ChatGPT

GM 工具开发说明文档

1. 工具目标与用途

GM 工具是用于游戏服务端管理的桌面应用，旨在为游戏运营和管理人员提供便捷的管理界面和操作手段。其主要目标和用途包括：

- **游戏管理：** 提供对游戏数据和玩家数据的查询、监控和修改功能，包括玩家账户、游戏内道具、资产、活动等管理。GM 可以通过图形界面完成原本需要直接操作数据库或后台命令的任务，提高效率并降低出错风险。

- **多平台支持：** 工具初期以桌面端形式提供（基于 Tauri + React + Tailwind 构建跨平台桌面应用），支持 Windows/Linux/Mac 等平台运行。未来计划拓展至移动端平台，方便随时随地进行游戏管理操作。

- **权限分级与安全控制：** 系统预设 **OWNER（所有者）**、**AGENT（代理管理员）**、**VIEWER（观察者）**三种角色权限等级，对应不同的功能访问权限：

- **OWNER：** 最高权限管理员，拥有所有功能的访问和操作权，包括管理其他 GM 账号和系统配置。
- **AGENT：** 常规GM或客服权限，可执行大部分玩家管理和运营操作，但可能受限制于某些敏感功能（例如权限配置、本身不能管理其他GM账号等）。
- **VIEWER：** 只读权限，只能查看数据，不能进行修改操作，适用于审核人员或实习GM等。

权限体系支持细粒度的功能点控制，每个功能页面、操作按钮、乃至具体数据字段都可以依据权限进行显示或禁用控制。通过角色绑定权限的方式，保证不同级别用户各司其职，避免越权操作。

- **高度可配置性：** **“功能点”和UI控制完全可配置**。也就是说，系统提供配置机制来定义哪些功能开启或关闭、哪些UI元素对哪些角色可见/可用。管理员可以通过配置文件或数据库表，灵活地增加新功能模块、调整权限策略，而无需修改代码。例如，可以新增一个功能点并赋予某些角色访问权，前端UI会自动识别配置变化呈现相应组件。

- **低侵入性设计：** 游戏服务端采用的是现有的 C++ 架构，我们在开发GM工具时**不对游戏服务器逻辑做修改**。GM工具通过访问游戏服务器所用的 MySQL 数据库来查询和变更数据，实现管理目的。服务端与GM工具松耦合：游戏服务器继续按照原有方式运行，GM工具仅通过安全受控的数据库读写或调用现有存储过程来影响游戏数据。例如，GM工具执行给玩家发放物品时，本质是插入一条物品记录到数据库或调用已有的发奖存储过程，从而让游戏服务器感知变化。

- **审计与追踪：** 内置操作审计日志功能。所有通过GM工具进行的关键操作都会记录日志，包括操作者、时间、目标玩家/数据、具体变更内容等，以备日后查询审计。这对于追踪问题、责任划分和保障安全非常重要。

综上，GM 工具旨在提供一个**安全、方便、直观**的游戏运营后台，提高管理效率，减少人工出错，并确保每个操作都有据可查。

2. 项目结构与技术栈

本项目采用前后端分离的架构，利用现代技术栈以确保性能和开发效率。总体结构如下：

- **前端：** 使用 **Tauri + React + Tailwind CSS** 构建。Tauri 提供桌面端容器，使我们的React前端可以打包成跨平台桌面应用，同时占用资源较Electron更低。React 负责构建丰富的用户界面，Tailwind CSS 用于快速样式开发和保持UI的一致性。前端主要实现界面展示、表单交互、权限判断，以及通过HTTP请求与后端通信。

- **后端：** 使用 **.NET 8 Minimal API** 构建RESTful服务。Minimal API 模式让我们以简洁的方式定义HTTP接口，无需样板代码。后端主要职责是处理业务逻辑和与数据库交互：

- 通过 MySQL 驱动（例如使用 Entity Framework Core 或 Dapper 等）访问游戏数据库和GM管理数据库。
- 实现各功能模块对应的REST接口（如玩家数据查询、发放物品、封禁玩家等），封装成统一的JSON响应格式。

- 执行权限校验和审计日志记录（例如在执行修改类操作前验证调用者权限，在操作后写入审计日志）。

- **数据库：** 采用 **MySQL**（或兼容的分布式数据库）存储游戏数据和GM工具数据。按照业务职责分为三个数据库：

- **玩家库（player）**：游戏服务器的玩家游戏数据（如角色属性、库存等）。
- **用户库（user）**：游戏服务器的用户账号数据（如登录帐号信息、账号状态等）。

- **GM管理库（gm_admin）**：GM工具自身的数据（如GM账号、角色权限配置、操作日志等）。

- **通信方式：** 前端通过 HTTP/HTTPS 请求调用后端提供的 REST API 接口。接口遵循统一规范：使用 JSON 作为数据格式，采用典型的 RESTful 风格路径和方法（GET用于查询，POST/PUT用于修改/新增，DELETE用于删除等）。**接口返回格式**统一为 JSON 对象，包含 `code`、`msg`、`data` 三部分，例如：

  ```
  {
   "code": 0,
   "msg": "OK",
   "data": { ... }
  }
  ```

  其中：

- `code` 为业务状态码，0表示成功，非0表示出错（错误码涵义另行约定）。
- `msg` 为对本次结果的描述信息，例如 `"OK"` 或错误提示字符串。
- `data` 为实际返回的数据内容（对象或数组）。出错情况下可以为空或提供错误详情。

前端根据 `code` 判断操作是否成功，并使用 `msg` 提示用户必要的信息。这样保持所有接口风格一致，便于前后端约定和维护。

- **模块划分：** 项目代码按功能模块划分，便于协作和扩展。主要模块包括：

- **Auth（认证模块）：** 处理GM工具的登录认证、登出、密码管理，保障只有授权的GM账号可访问系统。前端有登录页面，后端提供登录验证接口（验证GM账号用户名/密码），采用JWT或Session方式维持会话状态。
- **RBAC 权限模块：** 基于角色的权限控制模块。包含后台对GM角色及权限点的配置管理接口，以及前端在界面元素渲染时的权限判断逻辑。确保用户只有在拥有相应权限时才能看到或操作相关功能。
- **玩家管理模块：** 提供玩家信息的查询和管理，包括玩家基本资料查看、编辑（如调整玩家等级、金币等属性）、封禁/解封账号、查询玩家的登录状态等。
- **道具管理模块：** 包括玩家**背包管理**（查看玩家持有的道具/物品列表，增删改道具），**道具发放**（向指定玩家发送道具或奖励），以及**道具列表**（查看游戏内所有道具的静态列表信息，如道具ID、名称、类型，用于查询或下拉选择）。
- **赛车管理模块：** 针对游戏中“赛车”类资产的管理（如果游戏包含赛车玩法）。提供查看玩家拥有的赛车列表、赛车属性，以及增加/移除赛车等操作。
- **贵族/VIP 管理模块：** 管理玩家的 VIP 等级或贵族特权信息。提供查询玩家VIP等级、经验值、贵族有效期等，并可根据权限调整玩家的VIP状态（如授予VIP等级，调整VIP积分）。
- **签到管理模块：** 查看和修改玩家每日签到状态。包括查询玩家签到记录、手动补签/重置签到等功能，或全局签到活动配置的查看（例如当前月份的签到奖励配置）。
- **监控与运维模块：** 提供游戏运行状态的监控看板和服务器控制功能。监控内容可能包括当前在线玩家数、服务器CPU/内存利用率、网络延迟、服务器状态等，通常以仪表盘形式呈现。运维控制功能包括服务器启停、重启、公告发布等操作接口（高危操作仅限OWNER）。

