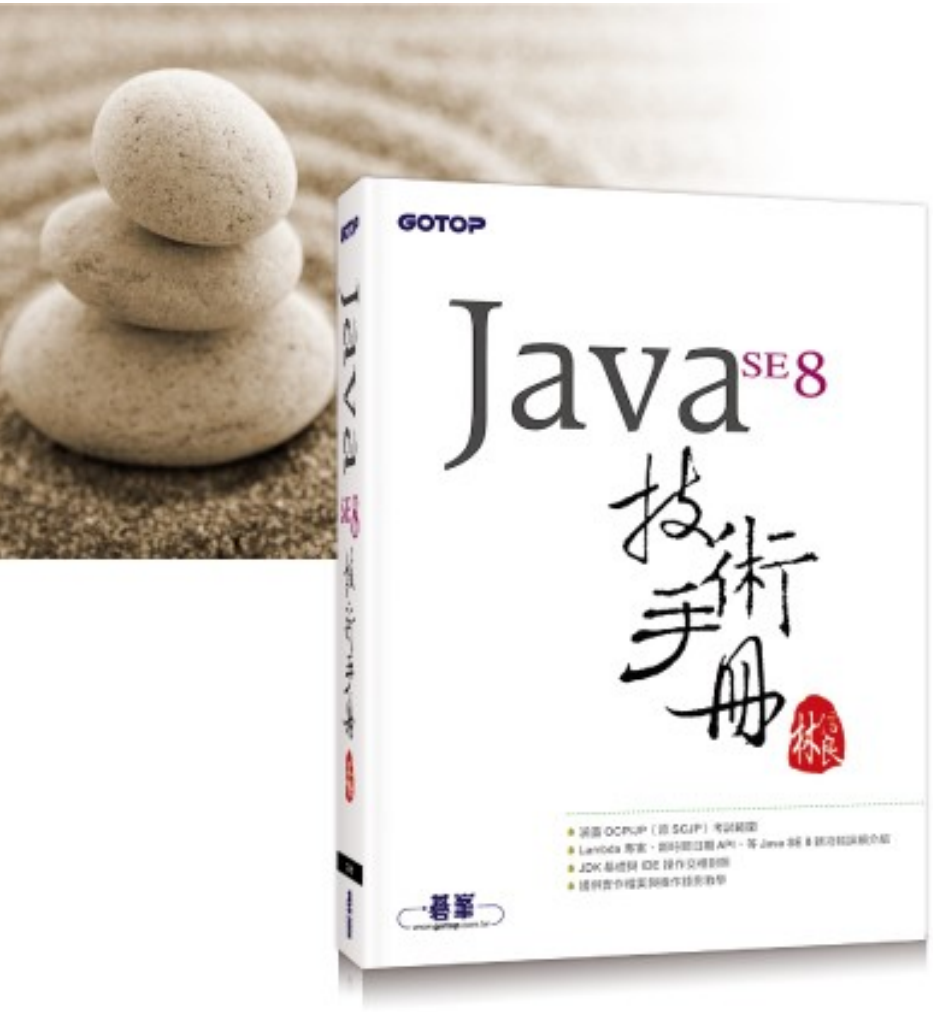


Java SE 8 技術手冊

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

NIO 與 NIO2

學習目標

- 認識 NIO
- 使用 `Channel` 與 `Buffer`
- 使用 NIO2 檔案系統

NIO 概觀

- 在 10.1.1 中看過的 dump() 方法

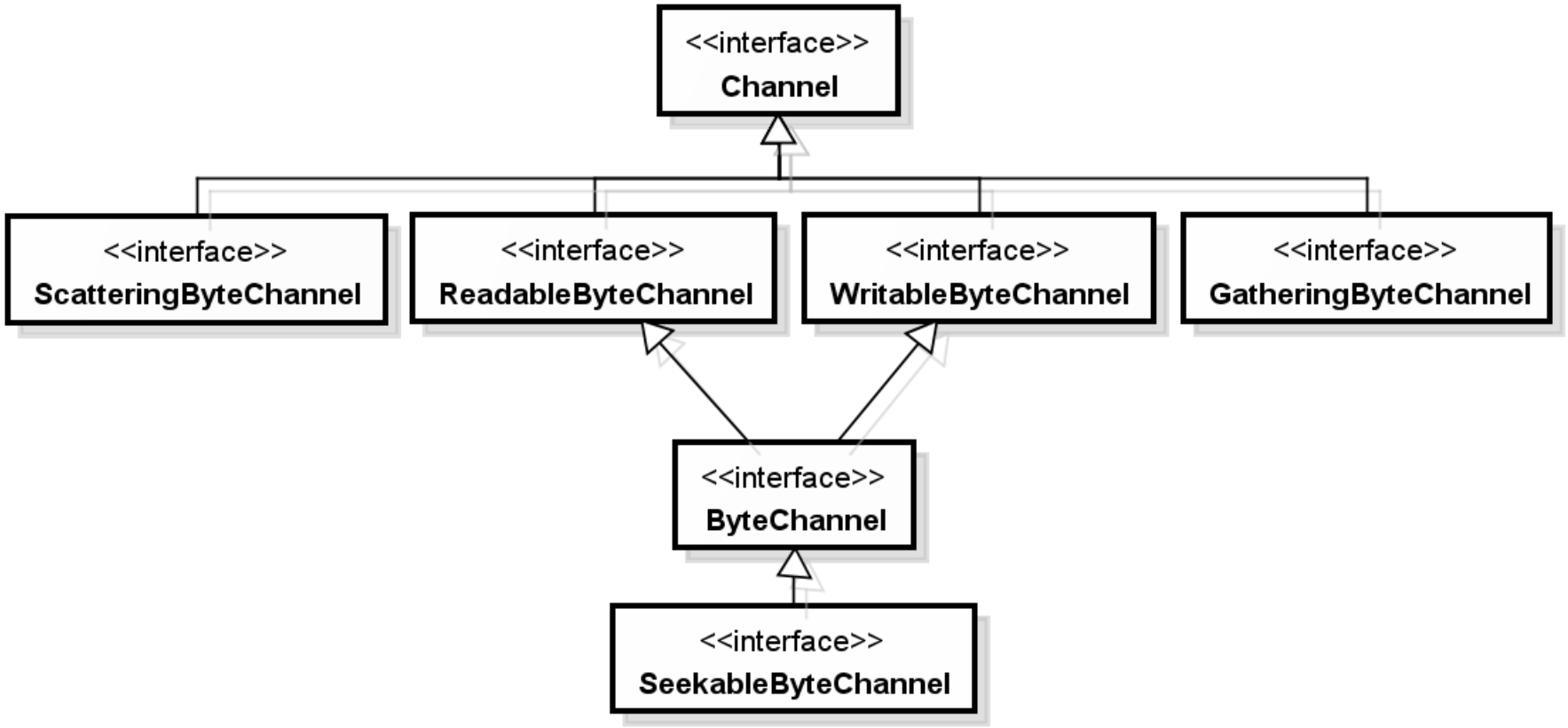
```
public static void dump(InputStream src, OutputStream dest)
                                throws IOException {
    try (InputStream input = src; OutputStream output = dest) {
        byte[] data = new byte[1024];
        int length;
        while ((length = input.read(data)) != -1) {
            output.write(data, 0, length);
        }
    }
}
```

