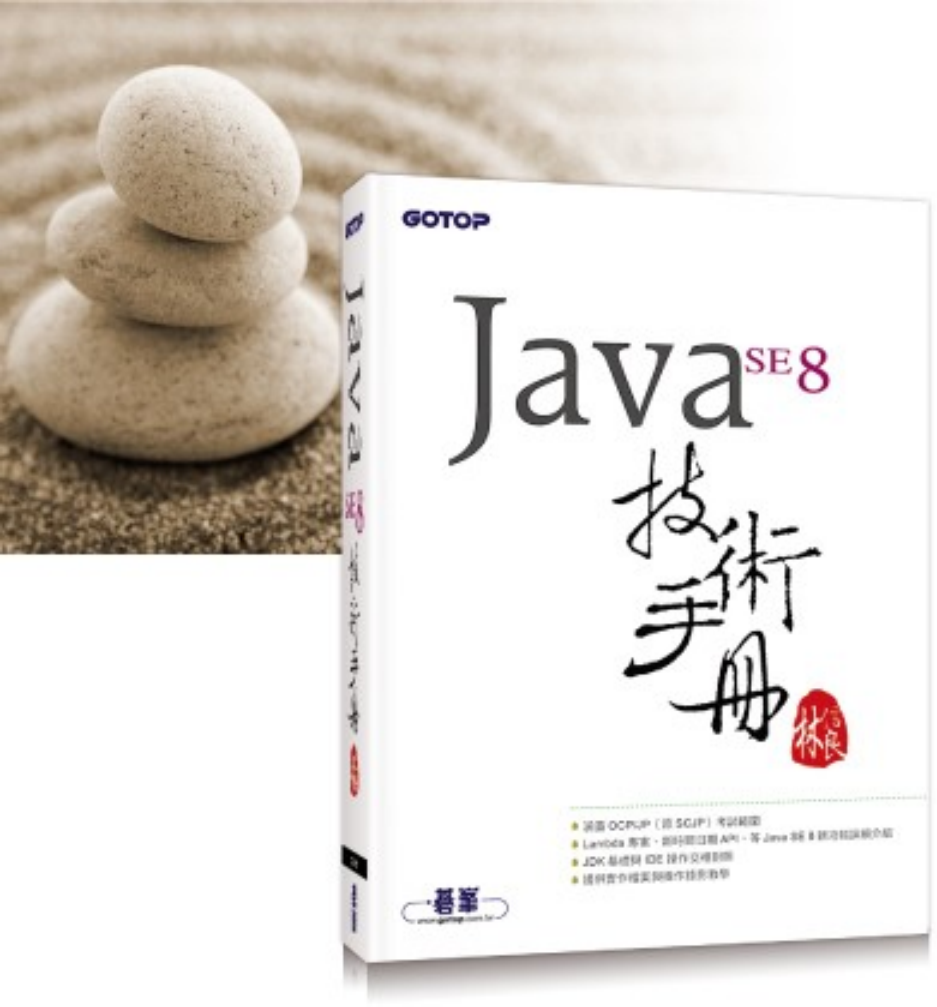


Java SE 8 技術手冊

林信良

- 涵蓋 OCP/JP（原 SCJP）考試範圍
- Lambda 專案、新時間日期 API、等 Java SE 8 新功能詳細介紹
- JDK 基礎與 IDE 操作交相對照
- 提供實作檔案與操作錄影教學

碁峰資訊

CHAPTER

反射與類別載入器

學習目標

- 取得 .class 檔案資訊
- 動態生成物件與操作方法
- 瞭解 JDK 類別載入器階層
- 使用 `ClassLoader` 實例

Class 與 .class 檔案

- Java 真正需要某個類別時才會載入對應的 .class 檔案
- `java.lang.Class` 的實例代表 Java 應用程式運行時載入的 .class 檔案
- `Class` 類別沒有公開（`public`）建構式，實例是由 JVM 自動產生

