ChatGPT

RimWorld 實例與運作機制深入解析

各類遊戲實例的分類與說明

- **Pawn（角色）**：Pawn 是 RimWorld 對**一切可行走移動的生物**的統稱，包括人類殖民者、敵對襲擊者、動物、機械體等 [1]。Pawn 具有複雜的屬性（如需求、技能、健康狀況），是遊戲中最核心的實例之一。

- **Thing（物件）**：Thing 是**遊戲中一切具體物件**的基類，包括道具、建築、植物甚至屍體等。許多遊戲實例（包含 Pawn 在內）都被實作為 Thing 或其子類。具體而言，RimWorld 中**能放置在地圖上的東西通常都是 Thing 的子類** [2]。Thing 定義了物件的基本屬性，如耐久（HitPoints）、位置、堆疊數量等。

- **Building（建築）**：Building 是 Thing 的一個子類，用於表示建築物實例。建築物如牆壁、床、工業設備等皆繼承自 Building 類別，具備 Thing 的通用屬性並擴充了一些建築專有行為（例如可連接電力等）。由於 Building 繼承自 ThingWithComps（Thing 的擴充類），建築物可附加組件（如電力組件）實現額外功能 [2]。建築的定義資料（如尺寸、材質需求）由相應的 Def 提供（通常還是以 ThingDef 定義）。

- **WorldObject（世界物件）**：WorldObject 指世界地圖上的物件實例，例如派系據點（settlements）、遠征隊（caravans）、臨時任務地點（sites）等。WorldObject 有別於地圖上的 Thing，**它存在於世界地圖層級**。每種世界物件通常由一個 WorldObjectDef 定義靜態資料，包括名稱、圖示等，而運行中則有對應的 WorldObject 實例，包含位置（所在世界座標）、所屬派系等動態資訊。

- **Incident（事件）**：Incident 是遊戲中的隨機事件（如襲擊、瘟疫、天氣變化等）的邏輯實例。事件由 **IncidentDef** 定義其參數（發生條件、效果等），並由對應的 **IncidentWorker** 類別執行。事件不像 Pawn/Thing 那樣持續存在的物件，**更像是一段觸發即執行的流程**；當事件發生時，IncidentDef 的 worker 方法會生成所需的瞬態實例（例如產生襲擊者 Pawn、生成天氣變量等），事件完成後並不保留一個長久的 “Incident 實例”。

- **Quest（任務）**：Quest 是隨資料片引入的多步驟事件/任務系統。一個任務由靜態的 **QuestScriptDef** 定義生成邏輯，包括可能的任務目標、獎勵等。當任務觸發時，遊戲會根據 QuestScriptDef **生成一個 Quest 實例**，其中包含一系列 **QuestPart**（任務片段）來追蹤任務進程 [3]。Quest 作為實例會顯示在玩家的任務列表中，持續存在直至任務完成或失敗為止。

實例與 Def 的關係

上圖概括了 RimWorld 中 Def 與實例類別的關係：左側橘色方框表示各種 **Def（定義數據類別）**，右側藍色圓節點表示 **實例對象的類別**。可以看到，每種 Def 都對應一種或多種運行時實例：

- **Def 是靜態的藍本數據**：例如，ThingDef、PawnKindDef、QuestScriptDef 等都是 `Verse.Def` 的子類，用於描述遊戲內物件或事件的**配置資料**。它們包含該物件的靜態屬性，如名稱（`defName`）、描述、基礎數值等 [4]。**實例則是運行時根據 Def 創建的對象**，例如 Pawn（由 PawnKindDef 和種族的 ThingDef 決定）、具體的 Thing 或 Building（由 ThingDef 定義）、Quest（由 QuestScriptDef 生成）等。

