ChatGPT

RimWorld 本體與 DLC 的代碼與模組載入機制詳解

1. 本體與 DLC 的代碼儲存與載入方式

整合編譯 vs. 動態載入： RimWorld 的遊戲本體（Core）和官方擴充內容（DLC，如 Royalty、Ideology 等）**並沒有像模組那樣以獨立 DLL 動態載入**。它們的程式碼都直接編譯進主程式（Unity 的 Assembly-CSharp.dll）中 [1] [2]。例如，有社群資料指出 **「Core 和 Royalty 這兩個內建“模組”不包含任何自己的程式碼」**，表示官方 DLC 並沒有在自己的 Assemblies 資料夾放置額外 DLL [1]。相對而言，玩家製作的模組通常會透過 Assemblies 資料夾加入自訂的 DLL 檔案 [3]。因此，**RimWorld 本體與 DLC 的程式功能是與遊戲本體一起編譯發行的**，而非在執行時透過反射從外部 DLL 載入。主程式在執行時已經載入了包含所有功能的 C# 程式集，**Core/DLC 並不存在獨立載入的 C# 程式庫** [2]。這樣的設計意味著即使未啟用 DLC，其相關程式碼仍隨遊戲執行，只是不會主動生效。

內容資料儲存： 儘管程式碼在主程式中，**本體與 DLC 的資源和定義（Defs）以類似模組的形式存放在遊戲資料夾**中。RimWorld 安裝目錄下有一個 `Data` 資料夾，其中包含 Core 資料夾（遊戲本體內容）以及各 DLC 資料夾（例如 Royalty, Ideology, Biotech 等） [4]。每個資料夾結構類似一個模組，包含 About 資料夾（內有 About.xml 描述檔）、Defs 資料夾（XML 定義）、Textures/Sounds 等資源。也就是說，**官方 DLC 被組織成與 Core 類似的“內建模組”**存放路徑，只是它們的代碼部分早已整合在遊戲中了。Steam 版 DLC 安裝後會自動放入上述 Data 子資料夾下，並在遊戲中以獨立模組形式出現且可啟用/停用 [5]。相較之下，Steam 工作坊的模組則下載到 Steam 專用目錄下（例如 Steam\steamapps\workshop\content\294100）或者手動放入本地 Mods 資料夾中 [4]。但不論來源如何，**本體、DLC 和玩家模組的內容檔案格式是相同的**：使用 XML 定義各種 Def，使用指定的資料夾結構放置美術素材（PNG 圖片）、音效（OGG/WAV）等資源，並在有程式碼時提供 Assemblies 內的 DLL [6] [7]。唯一的差別在於**官方本體與 DLC 通常不額外附帶 DLL**（因為其功能已內建），而玩家模組若需自定行為則會包含 DLL。

2. DLC 功能的啟用與停用控制機制

擴充內容的識別： 由於 DLC 的代碼始終隨遊戲存在，RimWorld 必須透過**軟體開關**來控制某些功能是否啟用。這首先透過「模組列表」(ModsConfig) 將 DLC 視為特殊的模組來管理：如果玩家擁有某 DLC，該 DLC 會在遊戲的擴充內容清單中出現，玩家可以選擇啟用或停用它。停用某 DLC 將使其 Def 資源不被載入，從而**有效關閉相關機制**。在程式碼中，開發者使用了條件檢查和元數據來區分 DLC 狀態。例如，遊戲提供 `ModLister` / `ModsConfig` 之類的介面來檢查 DLC 是否已安裝/啟用。典型用法如 `ModLister.RoyaltyInstalled` 等，可判斷玩家是否擁有某個 DLC [8]。值得注意的是，`RoyaltyInstalled` 等函式**檢查的是玩家是否擁有 DLC（已購買安裝）**，而不一定是當前是否在此存檔中啟用 [8]。這區分了擁有與啟用兩個概念：擁有表示程式碼和資源可用，但只有啟用後相關 Def 才會載入並影響遊戲。

功能 gating 與條件內容載入： 為了在程式碼層面根據 DLC 狀態啟閉功能，RimWorld 採用了**條件判斷與資料驅動相結合**的方式。舉例來說，Royalty DLC 引入了帝國貴族頭銜、超能力等系統。這些系統的大部分程式碼已融入遊戲本體更新，但**只有在檢測到 DLC 啟用時**，相關內容才會生效（例如生成貴族任務、允許角色獲得頭銜等）。開發者可能在關鍵邏輯處以 `if (ModsConfig.IsActive("Ludeon.RimWorld.Royalty"))` 或類似條件包圍，來決定是否執行 DLC 特有的內容。另一方面，XML 資料則使用了特殊屬性和目錄結構來實現**有條件地載入**：例如 XML 節點支援 `MayRequire` 屬性，可標註此內容依賴某個 DLC 或模組才會載入 [9] [10]。如果沒有對應的 DLC，則該節點會被跳過，不會影響遊戲。RimWorld 1.1+ 的更新中引入了這個屬性方便模組作者兼容官方 DLC [11]。官方也在部分定義中使用了類似技巧，以**確保非 DLC 環境下不會誤載入不應該存在的元素**。