Class 與 .class 檔案

- 可以透過 Object 的 **getClass()** 方法，或者是透過 **.class** 常量（Class literal）取得每個物件對應的 Class 物件
- 如果是基本型態，也可以使用對應的包裹類別加上 .TYPE 取得 Class 物件
  - 如 Integer.TYPE 可取得代表 int 的 Class 物件
  - 如果要取得代表 Integer.class 檔案的 Class，則必須使用 Integer.class

Class 與 .class 檔案

- 可以操作 Class 物件的公開方法取得類別基本資訊

```
Class clz = String.class;
out.println("類別名稱：" + clz.getName());
out.println("是否為介面：" + clz.isInterface());
out.println("是否為基本型態：" + clz.isPrimitive());
out.println("是否為陣列物件：" + clz.isArray());
out.println("父類別名稱：" + clz.getSuperclass().getName());
```

```
類別名稱：java.lang.String
是否為介面：false
是否為基本型態：false
是否為陣列物件：false
父類別名稱：java.lang.Object
```

Class 與 .class 檔案

- 載入 .class 檔案的時機
  - 使用指定類別生成物件時
  - 使用 `Class.forName()`
  - 使用 `java.lang.ClassLoader` 實例的 `loadClass()`
- 使用類別宣告參考名稱並不會載入 .class 檔案

Class 與 .class 檔案

```
public class Some {
    static {
        System.out.println("載入 Some.class 檔案");
    }
}

Some s;
out.println("宣告 Some 參考名稱");
s = new Some();
out.println("生成 Some 實例");
```

```
宣告 Some 參考名稱
載入 Some.class 檔案
生成 Some 實例
```

Class 與 .class 檔案

- 編譯時期若使用到相關類別，編譯器會檢查對應的 .class 檔案中記載之資訊，以確定是否可完成編譯
- 執行時期使用某類別時，會先檢查是否有對應的 `Class` 物件，如果沒有，會載入對應的 .class 檔案並生成對應的 `Class` 實例

Class 與 .class 檔案

- 預設 JVM 只會用一個 `Class` 實例來代表一個 .class 檔案（確切說法是，經由同一類別載入器載入的 .class 檔案，只會有一個對應的 `Class` 實例）
- 每個類別的實例都會知道自己由哪個 Class 實例生成。預設使用 `getClass()` 或 .class 取得的 `Class` 實例會是同一個物件

```
out.println("".getClass() == String.class);
```

使用 Class.forName()

- 可以使用 Class.forName() 方法實現動態載入類別，可用字串指定類別名稱來獲得類別相關資訊

```
try {
    Class clz = Class.forName(args[0]);
    out.println("類別名稱：" + clz.getName());
    out.println("是否為介面：" + clz.isInterface());
    out.println("是否為基本型態：" + clz.isPrimitive());
    out.println("是否為陣列：" + clz.isArray());
    out.println("父類別：" + clz.getSuperclass().getName());
} catch (ArrayIndexOutOfBoundsException e) {
    out.println("沒有指定類別名稱");
} catch (ClassNotFoundException e) {
    out.println("找不到指定的類別 " + args[0]);
}
```

使用 Class.forName()

- Class.forName() 另一版本可以讓指定類別名稱、載入類別時是否執行靜態區塊與類別載入器：

```
static Class forName(String name, boolean initialize, ClassLoader loader)
```

- 如果使用第一個版本的 Class.forName() 方法，等同於：

```
Class.forName(className, true, currentLoader);
```

使用 Class.forName()

```
class Some2 {
    static {
        out.println("[執行靜態區塊]");
    }
}

public class SomeDemo2 {
    public static void main(String[] args) throws ClassNotFoundException {
        Class clz = Class.forName("cc.openhome.Some2", false,
                SomeDemo2.class.getClassLoader());
        out.println("已載入 Some2.class ");
        Some2 s;
        out.println("宣告 Some 參考名稱");
        s = new Some2();
        out.println("生成 Some 實例");
    }
}
```

