React 書籍清單實作筆記

目錄

1. 基本概念
2. 核心功能實作
3. UI 實作
4. 最佳實踐
5. 常見問題與解決方案
6. 擴展建議
7. 補充說明
8. 實作範例
9. 進階功能
10. 測試範例

基本概念

1. 資料結構

```
// 書籍資料範例
{
  isbn: "9781593279509",
  title: "Eloquent JavaScript",
  author: "Marijn Haverbeke"
}
```

2. 狀態管理

```
// 定義書籤的狀態(陣列)，也可以是Set, Map, 物件(索引作為屬性)
// 狀態中記錄isbn(代表有加入到我的書籤)
const [bookmarks, setBookmarks] = useState(['9781593279509', '9781593277574'])
```

核心功能實作

1. 書籤切換功能

```
// 處理書籤布林值切換(toggle)
const onToggleBookmark = (bookIsbn) => {
  // 判斷目前這個bookIsbn是否有在狀態中
  if (bookmarks.includes(bookIsbn)) {
    // 如果有 ===> 移出陣列
    const nextBookmarks = bookmarks.filter((v) => v !== bookIsbn)
    setBookmarks(nextBookmarks)
```

```
    } else {
      // 否則 ===> 加入陣列
      const nextBookmarks = [...bookmarks, bookIsbn]
      setBookmarks(nextBookmarks)
    }
  }
```

UI 實作

1. 表格結構

```
<table>
  <thead>
    <tr>
      <th>ISBN</th>
      <th>書名</th>
      <th>作者</th>
      <th>加入書籤</th>
    </tr>
  </thead>
  <tbody>
    {data.map((book) => {
      return (
        <tr key={book.isbn}>
          <td>{book.isbn}</td>
          <td>{book.title}</td>
          <td>{book.author}</td>
          <td>
            <Image
              onClick={() => {
                onToggleBookmark(book.isbn)
              }}
              src={
                // 改為使用狀態中是否有isbn(代表有加入到我的書籤)
                bookmarks.includes(book.isbn) ? bookmarkIconFill : bookmarkIcon
              }
              alt=""
            />
          </td>
        </tr>
      )
    })}
  </tbody>
</table>
```

最佳實踐

1. 狀態管理

- 使用陣列存儲書籤狀態
- 使用不可變更新模式
- 保持狀態的純淨性

2. 使用者體驗

- 提供即時反饋
- 使用清晰的圖示
- 保持介面簡潔

3. 程式碼組織

- 將相關功能分組
- 使用有意義的函數名稱
- 保持程式碼簡潔

常見問題與解決方案

1. 狀態更新

- 問題：直接修改狀態
- 解決：使用不可變更新模式

2. 效能優化

- 問題：大量資料渲染效能
- 解決：使用虛擬列表或分頁

3. 使用者體驗

- 問題：操作反饋不明確
- 解決：加入動畫效果和提示

擴展建議

1. 加入本地儲存

```
useEffect(() => {
  localStorage.setItem('bookmarks', JSON.stringify(bookmarks))
}, [bookmarks])

useEffect(() => {
  const savedBookmarks = localStorage.getItem('bookmarks')
  if (savedBookmarks) {
    setBookmarks(JSON.parse(savedBookmarks))
  }
}, [])
```

2. 加入動畫效果

```
.bookmark-icon {
  transition: all 0.3s ease;
}

.bookmark-icon:hover {
  transform: scale(1.1);
}
```

3. 效能優化

```
// 使用 useMemo 緩存計算結果
const bookmarkedBooks = useMemo(
  () => data.filter((book) => bookmarks.includes(book.isbn)),
  [data, bookmarks]
)
```

補充說明

1. 狀態更新注意事項

- 使用 `filter` 方法時，確保過濾條件正確
- 使用 `setBookmarks` 時，確保傳入的是新的陣列
- 避免直接修改狀態

2. 效能優化建議

- 使用 `useMemo` 緩存計算結果
- 使用 `useCallback` 緩存函數
- 使用 `React.memo` 優化渲染

3. 使用者體驗優化

- 加入載入狀態
- 加入錯誤處理
- 加入成功提示

4. 程式碼組織建議

- 將相關功能分組
- 使用自定義 Hook
- 使用 Context API

5. 測試建議

- 單元測試
- 整合測試
- 端到端測試

實作範例

1. 書籍列表渲染

```
{
  data.map((book) => {
    return (
      <tr key={book.isbn}>
        <td>{book.isbn}</td>
        <td>{book.title}</td>
        <td>{book.author}</td>
        <td>
          <Image
            onClick={() => onToggleBookmark(book.isbn)}
            src={
              bookmarks.includes(book.isbn) ? bookmarkIconFill : bookmarkIcon
            }
            alt=""
          />
        </td>
      </tr>
    )
  })
}
```