擴充的模組化啟用流程： 從玩家角度看，啟用某個 DLC 後，遊戲會要求重啟並重新加載內容。實際上，這相當於**告訴遊戲將該 DLC 資料夾當作一個模組加入載入清單**。遊戲在初始化時會掃描所有 Data 子資料夾，根據購買情況列出擴充包並讀取其 About.xml。每個 DLC 的 About.xml 定義了它的 `packageId`（例如 Ludeon.RimWorld.Ideology）和支援的版本等資訊 [12]。也可能包含排序需求，例如 Ideology 的關於檔就宣告了 `<forceLoadAfter><li>Ludeon.RimWorld.Royalty</li></forceLoadAfter>`，表明 Ideology **必須在 Royalty 之後載入** [12]。啟用時，遊戲依照這些規則將 DLC 插入正確的載入順序。**程式碼中的 DLC 系統通常還會在啟動階段註冊或初始化**：例如，若 Ideology 啟用，則遊戲會建立思維信仰(Ideo)管理器，Royalty 啟用則初始化超能量系統等。反之，若未啟用，則可能使用空的預設物件或根本不建立，來避免不必要的計算。這種架構確保**同一份遊戲執行檔**可以同時支援有無 DLC 的兩種狀態，而不需要分離程式版本。

防止未購買者存取付費內容： 由於 DLC 資料檔實際上存在於所有玩家的安裝中（只是未購買者不會啟用），RimWorld 也實施了一些機制防範**透過模組繞過鎖定**的行為。例如，**嘗試在未擁有 DLC 的遊戲中調用 DLC 專屬代碼或 Def 通常會導致錯誤**。社群指南指出某些代碼被標記為「僅限 Royalty」且在無 DLC 時載入將拋出錯誤 [13]。這意味著開發者對部分類別或方法做了限制：如果玩家沒買 DLC，模組強行使用那些功能會直接報錯，從技術上防止了未購買者借助第三方模組解鎖付費內容的可能。再加上前述 Def 層面的 `MayRequire` 條件屬性，讓模組作者更容易編寫**「有則用、無則略」**的內容，確保兼容性的同時遵守 DLC 的邏輯邊界。總的來說，**RimWorld 通過統一的模組框架與條件檢查，實現了對 DLC 功能點的開關控制**：購買且啟用時無縫融合於遊戲，未啟用時則對玩家幾乎是不可見的。

3. XML Def 與資源的載入流程

3.1 本體與 DLC Def 的組織與載入順序

分層次的 Def 資料： RimWorld 將遊戲內絕大多數可配置內容都定義為 Def（Definition）。本體(Core)與各 DLC 都有自己的 Def 檔案集，通常按類型分類存放於各自的 Defs 資料夾中（例如 ThingDefs, RecipeDefs 等子資料夾）。載入時，遊戲會**按照模組載入順序**來讀取這些定義。首先**最先載入的是 Core（本體）**的全部 XML 定義 [14]。接著，若啟用了官方 DLC，則按照既定順序載入它們的 Def。例如 Royalty 通常緊隨 Core，其後是 Ideology，再來是 Biotech（這順序由 DLC 的 `forceLoadAfter/Before` 規則或預設順序決定） [12]。**最後**才載入玩家訂閱或安裝的其他模組 Def。換言之，從載入優先權看：**Core 最先，DLC 次之，第三方模組最後**。

載入的實際執行： RimWorld 在初始化階段會將所有啟用的模組（含本體和 DLC）逐一處理。在處理每個模組時，它會掃描該模組 Def 資料夾下的所有 XML 檔，並將它們的內容合併到遊戲的資料庫中。在程式實現上，有趣的一點是——**RimWorld 背後可能將所有 Def 檔案視作一個大型 XML 來解析** [15]。這表示如果其中一個檔案有XML結構錯誤（如標籤缺失），可能會導致整批 Def 解析失敗。因此載入時非常嚴格，一個錯誤能阻止後續所有 Def 載入 [15]。載入順序上，則如前所述，核心先行，因此**Core 提供了許多基礎定義**（包括抽象Def，見下文）供後續內容引用。官方 DLC 的 Def 在 Core 之後載入，這允許 DLC **新增新的 Def**（如新的物種、物品、勢力等）以及**修改本體 Def**。值得注意的是，**官方 DLC 本身也可以對 Core 的定義進行調整**：這通常透過 Patch 操作（XML Patching）或繼承機制實現，而不是直接覆蓋。同理，當所有內建內容載入後，再載入玩家模組的 Def。**整體順序確保前面的內容可被後面的內容所參考或修改**，例如模組可以假定本體和 DLC 的必要 Def 已存在。