NIO 概觀

- 若使用 NIO 的話，可以如下撰寫：

```
public static void dump(ReadableByteChannel src, WritableByteChannel dest)
                        throws IOException {
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    try(ReadableByteChannel srcCH = src; WritableByteChannel destCH = dest) {
        while(srcCH.read(buffer) != -1) {
            buffer.flip();
            destCH.write(buffer);
            buffer.clear();
        }
    }
}
```

Channel 架構與操作



Channel 架構與操作

- 想要取得 Channel 的實作物件，可以使用 Channels 類別
- 靜態方法 newChannel()，可以讓你從 InputStream、OutputStream 分別建立 ReadableByteChannel、WritableByteChannel

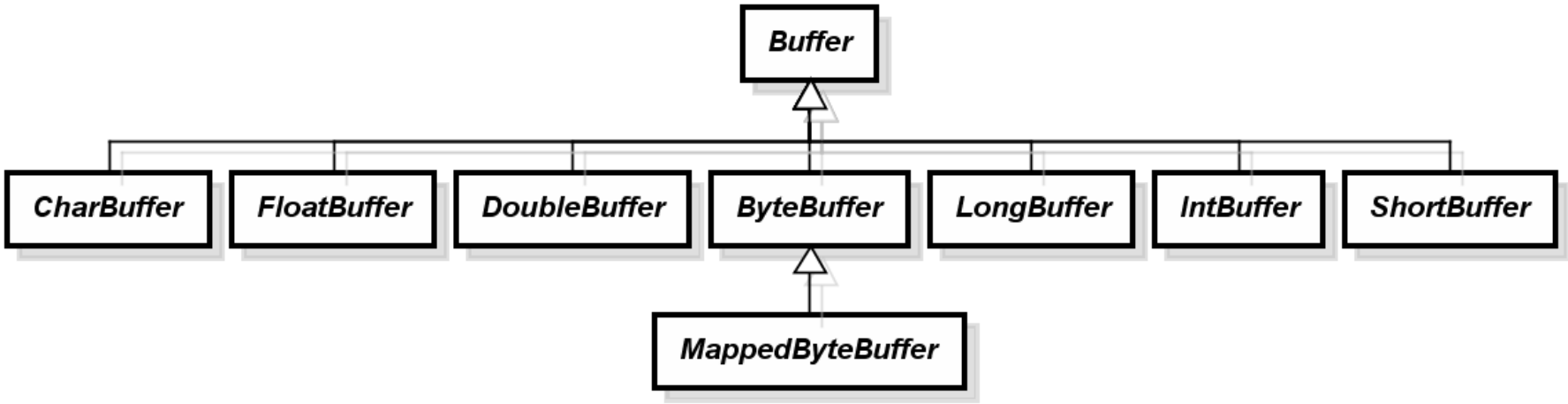
Channel 架構與操作

- 有些 InputStream、OutputStream 實例本身也有方法可以取得 Channel 實例
- FileInputSteam、FileOutputStream 都有個 getChannel() 方法可以分別取得 FileChannel 實例

Channel 架構與操作

- 已經有相關的 Channel 實例，也可以透過 Channels 上其他 newXXX() 靜態方法，取得 InputStream、OutputStream、Reader、Writer 實例

Buffer 架構與操作



Buffer 架構與操作

- 容量、界限與存取位置
- clear()、 flip() 與 rewind()
- mark()、 reset()、 remaining()

Buffer 架構與操作

- Buffer 的直接子類別們都有個 allocate() 靜態方法，可以讓你指定 Buffer 容量（Capacity）
- 如果想取得 Buffer 內部的陣列，可以使用 array() 方法
- 陣列想要轉為某個 Buffer 子類實例，每個 Buffer 子類別實例都有 wrap() 靜態方法

Buffer 架構與操作

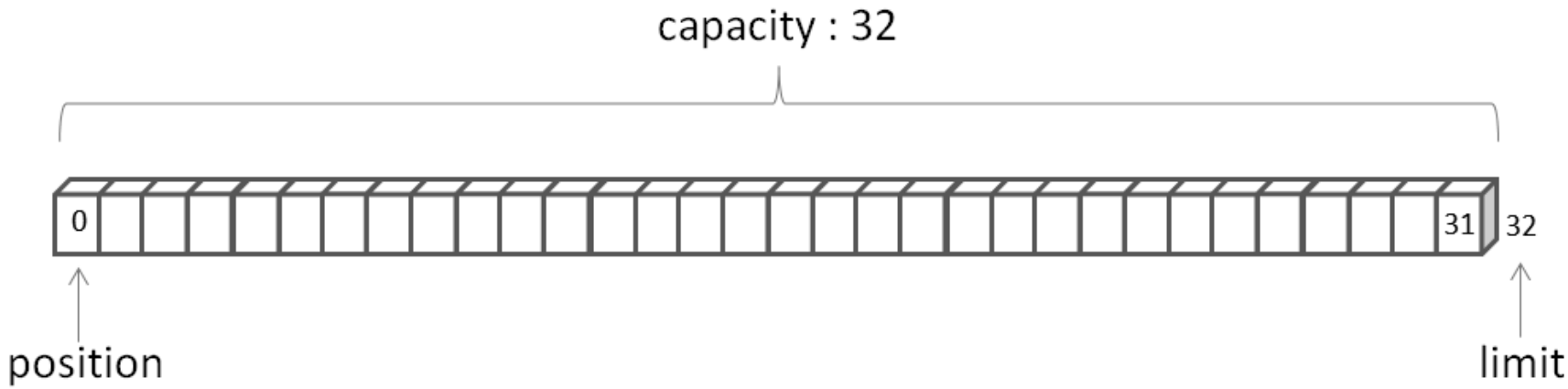
- ByteBuffer 還有一個 allocateDirect() 方法
- 會利用作業系統的原生 I/O 操作，試著避免 JVM 的中介轉接
- 建議用在大型、存活長的 ByteBuffer 物件

Buffer 架構與操作

- Buffer 是容器，填裝資料不會超過它的容量
- 實際可讀取或寫入的資料界限（Limit）索引值可以由 limit() 方法得知或設定
- 下一個可讀取資料的位置（Position）索引值，可以使用 position() 方法得知或設定