- **不同類別而非同一對象**：Def 本身並不是直接的遊戲物件類別。舉例而言，PawnKindDef 定義了一類 Pawn 的模板（如某種職業或派系成員），而 Pawn 類別則是真正的遊戲角色類別。兩者在類別結構上是截然不同的類別。一個 Pawn 實例**不會是** PawnKindDef 類別的物件，而是 Pawn 類別的物件。Pawn 實例會包含一個引用指向它的 PawnKindDef（例如 `pawn.kindDef` 字段）以了解其模板屬性 [5]。

- **由 Def 初始化實例**：當遊戲需要創建某個實例時，會**根據 Def 中指定的類別**進行初始化。例如，ThingDef 定義了一個 `thingClass`，指明對應的實例類型 [6]。遊戲使用 `ThingMaker.MakeThing` 等工廠方法，利用反射**生成 Def 對應的實例對象**，並將該 Thing 的 `def` 屬性設置為對應的 ThingDef [6]。隨後會調用 PostMake 等初始化方法完成實例的組件附加等操作 [7]。總結而言，實例並非從 Def “複製”而來，而是**按照 Def 規定的類別和屬性**在運行時創建並初始化。

- **Def 提供靜態欄位，實例擁有動態欄位**：實例的一部分屬性直接來源於其 Def。例如，道具的基礎傷害、建築的尺寸、Pawn 的種族基礎移動速度等皆由其 Def 定義的數值決定。實例會從自己的 `def` 引用讀取這些靜態數據（如 `pawn.def.race` 包含種族常數屬性） [8]。然而，實例還包含許多**只能在運行時確定或改變的欄位**：如 Pawn 當前的飽食度、心情值，物品當前的耐久度、品質等。這些屬性在 Def 中並不存在，而是在實例生成時初始化，並隨遊戲進行而變化。

- **初始化時機**：Def 在遊戲啟動時就全部加載完畢並存於 DefDatabase 中（詳見下節），而實例對象則通常**在需要時動態生成**。例如：載入一張地圖時生成其中的所有 Thing/Pawn 實例；事件觸發時生成相關 Pawn 或 WorldObject；建築物在玩家建造完成時生成 Building 實例。每當實例生成，遊戲都會基於對應的 Def 來設置其初始狀態（例如耐久度=最大值，Pawn 各項Need初始滿足度等），確保**靜態資料**正確應用於**動態對象** [7]。

靜態資料 (Def) 與動態實例

Def 的載入：RimWorld 啟動時會讀取核心和模組中定義的所有 XML Def，為每個定義創建相應的 Def 對象（如 ThingDef、PawnKindDef 等）。這些 Def 對象在載入時會被存入一個全域的 **DefDatabase**，透過類型進行分類管理 [4]。如此一來，遊戲隨處可以通過 `DefDatabase<DefType>.GetNamed("defName")` 來查詢所需的定義 [4]。例如遊戲在隨機產生任務時，會瀏覽 DefDatabase 中所有 QuestScriptDef，依其權重隨機挑選一個用於生成任務 [9]。**DefDatabase 相當於靜態資料的資料庫**，在遊戲運行期間通常不會再增刪（除非動態Mod特殊操作），確保所有實例引用的 Def 都可被查找到。

實例的生成：動態實例通常透過工廠/生成器模式創建。例如：- **物件 Thing 的生成**：使用 `ThingMaker.MakeThing(def, stuff)` 方法。它會依據給定的 ThingDef，反射創建對應類別的物件，並自動處理材質(stuff)參數。例如，如果 ThingDef標記需要材質而未提供，會記錄錯誤並以默認材質代替 [10]；若提供了材質但定義不允許，則忽略該材質 [11]。生成後會將 Thing 實例的 `def` 設為對應定義，`stuff` 設為選定材質，接著調用 PostMake 進行後續初始化 [6]。這期間，若 ThingDef 定義了 comps 列表，則實例會附加那些組件並初始化 [7]。

