技術筆記整理

Spring Boot 相關

1. Spring Boot 權限檢查機制

- **基本整合**
  - 使用 Spring Security 框架處理認證和授權
  - 實作 UserDetailsService 介面管理用戶資訊
  - 配置 SecurityFilterChain 定義安全規則
- **權限控制方式**
  - 使用 `@PreAuthorize` 進行方法級別的權限控制
  - 實作 `HandlerInterceptor` 進行請求攔截
  - 自定義 Filter 處理特殊認證邏輯
- **最佳實踐**
  - 採用 JWT 實現無狀態認證
  - 實作 RBAC (Role-Based Access Control) 角色權限模型
  - 使用快取機制優化權限檢查效能

需求規格理解

2. 規格理解與溝通策略

- **前置準備**
  - 仔細研讀需求文件，標記關鍵點
  - 繪製流程圖或使用案例圖輔助理解
  - 列出所有假設和限制條件
- **問題處理流程**
  - 將大需求拆分為小任務
  - 對每個任務進行可行性評估
  - 製作原型或概念驗證（POC）
- **團隊協作**
  - 定期與產品經理和利害關係人同步進度

- 建立明確的溝通管道和回饋機制
- 記錄所有決策和變更

JSP 技術

3. JSP 作用域變數詳解

- **page 作用域**
  - 生命週期：單一 JSP 頁面
  - 適用場景：暫存頁面計算結果
  - 存取方式：`pageContext.setAttribute()`
- **request 作用域**
  - 生命週期：單次請求週期
  - 適用場景：跨頁面傳遞參數
  - 存取方式：`request.setAttribute()`
- **session 作用域**
  - 生命週期：使用者會話期間
  - 適用場景：購物車、登入狀態
  - 存取方式：`session.setAttribute()`
- **application 作用域**
  - 生命週期：應用程式運行期間
  - 適用場景：系統配置、共享資源
  - 存取方式：`application.setAttribute()`

4. JSP 學習路線圖

1. **基礎知識**
   - JSP 語法和指令
   - Servlet 生命週期
   - 請求處理機制
2. **進階技能**
   - EL 表達式運用
   - JSTL 標籤庫使用
   - 自定義標籤開發
3. **整合應用**
   - 與 Spring Boot 整合

- MVC 架構實作
- 安全性考量

物件導向程式設計

5. 物件導向三大特性詳解

- **封裝 (Encapsulation)**
  - 原理：隱藏實作細節，只開放必要介面
  - 優點：提高安全性，降低耦合度
  - 實作：使用存取修飾符和 getter/setter
- **繼承 (Inheritance)**
  - 原理：子類繼承父類特性，支持擴展
  - 優點：代碼重用，建立類階層
  - 最佳實踐：優先使用組合而非繼承
- **多型 (Polymorphism)**
  - 原理：同一介面，不同實作
  - 優點：提高程式靈活性和擴展性
  - 應用：介面實作、方法重載

Java 集合框架

6. Collection Framework 深入解析

- **核心介面**
  - List：有序集合，允許重複元素
  - Set：不重複元素集合
  - Map：鍵值對映射
  - Queue：佇列操作
- **常用實作類**
  - ArrayList：動態陣列
  - HashMap：哈希表實作
  - TreeSet：排序集合
- **使用時機**
  - ArrayList：頻繁查詢，較少修改
  - LinkedList：頻繁增刪操作

- HashMap：快速鍵值查詢

7. List 特性與應用

- **有序性特點**
  - 維護元素插入順序
  - 支援索引訪問
  - 允許重複元素
- **常用操作**
  - 新增：`add()`, `addAll()`
  - 刪除：`remove()`, `clear()`
  - 查詢：`get()`, `indexOf()`
- **效能考量**
  - ArrayList：隨機訪問快
  - LinkedList：插入刪除快

8. TreeMap 特性與應用

- **核心特點**
  - 基於紅黑樹實作
  - 自動排序鍵
  - 不允許空鍵
- **適用場景**
  - 需要有序性的鍵值對
  - 範圍查詢操作
  - 需要維護數據順序
- **效能特性**
  - 插入/刪除：O(log n)
  - 查詢：O(log n)
  - 相比 HashMap 較慢

物件導向設計

9. 存取修飾符使用準則

- **private**
  - 範圍：僅類內部可見

    - 使用：封裝內部實作
    - 場景：類的屬性、輔助方法
- **protected**
    - 範圍：子類和同包可見
    - 使用：允許子類訪問/重寫
    - 場景：框架設計、抽象類
- **public**
    - 範圍：所有類可見
    - 使用：開放介面
    - 場景：API 設計、公共方法
- **default (package-private)**
    - 範圍：同包內可見
    - 使用：包內共享
    - 場景：內部實作類

分頁實作

10. 自定義分頁實作指南

- **後端實作**

```
public class PageResult<T> {
    private List<T> data;
    private int totalPages;
    private long totalElements;
    private int currentPage;
    private int pageSize;

    // constructors, getters, setters
}
```

- **SQL 查詢優化**

```
SELECT * FROM table_name
LIMIT :pageSize OFFSET :offset
```

- **前端處理**

```
const fetchPage = async (page, size) => {
  const response = await fetch(`/api/data?page=${page}&size=${size}`);
  return response.json();
};
```