2. 書籤切換功能

```
const onToggleBookmark = (bookIsbn) => {
  if (bookmarks.includes(bookIsbn)) {
    const nextBookmarks = bookmarks.filter((v) => v !== bookIsbn)
    setBookmarks(nextBookmarks)
  } else {
    const nextBookmarks = [...bookmarks, bookIsbn]
    setBookmarks(nextBookmarks)
  }
}
```

進階功能

1. 書籤持久化

```
const useBookmarkPersist = () => {
  const [bookmarks, setBookmarks] = useState(() => {
    const savedBookmarks = localStorage.getItem('bookmarks')
    return savedBookmarks ? JSON.parse(savedBookmarks) : []
  })

  useEffect(() => {
    localStorage.setItem('bookmarks', JSON.stringify(bookmarks))
  }, [bookmarks])

  return [bookmarks, setBookmarks]
}
```

2. 書籤動畫

```
import { motion } from 'framer-motion'

const BookmarkIcon = ({ isBookmarked, onClick }) => {
  return (
    <motion.div
      whileHover={{ scale: 1.1 }}
      whileTap={{ scale: 0.9 }}
      onClick={onClick}
    >
      <Image src={isBookmarked ? bookmarkIconFill : bookmarkIcon} alt="" />
    </motion.div>
  )
}
```

3. 書籤效能優化

```
// 使用 useMemo 優化計算
const BookmarkedBooks = React.memo(({ books, bookmarks }) => {
  const bookmarkedBooks = useMemo(
    () => books.filter((book) => bookmarks.includes(book.isbn)),
    [books, bookmarks]
  )

  return (
    <div>
      {bookmarkedBooks.map((book) => (
        <div key={book.isbn}>{book.title}</div>
      ))}
    </div>
  )
})
```

4. 書籤錯誤處理

```
const useBookmarkError = () => {
  const [error, setError] = useState(null)

  const handleError = (error) => {
    setError(error.message)
    setTimeout(() => setError(null), 3000)
  }

  return { error, handleError }
}

const BookList = () => {
  const { error, handleError } = useBookmarkError()

  const onToggleBookmark = (bookIsbn) => {
    try {
      // 書籤切換邏輯
    } catch (err) {
      handleError(err)
    }
  }

  return (
    <div>
      {error && <div className="error">{error}</div>}
      {/* 其他書籍列表內容 */}
    </div>
  )
}
```

測試範例

1. 單元測試

```
import { render, fireEvent } from '@testing-library/react'

test('切換書籤狀態', () => {
  const { getByAltText } = render(<BookList />)
  const bookmarkButton = getByAltText('')

  fireEvent.click(bookmarkButton)
  expect(bookmarkButton.src).toContain('bookmark-fill.svg')

  fireEvent.click(bookmarkButton)
  expect(bookmarkButton.src).toContain('bookmark.svg')
})
```

2. 整合測試

```
test('完整書籤流程', () => {
  const { getByAltText, getByText } = render(<BookList />)

  // 加入書籤
  fireEvent.click(getByAltText(''))
  expect(getByText('已加入書籤')).toBeInTheDocument()

  // 移除書籤
  fireEvent.click(getByAltText(''))
  expect(getByText('已移除書籤')).toBeInTheDocument()
})
```

3. 效能測試

```
test('大量書籍渲染效能', () => {
  const books = Array(1000)
    .fill()
    .map((_, i) => ({
      isbn: `978159327${i}`,
      title: `Book ${i}`,
      author: `Author ${i}`,
    }))

  const { getByAltText } = render(<BookList books={books} />)

  const startTime = performance.now()
  fireEvent.click(getByAltText(''))
  const endTime = performance.now()

  expect(endTime - startTime).toBeLessThan(100)
})
```

注意事項

1. **狀態管理**
    - 使用不可變更新模式
    - 避免直接修改狀態
    - 使用適當的狀態更新方法
2. **效能優化**
    - 使用 useMemo 緩存計算結果
    - 使用 useCallback 緩存函數
    - 使用 React.memo 優化渲染

3. **使用者體驗**

   - 提供適當的錯誤處理
   - 加入載入狀態
   - 提供清晰的提示訊息

4. **程式碼組織**

   - 將相關功能分組
   - 使用自定義 Hook
   - 保持程式碼簡潔

5. **測試**

   - 撰寫單元測試
   - 撰寫整合測試
   - 進行效能測試

實作方式比較

1. 陣列方式 (book-list-array)

```
// 狀態管理
const [bookmarks, setBookmarks] = useState(['9781593279509', '9781593277574'])

// 書籤切換
const onToggleBookmark = (bookIsbn) => {
  if (bookmarks.includes(bookIsbn)) {
    const nextBookmarks = bookmarks.filter((v) => v !== bookIsbn)
    setBookmarks(nextBookmarks)
  } else {
    const nextBookmarks = [...bookmarks, bookIsbn]
    setBookmarks(nextBookmarks)
  }
}

// UI 渲染
src={bookmarks.includes(book.isbn) ? bookmarkIconFill : bookmarkIcon}
```

