緯創國泰補充

database：

DDL(Data Definition Language)資料定義語言 ：CREATE、ALTER、DROP

DML(Data Manipulation Language)資料處理語言：INSERT、SELECT、UPDATE、DELETE

TCL(Transaction Control Language)交易控制語言：BEGIN、COMMIT、ROLLBACK、SAVEPOINT

DCL(Data Control Language)資料控制語言：主要用於控制不同用戶對資料庫對象（如資料表、檢視表(VIEW)）的訪問權限 GRANT、REVOKE

char：是固定長度的字段類型，如電話號碼、郵遞區號。

varchar ：是可變長度的欄位類型，使用者名稱、地址。

int 整數 要記住使用範圍

float、double不要使用

decimal(p 有效位數, s 小數位數) 小數使用

Transaction：

A－原子性（Atomicity）

事務是不可分割的單元，或全部成功執行，或全部失敗回滾。
如果交易中某個操作失敗，所有已執行的操作都會撤銷，資料庫狀態回滾到交易開始前的狀態。
C－一致性（Consistency）

事務的執行不會破壞資料庫的一致性約束。
I－隔離性（Isolation）

執行的事務之間互不干擾，每個事務的中間狀態對其他事務不可見。
資料庫透過隔離等級來控制交易之間的影響。
D－持久性（耐久性）

一旦提交事務（Commit），其結果將保存到資料庫中，即使系統永久崩潰也不會遺失。

生命週期：

開始事務

執行一系列的 SQL 操作。
在 MySQL 中，可以透過START TRANSACTION或BEGIN開始事務。
執行操作

修改包括INSERT、UPDATE、等資料操作。DELETE
提交事務（Commit）

提交事務的所有操作，使變更永久保存到資料庫中。
回滾事務（Rollback）

如果事務中出現錯誤或失敗，可以回滾事務，使資料庫恢復到事務開始前的狀態。

語法：

START/BEGIN、COMMIT、ROLLBACK、SAVEPOINT

Isolation Level(事務隔離層級)：低到高`READ UNCOMMITTED`、`READ COMMITTED`、`REPEATABLE READ`和`SERIALIZABLE`

隔離等級	贓讀	不可重複讀	幻讀	隨機性
未提交的閱讀	可能	可能	可能	最高
已提交讀	不可能	可能	可能	高
可重複讀取	不可能	不可能	可能	中
可串列化	不可能	不可能	不可能	最低

髒讀（Dirty Read）、不可重複讀（Non-repeatable Read）　　、幻讀（Phantom Read）

閉鎖：

SQL 閉鎖是指兩個或多個事務在執行過程中相互等待對方釋放資源（如行鎖或表鎖），導致所有事務無法繼續執行的情況。

SQL 閉鎖（Deadlock）說明

SQL 閉鎖是指兩個或多個事務在執行過程中相互等待對方釋放資源（如行鎖或表鎖），導致所有事務無法繼續執行的情況。閉鎖通常發生在多事務並發操作時，並可能影響系統性能甚至導致應用程序崩潰。

閉鎖形成的條件

閉鎖的發生需要滿足以下條件（又稱「死鎖條件」）：

1. **互斥條件**：資源同一時間只能被一個事務使用。
2. **請求與保持**：事務已經持有資源，並請求其他資源，但未釋放已持有的資源。
3. **不可剝奪**：資源不能被強制從事務中釋放，必須由事務自行釋放。
4. **環路等待**：多個事務形成資源等待的循環鏈，事務彼此等待其他事務釋放資源。

排除閉鎖：

1. 透過資料庫設置或工具
2. 設計良好的 SQL 操作順序
3. 減少鎖範圍
4. 事務粒度控制
5. 分析與優化查詢
6. 程式層級解決方案

閉鎖的最佳實踐

1. **優化查詢**：盡量使用索引，減少掃描範圍。
2. **控制事務大小**：將事務劃分為更小的單位。
3. **避免過多並發**：限制高峰期的同時事務數量。

4. **資源順序一致性**：所有事務統一資源的請求順序。

Java 中的 JAR、WAR、EAR 文件的區別與用途

1. JAR (Java ARchive)

用途

JAR 文件是用於打包 Java 應用程序或庫的壓縮文件，主要用於桌面應用程序、Java 庫或模塊的分發。

特點

- **結構簡單**：包括 `.class` 文件、資源文件（如圖片、配置文件）以及 `META-INF` 文件夾中的 `MANIFEST.MF` 描述文件。
- **可執行性**：可以生成可執行的 JAR 文件，通過 `java -jar` 命令執行。
- **用途廣泛**：通常用於工具類庫或獨立的 Java 應用。

