home";
    }
    @RequestMapping("/greeting")
    public @ResponseBody String greeting() {
        return service.greet();
    }
}
```


用寫成服務的形式 以範例來說此時輸入在網頁輸入localhost:8080/greeting則為直接回傳greet()所執行的也就是回傳"hello my service, send someting..."字串

創建test3 使用MVC可以在service傳回時傳回想要的內容

```
package com.system.demo1;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.test.web.servlet.MockMvc;

import static org.hamcrest.Matchers.containsString;
import static org.mockito.Mockito.when;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@WebMvcTest(RootController.class)
public class Test3 {
    @Autowired
    private MockMvc mockMvc;
    @MockBean
    private GreetingService service;
    @Test
    public void testService() throws Exception {
        when(service.greet()).thenReturn("I think it should be 29");
        mockMvc.perform(get("/greeting"))
            .andExpect(status().isOk())
            .andExpect(content().string(containsString("should be 29")));
    }
}
```

在build.gradle新增baan validator api

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-actuator'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'  
    implementation 'org.springframework.boot:spring-boot-starter-validation'  
    runtimeOnly 'com.h2database:h2'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

在com.system.demo1 新增一個java class 名稱為NormalUser1
@NotNull 不能為null，但可以為empty

```
package com.system.demo1;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.validation.constraints.NotNull;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class NormalUser1 {  
    @NotNull(message = "name should not be null")  
    private String name;  
}
```

在com.system.demo1 新增一個java class 名稱為NormalUser2
@NotEmpty不能為null，而且長度必須大於0

```
package com.system.demo1;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.validation.constraints.NotEmpty;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class NormalUser2 {
    @NotEmpty(message = "name should not be Empty")//不能為null，而且長度必須大於0
    private String name;
}
```

在com.system.demo1 新增一個java class 名稱為NormalUser3
@NotBlank 只能作用在String上，不能為null，而且调用trim()後，長度必須大於0

```
package com.system.demo1;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotEmpty;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class NormalUser3 {
    @NotBlank(message = "name should not be Blank")//只能作用在String上，不能為null，
    而且调用trim()后，長度必須大於0
    private String name;
}
```

創建test4 測試API

標記為@Before的代碼在每次測試之前執行，而@BeforeAll則在整個測試夾具之前運行一次。

```
package com.systex.demo1;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

import javax.validation.ConstraintViolation;
import javax.validation.Validation;
import javax.validation.Validator;
import java.util.Set;

import static org.assertj.core.api.Assertions.assertThat;

public class Test4 {
    private static Validator validator;
    @BeforeAll
    public static void setupValidationInstance() {
        validator = Validation.buildDefaultValidatorFactory().getValidator();
    }
    @Test
    public void test41() {
        NormalUser1 user = new NormalUser1("Mark");
        Set<ConstraintViolation<NormalUser1>>violations =
validator.validate(user);
        assertThat(violations.size()).isEqualTo(0);
    }
}
```