2. 物件方式 (book-list)

```
// 狀態管理
const initState = data.map((v) => {
  return { ...v, bookmark: false }
})
const [books, setBooks] = useState(initState)

// 書籤切換
```

```
const onToggleBookmark = (bookIsbn) => {
  const nextBooks = books.map((v, i) => {
    if (v.isbn === bookIsbn) {
      return { ...v, bookmark: !v.bookmark }
    } else {
      return v
    }
  })
  setBooks(nextBooks)
}

// UI 渲染
src={book.bookmark ? bookmarkIconFill : bookmarkIcon}
```

3. Set 方式 (book-list-set)

```
// 狀態管理
const [bookmarks, setBookmarks] = useState(new Set())

// 書籤切換
const onToggleBookmark = (bookIsbn) => {
  const updatedBookmarks = new Set(bookmarks)
  if (updatedBookmarks.has(bookIsbn)) {
    updatedBookmarks.delete(bookIsbn)
  } else {
    updatedBookmarks.add(bookIsbn)
  }
  setBookmarks(updatedBookmarks)
}

// UI 渲染
src={bookmarks.has(book.isbn) ? bookmarkIconFill : bookmarkIcon}
```

4. 三種實作方式的比較

優點

1. **陣列方式**
   - 狀態較簡單，容易管理
   - 記憶體使用較少，只存儲 ISBN
   - 程式碼較簡單，邏輯清晰
   - 適合簡單的書籤功能
   - 容易實現書籤列表的篩選
2. **物件方式**
   - 書籤狀態直接存在物件中，查詢效率高

- 資料一致性更容易維護
- 適合需要更多書籍相關屬性的場景
- 更容易實現書籍的排序和篩選
- 功能更完整

3. **Set 方式**

- 查詢效率高，O(1) 時間複雜度
- 自動處理重複值
- 記憶體使用適中
- 程式碼簡潔
- 符合不可變性原則

缺點

1. **陣列方式**

- 每次檢查書籤狀態需要遍歷陣列
- 需要確保 ISBN 的唯一性
- 需要手動維護書籤狀態和書籍資料的同步
- 功能擴展性較差

2. **物件方式**

- 狀態較複雜，需要維護更多資料
- 記憶體使用較多，每個書籍物件都包含書籤狀態
- 程式碼較複雜
- 初始設定較麻煩

3. **Set 方式**

- 需要手動創建 Set 的複本
- 不支援直接序列化（需要轉換為陣列）
- 某些瀏覽器可能不支援 Set 的所有方法
- 需要額外的記憶體來存儲 Set 的結構

使用場景

1. **陣列方式適合**

- 簡單的書籤功能
- 不需要頻繁更新書籍資料
- 記憶體資源有限
- 需要快速實現基本功能

2. **物件方式適合**

- 複雜的書籍管理功能
- 需要頻繁更新書籍資料
- 需要更好的效能

- 需要更多擴展功能

3. **Set 方式適合**

   - 需要高效的查詢操作
   - 需要自動處理重複值
   - 需要符合不可變性原則
   - 需要簡潔的程式碼

效能考量

1. **陣列方式**

   - 查詢效率：O(n)，需要遍歷陣列
   - 更新效率：O(n)，需要複製陣列
   - 記憶體使用：較少，只存儲 ISBN

2. **物件方式**

   - 查詢效率：O(1)，直接訪問物件屬性
   - 更新效率：O(n)，需要遍歷並複製物件
   - 記憶體使用：較多，每個物件都包含書籤狀態

3. **Set 方式**

   - 查詢效率：O(1)，使用 has 方法
   - 更新效率：O(1)，使用 add/delete 方法
   - 記憶體使用：適中，需要額外存儲 Set 結構

選擇建議

1. 如果只需要簡單的書籤功能，選擇**陣列方式**
2. 如果需要複雜的書籍管理功能，選擇**物件方式**
3. 如果需要高效的查詢和更新操作，選擇**Set 方式**
4. 如果重視記憶體使用，選擇**陣列方式**
5. 如果需要快速開發，選擇**陣列方式**或**Set 方式**
6. 如果需要長期維護和擴展，選擇**物件方式**或**Set 方式**