從 Class 獲得資訊

- 取得 Class 物件後，就可以取得與 .class 檔案中記載的的資訊，像是套件、建構式、方法成員、資料成員等訊息
  - **java.lang.Package**
  - **java.lang.reflect.Constructor**
  - **java.lang.reflect.Method**
  - **java.lang.reflect.Field**
  - …

```
Package p = String.class.getPackage();
out.println(p.getName());      // 顯示 java.lang
```

```
public static void main(String[] args) {
    try {
        ClassViewer.view(args[0]);
    } catch (ArrayIndexOutOfBoundsException e) {
        out.println("沒有指定類別");
    } catch (ClassNotFoundException e) {
        out.println("找不到指定類別");
    }
}

public static void view(String clzName) throws ClassNotFoundException {
    Class clz = Class.forName(clzName);

    showPackageInfo(clz);
    showClassInfo(clz);

    out.println("{");

    showFiledsInfo(clz);
    showConstructorsInfo(clz);
    showMethodsInfo(clz);

    out.println("}");
}
```

```
private static void showPackageInfo(Class clz) {
    Package p = clz.getPackage(); // 取得套件代表物件
    out.printf("package %s;%n", p.getName());
}

private static void showClassInfo(Class clz) {
    int modifier = clz.getModifiers();    // 取得型態修飾常數
    out.printf("%s %s %s",
            Modifier.toString(modifier), // 將常數轉為字串表示
            Modifier.isInterface(modifier) ? "interface" : "class",
            clz.getName() // 取得類別名稱
    );
}
```

```
    private static void showFiledsInfo(Class clz) throws SecurityException {
        // 取得宣告的資料成員代表物件
        Field[] fields = clz.getDeclaredFields();
        for (Field field : fields) {
            // 顯示權限修飾，像是 public、protected、private
            out.printf("\t%s %s %s;%n",
                    Modifier.toString(field.getModifiers()),
                    field.getType().getName(), // 顯示型態名稱
                    field.getName() // 顯示資料成員名稱
            );
        }
    }

    private static void showConstructorsInfo(Class clz) throws SecurityException
{
        // 取得宣告的建構方法代表物件
        Constructor[] constructors = clz.getDeclaredConstructors();
        for (Constructor constructor : constructors) {
            // 顯示權限修飾，像是 public、protected、private
            out.printf("\t%s %s();%n",
                    Modifier.toString(constructor.getModifiers()),
                    constructor.getName() // 顯示建構式名稱
            );
        }
    }
```

```
private static void showMethodsInfo(Class clz) throws SecurityException {
    // 取得宣告的方法成員代表物件
    Method[] methods = clz.getDeclaredMethods();
    for (Method method : methods) {
        // 顯示權限修飾，像是 public、protected、private
        out.printf("\t%s %s %s();%n",
                Modifier.toString(method.getModifiers()),
                method.getReturnType().getName(), // 顯示返回值型態名稱
                method.getName() // 顯示方法名稱
        );
    }
}
```

從 Class 建立物件

- 取得 Class 物件之後，利用其 newInstance() 方法建立類別實例

```
Class clz = Class.forName(args[0]);
Object obj = clz.newInstance();
```

從 Class 建立物件

- 你想採用影片程式庫來播放動畫，然而負責實作影片程式庫的部門遲遲還沒動工，怎麼辦呢？

```
public interface Player {
    void play(String video);
}

public class MediaMaster {
    public static void main(String[] args) throws ClassNotFoundException,
                                InstantiationException, IllegalAccessException {
        String playerImpl = System.getProperty("cc.openhome.PlayerImpl");
        Player player = (Player) Class.forName(playerImpl).newInstance();
        Scanner scanner = new Scanner(System.in);
        System.out.print("輸入想播放的影片：");
        player.play(scanner.nextLine());
    }
}
```

