

# 【Kotlin】sealed class

在Kotlin 中，`sealed class`（密封類）是一種特殊的類，它用來表示受限的類層次結構。`sealed class` 及其子類別的定義必須在同一個檔案中，從而確保了編譯時的類型檢查。這種類別通常用於表達某種有限數量的可能狀態或類型。以下是 `sealed class` 的一些主要特點和用途：

## 特點

1. **受限層次結構**：`sealed class` 的所有直接子類別必須在同一個檔案中定義，這樣可以確保在編譯時知道所有可能的子類別。
2. **抽象類別**：`sealed class` 本身是抽象的，不能直接實例化，只能透過其子類別來實例化。
3. **強制列舉**：在使用 `when` 表達式時，如果覆寫了所有的子類別分支，就不需要再加 `else` 分支，這樣可以在編譯時確保**列舉**所有情況。

## 使用場景

- **需要有限的類繼承**：您有一組預先定義的、有限的子類來擴展一個類，所有這些子類在編譯時都是已知的。
- **需要類型安全設計**：安全性和模式匹配在您的專案中至關重要。特別是對於狀態管理或處理複雜的條件邏輯。
- **使用封閉式 API**：您需要為程式庫提供強大且可維護的公共 API，以確保第三方用戶端按預期使用 API。

## UI 應用程式中的狀態管理

可以使用`sealed`來表示應用程式中的不同 UI 狀態。這種方法允許結構化且安全地處理 UI 變更。此範例示範如何管理各種 UI 狀態：

```
sealed class UIState {
    data object Loading : UIState()
    data class Success(val data: String) : UIState()
    data class Error(val exception: Exception) : UIState()
}

fun updateUI(state: UIState) {
    when (state) {
        is UIState.Loading -> showLoadingIndicator()
        is UIState.Success -> showData(state.data)
        is UIState.Error -> showError(state.exception)
    }
}
```

## 付款方式處理

實際業務應用中，高效處理各種支付方式是常見的需求。您可以使用具有 `when` 表達式的密封類別來實現此類業務邏輯。透過將不同的支付方式表示為密封類的子類，它建立了一個清晰且可管理的交易處理結構：

```
sealed class Payment {
    data class CreditCard(val number: String, val expiryDate: String) : Payment()
    data class PayPal(val email: String) : Payment()
    data object Cash : Payment()
}

fun processPayment(payment: Payment) {
    when (payment) {
        is Payment.CreditCard -> processCreditCardPayment(payment.number, payment.expiryDate)
        is Payment.PayPal -> processPayPalPayment(payment.email)
        Payment.Cash -> processCashPayment()
    }
}
```

`Payment` 是一個密封類，代表電子商務系統中的不同支付方式：`CreditCard`、`PayPal` 和 `Cash`。每個子類別都可以有其特定的屬性，例如 `number` 和 `expiryDate` for `CreditCard` 和 `email` for `PayPal`。

此 `processPayment()` 函數示範如何處理不同的付款方式。這種方法確保考慮所有可能的付款類型，並且系統對於將來添加新的支付方式保持靈活性。

## API 請求-回應處

您可以使用 sealed class 和密 sealed interface 來實作處理 API 請求和回應的使用者驗證系統。  
使用者認證系統具有登入和登出功能。

介面 ApiRequest 定義了特定的請求類型：LoginRequest 登入和 LogoutRequest 登出操作。sealed class ApiResponse 封裝了不同的回應場景：UserSuccess 使用使用者資料、UserNotFound 針對缺席的使用者以及 Error 針對任何故障。  
此 handleRequest 函數使用表達式以類型安全的方式處理這些請求 when，同時 getUserById 模擬使用者檢索：

```
// Import necessary modules
import io.ktor.server.application.*
import io.ktor.server.resources.*

import kotlinx.serialization.*

////////////////////////////////////
// Define the sealed interface for API requests using Ktor resources
@Resource("api")
sealed interface ApiRequest

@Serializable
@Resource("login")
data class LoginRequest(val username: String, val password: String) : ApiRequest

@Serializable
@Resource("logout")
object LogoutRequest : ApiRequest
////////////////////////////////////

// Define the ApiResponse sealed class with detailed response types
sealed class ApiResponse {
    data class UserSuccess(val user: UserData) : ApiResponse()
    data object UserNotFound : ApiResponse()
    data class Error(val message: String) : ApiResponse()
}
////////////////////////////////////

// User data class to be used in the success response
data class UserData(val userId: String, val name: String, val email: String)
////////////////////////////////////

// Function to validate user credentials (for demonstration purposes)
fun isValidUser(username: String, password: String): Boolean {
    // Some validation logic (this is just a placeholder)
    return username == "validUser" && password == "validPass"
}
////////////////////////////////////

// Function to handle API requests with detailed responses
fun handleRequest(request: ApiRequest): ApiResponse {
    return when (request) {
        is LoginRequest -> {
            // 合法使用者
            if (isValidUser(request.username, request.password)) {
                ApiResponse.UserSuccess(UserData("userId", "userName", "userEmail"))
            } else {
                ApiResponse.Error("Invalid username or password")
            }
        }
        is LogoutRequest -> {
            // Assuming logout operation always succeeds for this example
            ApiResponse.UserSuccess(UserData("userId", "userName", "userEmail")) // For demonstration
        }
    }
}

// Function to simulate a getUserById call
fun getUserById(userId: String): ApiResponse {
    return if (userId == "validUserId") {
        ApiResponse.UserSuccess(UserData("validUserId", "John Doe", "john@example.com"))
    }
```

```

    } else {
        ApiResponse.UserNotFound
    }
    // Error handling would also result in an Error response.
}

// Main function to demonstrate the usage
fun main() {
    // 使用帳密登入
    val loginResponse = handleRequest(LoginRequest("user", "pass"))
    println(loginResponse)
    // 登出
    val logoutResponse = handleRequest(LogoutRequest)
    println(logoutResponse)
    // 檢索使用者 (存在，回應成功)
    val userResponse = getUserById("validUserId")
    println(userResponse)
    // 檢索使用者 (不存在，回應失敗)
    val userNotFoundResponse = getUserById("invalidId")
    println(userNotFoundResponse)
}

```

## Java 中的 sealed 關鍵字

Java 17 引入了 sealed 類型，允許你限制哪些類可以繼承這個類。

```

public sealed class Shape permits Circle, Rectangle, NotAShape {
}

public final class Circle extends Shape {
    private final double radius;
    public Circle(double radius) {
        this.radius = radius;
    }
    public double getRadius() {
        return radius;
    }
}

public final class Rectangle extends Shape {
    private final double width;
    private final double height;
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
    public double getWidth() {
        return width;
    }
    public double getHeight() {
        return height;
    }
}

public final class NotAShape extends Shape {
}

```

## C# 中的 sealed 關鍵字

當你將一個類別標記為 sealed 時，這意味著這個類別不能被其他類別繼承。這在需要明確表達某個類別不能被擴展時非常有用，例如當你想要保證某個類別的行為不被改變時。

```

public sealed class MyClass
{

```

```
public void Display()
{
    Console.WriteLine("Hello from MyClass");
}
}

// 這將導致編譯錯誤，因為 MyClass 是 sealed 的
// public class DerivedClass : MyClass
// {
// }

public class Program
{
    public static void Main()
    {
        MyClass myClass = new MyClass();
        myClass.Display();
    }
}
```

---

🕒修訂版本 #12

★由 treeman 建立於 10 🕒C🕒🕒 2024 16:44:17

✍由 treeman 更新於 31 🕒C🕒🕒 2024 10:36:13