- **审计日志模块：** 记录并提供查询GM操作日志的功能。所有涉及数据变更的操作都会在后台写入日志表。前端提供日志查看页面，支持按时间、操作者、操作类型、目标玩家等筛选，方便审计追踪。

- **项目目录示例：**（简要展示可能的目录结构）

```
gmtool-backend/                 后端 .NET 8 项目根目录
  Program.cs                    启动和路由定义（Minimal API注册各模块路由）
  Controllers/                  （如果使用控制器风格则有此文件夹）
  Modules/
    AuthModule.cs               认证模块相关接口定义
    PlayerModule.cs             玩家管理模块接口定义
    ItemModule.cs               道具管理模块接口定义
    ...（其他模块）
  Services/                     后端服务层（如数据库访问、权限校验、审计记录等封装）
  Models/                       数据模型定义（如数据库实体或DTO）
  appsettings.json              数据库连接字符串、配置等
gmtool-frontend/                前端 Tauri+React 项目根目录
  src/
    pages/                      前端页面组件
      Login.jsx                 登录页
      PlayerProfile.jsx         玩家资料页
      Inventory.jsx             玩家背包页
      Cars.jsx                  玩家赛车页
      Vip.jsx                   贵族VIP页
      SignIn.jsx                签到管理页
      SendItem.jsx              发放物品页
      ItemList.jsx              道具列表页
      Dashboard.jsx             监控仪表盘页
      Permissions.jsx           权限设置页
    components/                 公用组件（表格、表单控件等）
    utils/                      工具类（如权限判断、API请求封装等）
```

```
  schema/                 （如果采用UI Schema配置，可存放各页面的schema定义）
tauri.conf.json           Tauri 配置文件（窗口大小、名称等）
package.json              前端依赖
```

（以上仅为示例，实际结构可根据需要调整）

- **开发与部署：**

- 开发阶段可使用 Node.js 启动 React 开发服务器以及 .NET 运行后端，两者通过localhost接口联调。Tauri开发模式可以加载本地React服务器方便热重载。
- 部署时，通过 Tauri 将前端打包为桌面应用，后端可选择集成在应用中或部署为独立的网络服务。若 GM 工具仅在公司内部网使用，后端Minimal API可部署在内网服务器，前端应用通过配置访问对应内网API地址。未来移动端支持时，可将前后端分离，移动App通过HTTPS访问同一套后端服务。

总之，本项目采用前后端解耦的模块化设计，在保证**高性能、高安全性**的同时，保持**良好的可扩展性**，能够随着游戏运营需求灵活添加新的功能模块和支持更多终端。

3. UI 页面与组件结构

前端界面采用 React 组件化的设计，将各功能模块划分为不同页面，每个页面由若干组件构成。这些页面和组件在设计时充分考虑了**权限控制**和**可配置性**，确保根据用户角色动态调整UI元素的显示和可用状态。下面详细列出各主要页面的结构、包含的字段和组件，以及权限控制方式。

3.1 玩家资料页面

页面描述： 玩家资料页面用于查看和管理单个玩家的基本信息。进入该页面通常需要提供玩家的唯一标识（例如玩家ID或用户名）进行查询。页面会展示玩家的核心属性，并提供部分管理操作（视权限开放）。

主要字段和显示信息：

- **玩家ID：** 玩家唯一标识符，通常为数据库中的主键ID。例如 `player_id` 。这一字段在页面中通常是**只读**显示，用于确认当前查看的玩家对象。
- **昵称：** 玩家在游戏中的昵称（角色名）。GM 可查看并确认玩家昵称是否正确。一般允许GM复制昵称用于搜索，但不允许直接修改昵称（昵称修改通常由游戏逻辑或专门接口处理）。
- **等级：** 玩家当前等级。如等级可以调整，GM（OWNER/AGENT）可在权限允许的情况下手动修改此值以测试游戏内容，但通常此操作应谨慎并记录日志。
- **经验值：** 当前等级下的经验进度。如有需要可编辑。
- **金币：** 玩家所持有的游戏币（软通货）数量。如玩家经济出现异常，可由GM调整金币数量。调整金币通常只开放给OWNER/AGENT。
- **钻石：** 玩家所持有的付费货币（硬通货）数量。GM可以增减钻石用于客服补偿或测试充值效果。**注意：**增加付费货币涉及营收，应严格控制权限和审计。
- **VIP等级：** 玩家贵族/会员等级（如果游戏有VIP系统）。这里可能只是显示，实际修改VIP等级或积分建议通过贵族页面操作，以保持业务逻辑清晰。

- **账号状态：** 玩家账号当前状态，例如“正常”、“封禁”、“禁言”等。GM可以查看状态并通过操作按钮改变状态（例如执行封禁/解封）。此字段可能根据user库记录（如封禁标记）实时显示。

- **注册时间：** 玩家创建账号或角色的时间。

- **最后登录：** 玩家最近一次登录游戏的时间。

- **所属服务器/渠道：** 如果游戏有多区多服或渠道概念，显示玩家所属的服务器编号、大区或渠道名称。这样GM在跨服管理时有依据（若本工具管理多服务器，这字段很重要）。

组件与布局： 玩家资料页通常包含一个信息卡或表单区域来排列上述字段，还有操作按钮区域： - 使用**表单(Form)**或**描述列表(List)**组件布局字段标签和值，每个字段占一行或网格布局。 - 关键属性（昵称、等级、VIP等）可以使用只读的文本组件，必要时可点击“编辑”按钮切换成输入框使GM修改值。 - 针对布尔状态（如账号封禁）可用**开关(Switch)**或**下拉选择(Select)**组件表示。例如“账号状态”字段可以是下拉选择正常/封禁，只有有权限的GM才能操作切换。 - 提供**操作按钮组**，例如： - **“保存修改”**：当GM更改了可编辑字段后提交。只有在Owner或Agent并且有编辑权限时按钮可点击；Viewer角色看不到或不可点。 - **“封停账户”/“解封账户”**：根据账号当前状态显示封停或解封操作按钮。点击后调用封禁/解禁接口。该按钮应仅对Owner或具有封禁权限的Agent可见。操作需二次确认（Modal 弹窗确认），且成功后刷新状态显示。 - **“重置密码”**（视需求）：如果支持GM重置玩家登录密码，则提供该按钮（仅Owner可见），点击弹出确认并执行密码重置流程。 - **“刷新”**：刷新玩家数据的按钮，使GM能够重新从数据库拉取最新数据，以防止过期信息。（所有有查看权限的角色都可用）