從 Class 建立物件

- 可以在啟動程式時，透過系統屬性 cc.openhome.PlayerImpl 指定

```
public class ConsolePlayer implements Player {
    @Override
    public void play(String video) {
        System.out.println("正在播放 " + video);
    }
}
```

從 Class 建立物件

- 執行 MediaPlayer 若指定了 -Dcc.openhome.PlayerImpl=cc.openhome.ConsolePlayer

```
輸入想播放的影片：Hello! Duke!
正在播放 Hello! Duke!
```

從 Class 建立物件

- 若類別定義有多個建構式，也可以指定使用哪個建構式生成物件
- 假設因為某個原因，必須動態載入 java.util.List 實作類別

```
Class clz = Class.forName(args[0]);  // 動態載入.class
Constructor constructor = clz.getConstructor(Integer.TYPE);  // 取得建構式
List list = (List) constructor.newInstance(100);  // 利用建構式建立實例
```

從 Class 建立物件

- 陣列的 Class 實例是由 JVM 生成，你並不知道陣列的建構式為何
- 若要動態生成陣列，必須使用 **java.lang.reflect.Array** 的 **newInstance()** 方法

```
Class clz = java.util.ArrayList.class;
Object obj = Array.newInstance(clz, 10);
```

從 Class 建立物件

- 可以使用 **Array.set()** 方法指定索引設值，或是使用 **Array.get()** 方法指定索引取值
- 比較偷懶的方式，直接當作 Object[] （或已知的陣列型態）使用

```
Class clz = java.util.ArrayList.class;
Object[] objs = (Object[]) Array.newInstance(clz, 10);
objs[0] = new ArrayList();
ArrayList list = objs[0];
```

從 Class 建立物件

- 為何要使用 Array.newInstance() 建立陣列實例？
- 回顧一下 9.1.7 中實作過的 ArrayList，如果現在為其設計一個 toArray() 方法：

```
public class ArrayList<E> {
    private Object[] elems;
    ...略
    public ArrayList(int capacity) {
        elems = new Object[capacity];
    }
    ...略
     public E[] toArray() {
         return (E[]) elems;
     }
}
```

從 Class 建立物件

- 現在有個使用者這麼使用 ArrayList，會拋出 java.lang.ClassCastException，告訴你不可以將 Object[] 當作 String[] 來使用

```
ArrayList<String> list = new ArrayList<>();
list.add("One");
list.add("Two");
String[] strs = list.toArray();
```

從 Class 建立物件

- 可以如下解決：

```
public E[] toArray() {
    E[] elements = null;
    if(size() > 0) {
        elements = (E[]) Array.newInstance(list[0].getClass(), size());
        for(int i = 0; i < elements.length; i++) {
            elements[i] = (E) list[i];  // 複製參考
        }
    }
    return elements;
}
```

操作物件方法與成員

- java.lang.reflect.Method 實例是方法的代表物件，可以使用 **invoke()** 方法來動態呼叫指定的方法

```
Class clz = Class.forName("cc.openhome.Student");
Constructor constructor = clz.getConstructor(String.class, Integer.class);
Object obj = constructor.newInstance("caterpillar", 90);
// 指定方法名稱與參數型態，呼叫 getMethod()取得對應的公開 Method 實例
Method setter = clz.getMethod("setName", String.class);
// 指定參數值呼叫物件 obj 的方法
setter.invoke(obj, "caterpillar");
Method getter = clz.getMethod("getName");
out.println(getter.invoke(obj));
```