- **Pawn 的生成**：Pawn 的產生過程更複雜，一般由 `PawnGenerator` 完成。當遊戲需要一個 Pawn（例如事件產生襲擊者或開局殖民者），會構造一個 PawnGenerationRequest，其中包含 PawnKindDef（種族/職業模板）、派系、年齡性別等參數。PawnGenerator 隨後依據 PawnKindDef 的種族屬性找到對應的 ThingDef（種族定義），用它來創建 Pawn 實例。Pawn 實例建立後，遊戲會為其**配置各項子系統**（如需要則生成關聯的裝備、服裝等），並應用 PawnKindDef 中預設的技能值、物品清單等資料。最終 Pawn 出現在地圖或世界中。整個過程中，Pawn 實例的 `def` 指向其種族的 ThingDef，而 `kindDef` 欄位則記錄了它所屬的 PawnKindDef 模板 [5]。

- **世界物件的生成**：世界事件（如產生任務據點）可能需要建立 WorldObject 實例。遊戲會使用對應 WorldObjectDef 中定義的 `worldObjectClass` 來實例化（例如生成一個 Site 或 Caravan 類的物件），設定其所屬派系、坐標等運行時資料，並加入世界經緯網列表中。WorldObject 實例的 `def` 屬性會指向創建它的 WorldObjectDef，以獲取靜態資訊（例如在世界地圖上的圖標、名稱格式等）。

靜態 vs 動態資料範例： 一件武器的 **ThingDef** 可能定義了其基本傷害、品質等級上限、可用材質類別等。當這件武器在遊戲中生成 (Thing 實例) 時： - **靜態欄位應用**：實例會引用 ThingDef 中的基礎屬性，如最大耐久度、重量、價值等作為初始值或常數。 [12] [13] 例如 ThingDef 定義了 `<MaxHitPoints>100</MaxHitPoints>`，那麼生成實例時其 HitPoints 上限為100。 - **動態欄位建立**：若此武器 Def 帶有品質（Quality）組件，則實例在 InitializeComps 時會創建一個 **CompQuality** 附加其上，用來存放該具體物品的品質等級 [14]。品質等級會在物品生成時決定（例如由製造者技能決定優良/精良），並存儲於 CompQuality 中，而非來自 Def。又如 Def 指定此武器可用不同材質，實例生成時就需要一個實際材質，如 鋼 或 木頭，該材質選擇是動態的，在 Def 中只以類別（Stuff Categories）描述可能性 [10]。

總之，Def 扮演**提供配置**的角色，而實例在創建時**讀取這些配置並將其具現化**為具體屬性值，同時創立自身的狀態欄位，以便在遊戲過程中更新。

遊戲實例的屬性運作機制

Pawn（角色）的動態屬性

Pawn 作為複雜的生命體，有多種子系統來管理其狀態：

- **需求（Needs）與心情**：每個 Pawn 在生成時都會附帶一組需求，如飢餓、休息、娛樂和（對人類）心情等。這些需求由 `Pawn_NeedsTracker` 管理。在 Pawn 創建時，若其 `needs` 尚未初始化，遊戲會建立一個新的 Pawn_NeedsTracker 並附加到 Pawn 上 [15]。Need 系統會根據 Def（NeedDef）的靜態設定來決定有哪些需求以及滿足/缺乏時的影響，但每個 Pawn 的當前需求值是動態變化的，例如**心情**會隨環境和事件上下波動。遊戲每刻（每 tick）都會更新 Pawn 的需求值，進而影響如心情指數等屬性。

- **技能（Skills）**：Pawn 擁有多種技能（射擊、近戰、勞動、醫療等）。技能的種類及上限由靜態的 SkillDef 列出，但**每個 Pawn 都有自己的技能等級與經驗值**。在 Pawn 初始化時，對於人形生物會創建 `Pawn_SkillTracker` 來持有該 Pawn 的技能列表 [16]。起始技能等級可由 PawnKindDef 或背景故事 (BackstoryDef) 提供模板值，然後隨遊戲行為增減。技能屬性純屬動態資料：Pawn 的 Def 並不直接包含技能等級，而只是規定有哪些技能存在及其作用。