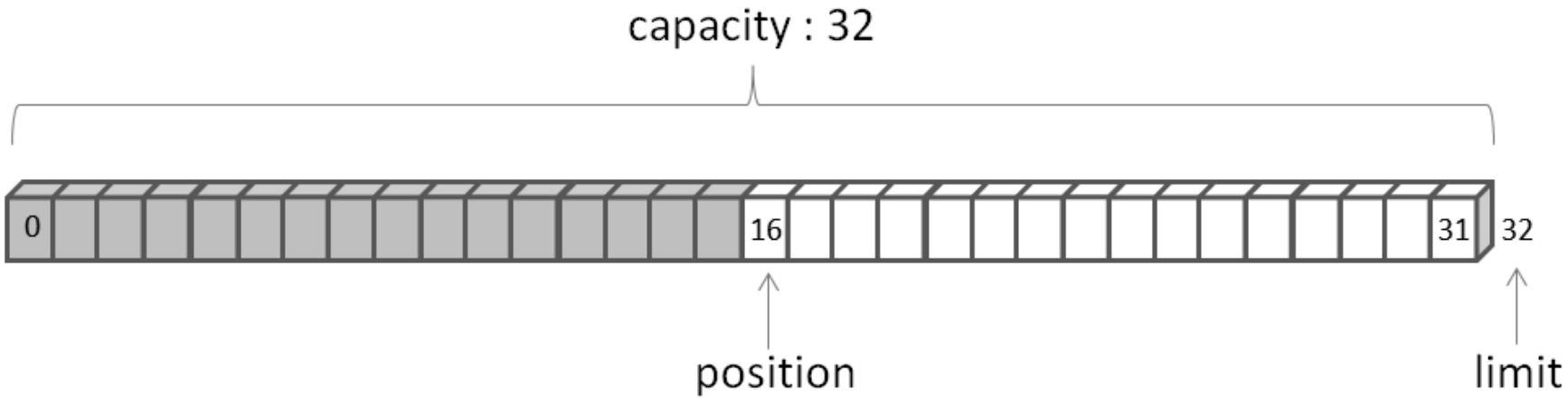
Buffer 架構與操作

- ByteBuffer 初建立或呼叫 clear()



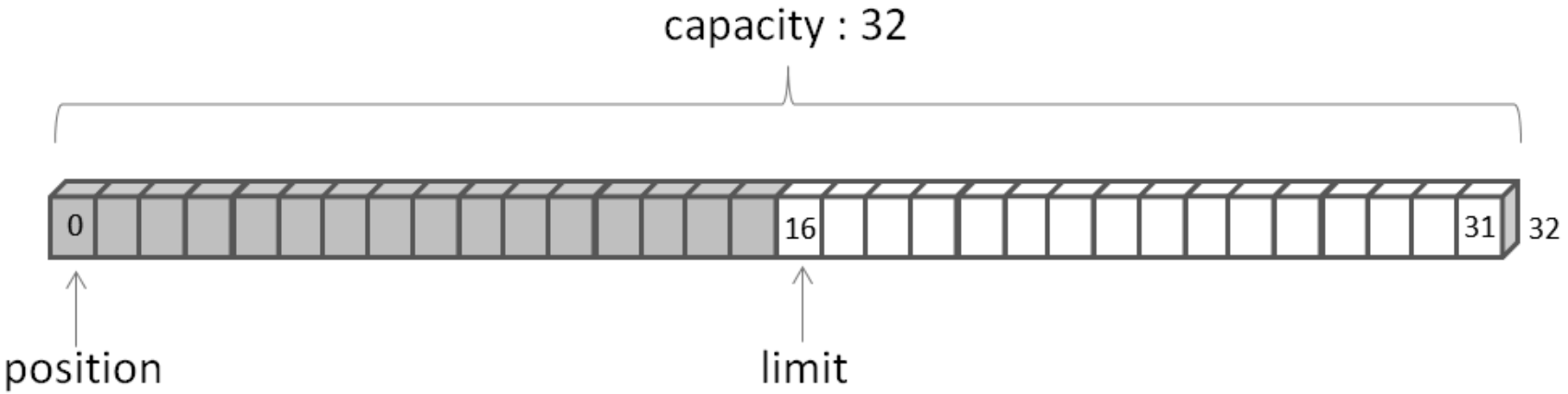
Buffer 架構與操作

- ByteBuffer 被寫入了 16 位元組



Buffer 架構與操作

- ByteBuffer 呼叫了 flip() 方法



```
public static void dump(ReadableByteChannel src,
                        WritableByteChannel dest) throws IOException {
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    try(ReadableByteChannel srcCH = src;
            WritableByteChannel destCH = dest) {
        while(srcCH.read(buffer) != -1) {
            buffer.flip();
            destCH.write(buffer);
            buffer.clear();
        }
    }
}

// 測試用的 main
public static void main(String[] args) throws Exception {
    URL url = new URL("http://openhome.cc");
    ReadableByteChannel src = Channels.newChannel(url.openStream());
    WritableByteChannel dest =
            new FileOutputStream("index.html").getChannel();
    NIOUtil.dump(src, dest);
}
```