排序與衝突避免： 一般來說，RimWorld 保證 Core 永遠處於第一位且無法移動；DLC 則通常也鎖定在 Core 之後、一切模組之前（遊戲內預設自動如此，或透過 DLC 的 About.xml 中 `forceLoadAfter` 等設定實現 [12]）。因此，玩家**無法**（或通常不應該嘗試）將一個普通模組排在官方 Core/DLC 之前。如果發生那種情況，很多 Def Patch 可能找不到目標，遊戲亦會報錯。實際案例顯示，當舊版遊戲未識別新DLC的排序時，就出現過「Core 產生紅錯誤，不知道 DLC 載入順序」的問題 [16] [17]——後來透過在 DLC 定義中加入排序標記解決 [17]。因此如今**DLC 在載入順序上有受保護的優先級**：它們彼此之間按規定順序排列，且一定在 mods 之前載入。這種機制確保 DLC 的內容（例如 Royalty 定義的新頭銜頭銜、能力等）已經在模組開始作用前就準備好。同時也避免了玩家模組“搶跑”在前、導致Patch操作找不到Core節點的情況 [18]。

3.2 Mods 的 Def 疊加與覆蓋方式

後載入者覆蓋前載入者： RimWorld 模組系統遵循「後面的模組**優先**」的原則，即**載入在越後面的定義，權重越高**。具體來說，如果兩個不同來源（例如本體和某模組）定義了相同的東西，最終生效的是後載入的版本[19][20]。不過在實踐中，**“覆蓋”並非透過重名Def直接取代**，而更常用**XML Patch（XPath 修補）**或繼承來達成調整。RimWorld 不鼓勵兩個模組定義完全相同的 defName——這會在載入時產生重複鍵衝突的錯誤或未定義行為。因此，玩家模組想修改本體或 DLC 的數值，不是重新複製整個 Def，而是使用 `<Patch>` 檔案描述要變動的節點。這種設計大幅提高了模組之間的相容性，避免硬覆蓋彼此內容。

XML PatchOperation 機制： RimWorld 提供了一套強大的 **PatchOperation** 系列，在載入所有 XML Def 之後執行。根據官方Wiki說明：**「在將所有 XML Def 載入記憶體之後，會依模組列表順序執行 XPath 修補」**[21]。也就是說，遊戲先把各模組的原始 Def 都讀進來，接著按順序套用每個模組定義的 Patch。這些 PatchOperation 通常放在模組的 Patches 資料夾或 About.xml 中引用，它使用 XPath 語法定位某個已載入的 XML節點，然後進行增加、替換、移除等操作[22][23]。**執行順序**上，本體 Core 一般沒有 PatchOperation（Core 原則上直接給出最終值），但**官方 DLC 可以帶有PatchOperation修改Core**：例如把皇家頭銜加入基礎遊戲的社交階級系統，或修改某些玩法參數。接著模組的PatchOperation按其出現在模組列表中的順序執行。因此，一個靠後的模組能修改先前任何模組（包括DLC）的Def資料。「**下方模組優先**」具體體現為：假如 Mod A 載入在 Mod B 之前，而兩者都修改了同一Def的同一值——Mod B 的修改因最後套用而勝出，成為遊戲最終狀態[19][20]。

Patch 與繼承的配合： 值得注意的是，Patch 操作**發生在 Def 繼承解析之前**[24]。RimWorld 的 Def 支援XML繼承：一個Def可以通過 `ParentName` 屬性繼承另一個Def的所有未覆蓋字段，而被繼承者可標記 `Abstract="True"` 使其本身不生成實體[25]。在載入過程中，遊戲會在所有 Patch 操作完成後才執行繼承樹的解析和數值繼承[24]。這意味著**Patch 可以影響繼承體系**：如果一個 Patch 修改了某個父Def的某字段，則所有子Def在之後繼承時都會反映這一改動[26]。但同時，因 Patch 發生在繼承之前，PatchOperation 的 XPath **無法直接選取繼承自父級而暫時不存在於子Def的節點**[24]。需要在Patch中有技巧地處理（例如直接改父Def或在Patch中加入條件）。舉例來說，如果Core定義了一個抽象Def `BaseGun`，其中傷害值為10，Mod可以寫一個Patch將BaseGun的傷害改為12，那麼**所有繼承自 BaseGun 的具體武器Def都會自動繼承新值12**[26]。若只想針對某一子Def調整，不影響其他繼承者，則Patch可在子Def上直接覆寫該值（或使用 `<Inherit="False">` 屬性防止繼承）[27][28]。總的來說，PatchOperation 與繼承機制讓 Def 資料具備**高度延展性**：本體提供抽象模板，DLC/Mod可以繼承並擴充，同時Patch允許在載入時動態改寫任何層級的節點。

Def 疊加與衝突解決： 由於有以上機制，模組之間通常透過 Patch 和繼承來疊加資料，而非互相覆寫整段 XML。這極大降低了衝突發生的概率。然而仍有可能出現兩個模組嘗試修改同一個值的情況。這時負責排序的就是**模組載入順序**，玩家可以透過調整 mod 列表次序或依賴關係設定（例如 About.xml 中的 `<loadBefore>`/`<loadAfter>`）來影響誰的改動生效[24][29]。開發者也提供了工具與指南鼓勵“防禦式”Patch——例如在添加新節點時，使用 `<PatchOperationConditional>` 先檢查目標節點有無，再決定是新增還是附加[30][31]，以避免重複添加造成錯誤[32]。總之，**模組Def的疊加遵循「最後載入者優先」原則，透過Patch及繼承精細控制**。官方DLC由於總是較早載入，其定義可被後面的玩家模組再修改或覆蓋。因此在RimWorld看來，DLC內容也屬於“vanilla”基礎的一部分，**玩家模組可以像修改本體一樣去修改DLC的定義**（前提是玩家擁有並啟用了該DLC）。這種彈性也使得大量「擴充內容加強」或「DLC相容」模組得以存在，利用同一套Def載入流程將各方內容融合在一起。