权限控制： - **页面访问：** 具备“玩家查看”权限（例如 Viewer以上角色）才能进入玩家资料页面。没有该权限的账号即使通过URL路由也应被重定向或看到无权限提示。 - **字段编辑：** 默认所有字段均只读显示；只有在用户具备相应“玩家编辑”权限时，特定字段才显示为可编辑状态。例如，Agent可以编辑金币钻石，但Viewer永远看不到编辑控件。 - **操作按钮：** 逐一绑定权限： - 保存修改按钮需要“玩家编辑”权限，否则不可点击或隐藏。 - 封停/解封按钮需要“封禁玩家”权限，否则对Agent隐藏或禁用（Viewer绝不可见，Agent如无权限也不可见）。 - 重置密码按钮可能需要“重置玩家密码”权限，仅Owner有，此按钮对其他角色完全隐藏。 - 权限判断由前端在渲染时通过当前登录GM的权限列表决定，同时后端在执行相关API时也会再次校验权限（双重校验）。UI上通过简单的条件渲染或封装的<Permission>组件来包裹相应元素：

```
{ hasPerm('PLAYER_BAN') &&
  <Button onClick={banPlayer} className="btn-danger">封停账户</Button>
}
```

如上，只有当`hasPerm('PLAYER_BAN')`为true时才会渲染封停按钮。

3.2 背包页面

页面描述： 背包页面用于查看指定玩家的物品库存（道具/背包物品）。GM 可通过此页面了解玩家拥有哪些道具，以及执行增删改的管理操作（如补发缺失道具、回收违规道具）。

主要字段与显示： 背包通常以**表格(table)**形式展示，包含以下列： - **道具ID：** 物品的唯一标识符。通常为整数ID。 - **道具名称：** 物品的名称。例如“强化石”、“金币”、“体力药水”等。便于GM识别道具。 - **数量：** 玩家当前持有该道具的数量。 - **类型：** 道具类别（如果有分类，如消耗品、装备、材料等）。可帮助GM过滤查看。 - **有效期：** 若道具有有效期或过期时间，该列显示具体日期或“永久”。没有有效期则显示“-”。 - **绑定状态：** 一些游戏道具有绑定/非绑定之分（不可交易等），此列指示该属性（可选）。

组件与布局： - 页面上方通常提供**玩家标识的输入或选择**组件（如一个搜索框或下拉）让GM先指定要查看哪个玩家的背包。如果从玩家资料页导航过来，可直接带入玩家ID并显示该玩家背包。 - 背包道具列表使用**表格组件**展示。支持按照道具名称或类型筛选搜索（提供表格自带的过滤功能或在顶部提供搜索框）。 - 每行物品后可带有**操作列**： - **“修改”**：点击可编辑该行道具数量（弹出对话框或行内编辑）。GM可以增加或减少玩家持有的数量。减少到0相当于删除该道具。 - **“删除”**：直接从玩家背包中移除该道具条目（从数据库删除此记录）。需要确认弹窗。 - 也可以在表格顶部或底部提供**批量操作按钮**： - **“添加道具”**：打开一个表单对话框，允许GM选择一个道具并输入数量后添加给玩家（相当于发放道具给该玩家）。 - **“一键清空”**：在特殊情况下清空玩家背包（危险操作，一般仅Owner可见）。

权限控制： - **页面访问：** 需要“背包查看”权限。Viewer具备查看权限时可以进入此页面查看列表。 - **添加/修改/删除操作：** 需要“背包修改”或更细粒度的权限。通常Owner和Agent具备，Viewer不具备。这具体可细分为： - 添加道具权限（ITEM_ADD）：无此权限则隐藏“添加道具”按钮。 - 修改数量权限（ITEM_EDIT）：无此权限则不渲染每行的修改按钮或禁止输入。 - 删除道具权限（ITEM_DELETE）：无权限则隐藏删除按钮。 - **数据范围限制：** 某些GM账号可能只被允许查看自己负责的服务器或渠道的玩家背包。系统可以在后台依据GM账号配置筛选玩家数据范围（这一点如有需要，可在查询接口根据GM的`allowed_server_id`等条件加上过滤）。

组件示例： - 玩家选择组件：`<PlayerSelector onSelect={loadInventory} />` 输入玩家ID或昵称自动提示，选择后调用`loadInventory`加载表格数据。 - 表格组件：使用React中的表格库或自定义`<Table dataSource={items} columns={columns} />`。定义columns时，如果当前用户没有ITEM_DELETE权限，可动态不包括删除这一列。 - 编辑操作可以用一个Modal表单：`<EditItemModal visible={editVisible} item={currentItem} onSubmit={saveItem}/>`。

UI Schema 结构： 背包页面也可以用UI Schema描述。例如每列对应一个 schema 字段定义，包含`visibleFor`属性控制权限。比如：

```
{
 "page": "inventory",
 "playerSelector": true,
 "columns": [
  { "title": "道具ID", "field": "item_id" },
  { "title": "道具名称", "field": "name" },
  { "title": "数量", "field": "quantity", "editable": false },
  { "title": "类型", "field": "type" },
  { "title": "有效期", "field": "expire_date" },
  { "title": "绑定", "field": "bound" },
  { "title": "操作", "actions": [
     { "label": "修改", "action": "editItem", "visibleFor": ["OWNER","AGENT"] },
     { "label": "删除", "action": "deleteItem", "visibleFor": ["OWNER"] }
   ]
  }
 ]
}
```

上述 schema 表示：只有 OWNER/AGENT 会看到“修改”按钮，只有 OWNER 看到“删除”按钮。前端渲染表格时会依据当前用户角色过滤不应显示的操作。

3.3 赛车页面

页面描述： 赛车页面用于管理玩家拥有的赛车（或载具）数据。如果游戏包含赛车或载具系统，玩家可能拥有多辆赛车，每辆赛车有独立属性。GM工具应允许查看玩家全部赛车列表，并对赛车进行调整（如赠送或移除赛车，调整赛车属性等）。

主要字段与显示： 类似背包，以表格形式罗列玩家的赛车： - **赛车ID：** 赛车的唯一标识（可能对应游戏内赛车的模板ID或实例ID）。 - **赛车名称：** 直观显示赛车类型的名称（如“闪电跑车”、“越野摩托”）。 - **星级/品质：** 赛车品质或改装星级（如果有培养系统）。 - **当前等级：** 赛车强化或改装等级。 - **最高速度：** 赛车关键属性（示例），可列出若干重要性能指标供参考。 - **拥有时长：** 如果赛车有获取时间或过期时间，显示相关日期。 - **状态：** 赛车状态（例如是否已锁定、正在使用等）。