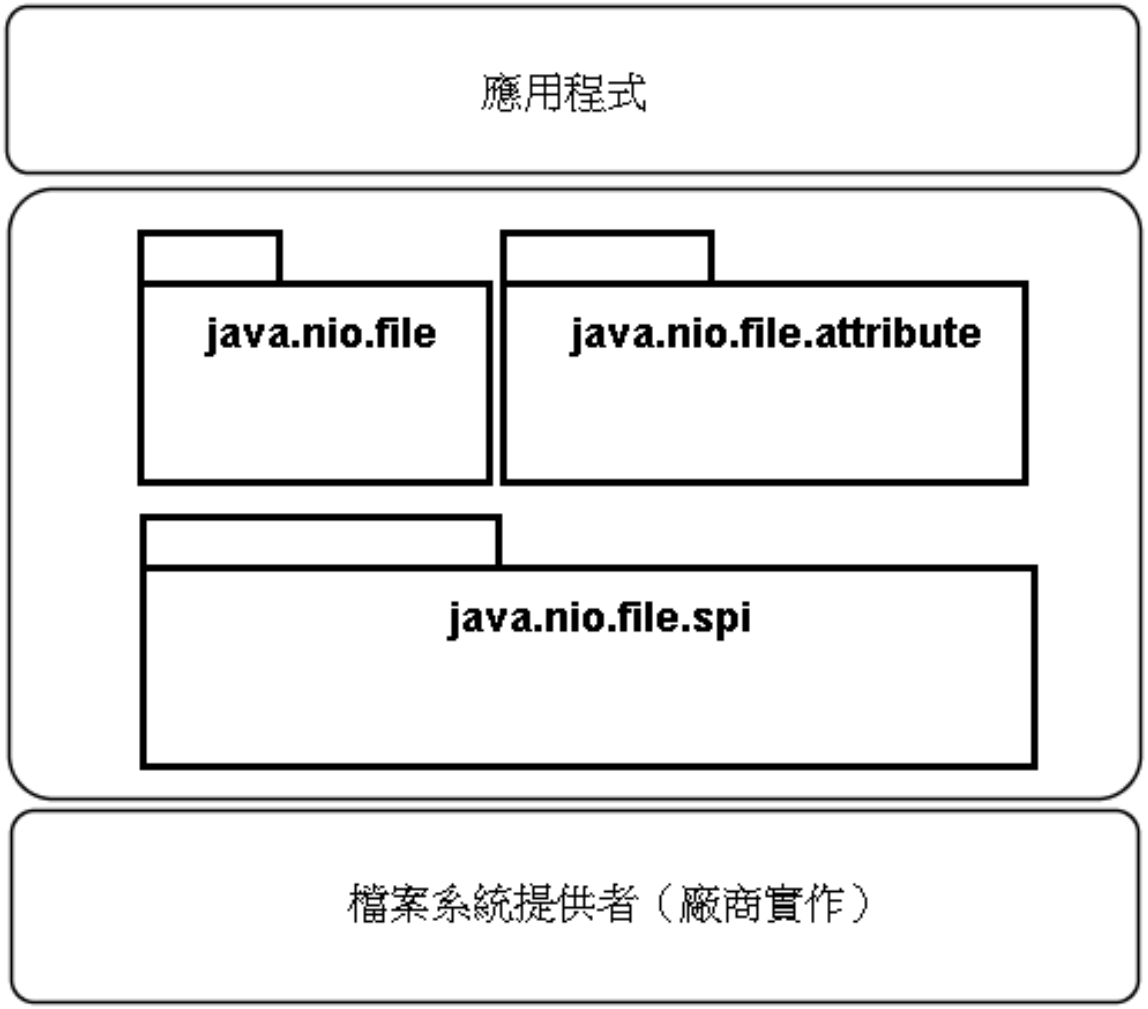
Buffer 架構與操作

- 呼叫 `rewind()` 方法的話，會將 position 設為 0，而 limit 不變
- 這個方法通常用在想要重複讀取 `Buffer` 中某段資料時使用，作用相當於單獨呼叫 `Buffer` 的 `position(0)` 方法。

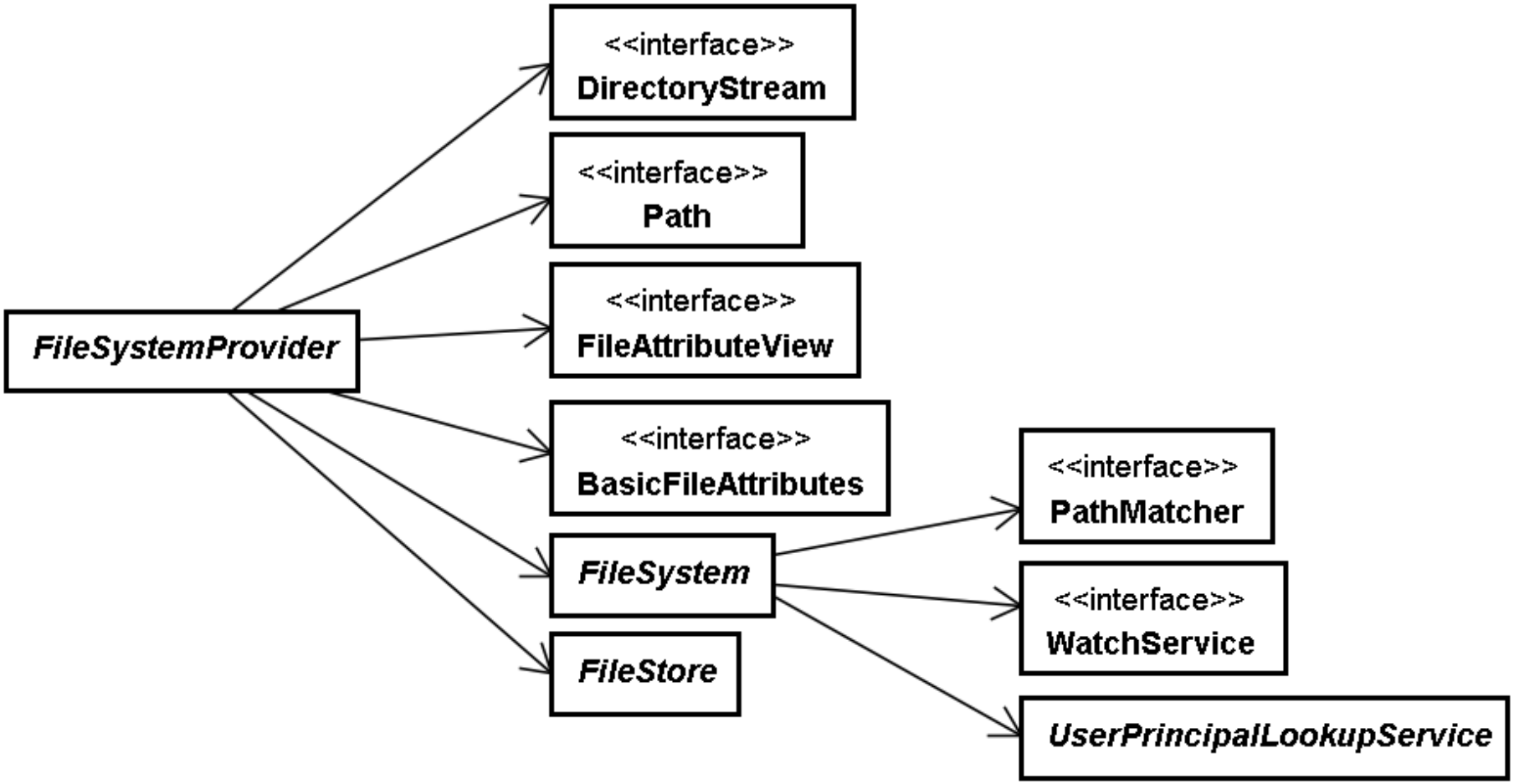
NIO2 檔案系統

- JDK7 在 **java.nio.file**、**java.nio.file.attribute** 與 **java.nio.file.spi** 套件中，提供了存取預設檔案系統進行各種輸入輸出的 API
  - 簡化現有檔案輸入輸出 API 的操作
  - 增加許多過去沒有提供的檔案系統存取功能

概述 API 架構


應用程式
java.nio.file
java.nio.file.attribute
java.nio.file.spi
檔案系統提供者（廠商實作）

概述 API 架構


<<interface>>
DirectoryStream
<<interface>>
Path
<<interface>>
FileAttributeView
FileSystemProvider
<<interface>>
BasicFileAttributes
<<interface>>
PathMatcher
FileSystem
<<interface>>
WatchService
FileStore
UserPrincipalLookupService