3.3 PatchOperation、抽象 Def 對流程的影響

PatchOperation 流程回顧： 如前所述，PatchOperations 在所有 Def 資料載入後統一執行。根據Wiki資料，其順序嚴格遵循模組順序進行[21]。這裡再強調幾點：首先，**PatchOperations覆蓋完畢後**，遊戲才會進一步做 Def 的引用解析與完整初始化，包括解析各Def之間的Cross-reference（例如解析Def中指定的其他Def名稱）以及遊戲內部各種資料結構的構建。也就是說，Patch能改動Def的任何部分（增刪節點、改值），而最終

遊戲所用的是修正後的最終版本 Def 資料庫。其次，**PatchOperation執行過程遇到的錯誤**（例如 XPath 找不到節點）通常會產生紅字錯誤提示，以方便模組作者排查 [33]。但是也有一些控制標記（如 `<silent>` 或 `<success>` 參數）可讓模組壓制錯誤輸出，在特定情況下使用 [34] [35]。總之，PatchOperation 機制讓 RimWorld 的資料驅動系統更具動態性，**把可能的衝突轉化為可控制的順序問題**，並提供工具來解決衝突（如 `loadAfter`）[24]。

抽象 Def (Abstract Def) 作用： RimWorld Core 定義了許多 Abstract="True" 的 Def，作為模板供具體內容使用 [25]。例如 Core 裡可能有 `ThingDef Name="GunBase" Abstract="True"`，定義了槍械的通用屬性；而每個具體槍械（手槍、步槍）Def 則 `ParentName="GunBase"` 繼承之。這種**繼承在載入時由引擎自動處理**：在所有 Patch 套用後，遊戲會遍歷每個 Def，若發現有 Parent，則將父 Def 中未被子 Def 覆寫的節點複製給子 Def [36] [27]。抽象 Def 本身不會出現在遊戲中（因為 Abstract=true 不會生成實體）[37]。這套機制極大減少了重複定義，並方便 DLC/模組重用核心設定。例如 Ideology DLC 可能新增多種祭祀活動，其中許多屬性相同，就可以定義一個抽象的 `RitualBase`，其他具體儀式繼承它，再各自添加少量差異。對載入流程的影響在於：**抽象 Def 必須先載入才能被繼承**，也就是為何 Core 裡的抽象基底要早於 DLC/Mods 的具體內容載入。好在RimWorld的載入順序天然滿足這點（Core最早）。另外，如果模組需要覆寫抽象基底的一些值，前面提到PatchOperation可以辦到——修改抽象Def，相當於影響所有子Def預設值 [26]。而**模組切忌“直接重寫抽象Def整個節點”**，因為這會覆蓋其他模組可能對它做的不同修改，應改用Patch有選擇地改變 [38] [39]。這也是Wiki中特別強調的最佳實踐：「不要覆寫抽象基底，除非必要」[38]。綜上，Abstract Def 提供**繼承架構**，PatchOperation 提供**動態調整**，兩者共同構成RimWorld Def 資料載入流程的重要特色。

4. 模組載入順序與優先級控制

ModsConfig 與載入順序： RimWorld 通過 ModsConfig（配置檔或遊戲內介面）讓玩家調整模組載入順序。**載入順序直接決定了XML定義與Patch應用的先後**，從而決定衝突時誰覆蓋誰 [19] [20]。一般原則是上面的（先載入的）提供基礎，下面的（後載入的）進行覆蓋。因此正如玩家常說的「底部模組獲勝」——模組列表越往下的，其改動在衝突中優先生效 [19]。RimWorld 對此提供了**顯式的依賴控制**：模組的 About.xml 可以聲明 `<loadBefore>` 或 `<loadAfter>` 列表，指定相對於某些模組的排序偏好 [24]。對於官方 DLC，正如前文提到的，開發者在 Ideology 等 About.xml 裡用了 `<forceLoadAfter>` 來強制排序 [12]。遊戲啟動時會根據這些依賴關係自動調整實際載入順序，以滿足所有約束。**一旦順序確定，載入過程就是嚴格按照順序線性進行，不會並行載入**，以避免資源衝突。