操作物件方法與成員

- 底下會設計一個 BeanUtil 類別，可以指定 Map 物件與類別名稱呼叫 getBean() 方法，這個方法會抽取 Map 內容並封裝為指定類別的實例
  - 例如 Map 中收集了學生資料，則以下傳回的就是 Student 實例

```
Map<String, Object> data = new HashMap<>();
data.put("name", "Justin");
data.put("score", 90);
Student student = (Student) BeanUtil.getBean(data, "cc.openhome.Student");
// 底下顯示(Justin, 90)
out.printf("(%s, %d)%n", student.getName(), student.getScore());
```

```
public class BeanUtil {
    public static <T> T getBean(Map<String, Object> data, String clzName)
                                        throws Exception {
        Class clz = Class.forName(clzName);
        Object bean = clz.newInstance();

        data.entrySet().forEach(entry -> {
            String setter = String.format("set%s%s",
                    entry.getKey().substring(0, 1).toUpperCase(),
                    entry.getKey().substring(1));
            try {
                // 根據方法名稱與參數型態取得 Method 實例
                Method method = clz.getMethod(
                        setter, entry.getValue().getClass());
                // 必須是公開方法
                if(Modifier.isPublic(method.getModifiers())) {
                    // 指定實例與參數值呼叫方法
                    method.invoke(bean, entry.getValue());
                }
            } catch(IllegalAccessException | IllegalArgumentException |
                    NoSuchMethodException | SecurityException |
                    InvocationTargetException ex) {
                throw new RuntimeException(ex);
            }
        });

        return (T) bean;
    }
}
```

操作物件方法與成員

- 想呼叫受保護的（protected）或私有（private）方法

```
Method priMth = clz.getDeclaredMethod("priMth", ...);
priMth.setAccessible(true);
priMth.invoke(target, args);
```

操作物件方法與成員

- 可以使用反射機制存取類別資料成員（Field）

```
Class clz = Student.class;
Object o = clz.newInstance();
Field name = clz.getDeclaredField("name");
Field score = clz.getDeclaredField("score");
name.setAccessible(true);    // 如果是private 的 Field，要修改得呼叫此方法
score.setAccessible(true);
name.set(o, "Justin");
score.set(o, 90);
Student student = (Student) o;
// 底下顯示(Justin  90)
out.printf("(%s, %d)%n", student.getName(), student.getScore());
```

動態代理

- 需要在執行某些方法時進行日誌記錄，你可能會如下撰寫：

```
public class HelloSpeaker {
    public void hello(String name) {
        // 方法執行開始時留下日誌
        Logger.getLogger(HelloSpeaker.class.getName())
                .log(Level.INFO, "hello() 方法開始....");
        // 程式主要功能
        out.printf("哈囉, %s%n", name);
        // 方法執行完畢前留下日誌
        Logger.getLogger(HelloSpeaker.class.getName())
                .log(Level.INFO, "hello() 方法結束....");
    }
}
```

靜態代理

- 在靜態代理實現中，代理物件與被代理物件必須實現同一介面

```
public interface Hello {
    void hello(String name);
}

public class HelloSpeaker implements Hello {
    public void hello(String name) {
        System.out.printf("哈囉, %s%n", name);
    }
}
```

靜態代理

- 代理物件同樣也要實現 Hello 介面

```
public class HelloProxy implements Hello {
    private Hello helloObj;

    public HelloProxy(IHello helloObj) {
        this.helloObj = helloObj;
    }

    public void hello(String name) {
        log("hello()方法開始....");       // 日誌服務
        helloObj.hello(name);             // 執行商務邏輯
        log("hello()方法結束....");       // 日誌服務
    }

    private void log(String msg) {
        Logger logger = Logger.getLogger(HelloProxy.class.getName())
                                    .log(Level.INFO, msg);
    }
}
```

靜態代理

- 可以如下使用代理物件：

```
Hello proxy = new HelloProxy(new HelloSpeaker());
proxy.hello("Justin");
```

