**ApplicationContext**

1. 在 Spring 中，「上下文」（Context）是指 **ApplicationContext**
2. 用途:
   * **管理 Bean 的生命週期**

負責創建、初始化、配置和銷毀 Bean

* + **提供依賴注入**

幫助我們將應用中的組件連接起來，實現鬆耦合設計

* + **資源管理**

能夠方便地管理文件、數據庫連接等外部資源

* + **事件發布和監聽**

支持事件驅動的編程模型，可以發布和監聽應用中的事件

* + **國際化支持**

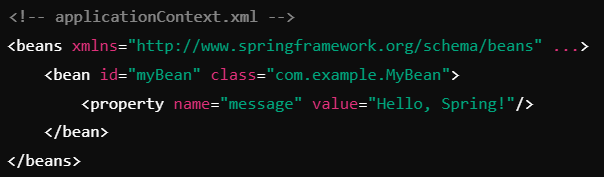
提供對國際化資源（如語言文件）的支持

1. 核心interface, Spring提供了幾個具體實作:
   * **ClassPathXmlApplicationContext：** 從類路徑加載 XML 配置。
   * **FileSystemXmlApplicationContext：** 從文件系統加載 XML 配置。
   * **AnnotationConfigApplicationContext：** 基於 Java 配置的上下文。
   * **WebApplicationContext：** 專門用於 Web 應用的上下文。

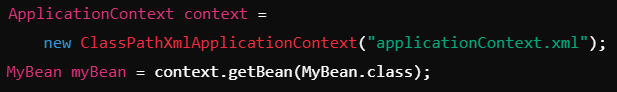
* 是ApplicationContext 的子接口，專門用於 Web 應用。
* 由Spring MVC 的DispatcherServlet或

ContextLoaderListener 創建和管理。

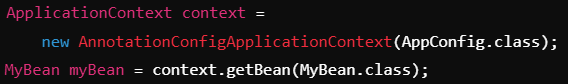
1. 設定上下文



* 透過加載xml配置文件



* 透過java配置



**ContextLoaderListener**

1. 用於初始化 Spring 應用上下文（ApplicationContext）的 Servlet 監聽器。通常與基於 Java EE 的 web 應用一起使用，用來在應用啟動時加載 Spring 配置並管理應用上下文的生命週期
2. 用途
   * **初始化Spring應用上下文**