DLC 的優先權與保護： 官方擴充內容在排序上具有**固定位階**：Core 固定第一，而已安裝的 DLC 依照預定順序排列在 Core 之後、第三方模組之前 [5]。通常遊戲介面不允許玩家將 DLC 拖動到無法啟動的位置。如果強行修改配置使順序錯亂（例如讓 Ideology 在 Core 前），遊戲會報錯並重置順序 [16] [17]。因此可以說 DLC **享有排序保護**，不會被玩家模組插隊到它們前面。同時，由於 DLC 內容被視作「Vanilla的一部分」，它們本身並沒有絕對優先覆蓋權：相反，**在載入後來的模組面前，DLC 資料可以被繼續Patch或修改**。舉例來說，Royalty DLC 添加的帝國勢力和頭銜制度，其相關Def在Royalty載入時成為遊戲資料的一部分，之後若某個模組針對這些Def（比如調整貴族需求）提供Patch且排序在Royalty之後，那麼該修改就會應用生效。DLC 並沒有對抗這種改動的特殊鎖定。所謂的優先權主要體現在**載入時序**上——確保DLC先行，避免模組「搶跑」。一旦進入運行階段，DLC 的內容與Core內容無異，都能被後面的流程（包含其他模組、遊戲機制）自由訪問或更改。

依賴與相容性： RimWorld 提供的排序控制還用於處理模組依賴。例如一些框架模組（HugsLib 等）要求比其他模組先載，作者會在About中宣告 `<loadBefore>` 大量模組包或建議玩家將它置頂。如果模組缺少必要的前置，遊戲在載入XML時就會報出錯誤提示缺少依賴 [29]。玩家則需要根據錯誤調整順序或安裝缺失的模組。對於官方DLC，玩家模組可以在 About.xml 的 `<supportedVersions>` 標籤之外，利用 `<requiredMod>` 或乾脆在Steam後台標註「需要XX DLC」。雖然技術上即使沒有DLC，載入對應模組只要避免調用缺失資源也能跑，但通常作者會直接禁止，避免未知錯誤。總的來說，**RimWorld 通過 ModsConfig + About.xml 的元數據，引導並部分自動管理了模組的載入優先級**。DLC 作為官方模組，其排序由官方定義且玩家無需操心，在這層面上是“優先”且“受保護”的，但它們並不會蓋過玩家後載入模組的變更。這種開放性保證了**擴充與模組可以靈活協同**：DLC 提供新系統，模組可以再調整其平衡或拓展功能，在正確的順序下各司其職。

5. RimWorld DLC 與 Steam 工作坊模組：載入流程之比較

5.1 檔案存放路徑與結構格式

官方 DLC： RimWorld 的 DLC 透過 Steam 或官網安裝後，會出現在遊戲安裝目錄下的 `Data` 資料夾內，以獨立子資料夾區分。例如 `Data/Core` 是本體，`Data/Royalty`、`Data/Ideology` 等則是各DLC [4]。每個DLC資料夾內含有 `About` 資料夾（內有 About.xml，描述名稱、版本、作者、SteamAppID等 meta 資訊）、`Defs` 資料夾（各種XML定義檔案）、`Textures`/`Sounds` 等資源子資料夾，其結構與 Mods 的格式完全一致 [40] [7]。值得一提的是，官方 DLC 的 About.xml 中還列有 `packageId`，通常形如 `Ludeon.RimWorld.<DLCName>`，例如 Ludeon.RimWorld.Royalty [41]。遊戲利用這個唯一ID識別 DLC 並處理依賴關係。此外，DLC 的 About.xml 也會標註其相容的 RimWorld 版本號（如 Royalty 支援1.1起，Ideology支援1.3起等等） [17]，以避免版本不符時載入。**DLC 資料大多採用 XML（Def）和對應資源檔**；如前述，它們通常**不含獨立DLL**，因為行為邏輯已在主程式中實現。

工作坊模組： 來自 Steam Workshop 的 RimWorld 模組會下載至 Steam 資料夾中對應的AppID目錄（RimWorld的Workshop路徑通常是 steamapps/workshop/content/294100）下，每個訂閱的模組以其 Workshop ID 命名的子資料夾存放 [4]。玩家也可以將本地模組放在 RimWorld 安裝目錄的 Mods 資料夾中。無論Workshop或本地，其結構與DLC類似：擁有 About/Defs/Assemblies/Textures/... 等標準目錄 [42] [43]。不同的是，玩家模組的 `About.xml` 由模組作者自定義，`packageId` 通常以「作者.模組名」形式命名，而不是 Ludeon 前綴。模組可以包含自己編譯的DLL（放在 Assemblies 資料夾）來增加新程式行為；也可以僅靠 XML 達到資料修改目的。**檔案格式上**，模組與DLC皆遵循 RimWorld 定義的格式：XML 用於Def、PNG用於貼圖、OGG/WAV用於音效等。RimWorld對模組資源有約定，如貼圖需置於Textures下並以特定路徑引用，Sound需在 Sounds 下並在 Def 中註冊等等 [44] [45]。**簡而言之，DLC 和Mods除了存放路徑不同（內建 vs. 用戶路徑）外，在內容格式和結構上幾乎完全相同**，這也是為何引擎能用同一套機制加載它們。