動態代理

- 使用動態代理機制，可使用一個處理者（Handler）代理多個介面的實作物件

```
public class LoggingHandler implements InvocationHandler {
    private Object target;

    public Object bind(Object target) {
        this.target = target;
        return Proxy.newProxyInstance(
                            target.getClass().getClassLoader(),
                            target.getClass().getInterfaces(),
                            this);
    }

    public Object invoke(Object proxy, Method method,
                         Object[] args) throws Throwable {
        Object result = null;
        try {
            log(String.format("%s() 呼叫開始...", method.getName()));
            result = method.invoke(target, args);
            log(String.format("%s() 呼叫結束...", method.getName()));
        } catch (Exception e){
            log(e.toString());
        }

        return result;
    }
```

❶ 動態建立代理物件

❷ 代理物件的方法被呼叫時會呼叫此方法

❸ 實現日誌

❹ 實現被代理物件職責

❺ 實現日誌

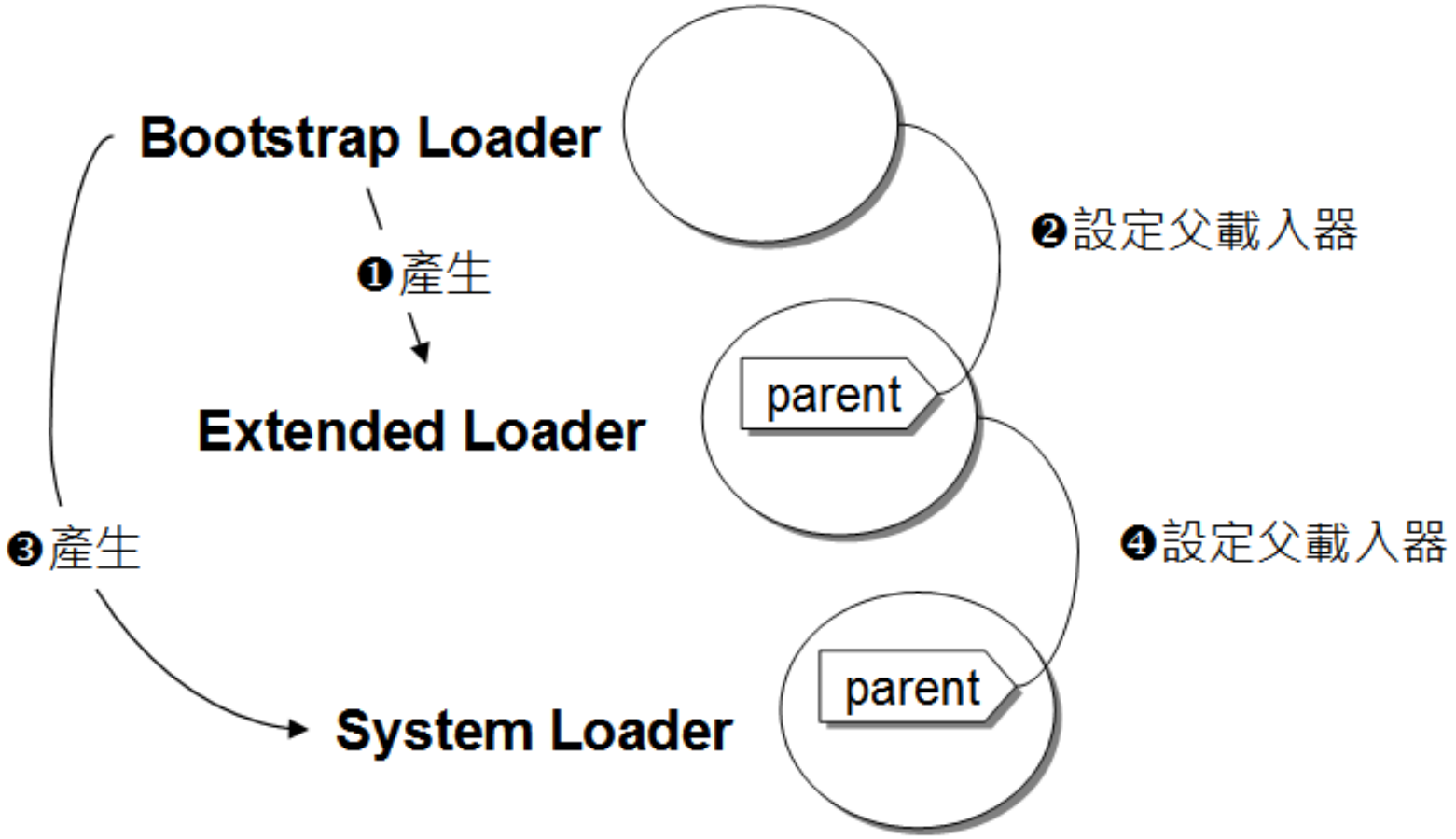
動態代理

- 使用 `LoggingHandler` 的 `bind()` 方法來綁定被代理物件

```
LoggingHandler loggingHandler  = new LoggingHandler();
Hello helloProxy = (Hello) loggingHandler.bind(new HelloSpeaker());