應用場景

- 單機 Java 應用程序。
- 第三方庫或依賴包（如 Spring 框架、Hibernate）。

---

2. WAR (Web Application Archive)

用途

WAR 文件是用於打包 Web 應用程序的壓縮文件，專為部署在應用伺服器（如 Tomcat、Jetty）設計。

特點

- **Web 專用結構**：包括 `WEB-INF` 文件夾，內含 `web.xml` 配置文件、類文件、靜態資源（如 HTML、CSS、JS）。
- **部署方便**：可以直接部署到應用伺服器，伺服器會自動解壓和運行。

- **支持多種組件**：支持 JSP 文件、Servlet、靜態資源以及其他 Java Web 組件。

應用場景

- Java Web 應用程序（如網站、REST API）。

---

3. EAR (Enterprise Archive)

用途

EAR 文件是用於打包企業級應用程序的壓縮文件，主要針對大型的 JEE（Java EE）應用，支持分佈式系統和多模塊部署。

特點

- **多模塊支持**：可以包含多個 JAR 和 WAR 文件。
- **企業級功能**：適合需要 EJB（Enterprise JavaBeans）、分散式事務、JMS（Java Message Service）等功能的應用。
- **高複雜性**：包括 `META-INF` 文件夾中的 `application.xml` 配置文件，用於描述應用程序結構。

應用場景

- 大型企業應用。
- 需要多模塊整合和企業級功能（如事務管理、消息隊列）的應用。

對比總結

特性	JAR	WAR	EAR
用途	Java 程式庫或獨立應用程序	Web 應用程序	企業級應用程序
內容	`.class` 文件、資源文件	JAR 文件、JSP、靜態資源等	多個 JAR 和 WAR 文件
部署環境	任意支持 JVM 的環境	Web 應用伺服器（如 Tomcat）	Java EE 伺服器（如 WildFly）
使用場景	工具類或桌面應用	網站、REST API	分佈式企業應用

結構	簡單	中等複雜度	高度複雜

如何選擇？

1. 如果是 **桌面應用** 或 **工具庫**，使用 **JAR**。
2. 如果是 **Web 應用**，使用 **WAR**。
3. 如果是 **企業級應用**，使用 **EAR**。

Java 的 Call by Value 與 Call by Reference

1. Call by Value（值傳遞）

在 Java 中，方法調用時，實際傳遞的是參數的**副本**（值），而不是參數本身。這意味著在方法內對參數的更改，不會影響方法外部的原始參數。

2. 物件參數的處理

物件作為參數時，Java 傳遞的仍然是**物件引用的副本**，而不是物件本身。因此，物件參數既不完全是值傳遞，也不是真正的引用傳遞。

3. 不支持 Call by Reference（引用傳遞）的證明

如果 Java 支持引用傳遞，那麼我們應該能夠改變物件參考本身（即讓外部的參考指向一個新物件），但實際上不行。

總結比較

傳遞方式	行為	Java 支援
Call by Value	傳遞參數的副本，方法內的修改不影響原始參數。	✅ 支援
Call by Reference	傳遞參數的記憶體地址，方法內的修改會直接影響原始參數（參考指向的改變也有效）。	❌ 不支援
物件參數特殊性	傳遞物件引用的副本，可修改物件的內容，但不能改變參考指向。	✅ 支援

在 Java 中，所有參數的傳遞都是 **Call by Value**，即使是對象類型的參數也僅僅是傳遞對象引用的副本。Java 並不支援 **Call by Reference**。但為了更清楚地說明這一點，我會展示兩個範例：一個是使用基本類型的 **Call by Value**，另一個是使用對象的 **Call by Value**。

1. 基本數據類型的 Call by Value 範例

對於基本數據類型（如 `int`、`float` 等），傳遞的其實是變數的副本。這意味著即使方法內部修改了參數，也不會影響到外部的原始變數。

2. 對象類型的 Call by Value 範例

對於對象類型（如 `String`、`Array`、自定義的 `Class` 等），傳遞的是引用的副本。這意味著方法內部的對象內容可以被修改，但修改引用本身不會改變外部的引用。

為什麼 Java 沒有 Call by Reference？

Java 使用 **Call by Value**，即使對於對象也是傳遞引用的副本。這意味著：

- **對於基本類型**，Java 傳遞的是數據本身的副本，修改不會影響原始數據。
- **對於對象類型**，Java 傳遞的是對象引用的副本，這樣方法內部可以修改對象的內容，但不能改變原始引用指向的對象。

因此，Java 中不會直接支持 **Call by Reference** 的方式，也就是說，無法像 C++ 中那樣直接改變外部變數的引用。

總結

- Java 中所有的參數傳遞都是 **Call by Value**，這一點適用於基本數據類型和對象引用。
- **基本類型**：傳遞的是變數的值副本，修改參數不會影響原始變數。
- **對象類型**：傳遞的是引用的副本，修改對象的內容會影響外部對象，但修改引用本身不會影響外部的引用。