概述 API 架構

- 應用程式開發者可以透過 `java.nio.file` 套件中 **`FileSystems`**、 **`Paths`**、 **`Files`** 等類別提供的靜態方法，取得相關實作物件或進行各種檔案系統操作
- 這些靜態方法內部會運用 `FileSystemProvider` 來取得所需的實作物件，完成應有的操作

概述 API 架構

- 想要取得 **java.nio.file.FileSystem** 實作物件：

```
FileSystem fileSystem = FileSystems.getDefault();
```

- 可以使用系統屬性 "java.nio.file.spi.DefaultFileSystemProvider" 指定該廠商實作的類別名稱

操作路徑

- Path 實例是在 JVM 中路徑的代表物件，也是 NIO2 檔案系統 API 操作的起點

```
Path workspace = Paths.get("C:\\workspace");   \\ Windows 下絕對路徑
Path books = Paths.get("Desktop\\books");       \\ Windows 下相對路徑
Path path = Paths.get(args[0]);
```

- Paths.get() 的第二個參數開始接受不定長度引數

```
Path path = Paths.get(
    System.getProperty("user.home"), "Documents", "Downloads");
```

操作路徑

- Path 實例僅代表路徑資訊，路徑實際對應的檔案或資料夾（也是一種檔案）不一定存在
- Path 提供一些方法取得路徑的各種資訊

```
Path path = Paths.get(
        System.getProperty("user.home"), "Documents", "Downloads");
out.printf("toString: %s%n", path.toString());
out.printf("getFileName: %s%n", path.getFileName());
out.printf("getName(0): %s%n", path.getName(0));
out.printf("getNameCount: %d%n", path.getNameCount());
out.printf("subpath(0,2): %s%n", path.subpath(0, 2));
out.printf("getParent: %s%n", path.getParent());
out.printf("getRoot: %s%n", path.getRoot());
```

操作路徑

- Path 實作了 Iterable 介面

```
Path path = Paths.get(
    System.getProperty("user.home"),"Documents", "Downloads");
path.forEach(out::println);
```

- 路徑中若有冗餘資訊，可以使用 **normalize()** 方法移除

```
Path path1 = Paths.get(
       "C:\\Users\\Justin\\.\\Documents\\Downloads").normalize();
Path path2 = Paths.get(
       "C:\\Users\\Monica\\..\\Justin\\Documents\\Downloads").normalize();
```

操作路徑

- **`toAbsolutePath()`** 方法可以將（相對路徑） `Path` 轉為絕對路徑 `Path`
- 如果路徑是符號連結（Symbolic link），**`toRealPath()`** 可以轉換為真正的路徑，若是相對路徑則轉換為絕對路徑，若路徑中有冗餘資訊也會移除

操作路徑

- 路徑與路徑可以使用 **`resolve()`** 結合。例如以下最後得到代表 C:\Users\Justin 的 Path 實例：

```
Path path1 = Paths.get("C:\\Users");
Path path2 = Paths.get("Justin");
Path path3 = path1.resolve(path2);
```

操作路徑

- 想知道如何從一個路徑切換至另一路徑，則可以使用 **`relativize()`** 方法

```
Path p1 = Paths.get(System.getProperty("user.home"),
                    "Documents", "Downloads");
Path p2 = Paths.get("C:\\workspace");
Path p1ToP2 = p1.relativize(p2);
out.println(p1ToP2);        // 顯示..\..\..\..\workspace
```

操作路徑

- 使用 **equals()** 方法比較兩個 Path 實例的路徑是否相同
- 使用 **startsWith()** 比較路徑起始是否相同
- 使用 **endsWith()** 比較路徑結尾是否相同
- 如果檔案系統支援符號連結，兩個路徑不同的 Path 實例，有可能是指向同一檔案
  - 可以使用 **Files.isSameFile()** 測試看看是否如此

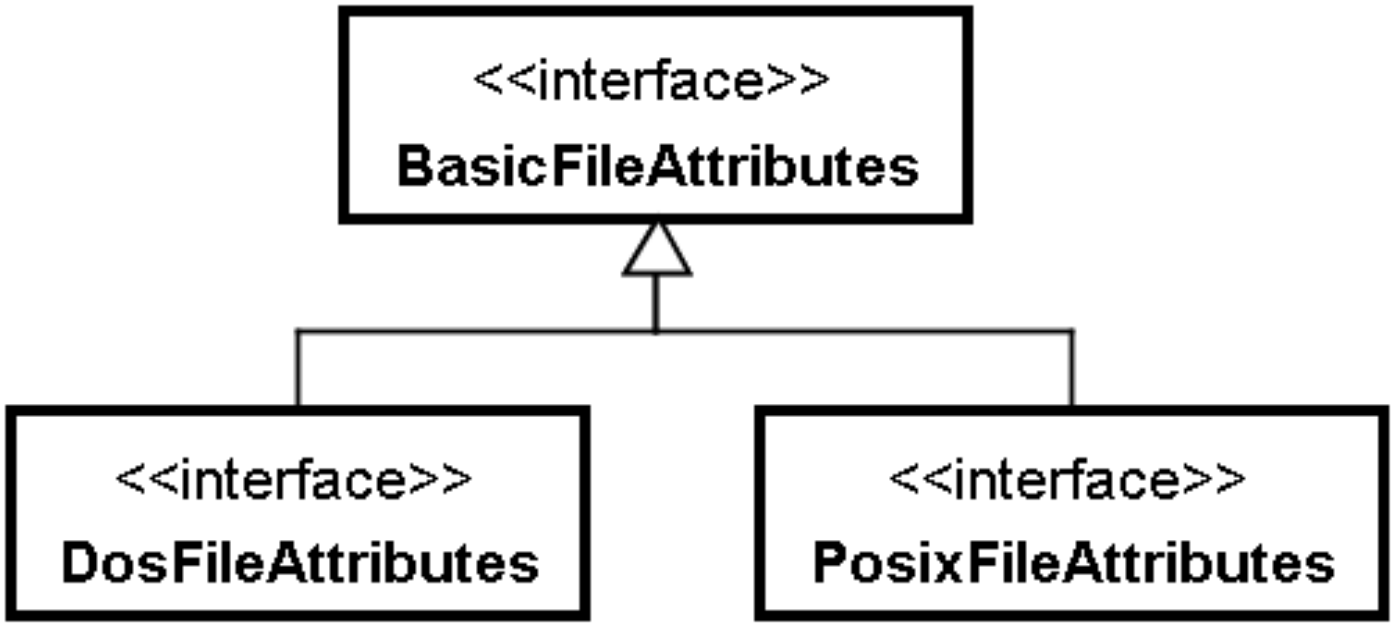
操作路徑

- 想確定 Path 代表的路徑，實際上是否存在檔案，可以使用 **Files.exists()** 或 **Files.notExists()**
  - Files.exists() 僅在檔案存在時傳回 true，如果檔案不存在或無法確認存不存在（例如沒有權限存取檔案）則傳回 false
  - Files.notExists() 會在檔案不存在時傳回 true，如果檔案存在或無法確認存不存在則傳回 false