helloProxy.hello("Justin");
```

```
五月 13, 2014 2:16:58 下午 cc.openhome.LoggingHandler log
資訊: hello() 呼叫開始...
哈囉, Justin
五月 13, 2014 2:16:58 下午 cc.openhome.LoggingHandler log
資訊: hello() 呼叫結束...
```

類別載入器階層架構


Bootstrap Loader
❶產生
Extended Loader
❷設定父載入器
parent
❸產生
System Loader
❹設定父載入器
parent

類別載入器階層架構

- 如果是 Oracle 的 JDK， Bootstrap Loader 會搜尋系統參數 sun.boot.class.path 中指定位置的類別

```
C:\Program Files\Java\jdk1.8.0\jre\lib\resources.jar;
C:\Program Files\Java\jdk1.8.0\jre\lib\rt.jar;
C:\Program Files\Java\jdk1.8.0\jre\lib\sunrsasign.jar;
C:\Program Files\Java\jdk1.8.0\jre\lib\jsse.jar;
C:\Program Files\Java\jdk1.8.0\jre\lib\jce.jar;
C:\Program Files\Java\jdk1.8.0\jre\lib\charsets.jar;
C:\Program Files\Java\jdk1.8.0\jre\lib\jfr.jar;
C:\Program Files\Java\jdk1.8.0\jre\classes
```

類別載入器階層架構

- Extended Loader 由 Java 撰寫而成，會搜尋系統參數 java.ext.dirs 中指定位置的類別

```
C:\Program Files\Java\jdk1.8.0\jre\lib\ext;
C:\WINDOWS\Sun\Java\lib\ext
```

類別載入器階層架構

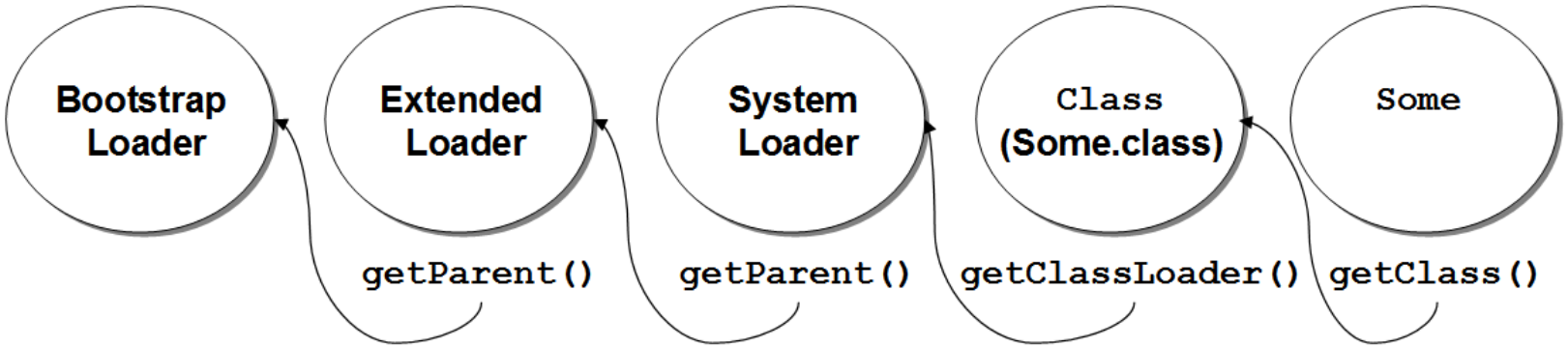
- System Loader 由 Java 撰寫而成，會搜尋系統參數 java.class.path 指定位置的類別，也就是 CLASSPATH 路徑
- 使用 java 執行程式時，可以加上 -cp 來覆蓋原有的 CLASSPATH 設定