處理WebApplicationContext是特化的ApplicationContext，

專注於處理Web的相關配置，常用於加載應用的核心配置

備註：DispatcherServlet 則會創建自己專屬的子上下文，

用於處理特定於 Web 層的配置

* + **管理應用上下文的生命週期**

Web 應用啟動時，初始化應用上下文，

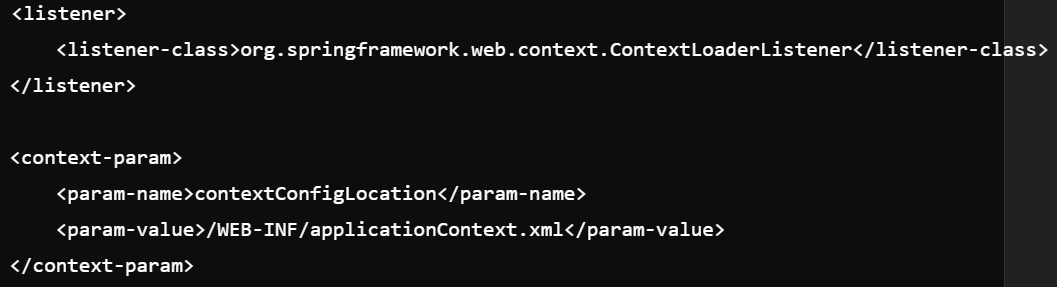
並在 Web 應用關閉時，自動銷毀應用上下文

* + **與 Servlet 容器整合**

將 Spring 的應用上下文與 Servlet 容器(Servlet Container)綁定，

提供全局的依賴注入和配置支持

1. Web.xml中的部分配置



**DispatcherServlet**

1. 是Spring MVC的核心套件(Spring MVC中扮演Servlet前端控制器)，

主要處理HTTP請求並將其轉發給相對應的處理器

1. 用途:
   * 接收和分發請求

所有的HTTP請求首先會被DispatcherServlet接收

根據請求的URL和參數來選擇合適的處理器（如控制器方法），

然後將請求交給相應的處理器

* + 請求處理流程的協調者
* 收到HTTP請求後，DispatcherServlet根據配置的HandlerMapping  
  找到對應的控制器方法(處理器)，然後調用該方法處理請求。
* 負責調用處理器、處理模型數據並選擇合適的視圖來生成響應
  + 處理請求與響應的過程

1. 解析請求： 接收HTTP請求
2. 查找處理器： 查詢一個或多個HandlerMapping

HandlerMapping主要將URL映射到一個處理器物件(Handler 就是Controller類別)

備註：

HandlerMapping同時把Handler和HandlerInterceptor封裝成一個HandlerExecutionChain

1. 調用處理器: 將找到 負責處理請求的處理器

交由可以處理的HandlerAdapter

1. 交付前的處理: 請求交給處理器處理前，

調用Interceptor的preHandler()(如果有)

對資料進行處理

(資料轉換、資料格式化及資料驗證)

(此Spring MVC透過參數解析ex:@RequestBody)，

接著呼叫處理器

1. 返回模型數據: 完成業務邏輯後，

處理器傳回一個ModelAndView或一個檢視物件，

ModelAndView包含模型 (model)

和檢視物件的邏輯名稱(view name)

接著調用Interceptor的postHandler() (如果有)

1. 選擇視圖: DispatcherServlet依據ViewResolver

將ModelAndView解析成View

1. 繪製: DispatcherServlet將Model資料放入View，

View呼叫render()再將Model資料繪製出頁面

1. 響應: 響應前

調用Interceptor的afterCompletion()(如果有)

接著DispatcherServlet將View的輸出內容寫入repsone

**啟動順序**

1. **掃描 web.xml：**
   * Tomcat 啟動後會掃描 web.xml 配置文件。
   * 根據配置，找到需要初始化的 Listener、Servlet 等。
2. **初始化 Listener（包括 ContextLoaderListener）：**
   * 初始化所有配置的 Listener。
   * 當執行到 ContextLoaderListener 時，開始創建和初始化根 Spring 容器。
3. **ContextLoaderListener 建立根容器（ApplicationContext）：**
   * 創建一個根容器（通常是 WebApplicationContext 的實例）。
   * 根容器主要管理全局的 Bean（例如 Service 層和 Repository 層的 Bean）。
4. **ContextLoaderListener 初始化 Spring 全局上下文：**
   * 透過 context-param 加載全局設定（如果有配置）。
   * 讀取 Spring 配置文件（如 applicationContext.xml 或 Java 配置類），加載並初始化全局的 Bean。
   * 初始化完成後，將根容器放入 ServletContext，以便後續使用。
5. **初始化 Servlet（包括 DispatcherServlet）：**
   * 初始化所有配置的 Servlet。
   * 當執行到 DispatcherServlet 時，開始創建和初始化子容器。
6. **DispatcherServlet 初始化時建立子容器（WebApplicationContext）：**
   * 創建一個子容器（WebApplicationContext）。
   * 子容器專門管理 Web 層的 Bean（例如 Controller 和 Spring MVC 的組件）。
7. **DispatcherServlet 加載 Spring MVC 配置：**
   * 透過 contextConfigLocation 指定的配置文件加載 Spring MVC 的組件：
     + HandlerMapping（處理請求的映射）。
     + HandlerAdapter（執行處理器）。
     + 其他 MVC 組件（如 ViewResolver）。
8. **初始化 Spring MVC 組件：**
   * 完成 MVC 組件的初始化。
   * 準備好處理 HTTP 請求。

<load-on-startup>

數值無配置或是負數 ->請求第一次到達時才初始化

數值配置正數(越小越優先)Tomcat啟動時初始化 Servlet (不影響Listener)

多個servelet依據數值高低決定初始順序