操作路徑

- 對於檔案的一些基本屬性，可以使用 `Files` 的 **`isExecutable()`**、**`isHidden()`**、**`isReadable()`**、**`isRegularFile()`**、**`isSymbolicLink()`**、**`isWritable()`** 等方法來得知
- 如果需要更多檔案屬性資訊，則必須透過 `BasicFileAttributes` 或搭配 `FileAttributeView` 來取得

屬性讀取與設定


<<interface>>
BasicFileAttributes
<<interface>>
DosFileAttributes
<<interface>>
PosixFileAttributes

屬性讀取與設定

```
Path file = Paths.get("C:\\Windows");
BasicFileAttributes attrs =
        Files.readAttributes(file, BasicFileAttributes.class);
out.printf("creationTime: %s%n", attrs.creationTime());
out.printf("lastAccessTime: %s%n",  attrs.lastAccessTime());
out.printf("lastModifiedTime: %s%n", attrs.lastModifiedTime());
out.printf("isDirectory: %b%n", attrs.isDirectory());
out.printf("isOther: %b%n", attrs.isOther());
out.printf("isRegularFile: %b%n",  attrs.isRegularFile());
out.printf("isSymbolicLink: %b%n", attrs.isSymbolicLink());
out.printf("size: %d%n", attrs.size());
```

屬性讀取與設定

- **creationTime()**、**lastAccessTime()**、**lastModifiedTime()** 傳回的是 **FileTime** 實例，也可以透過 **Files.getLastModifiedTime()** 取得最後修改時間
- 若想設定最後修改時間，可以透過 **Files.setLastModifiedTime()** 指定代表修改時間的 **FileTime** 實例：

```
long currentTime = System.currentTimeMillis();
FileTime ft = FileTime.fromMillis(currentTime);
Files.setLastModifiedTime(Paths.get("C:\\workspace\\Main.java"), ft);
```

屬性讀取與設定

- 屬性設定可透過 **`Files.setAttribute()`** 方法。例如設定檔案為隱藏：

```
Files.setAttribute(Paths.get(args[0]), "dos:hidden", true);
```

- `Files.setAttribute()` 第二個引數必須指定 **`FileAttributeView`** 子介面規範的名稱，格式為 **[view-name:]attribute-name**
  - view-name 可以從 **`FileAttributeView`** 子介面實作物件的 **`name()`** 方法取得（亦可查看 API 文件），如果省略就預設為“basic”
  - **attribute-name** 可在 `FileAttributeView` 各子介面的 API 文件中查詢

屬性讀取與設定

- 例如同樣設定最後修改時間，改用 `Files.setAttributes()` 可以如下撰寫：

```
long currentTime = System.currentTimeMillis();
FileTime ft = FileTime.fromMillis(currentTime);
Files.setAttribute(
     Paths.get("C:\\workspace\\Main.java"), "basic:lastModifiedTime", ft);
```

屬性讀取與設定

- 可以透過 **Files.getAttribute()** 方法取得各種檔案屬性，使用方式類似 setAttributes()
- 也可透過 **Files.readAttributes()** 另一版本取得 Map<String, Object> 物件，鍵部份指定屬性名稱，就可以取得屬性值

```
Map<String, Object> attrs = Files.readAttributes(
        Paths.get(args[0]), " size,lastModifiedTime,lastAccessTime");
```

屬性讀取與設定

- 可以如下取得 `DosFileAttributes` 實例

```
Path file = Paths.get(args[0]);
DosFileAttributes attrs =
                Files.readAttributes(file, DosFileAttributes.class);
```

屬性讀取與設定

- 如果想取得儲存裝置本身的資訊，可以利用 **`Files.getFileStore()`** 方法取得指定路徑的 **`FileStore`** 實例
- 或透過 `FileSystem` 的 **`getFileStores()`** 方法取得所有儲存裝置的 `FileStore` 實例

```
public static void main(String[] args) throws IOException {
    if (args.length == 0) {
        FileSystem fs = FileSystems.getDefault();
        for (FileStore store: fs.getFileStores()) {
            print(store);
        }
    }
    else {
        for (String file: args) {
            FileStore store = Files.getFileStore(Paths.get(file));
            print(store);
        }
    }
}

public static void print(FileStore store) throws IOException {
    long total = store.getTotalSpace();
    long used = store.getTotalSpace() - store.getUnallocatedSpace();
    long usable = store.getUsableSpace();
    DecimalFormat formatter = new DecimalFormat("#,###,###");
    out.println(store.toString());
    out.printf("\t- 總容量\t%s\t位元組%n", formatter.format(total));
    out.printf("\t- 可用空間\t%s\t位元組%n", formatter.format(used));
    out.printf("\t- 已用空間\t%s\t位元組%n", formatter.format(usable));
}
```

操作檔案與目錄

- 想要刪除 Path 代表的檔案或目錄，可以使用 **Files.delete()** 方法
  - 如果不存在，會拋出 **NoSuchFileException**
  - 如果因目錄不為空而無法刪除檔案，會拋出 **DirectoryNotEmptyException**
- 使用 **Files.deleteIfExists()** 方法也可以刪除檔案，這個方法在檔案不存在時呼叫，並不會拋出例外

操作檔案與目錄

- 如果想要複製來源 Path 的檔案或目錄至目的地 Path，可以使用 **Files.copy()** 方法
  - 第三個選項可以指定 **CopyOption** 介面的實作物件，CopyOption 實作類別有以 Enum 型態實作的 **StandardCopyOption** 與 **LinkOption**
  - StandardCopyOption 的 **REPLACE_EXISTING** 實例進行複製時，若目標檔案已存在就會予以覆蓋，**COPY_ATTRIBUTES** 會嘗試複製相關屬性
  - LinkOption 的 **NOFOLLOW_LINKS** 則不會跟隨符號連結

操作檔案與目錄

- 一個使用 `Files.copy()` 的範例如下：

```
Path srcPath = ...;
Path destPath = ...;
Files.copy(srcPath, destPath, StandardCopyOption.REPLACE_EXISTING);
```

- `Files.copy()` 還有重載兩個版本
  - 接受 `InputStream` 作為來源，可直接讀取資料，並將結果複製至指定的 `Path` 中
  - 將來源 `Path` 複製至指定的 `OutputStream`