5.2 載入時機與方式比較

載入時機： 當玩家啟動遊戲時，RimWorld 會在顯示主選單之前載入所有啟用的內容。這包括Core、已啟用的 DLC、以及在 ModsConfig 中標記為啟用的所有模組。**無論DLC還是一般模組，都需要重新啟動遊戲才能完成載入或卸載**（因為Unity遊戲通常不支持運行時即時卸載C#模組）。例如玩家在主選單啟用一個新的DLC或模組，遊戲會提示重啟；重啟過程中才執行實際的載入。載入順序如前所述，Core -> DLC -> 其他模組 [14]。**對比**而言，本體與DLC往往在**每次遊戲啟動**時都會載入（除非玩家手動停用DLC），而工作坊模組則取決於玩家當前的啟用清單。值得一提的是，當遊戲版本更新、特別是主要版本變更時，官方DLC通常會同步更新以保持相容；玩家模組則需要作者手動更新 targetVersion。啟動時遊戲會檢查每個模組的 `<supportedVersions>`，如果發現版本不匹配，會在模組列表中標示“可能不相容”或在控制台警告，DLC 亦然 [16] [17]。

載入方式（DefDatabase 與 PatchOperation）： Core、DLC 和 Mods都是透過**DefDatabase**機制由引擎統一載入的。也就是說，遊戲並沒有給DLC特殊的載入通道；它們的 XML Def 資料由引擎讀入後，都註冊到對應的 Def 資料庫中（例如所有 ThingDef 進入全局 ThingDef 列表）。載入DLC的 Def 與載入模組的 Def 沒本質區別，只是**載入順序決定了它們進入資料庫的先後**。緊接著，PatchOperation 流程也適用於所有來源：例如 Royalty DLC 本身若附帶 Patch（如增加Royalty內容到Core定義中），會在Core Def之後、其他模組Patch之前執行。再後面載入的玩家模組Patch亦可針對Royalty或Ideology的Def進行修改，前提是那些DLC處於啟用且已載入狀態。換句話說，**DLC 與 Mods 共用同一套 Def-載入-繼承-Patch 的管線** [14] [24]。**在執行階段**，遊戲也不再區分某Def來自本體、DLC還是模組——它們都成為遊戲內容的一部分，由 DefDatabase 和各種 Manager 管理。例如，所有生物的 ThingDef 不管來源如何都在同一列表中，事件系統隨機抽取事件時也會包含已載入的 DLC 事件Def等等。因此，一旦載入完成，DLC內容的運作與基礎遊戲內容並無二致。需要強調的是，由於**官方DLC的C#程式已整合**，因此DLC功能的執行也走跟本體相同的程式碼路徑，只是資料差異。反觀玩家模組的 C# 則是在所有 Def 加載完成後再逐個載入其 Assemblies [20]。玩家模組的代碼（例如通過 Harmony patch遊戲方法）可以作用於本體或DLC的程式，但**必須考慮DLC是否存在**。總之，在載入方式上DLC

和Mods都是“模組”，沒有特權跳過DefDatabase，也沒有繞開Patch流程；**它們遵循相同的加載時序與邏輯**，保證所有內容在資料層面彼此兼容。

5.3 版本相依性與相容性差異

版本相依性（官方DLC）： 每個官方DLC都與特定的遊戲版本同步發布，例如Royalty隨1.1版本推出，Ideology隨1.3，Biotech隨1.4等。因此DLC的 About.xml `supportedVersions` 通常只列出一個主要版本，例如 Ideology 標註 `<supportedVersions><li>1.3</li></supportedVersions>` [17]。當遊戲升級到更新的大版本時（如1.3升1.4），官方通常會發布DLC的兼容更新，以便其內容與新系統協同。玩家不需要擔心DLC相容性——只要遊戲本體更新到對應版本，DLC就可以繼續運作（Steam會自動更新DLC檔案）。但像上述論壇例子所示，如果玩家嘗試在不相容的舊版本遊戲中載入新DLC（如1.2遊戲裝了Ideology），遊戲會直接報錯拒絕載入 [16] [17]。**因此官方DLC的相依性很嚴格：必須匹配遊戲版本**。同時，DLC之間一般**不強制要求**彼此（每個DLC可獨立啟用），但在有多個DLC時官方已設計好它們的交互順序和條件邏輯。例如 Ideology + Royalty 同時啟用時，兩者內容可以一起運作且部分互動，由於主程式考慮了這些情況（如貴族也有信仰的情況），在程式碼和Defs中加入了對雙DLC的支持。這些交互通常用條件Def（比如帶 `MayRequire` 的節點）或程式碼分支實現。