组件与布局： - **玩家选择**：同背包页面，需要先选择玩家。 - **赛车列表表格**：显示上述字段列。 - **操作**： - **“添加赛车”**：按钮/表单，允许GM赠送一辆赛车给玩家。需要提供赛车ID/类型选择和初始属性设置（如默认等级）。 - **“删除赛车”**：从玩家账户移除某辆赛车（回收）。需谨慎操作，通常仅Owner可见此按钮。 - **“调整属性”**：针对已拥有的赛车，支持修改其属性（如星级、等级）。实现方式可以是点击某行赛车打开详情编辑窗口，让GM修改属性后保存。也可细化为若干操作按钮，比如“升级+1星”、“降级”等快捷按钮（视需求）。

权限控制： - **页面访问：** 需要“赛车查看”权限。 - **操作权限：** 细分为“赛车添加”、“赛车删除”、“赛车编辑”等权限点。通常： - Agent具备添加赛车和调整属性权限（用于客服补偿或纠错），Viewer没有任何修改权限。 - 删除赛车（永久移除）可能视为高风险操作，仅Owner允许。 - **UI控制：** 前端根据权限渲染操作列。例如：

```
columns.push({
 title: '操作',
 render: (car) => (
  <>
   { hasPerm('CAR_EDIT') && <a onClick={()=>editCar(car)}>调整属性</a> }
   { hasPerm('CAR_DELETE') && <a onClick={()=>removeCar(car)}>删除</a> }
  </>
 )
});
```

如上，不具备CAR_DELETE权限的GM看不到“删除”链接。

备注： 如果赛车较为复杂（比如有独立的实例ID、关联多个子表数据），GM工具可能需要在点击某辆赛车时展示更多详情。可以使用**抽屉(Drawer)**或**弹窗(Modal)**形式展现赛车详情（包括附加属性、改装件等），并在其中提供编辑保存功能。Viewer可查看详情但不能修改；Agent/Owner可进行修改，修改提交后通过API更新数据库并记录日志。

3.4 贵族页面 (VIP 管理)

页面描述： 贵族页面用于查看和管理玩家的 VIP 特权或贵族等级信息。许多游戏都有VIP系统，玩家通过充值累积积分升级VIP等级，享受额外福利。GM 工具应该让管理员查看玩家VIP状态，必要时进行调整（例如补偿VIP经验、直接提升VIP等级用于测试）。

主要字段与显示： - **当前 VIP 等级：** 玩家目前的VIP级别（例如 0-10级）。以徽章或数字显示。 - **当前 VIP 积分/经验：** 玩家已累积的VIP经验值。通常升级VIP需要一定经验，此值记录累积充值额度。 - **下一等级所需经验：** （只读）计算并显示升至下一级还需多少经验，方便GM告知玩家或进行判断。 - **VIP 有效期：** 如果VIP特权是限时的（有的游戏VIP按月购买），则显示到期日期；若为永久累积制VIP则不适用。 - **VIP 特权列表：** 可选显示玩家当前VIP等级享受的主要特权说明（如每日礼包、更高上限等），此为静态信息协助GM理解玩家权益。该部分一般不可编辑。

组件与布局： - 以**卡片(Card)**或表单样式展示上述信息。VIP等级和图标可以比较醒目放置。 - 如果GM有编辑权限，提供**表单控件**： - VIP经验值字段可切换为可编辑的数字输入框，让GM手动增加/减少经验（例如为玩家补加充值积分）。 - VIP等级字段一般不直接编辑（因为等级由经验累计计算）。然而，为了测试方便，可以在特殊权限下允许GM直接提升或降低玩家VIP等级（这可能需要同步调整经验值）。可以实现为**按钮**：“直升至 X 级”或“降级”，或者干脆提供一个VIP等级下拉选择，通过后台逻辑计算应设置的经验。 - **操作按钮：** - **“调整VIP经验”**：当GM输入新的经验值后，点击“保存”将修改玩家的VIP经验。服务器应重新计算等级并更新。

- **“赠送VIP等级”**：如果需要，可以有一个操作直接把玩家VIP升一级或降一级，用于活动赠送。点击后后台会按规则增加对应的经验值并更新等级。 - **“刷新”**：重新加载玩家最新VIP信息（与玩家资料类似）。

权限控制： - 查看VIP信息需要“VIP查看”权限。Viewer一般可以查看。 - 修改VIP需要“VIP修改”权限，通常仅开放给高级别GM： - Owner：可以修改VIP经验或等级。 - Agent：视运营策略决定是否允许，一般不允许直接改VIP等级，但可能允许补偿少量VIP经验（例如纠正充值记录错误）。这可通过更细权限“VIP经验调整”来控制，如果Agent有则开放经验字段编辑，没有则经验字段也是只读。 - Viewer：没有编辑权限。 - UI 控制体现为：无“VIP修改”权限则所有输入框禁用、调整按钮隐藏，仅呈现纯信息；有权限则启用相应控件。 - 例如：

```
<InputNumber value={vipExp} disabled={!hasPerm('VIP_MODIFY_EXP')} />
{ hasPerm('VIP_MODIFY_EXP') && <Button onClick={saveVipExp}>保存</Button> }
```

仅当有权限时输入框可用且显示保存按钮。

注意： VIP相关调整需极其谨慎，因为VIP往往和玩家充值金额挂钩，修改等同影响玩家付费记录。因此系统应在后台对VIP修改操作进行**特别审计**，记录修改前后值、原因等，并可能要求输入二级密码或额外确认（可以在UI上实现弹窗要求再次输入管理员密码确认）。

3.5 签到页面

页面描述： 签到页面用于管理玩家的每日签到数据。许多游戏每天登录可签到领取奖励，GM工具需要查看某玩家的签到进度，或帮玩家补签、重置签到状态等。也可能用于查看全服签到活动配置，但本说明倾向于玩家个人的签到数据管理。

主要字段与显示： - **当月签到天数：** 玩家本月已签到的总天数。 - **连续签到天数：** 玩家连续不间断签到的天数。 - **上次签到日期：** 玩家最近一次签到的日期。 - **缺勤天数：** 本月漏签的天数（如果有补签道具或机会，GM可据此判断是否给予补签）。 - **签到日历：** 可视化显示当前月份每天签到情况的组件。例如一个日历网格，标记已签到的日期（✔）和未签到的日期（✖）。方便直观了解玩家本月签到记录。对于未签到的过去日期，可允许GM点击执行补签。 - **签到奖励状态：** 若签到奖励有阶段性（如累计签到7天领奖），显示玩家是否已领取相应奖励。