- **裝備與物品**：Pawn 可以攜帶和裝備物品。對於能使用工具/武器的 Pawn（通常 RaceProperties.ToolUser = true），遊戲在其生成時會建立 `Pawn_EquipmentTracker` 和（如適用）`Pawn_ApparelTracker`，分別管理武器裝備和穿戴服裝 [17]。例如殖民者 Pawn 創建時若有預設武器，會通過 PawnKindDef 指定，生成對應物品實例並由 EquipmentTracker 託管。這些裝備物品本身是 Thing 實例，Pawn 的裝備系統只是保存引用並提供介面（如切換武器）。裝備、物品的清單會隨玩家操作或事件改變，是完全動態的內容。

- **健康（Health）**：每個 Pawn 有一個 `Pawn_HealthTracker`，其下管理著身體狀況，包括各部位的創傷、疾病（Hediff列表）等。Pawn 創建時會初始化 healthTracker [18]。健康系統結合靜態的 BodyDef（身體結構定義）和 HediffDef（健康差異定義）來作用：Pawn 的種族 Def 定義了其身體構造（有哪些部位，基礎體力等），但**具體的傷勢和健康狀態是實例層面的**。比如一個人類 Pawn 的 BodyDef 定義了其有腿有臂，而當前左臂是否受傷則由 Pawn 的 healthTracker 裡的 Hediff 列表決定。

隨戰鬥受傷、療傷休養，Pawn 的健康狀況會即時更新。健康屬性還影響 Pawn 的其他動態狀態，例如失去一條腿會降低移動速度等——這些計算都是在實例運行時根據狀況評估的。

- **其它子系統**：Pawn 還有許多其他屬性系統，例如 **AI Mindstate**（`Pawn_MindState`，控制AI行為狀態，如狂暴、閒逛等）、**工作設定**（`Pawn_WorkSettings`，記錄該Pawn擅長和允許從事的工作類型）[19]、**關係**（`Pawn_RelationsTracker`，追蹤與其他Pawn的社交關係）等。這些都在 Pawn 創建或加入地圖時初始化[20][21]。它們的存在使 Pawn 具備豐富的行為和狀態，而且大多數數值在遊戲過程中會改變（例如關係好感度隨互動變化，工作優先級可由玩家調整）。**總之，Pawn 的 Def（通常是種族的 ThingDef 和 PawnKindDef）提供了生物的靜態參數**（如種族基礎移動速度、食性、初始技能範圍等），**而 Pawn 自身承載所有實際運行中變量**，透過一系列 Tracker/Manager 類來動態維護這些狀態。

Thing（物件）的動態屬性

非生物的物件（Thing）在遊戲中也有靜態與動態結合的屬性機制：

- **品質（Quality）**：部分物品具有品質等級（從劣劣到傳奇等），這決定了物品的價值和效能。品質並非每個 Thing 都有，而是由 Def 決定。**如果一個 ThingDef 在其 `<comps>` 列表中包含了 `CompQuality`**，那麼該物品實例就會生成對應的品質組件，並擁有一個品質屬性[14]。品質等級是在物品製造時根據製造者技能和隨機因素確定，屬於實例數據，後天也可能透過開發者工具等修改。但品質框架本身（等級類別及效果）由靜態定義（QualityCategory Enum 和相關 StatFactors）決定。

- **材質（Stuff）**：許多建築和物品可以由不同材料製成，如木頭、鋼鐵、白銀等。ThingDef 通過 `MadeFromStuff` 屬性指定是否允許選擇材質。對於 **MadeFromStuff=true** 的定義，實例在創建時**必須指定一種具體材質 (ThingDef)**。遊戲在生成該物品時會檢查並確保提供了合法的材質：若缺失則自動填入預設材質，若不需要材質卻提供了則忽略[10]。實例會將所選材質存儲（例如 `thing.stuff`），影響其屬性（比如鋼製武器與木製武器的耐久和價值不同）。材質屬性一旦創建後通常不可改變（物品不會在運行時改變材質）。