類別載入器階層架構

- 在載入類別時，每個類別載入器會先將載入類別的任務交由給父載入器，如果父載入器找不到，才由自己載入
  - 所以會以 Bootstrap Loader→Extended Loader→System Loader 順序尋找類別
- 如果所有類別載入器都找不到指定類別，就是 **`java.lang.NoClassDefFoundError`**

類別載入器階層架構

- 類別載入器都繼承自抽象類別 **`java.lang.ClassLoader`**，可以由 `Class` 的 **`getClassLoader()`** 取得



類別載入器階層架構

- 如果 Some 可在 CLASSPATH 中載入

```
Some some = new Some();                          // 生成 Some 實例
Class clz = some.getClass();                     // 取得 Some.class 的 Class 實例
ClassLoader loader = clz.getClassLoader();  // 取得 ClassLoader
out.println(loader);
out.println(loader.getParent());                    // 取得父 ClassLoader
out.println(loader.getParent().getParent());  // 再取得父 ClassLoader
```

```
sun.misc.Launcher$AppClassLoader@177b3cd
sun.misc.Launcher$ExtClassLoader@1bd7848
null
```

類別載入器階層架構

- 如果把 Some.class 檔案（包括套件資料夾）移至 JRE 目錄的 lib\ext\classes 下

```
sun.misc.Launcher$ExtClassLoader@1bd7848
null
Exception in thread "main" java.lang.NullPointerException
        at cc.openhome.ClassLoaderHierarchy.main(ClassLoaderHierarchy.java:10)
Java Result: 1
```

建立 ClassLoader 實例

- Bootstrap Loader、 Extended Loader 與 System Loader 在程式啟動後，就無法再改變它們的搜尋路徑
- 可以使用 URLClassLoader 來產生新的類別載入器

```
URL url = new URL("file:/d:/workspace/");
ClassLoader loader = new URLClassLoader(new URL[] {url});
Class clz = loader.loadClass("cc.openhome.Some");
```

建立 ClassLoader 實例

- 由同一類別載入器載入的 .class 檔案，只會有一個 `Class` 實例
- 如果同一 .class 檔案由兩個不同的類別載入器載入，則會有兩份不同的 `Class` 實例

```
public class ClassLoaderDemo {
    public static void main(String[] args) {
        try {
            String path = args[0];     // 測試路徑
            String clzName = args[1]; // 測試類別

            Class clz1 = loadClassFrom(path, clzName);
            out.println(clz1);
            Class clz2 = loadClassFrom(path, clzName);
            out.println(clz2);

            out.printf("clz1 與 clz2 為%s 實例",
                    clz1 == clz2 ? "相同" : "不同");
        } catch (ArrayIndexOutOfBoundsException e) {
            out.println("沒有指定類別載入路徑與名稱");
        } catch (MalformedURLException e) {
            out.println("載入路徑錯誤");
        } catch (ClassNotFoundException e) {
            out.println("找不到指定的類別");
        }
    }

    private static Class loadClassFrom(String path, String clzName)
            throws ClassNotFoundException, MalformedURLException {
        ClassLoader loader = new URLClassLoader(new URL[] {new URL(path)});
        return loader.loadClass(clzName);
    }
}
```