组件与布局： - **日历组件(Calendar)**：中间主体可以使用一个日历控件或自定义网格，显示月份天数1~31，每天格子上用不同颜色或图标表示签到/未签状态。当前日期突出显示。 - **详情区域**：日历下方列出详细信息，例如上述连续天数、上次签到等统计。 - **操作按钮：** - **“补签”**：允许GM将玩家某日标记为已签到（通常选中未签到的日期点击“补签”）。可以一次补签一天，也可以批量补签多天（根据权限可能开放）。补签通常需要消耗补签道具或特殊批准，因此要记录原因。 - **“重置签到”**：将玩家当前月的签到记录清零/重置。这一般仅用于测试或纠错（如玩家签到数据异常），属于高权限操作，需Owner审批。 - **“发放累计奖励”**：如果玩家因漏签错过了累计签到的奖励，GM可以手动触发发放。例如玩家连续签到30天可得大奖，但因系统BUG未发，GM可在此强制发放。该操作应调用发奖接口或标记奖励已领取。

权限控制： - **页面访问：** 需要“签到查看”权限。 - **补签操作：** 需要“签到补签”权限。通常Agent有权执行玩家要求的补签（前提可能玩家提供补签道具等），Viewer无权。 - **重置操作：** 需要“签到重置”或更高权限，仅Owner可。 - **发奖励：** 需要“签到奖励发放”权限，可给予Agent。 - **UI 表现：** 没有补签权限时，日历中的未签日期点击应无效或者不提供点击事件，有权限则点击弹出确认补签对话框（可选填写补签原因）。没有重置权限则重置按钮不显示。 - **例如：**

```
<Button onClick={resetSignIn} disabled={!hasPerm('SIGN_RESET')}>重置签到</Button>
```

如果当前用户无SIGN_RESET权限，该按钮直接隐藏或禁用。

实现细节： 当GM补签或重置后，后台需要更新玩家签到记录表。考虑游戏服务可能有自己签到逻辑，GM工具通过直接修改数据库需要谨慎，例如更新`player_signin`表相应字段，并可能插入一条“人工补签”记录供后台审计。UI上的日历则在操作成功后即时更新显示。

3.6 发物品页面

页面描述： 发物品页面提供一个便利界面，允许GM将道具、货币或礼包发送给指定玩家。常用于客服补偿、活动奖励补发，支持一次发送一件或多件道具到单个玩家。此页面是GM日常使用频率较高的功能之一。

主要字段与表单项： - **目标玩家：** 输入或选择要发送物品的玩家。通常提供一个玩家搜索框（可以根据玩家ID、昵称搜索），确保准确选定目标。字段类型：文本输入 + 自动完成下拉 或 明确的玩家ID输入框。 - **道具选择：** 选择将要发送的道具。可通过下拉框列出常用道具或提供道具ID输入。理想情况下，这个组件联动道具列表数据源，允许按名称关键词检索道具ID。字段类型：下拉选择或自定义**道具选择器**组件。 - **数量：** 发送道具的数量。字段类型：数字输入框，需限定最小1，最大值可依据道具类型或GM权限设定上限（比如一次最多发100个，以防误操作）。 - **附加信息：** 一些实现中，发送物品会通过游戏内邮件系统送达，可能需要填写**邮件标题**和**邮件内容**字段。如果直接添加到背包则不需要邮件信息。本工具可简化为自动使用默认邮件模板，也可以让GM填写一段理由作为邮件内容或日志备注。 - **发送原因：** （可选）文字描述此次发放的原因或备注。虽然不一定发送给玩家，但应记录进审计日志。例如“补偿活动遗漏奖励”或“客服处理举报奖励”。

组件与布局： - 使用**表单(Form)**布局上述字段，每个字段配以清晰的标签和必要的说明： - 玩家字段如果用搜索框，当输入部分昵称时弹出下拉候选列表供选择。 - 道具字段可以是下拉列表（如果道具种类不多）或者一个对话框式选择器（点击后弹出道具列表让GM筛选后选择）。 - 数量字段紧随道具字段之后，输入框并附带单位（如“个”）。 - 邮件内容/理由可以用**多行文本(TextArea)**。 - 页面下方或右下提供**“发送”**主按钮，点击后提交表单。还可以有**“重置”**按钮以清空表单方便重新填写。 - 为防止误操作，点击发送应有二次确认对话框（尤其是数量较大时弹警告：“确认发送X个[道具名称]给玩家Y吗？”）。

权限控制： - **页面访问：** 需要“发放道具”权限（ITEM_SEND）。没有此权限的GM不会看到此菜单页面。 - **可发送道具范围：** 某些GM可能只允许发放特定类型道具或有数量限制。基础实现中不细分此权限，但可扩展：例如权限里关联可发放道具的“道具白名单”或最大数量。如果有此配置，前端在道具选择组件或数量输入上应进行相应限制（如Agent不能发送稀有道具或数量上限10）。 - **UI控制：** - 无权限则菜单入口不出现。 - 具有权限则全部表单字段可见。Viewer绝对不可见该页面。 - 如果实现道具发送限制，可在UI Schema配置中体现，例如：

```
{
 "field": "item_id",
 "component": "ItemSelector",
 "props": { "allowedCategories": ["consumable","currency"] }
}
```

表示某角色的UI只允许选择消耗品或货币类别的物品。 - **操作审计：** 点击发送后，无论成功与否都应该记录操作日志。成功则日志包含发送物品详情；失败（如玩家不存在）也记录尝试行为。前端不负责审计，但可确保把“原因”字段传给后端日志用途。

示例： 发物品页面的UI Schema定义示例：

```
{
 "page": "send_item",
```

```
  "title": "发放物品",
  "fields": [
    { "label": "玩家ID或昵称", "field": "player_id", "component": "PlayerSelector", "required": true },
    { "label": "道具", "field": "item_id", "component": "ItemSelector", "required": true },
    { "label": "数量", "field": "quantity", "component": "NumberInput", "props": { "min": 1, "max":
1000 }, "required": true },
    { "label": "邮件内容/备注", "field": "mail_msg", "component": "TextArea", "props": { "placeholder":
"可填写发送原因或附加说明" } }
  ],
  "actions": [
    { "type": "submit", "label": "发送", "primary": true, "confirm": "确认将道具发送给玩家吗？",
"permission": "ITEM_SEND" }
  ]
}
```

上述 schema 指定表单字段及属性，以及一个提交操作按钮（只有拥有ITEM_SEND权限者可以触发）。

3.7 道具列表页面

页面描述： 道具列表页面展示游戏内所有道具的静态信息数据库，让GM查询道具详情，辅助其他操作（如发放物品时需要知道道具ID和名称的对应关系）。此页面相当于游戏道具词典，也可用来管理道具配置（视权限决定是否允许编辑）。