String call by value

`String` 類型在 Java 中的行為與其他對象類型的行為有所不同，這是因為 `String` 是 **不可變（immutable）** 的類型。這意味著一旦創建了 `String` 對象，它的內容就不能再被改

變。即使 `String` 是對象類型，因為其不可變性，這也導致了它在傳遞過程中的一些特殊性。

`String` 的傳遞：為什麼看起來像 "call by reference"？

儘管 `String` 是對象類型，並且 Java 的傳遞機制是 **按值傳遞（call by value）**，但由於 `String` 是不可變的，這造成了 `String` 在使用中的一些特別行為。

1. 傳遞過程中的參數是引用的副本

就像其他對象類型一樣，當 `String` 作為方法參數傳遞時，傳遞的是對該 `String` 物件引用的副本。因此，方法內部可以使用這個副本對 `String` 進行操作。但因為 `String` 是不可變的，所有看似改變 `String` 值的操作其實是創建了一個新的 `String` 對象。

2. 不可變性導致的方法內部操作不會改變原始對象

由於 `String` 不可變，即使方法內部對 `String` 參數進行了改變（例如字符串拼接或替換），這些操作並不會影響到原始的 `String` 對象。實際上，這些操作會生成一個新的 `String` 物件，而原來的 `String` 仍然保持不變。

java 字串相加

建議使用
`StringBuilder` 或 `StringBuffer`，而不是直接使用 `+` 來進行字串連接。這是因為每次使用 `+` 進行字串拼接時，會創建新的字串對象，從而導致性能低下，特別是在迴圈中進行大量拼接時。接下來，我會詳細介紹 `StringBuilder` 和 `StringBuffer` 的區別，並說明為什麼它們比直接使用 `+` 更有效率。

`StringBuilder` vs `StringBuffer`

`StringBuilder` 和 `StringBuffer` 都是可變的字串類別，允許在不創建新對象的情況下修改字串的內容。這些類別非常適合用於需要多次修改字串的情況。它們的區別主要在於 **執行緒安全性**：

- `StringBuilder`：
  - 是非執行緒安全的（即不會在多線程環境中自動進行同步處理）。
  - 由於它不需要同步機制，性能較 `StringBuffer` 更好，適用於單線程或不需要同步的情況。

- `StringBuffer`：
  - 是執行緒安全的（即它會自動進行同步，確保在多線程環境中是安全的）。
  - 因為有同步機制，性能會比 `StringBuilder` 慢，適用於多線程環境中對字串進行修改的情況。

`StringBuilder` 和 `StringBuffer` 的優勢

- **性能更好**：由於 `StringBuilder` 和 `StringBuffer` 是可變的字串類型，它們會直接操作內部的字符數組，避免了多次創建新的字串對象，這樣能顯著提高性能。
- **內存使用更高效**：不會頻繁創建新對象，減少了垃圾回收的負擔。
- `StringBuffer` **適合多線程**：如果你的應用中需要在多線程環境下修改字串，`StringBuffer` 是合適的選擇，因為它具有同步機制來保證執行緒安全。

要考慮到使用情境，現在的環境都已`StringBuffer` 為主。

static

`static` 是一個關鍵字，用來標記類別層級的成員（如變數、方法、區塊、內部類別），而不是物件層級的成員。

- **靜態變數**：屬於類別本身，所有實例共享。
- **靜態方法**：屬於類別本身，可以直接訪問靜態變數和其他靜態方法，無法訪問實例變數和方法。
- **靜態區塊**：用來初始化靜態變數，並且在類別加載時執行一次。
- **靜態內部類別**：內部類別的一種，它不需要外部類別的實例。

使用 `static` 使得某些成員可以不依賴於物件實例就能被訪問，但也要小心它的使用，以免過度依賴靜態成員，導致程式不夠靈活。

`static final` 常量可以提高程式執行效率，因為它在內存中是共享且不可更改的。

交易控制

Atomicity (原子性)

交易是一個不可分割的單位，所有操作必須全部成功或全部失敗。若交易中途出錯，資料庫會回滾至交易開始前的狀態，確保資料一致性。
Consistency (一致性)

交易必須使資料庫從一個一致的狀態轉換到另一個一致的狀態。在交易開始和結束時，資料庫都必須遵循其所有的完整性約束。
Isolation (隔離性)

多個交易同時執行時，彼此不應相互干擾。每個交易的執行效果應與其獨立執行時的效果相同。
Durability (持久性)

一旦交易完成，其結果應永久保存，即使系統發生故障也不會丟失。

MVC 與 MVVM

MVC：

- **Model**：處理資料邏輯和業務邏輯。負責與資料庫交互，管理資料的狀態。
- **View**：負責顯示資料的界面。它是使用者看到的部分。
- **Controller**：負責處理使用者輸入，協調 Model 和 View 之間的互動。它是應用的中介。