- **堆疊數量**：同類可堆疊物品在同一格可以存在多個，這由 Thing 的 `stackCount` 屬性表示。Def 裡定義了該物品 `stackLimit` 上限，但**當前堆疊數**是實例狀態，隨遊戲中合併或分拆物品而變動。比如藥草的 ThingDef.stackLimit=75，但某堆藥草實例可能當前只有20個，玩家使用幾個後變成17個，這都是實例的動態變化，Def 並不記錄。

- **耐久度與損壞**：大部分物件有耐久值（Hit Points）。Def 的統計值 `MaxHitPoints` 提供了該物品滿血時的耐久上限[12]。實例生成時耐久通常為上限值，存於 Thing 的 `HitPoints` 屬性。**隨物件受損或磨損**，HitPoints 會降低；若降為0物品即毀壞消失。這些損壞過程中，Def 不變，但實例的 HitPoints 是即時更新的。此外，Def 可能通過 `DeteriorationRate` 等設定影響耐久隨時間自然減少的速率[22]，但當前耐久度數值仍是每個實例獨立維護。

- **組件 (Comps) 及其他動態欄位**：很多 Thing 可以附加 **ThingComp** 來提供額外屬性，例如帶有燃料的建築有 CompRefuelable，帶開關的有 CompFlickable 等。這些組件透過 Def 的 `<comps>` 配置增加，**但在實例中以組件對象形式存在並保存狀態**。舉例而言，一座電器設備建築的 ThingDef 定義了 CompPowerTrader，那麼實際放置在地圖上的該建築實例就會有一個 CompPowerTrader 實例附加其上，內含當前電力狀態（是否通電、耗電量等）——這部分資料明顯是隨遊戲變化的。總體來說，Thing 實例的額外欄位大部分都通過 ThingComp 來實現，**Def 決定有哪些 Comp，而 Comp 的內部數值則是實例態**。

綜上，靜態的 Def 決定了一個物件可能有哪些**屬性維度**（有無品質？可否由材質製成？最大耐久多少？），而實例則在運行中填充和改寫這些屬性的**具體值**。靜態資料提供統一規則與上限，動態實例呈現實時的具體狀態。

模組如何影響實例運作

RimWorld 的模組(Mod)能透過多種機制介入遊戲實例，從改變屬性到擴充行為：

- **Harmony Patch (方法攔截修改)**：Harmony庫允許Mod在運行時**修改原始程式碼的行為**。Mod作者常用它來對遊戲方法進行前綴/後綴或替換。例如，可以Patch角色受傷的函式，使當Pawn受到傷害時執行額外邏輯（如特殊護盾減傷）或改變其結果。Harmony修改直接作用於遊戲邏輯層面，**影響所有相應實例**的行為。比如修改Pawn.Tick()可以改變每個Pawn每tick的更新過程。這種方式威力強大，但需要謹慎避免與其他Mod衝突。

- **ThingComp / PawnComp（附加組件）**：如前述，ThingWithComps類的對象（包括大多數物品、建築甚至Pawn）都支持附加組件以延展功能[2]。Mod 可以定義自訂的 ThingComp 或其衍生類（PawnComp 是針對Pawn的子類）來附加到對象上。透過XML的 `<comps>`，Mod能將新的 CompProperties 插入現有ThingDef，使**新組件自動附加到該類物件的所有實例**。組件可以保存自己的數據並覆寫如 CompTick、PostSpawnSetup 等方法來實現持續效果[23]。例如，一個武器模組可給武器附加一個Comp，實現武器每擊中目標時附加中毒效果的功能。組件方法在遊戲過程中被調用，等於為實例**注入了新的運行時行為**而無需修改原有類別。此外，PawnComp（本質上也是ThingComp，只是特化用於Pawn）可以為Pawn增加新屬性，如某Mod給Pawn增加“士氣”值，透過PawnComp每tick改變士氣並影響Pawn狀態。