主要字段与显示： 道具列表通常包括： - **道具ID：** 唯一标识。 - **道具名称：** 道具的名字。 - **类型：** 道具类型/分类（装备、消耗品、材料、货币等）。 - **稀有度：** 道具品质（普通、稀有、史诗等，如果有此概念）。 - **堆叠上限：** 单个堆叠的最大数量（如金币可能无限，药水可能99等）。 - **描述：** 道具的文字描述或效果说明。 - **其他属性：** 例如是否可交易、是否绑定、基础价等，可按需要添加列。

组件与布局： - **搜索过滤栏：** 页面顶部提供搜索框，可按道具名称关键字搜索，或下拉筛选类型。 - **道具表格：** 列出道具记录。由于道具数目可能很多，需要支持分页加载或虚拟滚动。每页显示固定数量（如50条），可翻页浏览。 - **详情查看：** 点击某行可展开展示更详细的道具信息。可以通过扩展行或弹出侧边栏的方式显示，包括道具图标、大图描述等。 - **编辑功能（如果开放）：** 若GM需要修改道具配置（一般仅限测试或运营调整，比如修改道具描述文字、开关某道具投放），可以在详情视图中提供可编辑项并保存。大多数情况下游戏道具配置是不会在运营平台修改的，因为这属于设计数据。但系统可以预留此能力给Owner角色。

权限控制： - **页面访问：** 需要“道具列表查看”权限。通常Viewer/Agent/Owner都可以查看此页面，因为不涉及敏感操作。 - **编辑权限：** 只有拥有“道具编辑”权限的角色（默认可能只有Owner）才能对道具进行修改。大部分GM账号不会有此权限。 - **UI控制：** 默认所有字段只读。如果 `hasPerm('ITEM_CONFIG_EDIT')`，则在详情面板中那些可调整的字段（如描述）显示为输入框和保存按钮，否则只是文本。 - **操作列：** 可选实现一个“编辑”按钮列，但仅当有编辑权限时渲染。点击进入编辑模式。没有权限则不出现该列。

实现说明： 由于道具列表数据通常来自游戏策划配置（可能是Excel导入或静态文件），通过GM工具修改可能不会实时作用于游戏服务器（除非服务器在运行时从数据库读取配置）。因此在技术上，可以提供修改并将更改记录到数据库配置表，但要注意和游戏实际使用的数据源保持同步。必要时，可结合**服务器控制模块**实现“热更”配置（通知游戏服务器重新加载道具表）。这些属于扩展功能，超出基本工具要求的可在后续扩展章节说明。

3.8 监控仪表盘页面

页面描述： 监控仪表盘页面汇总展示游戏服务器的实时运行状态和关键指标，帮助GM及时了解服务器健康状况和在线情况。它通常是一个由多个**数据面板（Card）**和**图表（Chart）**组成的综合页面。

主要内容和指标： - **在线玩家数：** 当前服务器实时在线的玩家数量。可能细分为总在线、各服务器/各渠道在线等。通常以一个突出的大数字显示，并附随一个曲线图展示最近一段时间在线人数变化趋势。 - **服务器CPU利用率：** 游戏服务器所在机器的CPU当前使用率（百分比）。可用进度条或仪表盘形式表示。最好包括实时数值及历史曲线。 - **内存使用率：** 服务器内存占用，同样用百分比和曲线表示。 - **网络延迟：** 服务器与监控节点之间的 ping 值或玩家平均延迟。如有分区，也可显示不同地区平均延迟。 - **服务器状态：** 例如“运行中”、“维护中”、“停止”等状态指示。如果有多个服务组件（网关、战斗服等），可以显示各自状态灯。 - **日志警报：** 如果服务器出现错误或警告日志，或者有重要事件（如异常崩溃）可以在仪表盘上以通知方式提醒GM。 - **其他业务指标：** 例如今日新增玩家数、今日充值金额（如有接入）、当前活动参与人数等，可以根据运营需要添加。

组件与布局： - **概览卡片：** 页面顶部以卡片形式平铺几个核心指标（在线数、新增数、充值等）。卡片内用大字体显示数值，小字注释指标名称，底部或侧边可有同比变化等信息。 - **折线/区域图：** 在线人数趋势、CPU/内存曲线等采用图表组件（如 Chart.js, ECharts）绘制。通常几个图表以网格布局放置，占据页面主要部分。 - **服务器列表/状态：** 如果监控多台服务器，可有一个服务器列表表格，每行显示服务器ID、名称、IP、在线人数、CPU、内存等，点击特定服务器行可以筛选/高亮对应图表。 - **刷新频率：** 前端可以每隔N秒自动刷新数据（通过定时调用后端API获取最新指标），或者提供“刷新”按钮手动刷新。频率高的指标（如在线数）可以设定定时刷新，比如每30秒刷新一次。 - **异常告警：** 可在页面底部或角落设置一个**日志列表/告警框**，显示最近的异常事件，如服务器报错日志摘要、玩家峰值预警等。

权限控制： - 一般**所有GM角色都允许访问监控页面**（运营需要了解在线情况）。因此“监控查看”权限应该赋予Viewer以上角色都具备。只有极特殊的只读角色才可能没有。 - **服务器控制**（重启/停机等危险操作）通常集成在此页或相关页面，但仅OWNER可执行。可以在监控页中放置控制按钮： - **“重启服务器”**按钮：仅OWNER可见。点击后请求后端执行服务器重启操作（可能调用脚本或设置维护标志等）。需要二次确认。 - **“发送公告”**按钮：Agent/Owner可见。用于发送全服公告广播，点击弹窗输入公告内容后发送。Viewer无权。 - **“进入维护模式”**开关：仅Owner。打开后服务器停止新玩家登录，现有玩家下线后进入维护状态。 - **UI控制：** 这些控制按钮用 `hasPerm('SERVER_CONTROL')` 等判断。例如：

```
{ hasPerm('SERVER_RESTART') && <Button onClick={restartServer}>重启服务器</Button> }
```

无权限则按钮不渲染。 - 查看类指标一般不限制，可见即可见。如果要细分权限，比如某GM只负责某台服务器的数据，可以在后端过滤数据或前端只呈现授权范围内服务器的指标（通过GM账号配置关联的服务器ID列表实现）。

数据获取： 仪表盘所显示的数据由后端汇总提供。后端可能通过以下途径获取： - 直接查询游戏数据库（如在线人数可能存在于缓存或专门的在线统计表，定时刷新）。 - 调用游戏服务器开放的状态接口（如果有HTTP/RPC接口提供内部状态）。 - 读取监控系统的数据，如Prometheus等（如果基础设施有监控，则GM后端充当一个代理去取数据）。 - 以上细节在本工具范围内可以简单实现：例如在数据库中维护一些统计表（游戏服务端每分钟写当前在线人数、CPU等指标到stats表），GM接口直接查最新记录即可。

3.9 权限设置页面

页面描述： 权限设置页面用于管理GM工具本身的用户、角色和权限分配。只有最高权限用户 (OWNER) 能访问此页面，用于创建/删除 GM 账号、分配角色，以及配置各角色拥有的权限点。其目的是实现**勾选式配置**的权限管理界面，使运维人员无需直接修改数据库就能调整权限。