MVVM：

- **Model**：與 MVC 的 Model 相似，負責資料邏輯。
- **View**：同樣負責界面顯示，但它不直接與 Model 交互。
- **ViewModel**：作為 View 與 Model 的中介，負責處理與界面邏輯相關的行為。它實現了雙向資料綁定（Two-way Data Binding），讓 View 的更新自動反映在 ViewModel 上，反之亦然。

Model 1 跟 Model 2

是兩種不同的架構模式，主要用於早期的 Web 開發中，特別是在基於 Java 的應用中（例如 JSP/Servlet）。這兩種模式的主要區別在於控制邏輯的設計和分工方式。

特性	Model 1	Model 2

結構設計	單層架構，JSP 包含控制和業務邏輯	多層架構，分為 Controller、Model 和 View
控制邏輯位置	位於 JSP	位於 Controller（通常是 Servlet）
耦合性	高，業務邏輯和界面混雜在一起	低，每層有明確職責
適用場景	小型應用，簡單頁面	中大型應用，需要高可維護性和複用性
開發難度	簡單，適合初學者	較難，適合有經驗的開發者

C語言網頁的優缺點

優點

1. 高效性能

- **低層級語言特性**：C 語言非常接近底層硬體，具有高效的內存管理能力，能實現比其他高層語言（如 PHP、Python 等）更高效的程式執行速度。
- 適合開發需要處理大量資料或高並發的應用，如高性能 Web 伺服器（Nginx 就是部分用 C 寫的）。

2. 靈活控制

- **直接控制硬體資源**：開發者可以直接操作網路套接字（sockets）或文件系統，適合設計需要精細控制的網頁伺服器。
- 可以編寫高效的自定義網路協議或實現針對特定硬體的優化。

3. 可靠性和穩定性

- **成熟穩定的語言**：C 語言誕生於 1970 年代，擁有豐富的標準庫和生態系統，許多底層網頁技術（如 HTTP 伺服器）都以 C 語言為基礎開發。
- 適合構建需要長時間穩定運行的核心服務。

4. 移植性強

- **跨平台能力**：C 語言編寫的程式可以在不同的操作系統（Linux、Windows、macOS 等）上進行編譯和運行，這對於伺服器端應用特別重要。

5. 學習底層網路基礎

- **學習機會**：用 C 寫網頁相關應用需要理解網路基礎（如 HTTP 協議、TCP/IP 協議），對學習者深入掌握網頁運行原理非常有幫助。

缺點

1. 開發效率低

- **缺少高層次工具**：C 語言沒有內建的網頁處理功能（如模板引擎、路由系統），需要開發者從零開始實現許多功能，導致開發效率低。
- 現代網頁開發通常使用高層語言（如 Python、Node.js、PHP）搭配框架，這些語言提供了豐富的開發工具和簡化的功能。

2. 安全性挑戰

- **內存管理問題**：C 語言需要手動管理內存，容易出現內存洩漏、緩衝區溢出等問題，這對網頁開發來說是一個安全隱患。
- 如果處理用戶輸入不當，可能會導致安全漏洞（如 SQL 注入、跨站腳本攻擊等）。

3. 學習門檻高

- **需要更深入的知識**：用 C 開發網頁需要理解網路協議、伺服器運作原理，對初學者來說學習曲線較陡。
- 與之相比，Python 或 PHP 提供了更高層的抽象，適合快速入門。

4. 缺乏現代化生態

- C 語言缺少現代網頁開發所需的生態工具（如 ORM、模板引擎、框架等），開發者需要手動實現，開發過程繁瑣且容易出錯。

5. 社群支持較少

- 用 C 語言進行網頁開發的開發者相對較少，出現問題時可能很難找到現成的解決方案或社群支持。

`doGet()` 和 `doPost()`

- `doGet()`：用來處理 **HTTP GET 請求**。這是瀏覽器預設的請求方式，通常用於請求網頁或數據，並且這些請求是**只讀的**。

    - 通常用於**獲取**資源，例如顯示頁面、查詢數據等。
    - GET 請求會將所有參數以**URL 查詢字串**的方式附加在 URL 後面，數據暴露於瀏覽器地址欄中。
    - URL 長度有限制（一般為 2048 個字符左右，具體限制因瀏覽器不同而異）。
- `doPost()`：用來處理 **HTTP POST 請求**。POST 請求通常用於提交數據（例如表單提交、上傳文件等）。
    - 通常用於**提交**數據給伺服器，並且可以包含大量的數據。
    - POST 請求的數據會包含在請求體（Request Body）中，不會顯示在 URL 中，相對來說更加隱私。
    - 沒有 URL 長度的限制，適合處理大量數據。