- **Def 擴充屬性 (DefModExtension)**：這是一種相對非侵入的擴充方式。DefModExtension允許Mod為任何Def添加自定義欄位，以存放Mod需要的額外靜態數據[24]。實作上，每個Def有一個列表可容納多個擴充物件。Mod可定義新的 DefModExtension 類（繼承自 DefModExtension），在XML裡把它附加到目標Def的 `<modExtensions>` 節點[25]。這並不直接改變實例，但**Mod的程式碼可以在運行時讀取Def上的擴充資料**，據此決策影響實例行為。例如，一個Mod可以給某些動物的種族ThingDef附加一個擴展標記“canSwim: true”，然後在Pawn移動判定的程式碼中檢查該Pawn的種族Def是否有此擴充，以決定其能否下水。相比於ThingComp，Def擴充不會每個實例保存一份（因為它屬於Def共享），適合描述種族或物品**全局不變的特性**[26]。但它只能提供靜態資訊，若需要在實例間不同或會改變的數值，仍需配合其他機制。

- **運行時直接操作**：Mod也可以撰寫自身的遊戲邏輯，在適當時機直接修改實例屬性。RimWorld 提供許多管理器類單例（如 `GameComponent`、`MapComponent`、`Ticker` 等）讓Mod能每幀或定期執行代碼。Mod作者可以在這些更新迴圈中查找特定實例並調整其數據。例如，一個天氣模組可以每隔一段時間掃描地圖上所有植物（Thing類的Plant子類），逐個改變其生長速度以模擬季節影響。這種直接操作需要開發者熟悉遊戲的API，比如使用 Map.listerThings 獲取地圖上的Thing清單等。相對而言，Harmony Patch和Comp更多屬於“被動式”在既有架構下運行，而**直接操作**則是Mod主動執行邏輯影響實例。

需要注意的是，以上幾種方式常常結合使用。例如，一個大型Mod可能**同時**：使用Patch更改核心更新頻率、透過Def擴充標記出特定對象、並給這些對象附加Comp以儲存額外數據及提供介面。與簡單的XML Patch（修改Def數值）不同，這些方法允許Mod在**遊戲運行過程中**改變和控制實例的狀態。總結而言，RimWorld 的架構提供了多層次的擴充點：從**定義階段**的數據擴充，到**對象級別**的組件附加，再到**方法級別**的攔截修改，滿足進階Mod對遊戲實例幾乎任意方面的調整需求。

Def 載入與實例生成的流程示意

為了更清晰理解靜態Def資料如何轉化為動態實例，下列以遊戲流程總結各步驟：

1. **遊戲啟動–載入所有 Def**：在主選單載入時，遊戲會讀取核心和各模組中的XML定義。對於每種Def類型（ThingDef、PawnKindDef、IncidentDef等），解析XML創建對應的Def對象，並將其加入 DefDatabase 的列表中 [9]。此時所有靜態資料已就緒，可供遊戲查詢使用。

2. **開始遊戲/生成世界**：玩家開始新劇本時，遊戲根據情境從Def中抽取需要的信息。例如隨機地形從 BiomeDef中取得，開局陣營從FactionDef中選擇。建立世界地圖和初始定居點過程中，可能會產生一些 WorldObject（如玩家初始基地），這些是按照WorldObjectDef創建的實例。

3. **生成地圖與物件**：進入遊戲地圖後，地圖上初始存在的所有物件（如地形、植物、動物、殖民者、資源等）都被逐一創建實例。例如，地形根據 TerrainDef 塗佈，植物按 PlantDef 生成對應的 Plant 實例，開局Pawn按 PawnKindDef 和種族 ThingDef 創建 Pawn 實例並初始化需要的Tracker [27] [28]。這個階段可視為將靜態世界轉化為動態遊戲對象的批量實例化過程。