主要功能和组件:

- **GM账号管理:**
- **GM账号列表:** 显示所有已有的GM用户账户。字段包括：账号ID、用户名、姓名（可选）、角色、最近登录时间、账号状态（正常/禁用）。
- **新增账号按钮:** 弹出表单创建新GM账号。字段：用户名、初始密码、选择角色、备注。点确认调用后端创建用户接口。
- **编辑账号:** 在账号列表每行提供编辑和删除操作。编辑可以修改该GM的角色、姓名、密码重置、启用禁用状态。删除用于移除GM账号（需二次确认）。
- **搜索过滤:** 按用户名筛选功能。
- **角色权限配置:**
- **角色列表:** 显示当前系统已有的角色（如 OWNER, AGENT, VIEWER，以及将来可能新增的角色）。字段：角色ID、名称、描述、拥有用户数等。
- **权限点矩阵:** 以角色为单位展示权限点勾选情况。例如采用表格形式，左侧列出所有权限项，右侧各列对应角色，在交叉单元以**Checkbox**形式表示该角色是否拥有该权限。这样直观呈现和修改。
  - 权限项可以按模块分类分组显示（比如玩家管理、道具管理、监控、系统管理等）。
  - GM可通过勾选/取消选项来赋予或收回某角色的权限，提交保存后生效。
- **编辑角色:** 允许新增自定义角色或修改角色名称、描述。新增角色之后出现在矩阵中可配置权限。
- **一键全选:** 对于某角色可能提供“全选所有权限”或“全不选”按钮，方便快速配置（Owner通常全选）。
- **权限点列表说明:** 为了方便管理者理解，各权限点可以有一个描述提示（Tooltip）。如悬停在权限名称上，显示该权限具体控制哪些页面或操作。例如“PLAYER_EDIT：允许修改玩家基本信息（等级、金币等）”。

权限控制: - 此页面原则上**仅OWNER角色**可以访问（需要权限例如`ADMIN_MANAGE`）。其他角色即使尝试访问路由，也应被阻止。 - 在实现上，可以在进入此页面时做权限判断，或干脆不在菜单中显示此页面给非Owner用户。 - 操作上，仅Owner能新增/删除GM账号、增删角色、勾选权限。Agent和Viewer无权访问也无权修改。 - 如果公司有不同超级管理员，可以赋予多个账号Owner角色。

审计要求: 权限变更属于高风险操作，应记录审计日志。例如: - 哪个GM在何时修改了角色X的哪些权限（列表）。 - 哪个GM创建了新账号Y，赋予了什么角色。 - 哪个GM删除/禁用了账号Z。 这些日志方便以后追查权限被错误修改的情况。

UI实现细节: - **账号列表**可以用表格+分页展示。编辑账号弹窗内密码字段应支持自动生成或手动输入。考虑安全，密码不以明文显示。 - **角色权限矩阵**实现可以考虑将权限项和角色用双重列表渲染Checkbox:

```
permissions.map(perm => (
 <div key={perm.code} className="perm-row">
  <span>{perm.name}</span>
  { roles.map(role => (
    <Checkbox
     key={role.id}
     checked={rolePermMatrix[role.id][perm.code]}
     onChange={(e)=>togglePerm(role.id, perm.code, e.target.checked)}
     disabled={!hasPerm('ADMIN_MANAGE')}
```

```
      />
    ))}
  </div>
))
```

仅当有ADMIN_MANAGE权限时Checkbox可操作，否则禁用。 - **UI Schema**也可用于描述权限配置界面的结构。例如：

```
{
  "page": "permissions",
  "sections": [
    {
      "title": "GM账号列表",
      "component": "AccountTable",
      "props": { "editable": true }
    },
    {
      "title": "角色权限配置",
      "component": "RolePermMatrix",
      "props": { "roles": [...], "permissions": [...] }
    }
  ]
}
```

这样通过schema可以动态增加权限项或角色列而无需调整前端代码，前提是前端组件能够根据传入数据渲染适应变化。

4. API 接口说明

后端提供一系列 RESTful API 接口以供前端调用，实现上述功能。这些接口以 `/api` 为基础路径，遵循统一的请求和响应格式。下面按照模块分类列出主要接口的路由、请求参数、响应结构，以及对应的权限与审计要求。

通用说明： 所有需要请求体的接口均使用 JSON 格式提交参数；响应统一为 JSON，结构包含 `code`, `msg`, `data`。调用需附带身份认证信息（如JWT Token或Session），后端会验证权限。

4.1 身份认证接口 (Auth)

- **登录接口** (`POST /api/auth/login`):
- **请求参数：** JSON 对象，包含：
  - `username`：字符串，GM账户用户名。
  - `password`：字符串，密码明文或经前端加密后的密文（视实现，简单起见明文传输经HTTPS保护）。
- **响应：**
  - 成功: `code=0`, `data` 包含认证信息，如 `{"token": "<JWT令牌>", "user": {"id": 1, "name": "admin", "role": "OWNER"}}`。`msg` 可为"登录成功"。
  - 失败: `code` 为相应错误码（例如 401），`msg` 提示错误原因（如"用户名或密码错误"）。
- **权限：** 任意用户可调用（无需登录）。成功后系统将建立会话。

- **审计：** 可记录登录日志（包括IP、时间）。但一般不计入审计日志表，只做安全日志用途。

- **登出接口** (`POST /api/auth/logout`):

- **请求参数：** 无（或在Header/Cookie中标识用户会话）。
- **响应：** 成功`code=0`表示已注销。
- **权限：** 登录用户自己操作，无额外权限要求。

- **审计：** 可不记录或简要记录登出事件。

- **获取当前用户信息** (`GET /api/auth/me`):

- **功能：** 返回当前登录GM用户的详细信息和权限列表，用于前端初始化权限状态。
- **响应示例：**

```
{
 "code": 0,
 "data": {
  "id": 5,
  "username": "agent1",
  "role": "AGENT",
  "permissions": ["PLAYER_VIEW","PLAYER_EDIT","ITEM_SEND", "..."]
 }
}
```

- **权限：** 登录用户通用。
- **审计：** 无需。

4.2 GM用户及权限管理接口 (Admin/RBAC)

这些接口用于权限设置页面，由OWNER调用。

- **获取GM用户列表** (`GET /api/admin/users`):
- **功能：** 列出GM工具的所有用户。
- **请求参数：** 可选查询参数`?role=`按角色过滤，`?keyword=`按用户名搜索。
- **响应：**

```
{
 "code": 0,
 "data": [
  { "id": 1, "username": "admin", "role": "OWNER", "lastLogin": "2025-09-01 12:00", "status": "active" },
  { "id": 2, "username": "gm_test", "role": "AGENT", "lastLogin": null, "status": "inactive" },
  ...
 ]
}
```