版本相依性（玩家模組）： 相對地，工作坊模組的作者需要自行指定模組適配的遊戲版本。同一模組可能隨遊戲更新推出不同版本（作者上傳多個Workshop項目或更新SupportedVersions）。遊戲在載入模組前會檢查其 `<supportedVersions>` 列表是否包含當前遊戲版本號。若不包含，則在Mods列表界面將該模組標記為不適配版本，並**可能導致載入錯誤**。很多模組只是XML數據調整，較小版本更新時即使標記不相容也能正常工作，但複雜模組（帶C#）通常需要作者更新。這形成了玩家社群常見的模組相容性維護工作。相比之下，**DLC由官方保證與新版本同步，玩家不用擔心其相容問題**（除非選擇執行老版本遊戲，在那種情況下新DLC無法用）。在依賴性方面，玩家模組還可以依賴官方DLC。常見做法是在About.xml中聲明 `<li>Ludeon.RimWorld.<DLC></li>` 於 `requires` 或 `dependencies` 列表（或直接在Steam標記需要某DLC）。這樣遊戲就會在未啟用該DLC時提示玩家。本質上，**模組與DLC的關係也通過packageId機制處理**：模組可以用PatchOperationFindMod/Package或 `MayRequire` 等來有條件支持DLC [46] [9]。在無該DLC時，模組應保持功能降級但不報錯，這考驗作者功力。值得一提的是，官方對於DLC內容有社群規範，不允許模組「盜用」DLC素材給沒買DLC的玩家 [13]。技術上透過前述報錯機制實現，法律上也在社群規則中明確。總而言之，**玩家模組的版本及相依處理更分散，由每個作者控制**；官方DLC則由Ludeon統一維護相容性，玩家只需更新遊戲本體即可。

相容性與衝突： 由於DLC皆為Ludeon自家內容，**它們與本體的相容性是內建的**（發行前已經過完整測試）。多個DLC同時啟用也在開發時就設計好了交互（例如雙持DLC時新增的聯動內容）。這種**一體化**使得官方DLC幾乎不會和本體產生衝突問題。而玩家模組彼此獨立開發，可能改動同一系統而發生衝突，需要靠前述的載入順序調整或作者協調來解決 [47] [39]。不過，有趣的是，從引擎角度看，**官方DLC和模組其實享有同等的“修改權”**。例如，一個模組如果載入順序在Royalty之後，它甚至可以移除或更改Royalty新增的內容（可以用PatchOperation把某Royalty的Def整段刪除，儘管一般沒人這麼做）。引擎不會阻止這種行為——只要玩家有Royalty，這些Def就在資料庫，可被修改。這也意味著**DLC沒有賦予內容“不可被改”的保護鎖**，相容性更多是靠作者自律和玩家自行排序維護的。當然，大多數模組作者會針對有無DLC做相容處理，如前述利用 `MayRequire` 或提供兩套配置，以避免在缺少DLC時改動不存在的東西。此外，由於DLC通常引入大型新系統，許多模組會特地針對“有DLC時怎樣、無DLC時怎樣”做適配。這種情況下DLC的存在與否對模組功能影響較大，但這是高層設計問題，不屬於引擎載入衝突。

設計哲學與優劣勢分析

統一的模組架構哲學： 從上述技術細節可以看出，RimWorld 採用了**模組驅動的架構**來實現其核心遊戲與擴充內容。官方把本體和DLC都當作模組來對待，只是本體模組無法禁用，DLC模組需購買啟用。這種設計哲學強調

內容的解耦與資料導向：遊戲邏輯盡可能寫死在C#裡，而實際具體數值、物件、事件則透過XML定義，方便增減修改。優點是明顯的：

- **擴充與基礎共用框架，降低重複開發：** 官方在發布DLC時，只需在原有系統基礎上添加資料和少量必要的程式分支，無需fork一個新程式版本給有DLC玩家。這確保所有玩家運行相同的執行檔，減少分裂。同時讓**擴充功能可與原版內容無縫整合**。例如Royalty的帝國勢力、超能力可以影響原版劇情事件，因為底層是一樣的，只是多了新Def和新邏輯。由於一體化，DLC內容也能夠**很容易地被mods再度修改**，這迎合了RimWorld重視模組生態的理念。

- **高度模組化，促進社群創作：** 將Core和DLC都模組化意味著遊戲天生支持改裝（Modding）。事實上，RimWorld在設計時就考慮了可修改性，提供了Harmony庫讓作者熱補丁代碼，提供了XML Patch機制避免覆蓋衝突。官方DLC沿用同一套系統，等於做了範例。這種一致性降低了學習成本——模組作者只要學會修改Core，就能修改DLC內容。長遠看，**社群可以基於DLC做二次創作**（比如擴展新的帝國頭銜、宗教理念等），進一步豐富遊戲，而Ludeon的擴充正是為此鋪路。

- **動態內容管理，提高相容性：** 通過載入順序和Patch系統，RimWorld 幾乎可以容忍上百個模組同時存在而**在資料層沒衝突**，只要順序妥當。DLC 作為“官方模組”，也受惠於此——玩家可以將上百模組與多個DLC一起用，很多情況下能正常遊玩。這相比某些遊戲擴充必須開新exe或者互斥，顯然更彈性。官方擴充因為和mods共享環境，也能**迅速得到社群支援**：新DLC出了，很快就有相容模組、平衡調整模組等，增進了DLC價值。

當然，此架構也有一些潛在**缺點與挑戰**：

- **載入依賴的複雜度：** 讓一切皆模組意味著**載入順序非常關鍵**。對新手玩家而言，大量模組的排序是一門學問，稍有不慎就可能錯亂導致紅錯。雖然有自動排序工具和依賴標記協助，但仍需要玩家理解一些原理。官方DLC雖然自動排序，但當多個DLC和上百模組混合時，依然可能遇到比如「某模組應在Royalty前才能改某值」這類要求，增加瞭解決衝突的難度。總體而言，系統**將衝突交由順序解決**，這在大量模組環境下偶爾會產生繁瑣的調整工作。