MVC 架構

11. MVC 架構深入解析

- **Model 層**
  - 職責：業務邏輯和數據處理
  - 組件：Entity、Repository、Service
  - 設計原則：單一職責、高內聚低耦合
- **View 層**
  - 職責：使用者介面展示
  - 技術：JSP、Thymeleaf、前端框架
  - 設計原則：關注點分離
- **Controller 層**
  - 職責：請求處理和路由
  - 設計：RESTful API、請求參數驗證
  - 最佳實踐：輕量控制器

12. Servlet vs JSP

- **Servlet 特性**
  - Java 類，繼承 HttpServlet
  - 處理請求邏輯
  - 生命週期管理
- **JSP 特性**
  - 動態網頁技術
  - 簡化頁面開發
  - 內建隱含對象
- **使用建議**
  - Servlet：複雜業務邏輯
  - JSP：簡單頁面渲染
  - 結合：MVC 模式

程式設計實踐

13. equals 方法最佳實踐

```java
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;

    MyClass other = (MyClass) obj;
    return Objects.equals(field1, other.field1) &&
           Objects.equals(field2, other.field2);
}
```

14. 日誌配置最佳實踐

- **Spring Boot 配置**

```yaml
logging:
  level:
    root: INFO
    com.example: DEBUG
  file:
    name: logs/application.log
  pattern:
    console: "%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%r
```

- **Logback 配置**

```xml
<configuration>
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppende
    <!-- 配置詳情 -->
  </appender>
</configuration>
```

SQL 進階

15. JOIN 操作詳解

- **INNER JOIN**

```sql
SELECT a.*, b.*
FROM table_a a
INNER JOIN table_b b ON a.id = b.a_id
```

- **LEFT JOIN**

```sql
SELECT a.*, b.*
FROM table_a a
LEFT JOIN table_b b ON a.id = b.a_id
```

- **性能考量**
  - 使用索引優化

- 避免多表連接
- 選擇合適的 JOIN 類型

16. 進階 SQL 技巧

- **子查詢最佳實踐**

```
SELECT *
FROM orders
WHERE customer_id IN (
    SELECT id
    FROM customers
    WHERE country = 'TW'
)
```

- **CTE 使用**

```
WITH RECURSIVE cte AS (
    SELECT id, parent_id, name, 1 as level
    FROM categories
    WHERE parent_id IS NULL
    UNION ALL
    SELECT c.id, c.parent_id, c.name, cte.level + 1
    FROM categories c
    JOIN cte ON c.parent_id = cte.id
)
SELECT * FROM cte
```

前端技術

17. HTML vs JSP 比較

- **HTML**
  - 靜態內容
  - 客戶端渲染
  - 檔案較小
- **JSP**
  - 動態內容
  - 伺服器端渲染
  - 支援 Java 程式碼

18. EL & JSTL 使用指南

- **EL 表達式**

```
${user.name}
${requestScope.message}
```

```
${sessionScope.cart.items}
```

- **JSTL 標籤**

```
<c:if test="${not empty users}">
    <c:forEach items="${users}" var="user">
        ${user.name}
    </c:forEach>
</c:if>
```

19. JSTL vs Scriptlet

- **JSTL 優點**
  - 可讀性高
  - 維護性好
  - 支援 i18n
- **Scriptlet 缺點**
  - 程式碼混雜
  - 難以維護
  - 違反 MVC

20. Servlet vs Spring @RequestMapping

- **Servlet 優勢**
  - 輕量級
  - 直接控制
  - 學習基礎
- **@RequestMapping 優勢**
  - 註解驅動
  - 整合度高
  - 功能豐富

21. 資源關閉最佳實踐

```
try (Connection conn = dataSource.getConnection();
     PreparedStatement stmt = conn.prepareStatement(sql)) {
    // 使用連接
} catch (SQLException e) {
    logger.error("Database error", e);
}
```

22. AJAX vs Form Submit

- **AJAX 特點**
  - 非同步處理
  - 局部更新
  - 更好的用戶體驗
- **Form Submit 特點**
  - 同步處理
  - 整頁刷新
  - 簡單直接

23. 非同步處理模式

```
// Promise 方式
async function fetchData() {
    try {
        const response = await fetch('/api/data');
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('Error:', error);
    }
}
```

24. 編碼風格規範

- **命名規範**
  - 類名：PascalCase
  - 方法名：camelCase
  - 常量：UPPER_SNAKE_CASE
- **檔案組織**
  - 按功能分包
  - 相關類放在一起
  - 適當的註解

25. DOM 操作最佳實踐

```
// 查詢元素
const element = document.querySelector('.class-name');

// 事件委派
document.addEventListener('click', (e) => {
    if (e.target.matches('.button-class')) {
        handleClick(e);
    }
});
```

```javascript
// 批量更新
const fragment = document.createDocumentFragment();
items.forEach(item => {
    const div = document.createElement('div');
    div.textContent = item;
    fragment.appendChild(div);
});
container.appendChild(fragment);
```

26. 交易控制實作

```java
@Transactional
public void transferMoney(String fromAccount, String toAccount, BigDecimal amount
    try {
        accountService.withdraw(fromAccount, amount);
        accountService.deposit(toAccount, amount);
    } catch (Exception e) {
        throw new TransactionException("Transfer failed", e);
    }
}
```