操作檔案與目錄

- 可改寫 10.1.1 中的 Download 為以下：

```
public class Download {
    public static void main(String[] args) throws IOException {
        URL url = new URL(args[0]);
        Files.copy(url.openStream(), Paths.get(args[1]), REPLACE_EXISTING);
    }
}
```

操作檔案與目錄

- 若要進行檔案或目錄移動，可以使用 **Files.move()** 方法
- 如果要建立目錄，可以使用 **Files.createDirectory()** 方法，如果呼叫時父目錄不存在，會拋出 **NoSuchFileException**
- **Files.createDirectories()** 會在父目錄不存在時一併建立
- 如果要建立暫存目錄，可以使用 `Files.createTempDirectory()` 方法

操作檔案與目錄

- 對於 `java.io` 中的基本輸入輸出 API，NIO2 也作了封裝
  - 可使用 **`Files.readAllBytes()`** 讀取整個檔案，然後以 `byte[]` 傳回檔案內容
  - 如果檔案內容都是字元，則可使用 **`Files.readAllLines()`** 指定檔案 `Path` 與編碼，讀取整個檔案，將檔案中每行收集在 `List<String>` 中傳回

操作檔案與目錄

- 如果本來有個建立 BufferedReader 的片段如下：

```
BufferedReader reader =
     new BufferedReader(
          new InputStreamSReader(
               new FileInputStream(args[0]), "UTF-8"));
```

- 可使用 Files.newBufferedReader() 改寫如下：

```
BufferedReader reader =
        Files.newBufferedReader(Path.get(args[0]), "UTF-8");
```

操作檔案與目錄

- 如果想要以 `InputStream`、`OutputStream` 處理檔案，也有對應的 **`Files.newInputStream()`**、**`Files.newOutputStream()`** 可以使用

讀取、走訪目錄

- 如果想要取得檔案系統根目錄路徑資訊，可以使用 `FileSystem` 的 **`getRootDirectories()`** 方法

```
Iterable<Path> dirs = FileSystems.getDefault().getRootDirectories();
dirs.forEach(out::println);
```

讀取、走訪目錄

- 可以使用 **Files.newDirectoryStream()** 方法取得 **DirectoryStream** 介面實作物件，代表指定路徑下的所有檔案
- 在不使用 DirectoryStream 物件時必須使用 close() 方法關閉相關資源
- DirectoryStream 繼承了 Closeable 介面，其父介面為 AutoClosable 介面，所以可搭配嘗試關閉資源語法來簡化程式撰寫

讀取、走訪目錄

- `Files.newDirectoryStream()` 實際傳回的是 `DirectoryStream<Path>`
- `DirectoryStream` 也繼承了 `Iterable` 介面，所以可使用增強式 `for` 迴圈語法來逐一取得 `Path`

讀取、走訪目錄

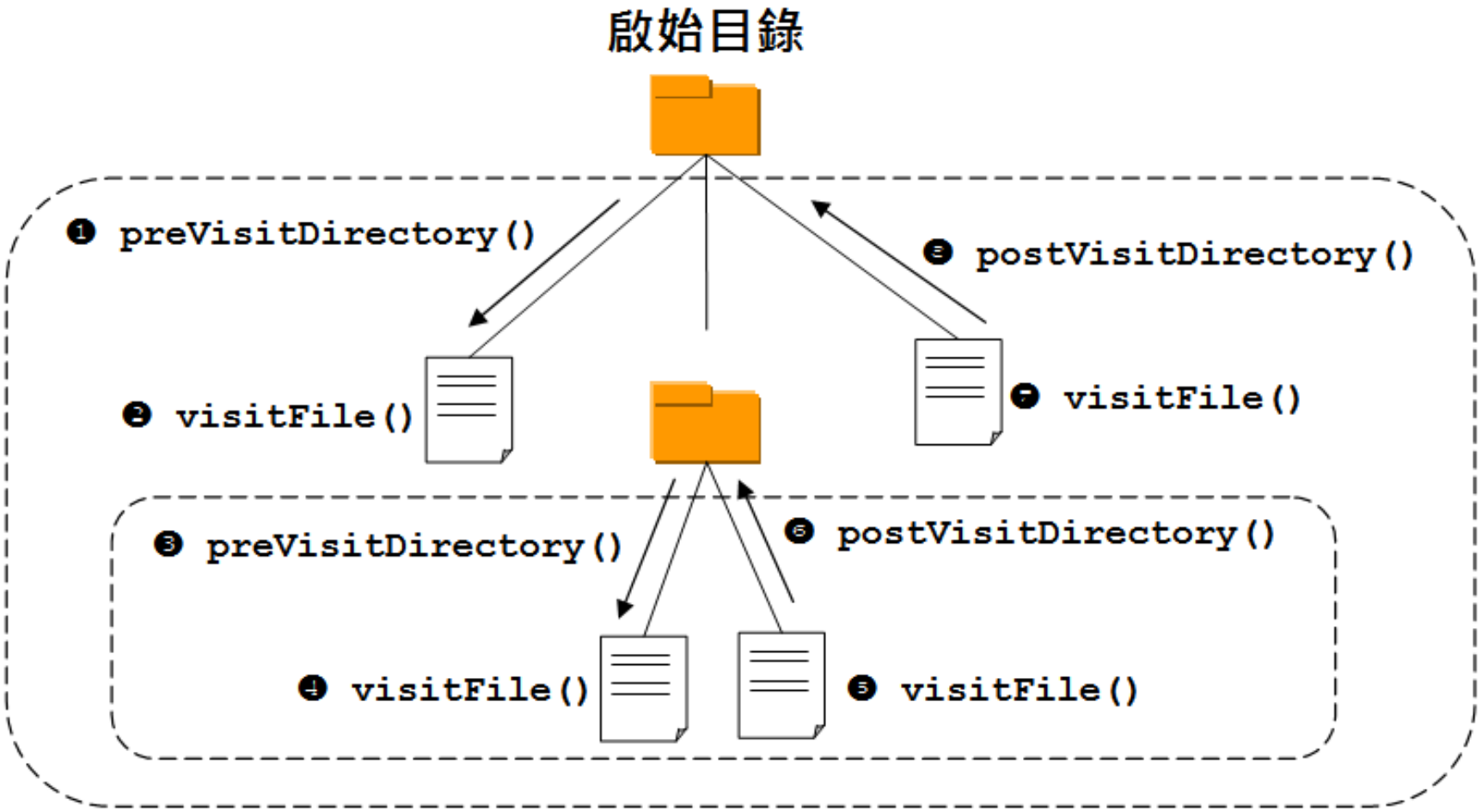
```
try(DirectoryStream<Path> directoryStream =
                    Files.newDirectoryStream(Paths.get(args[0]))) {
    List<String> files = new ArrayList<>();
    for(Path path : directoryStream) {
        if(Files.isDirectory(path)) {
            out.printf("[%s]%n", path.getFileName());
        }
        else {
            files.add(path.getFileName().toString());
        }
    }
    files.forEach(out::println);
}
```