- **性能與內存負擔：** RimWorld 的資料驅動架構需要在啟動時解析大量XML，應用大量Patch，再構建所有Def對象。當模組特別多時（上百個、上萬行XML），載入時間顯著增加，內存佔用也提高。官方DLC本身增加的內容量不小，跟大量模組疊加後，新遊戲啟動可能幾分鐘才能進入。雖然這是模組友好帶來的自然代價，但對某些玩家體驗來說算一個弱點。此外，由於Unity的單線程加載，這部分性能損耗無法利用多核緩解。

- **模組對DLC的潛在破壞：** 由於沒有嚴格區隔，**模組完全可以改到DLC的核心機制**，這有時會出現破壞性行為。比如一個模組不當地patch了Royalty的心理狀態計算，可能導致擁有Royalty的玩家遊戲出錯。從Ludeon角度看，這樣容易讓人誤以為是DLC本身bug。雖然社群有共識去規範模組行為，但開放架構不可避免地帶來不可控因素。官方能做的只是在API層面提供檢查（如前述RoyaltyInstalled檢查）和約束（報錯未授權代碼），但無法杜絕“惡意”或“不當”Mod修改。因此，**開放性是雙刃劍**：一方面賦予創造力，另一方面也需承擔風險。

- **DLC功能取決於本體更新：** 官方採用把DLC代碼整合進本體的方式發佈，這意味著每次出DLC都伴隨一次大型版本更新（例如1.3配合Ideology）。沒購買DLC的玩家也會升級本體並得到相關代碼（只是沒有內容）。從產品角度看，這讓非DLC玩家也承受了部分升級變動，有時可能改動遊戲平衡或新增Bug（與DLC無關的基礎變更）。這點在Ideology推出時明顯：1.3版本改動了動物馴養、奴役等機制，所有玩家都得接受，不管是否購買DLC，引發過一些討論。這是**一體化更新**帶來的影響，優點是維持代碼一致性，缺點是付費內容的更新波及免費玩家體驗。儘管官方會盡量把重大變更合理化（比如以免費內容更新名義加入），這依然是值得平衡考量的地方。

總結設計取向： RimWorld 的擴充與模組系統顯示出明確的取向——**極高的模組化與擴展性**，這與其作為沙盒模擬遊戲、鼓勵玩家塑造體驗的理念吻合。其設計哲學不是將DLC功能封裝得嚴嚴實實，而是提供一個框架，讓DLC成為這框架下的“官方模組”。這種做法的哲學在於相信社群的創造力，以及透過開放來延長遊戲生命力。優點在於靈活、多變、社群共創，缺點是需要在相容性與體驗一致性上投入更多管理精力。總的來說，RimWorld 選擇了賦予玩家和模組作者**更大自由度**的路徑，從而換取遊戲內容的繁榮生長——其DLC與模組載入設計正是服務於此一理念的具體體現。

參考資料：

1. RimWorld 模組檔案結構與載入教學 – RimWorld Wiki [6] [4]
2. RimWorld XML PatchOperations 載入順序 – RimWorld Wiki [24] [14]
3. 玩家社群對 Core/DLC 代碼關係的討論 [1] [2]
4. RimWorld `MayRequire` 與 DLC 依賴屬性 – RimWorld Wiki [9] [8]
5. Ludeon 官方論壇 – DLC 載入排序錯誤回報 [16] [12]

---

[1] [18] Harmony - the full story : r/RimWorld
https://www.reddit.com/r/RimWorld/comments/fbnm45/harmony_the_full_story/

[2] [14] [19] [20] [29] [30] [31] [38] [39] [47] Modding Tutorials/Compatibility - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Compatibility

[3] [6] [7] [40] [42] [43] [44] [45] Mod Folder Basics - RimWorld Modding Wiki
https://rimworldmodding.wiki.gg/wiki/Mod_Folder_Basics

[4] Modding Tutorials/Getting started with mods - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Getting_started_with_mods

[5] How the heck do i install Royalty DLC? :: RimWorld General Discussions
https://steamcommunity.com/app/294100/discussions/0/2295094230834708439/

[8] [11] [13] [46] User:Dninemfive - RimWorld Wiki
https://rimworldwiki.com/wiki/User:Dninemfive

[9] [10] [41] MayRequire - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/MayRequire

[12] [16] [17] (1.2.3062) Core red error - The STRIPPED 1.2 Core doesn't like disabled Ideology
https://ludeon.com/forums/index.php?topic=54627.0

[15] Modding Tutorials/XML Defs - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/XML_Defs

[21] [22] [23] [24] [26] [28] [32] [33] [34] [35] PatchOperations - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/PatchOperations

[25] [27] [36] [37] XML Inheritance - RimWorld Modding Wiki
https://rimworldmodding.wiki.gg/wiki/XML_Inheritance