4. **實例初始化**：每個實例創建後，會執行自身的初始化邏輯。例如Thing通過 ThingMaker.MakeThing 創建後，調用 PostMake -> InitializeComps，把 Def 中配置的各種Comp實例化並附加 [7]；Pawn 則在生成後由 PawnComponentsUtility 配置生命、AI、需求、技能等各個子系統 [29] [27]。這一步確保實例帶有完成的屬性集（靜態來自Def，動態部分已設初值）。

5. **遊戲運行**：在隨後的遊戲過程中，Tick管理器每隔一定遊戲時間推進所有實例的狀態。Pawn 會按照各自的需求消耗值、執行AI行動，物品可能因環境腐壞降低耐久，建築的Comp持續運作（比如發電機的燃料Comp每秒消耗燃料）等。玩家與事件也不斷創建或銷毀實例（如建造建築、擊殺敵人掉落物品等）。**此時靜態Def依然提供規則參考**（例如計算傷害時會查詢武器的ThingDef和Projectile Def），但實例的即時變化完全由遊戲邏輯和玩家操作驅動。

6. **存檔與載入**：當玩家存檔時，遊戲將當前所有動態實例的狀態序列化（通過Scribe系統），包括它們引用的Def（以defName標識）。下次載入該存檔時，遊戲會先重新加載相同版本的所有Def到 DefDatabase，然後根據存檔資料重新生成實例並恢復其屬性（例如Pawn重新建立並套用之前保存的健康、裝備等）。因為Def是靜態且一致的，所以存檔只需記錄每個實例**引用哪個Def**以及動態改變的數值。這機制確保不同玩家環境下（只要Mod列表相同）存檔也能準確還原實例狀態。

7. **模組介入**：在上述流程的各階段，Mod都可能參與進來。比如有些Mod會在Def加載後對DefDatabase 進行調整（添加/刪除Def或修改數值）；更多Mod是在遊戲運行階段介入，如通過Harmony Patch改變Tick行為，或在物件初始化時（PostSpawn等事件）執行自訂邏輯。得益於RimWorld靈活的設計，**模組可以在不破壞核心流程的情況下鉤入幾乎任何一步**。這也正是前述Mod影響實例的方法能發揮作用的原因。

總而言之，RimWorld 將靜態定義與動態運行清晰分離：Def階段先定義好「有哪些東西」「基本屬性如何」，實例階段再按照需要「生成東西」「隨遊戲變化」。了解這兩部分如何交互，對進階Mod製作和深入分析遊戲機制都至關重要。以上內容希望能清晰詳盡地闡明 RimWorld 中 Pawn、Thing、Building、WorldObject、Incident、Quest 等實例的架構與運作，協助模組作者和高級玩家更好地理解遊戲底層原理。 [2] [3]

---

[1] Pawns - RimWorld Wiki
https://rimworldwiki.com/wiki/Pawns

2 7 23 Modding Tutorials/ThingComp - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/ThingComp

3 9 Modding Tutorials/Quests - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Quests

4 Modding Tutorials/BigAssListOfUsefulClasses - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/BigAssListOfUsefulClasses

5 8 Pawn.cs
https://github.com/YoOy0o/Rim_World_Source_Code/blob/49047c0a99ca5fefeec05e4ac28b1908e8d5c2c1/RimWorldDecompile/Verse/Pawn.cs

6 10 11 ThingMaker.cs
https://github.com/josh-m/RW-Decompile/blob/d5bbfd741a46452bbfbec3a38b11a122f766f057/Verse/ThingMaker.cs

12 13 22 Modding Tutorials/Weapons Guns - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Weapons_Guns

14 RIMMSqol :: Discussions - RimWorld - Steam Community
https://steamcommunity.com/workshop/filedetails/discussion/1084452457/2797251375474390875/?ctp=3

15 16 17 18 19 20 21 27 28 29 PawnComponentsUtility.cs
https://github.com/josh-m/RW-Decompile/blob/d5bbfd741a46452bbfbec3a38b11a122f766f057/RimWorld/PawnComponentsUtility.cs

24 25 26 Modding Tutorials/DefModExtension - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/DefModExtension