讀取、走訪目錄

- 如果想要走訪目錄中所有檔案與子目錄，可以實作 **FileVisitor** 介面

```
package java.nio.file;
import java.nio.file.attribute.BasicFileAttributes;
import java.io.IOException;
public interface FileVisitor<T> {
    FileVisitResult preVisitDirectory(T dir, BasicFileAttributes attrs)
        throws IOException;
    FileVisitResult visitFile(T file, BasicFileAttributes attrs)
        throws IOException;
    FileVisitResult visitFileFailed(T file, IOException exc)
        throws IOException;
    FileVisitResult postVisitDirectory(T dir, IOException exc)
        throws IOException;
}
```

讀取、走訪目錄


啟始目錄
❶ preVisitDirectory()
❷ visitFile()
❸ preVisitDirectory()
❹ visitFile()
❺ visitFile()
❻ postVisitDirectory()
❼ visitFile()
❽ postVisitDirectory()

讀取、走訪目錄

- 可以繼承 **`SimpleFileVisitor`** 類別，這個類別實作了 `FileVisitor` 介面，你只要繼承之後重新定義感興趣的方法就可以了

```
public class ConsoleFileVisitor extends SimpleFileVisitor<Path> {
    @Override
    public FileVisitResult preVisitDirectory(Path path,
                                             BasicFileAttributes attrs) throws IOException {
        printSpace(path);
        out.printf("[%s]%n", path.getFileName());
        return CONTINUE;
    }

    @Override
    public FileVisitResult visitFile(Path path, BasicFileAttributes attr) {
        printSpace(path);
        out.printf("%s%n", path.getFileName());
        return CONTINUE;
    }

    @Override
    public FileVisitResult visitFileFailed(Path file, IOException exc) {
        err.println(exc);
        return CONTINUE;
    }

    private void printSpace(Path path) {
         out.printf("%" + path.getNameCount() * 2 + "s", "");
    }
}
```

讀取、走訪目錄

- 如果要使用 `FileVisitor` 走訪目錄，可以使用 **`Files.walkFileTree()`** 方法：

```
Files.walkFileTree(Paths.get(args[0]), new ConsoleFileVisitor());
```

讀取、走訪目錄

- Files 新增了 list() 與 walk() 靜態方法，它傳回的是 Stream<Path>
- lines()、list() 與 walk() 傳回的 Stream 不使用時需要呼叫 close() 方法來釋放資源

```
try (Stream<Path> paths = Files.list(Paths.get(args[0]))) {
    paths.forEach(out::println);
}

try (Stream<Path> paths = Files.walk(Paths.get(args[0]))) {
     paths.forEach(out::println);
}
```

過濾、搜尋檔案

- 想顯示 .class 與 .jar 檔案：

```
try(DirectoryStream<Path> directoryStream = Files.newDirectoryStream(
                                    Paths.get(args[0]), "*.{class,jar}")) {
    directoryStream.forEach(path -> out.println(path.getFileName()));
}
```

- 像 `*.{class,jar}` 這樣的語法稱之為 Glob

過濾、搜尋檔案

符號	說明
*	比對零個或多個字元
**	跨目錄比對零個或多個字元
?	比對一個字元
{}	比對收集的任一子模式，例如{class,jar}比對 class 或 jar，{tmp,temp*}比對 tmp 或 temp 開頭
[]	比對收集的任一字元，例如[acx]比對 a、c、x 任一字元，可使用-比對範圍，例如[a-z]比對 a 到 z 任一字元，[A-Z,0-9]比對 A 到 Z 或 0-9 任一字元，在[]中的*、?、\就是進行字元比對，例如[*?\]就是比對*、?、\任一字元
\	忽略符號，例如要比對*、?、\，就要撰寫為*、\?、\\。
其他字元	比對字元本身

過濾、搜尋檔案

- `*.java` 比對 .java 結尾的字串。
- `**/*Test.java` 跨目錄比對 Test.java 結尾的字串，例如 BookmarkTest.java 、 CommandTest.java 都符合。
- `???` 符合三個字元，例如 123 、 abc 會符合。
- `a?*.java` 比對 a 之後至少一個字元，並以 .java 結尾的字串。
- `*.{class,jar}` 符合 .class 或 .jar 結尾的字串。
- `*[0-9]*` 比對的字串中要有一個數字。
- `{*[0-9]*,*.java}` 比對字串中要有一個數字，或者是 .java 結尾。

過濾、搜尋檔案

```
public class LS {
    public static void main(String[] args) throws IOException {
        // 預設取得所有檔案
        String glob = args.length == 0 ? "*" : args[0];

        // 取得目前工作路徑
        Path userPath = Paths.get(System.getProperty("user.dir"));
        try (DirectoryStream<Path> directoryStream =
                Files.newDirectoryStream(userPath, glob)) {
            directoryStream.forEach(path -> out.println(path.getFileName()));
        }
    }
}
```

過濾、搜尋檔案

- 如果 Glob 語法無法滿足條件過濾需求時，可以自行實作 `DirectoryStream.Filter` 的 **`accept()`** 方法自訂過濾條件

```
DirectoryStream.Filter<Path> filter = new DirectoryStream.Filter<Path>() {
    public boolean accept(Path path) throws IOException {
        return (Files.isDirectory(path));
    }
};
try (DirectoryStream<Path> directoryStream =
     Files.newDirectoryStream(Paths.get(args[0]), filter)) {
    directoryStream.forEach(path -> out.println(path.getFileName()));
}
```

過濾、搜尋檔案

- 可以使用 `FileSystem` 實例的 **`getPathMatcher()`** 取得 **`PathMatcher`** 介面實作物件
  - 可以指定使用哪種模式比對語法，**`"regex"`** 表示使用規則表示式語法、`"glob"` 表示 Glob 語法

```
PathMatcher matcher = FileSystems.getDefault()
                                 .getPathMatcher("glob:*.{class,jar}");
```

```
public static void main(String[] args) throws IOException {
    // 預設使用 Glob 取得所有檔案
    String syntax = args.length == 2 ? args[0] : "glob";
    String pattern = args.length == 2 ? args[1] : "*";
    out.println(syntax + ":" + pattern);

    // 取得目前工作路徑
    Path userPath = Paths.get(System.getProperty("user.dir"));
    PathMatcher matcher = FileSystems.getDefault()
                              .getPathMatcher(syntax + ":" + pattern);
    try (DirectoryStream<Path> directoryStream =
           Files.newDirectoryStream(userPath)) {
       directoryStream.forEach(path -> {
           Path file = Paths.get(path.getFileName().toString());
            if(matcher.matches(file)) {
                out.println(file.getFileName());
            }
       });
    }
}
```