- **权限：** 需要ADMIN_MANAGE权限（仅Owner具备）。

- **审计：** 查询操作通常不审计。

- **创建GM用户** (`POST /api/admin/users`):

- **请求：** JSON: `{ "username": "newgm", "password": "P@ssw0rd", "role": "AGENT", "status": "active" }`
- **响应：** 成功返回新用户ID等信息。
- **权限：** ADMIN_MANAGE（Owner）。

- **审计：** 记录操作日志："Admin X 创建GM用户 newgm (角色AGENT)"。

- **编辑GM用户** (`PUT /api/admin/users/{id}`):

- **功能：** 修改指定GM用户的信息。如改角色、锁定/解锁账号或重置密码等。
- **请求：** JSON 包含可修改字段，例如：
  - `role`: (可选) 修改后的角色标识字符串。
  - `resetPassword`: (可选) 若存在且为true则后端重置该用户密码（通常随机新密码或固定初始密码并通知）。
  - `status`: (可选) "active"或"inactive"用于启用/禁用账户。
- **响应：** 成功或错误信息。重置密码后可在`data`返回新密码明文（也可通过别的安全渠道发送给用户）。
- **权限：** ADMIN_MANAGE。

- **审计：** 记录日志，细化内容如"Admin X 修改GM用户Y: 改角色->AGENT, 状态->inactive"。

- **删除GM用户** (`DELETE /api/admin/users/{id}`):

- **功能：** 删除指定GM账号（或视为永久禁用）。
- **权限：** ADMIN_MANAGE。

- **审计：** 记录日志，"Admin X 删除GM用户Y"。

- **获取角色列表和权限点配置** (`GET /api/admin/roles`):

- **功能：** 返回所有角色以及对应权限点列表，用于前端渲染矩阵。
- **响应：**

```
{
 "code": 0,
 "data": {
  "roles": [
   { "id": 1, "name": "OWNER" },
   { "id": 2, "name": "AGENT" },
   ...
  ],
  "permissions": [
   { "code": "PLAYER_VIEW", "desc": "查看玩家信息" },
   { "code": "PLAYER_EDIT", "desc": "修改玩家信息" },
   ...
  ],
  "rolePerms": [
   { "roleId": 1, "permCode": "PLAYER_VIEW" },
   { "roleId": 1, "permCode": "PLAYER_EDIT" },
```

```
    { "roleId": 2, "permCode": "PLAYER_VIEW" },
    ...
   ]
  }
}
```

- **权限：** ADMIN_MANAGE (仅Owner)。
- **审计：** 查询不记录。

- **更新角色权限** (`PUT /api/admin/roles/{roleId}/permissions`):

- **功能：** 批量更新某角色的权限分配。
- **请求：** JSON 示例：`{ "permissions": ["PLAYER_VIEW","PLAYER_EDIT","ITEM_SEND"] }` 表示将该角色的权限点设置为这几个（通常后端实现是先清空再添加传入的列表）。
- **响应：** 成功返回code=0。
- **权限：** ADMIN_MANAGE。
- **审计：** 记录日志：“Admin X 更新角色RoleName权限：新增N个，移除M个权限项”。

- **创建/删除角色** (`POST /api/admin/roles`, `DELETE /api/admin/roles/{roleId}`):

- **功能：** 新增自定义角色，或删除已有自定义角色（系统默认角色不建议删除）。
- **请求：** 新增时 JSON `{ "name": "TESTER", "desc": "测试专用角色" }`。删除时路径携带角色ID。
- **权限：** ADMIN_MANAGE。
- **审计：** 记录角色增删操作。

4.3 玩家管理接口

针对游戏玩家数据的查询和修改操作接口。除查看外，大多属于敏感操作，需要严格权限校验和审计日志。

- **查询玩家基本信息** (`GET /api/player/{playerId}`):
- **功能：** 获取指定玩家的基本资料。
- **响应：**

```
{
 "code": 0,
 "data": {
  "playerId": 1001,
  "nickname": "PlayerOne",
  "level": 35,
  "exp": 12345,
  "gold": 50000,
  "diamond": 100,
  "vipLevel": 3,
  "vipExp": 2500,
  "status": "正常",
  "lastLogin": "2025-09-18 08:30",
  "registerTime": "2025-01-10 16:05",
  "serverId": 1
```

```
  }
}
```

- **权限：** 需要 PLAYER_VIEW 权限。Viewer及以上可调用。
- **审计：** 纯查询不记录日志（可在安全日志记录查询频次）。

- **搜索玩家** (`GET /api/player/search`):

- **功能：** 按玩家昵称或帐号模糊搜索玩家。用于在没有ID时找到玩家ID。
- **请求参数：** `?name=部分昵称` 或 `?account=xxx` 等。
- **响应：** 列表包含匹配玩家的ID和昵称等基本信息，可能多条。
- **权限：** PLAYER_VIEW。
- **审计：** 不记录。

- **修改玩家基本属性** (`PUT /api/player/{playerId}`):

- **功能：** 编辑玩家资料中允许修改的字段（等级、金币、钻石等）。
- **请求：** JSON中仅包含需要修改的字段。例如：`{ "level": 36, "gold": 60000 }` 表示将等级改为36，金币改为60000。
- **响应：** 成功或失败信息。成功`code=0`，`msg="修改成功"`。
- **权限：** PLAYER_EDIT 或更细分如 LEVEL_EDIT, GOLD_EDIT 等。一般Agent以上允许。
- **审计：** **必须记录**。日志应包括操作者，目标玩家，修改前后的值。可以逐字段记录，例如：“GM用户X修改玩家1001：等级 35->36，金币 50000->60000”。

- **封禁/解封玩家** (`POST /api/player/{playerId}/ban` 或 `PUT /api/player/{playerId}/status`):

- **功能：** 改变玩家账号状态为封禁或恢复正常。
- 两种设计：
  - 单独的动作接口：`POST /api/player/{id}/ban` 封禁，`POST /api/player/{id}/unban` 解封。
  - 或统一用状态字段：`PUT /api/player/{id}/status`，请求`{ "status": "banned" }`或`{ "status": "active" }`。
- **权限：** PLAYER_BAN 权限，通常只有Owner和特定Agent有。
- **审计：** 记录日志：“GM X 封禁玩家1001，原因: YYY”。应要求GM提供封禁原因（前端弹窗输入）传到后台记录。

- **重置玩家密码** (`POST /api/player/{playerId}/reset_password`):

- **功能：** 重置玩家登录密码。
- **权限：** PLAYER_PASSWORD_RESET 权限，仅Owner。
- **审计：** 记录操作及后续处理（新密码如何通知玩家等）。

注：实际实现中，很多玩家管理修改操作也可以通过发物品（例如发放货币代替直接修改金币）或通过内部GM指令接口完成。但本工具直接操作数据库提供了简捷途径，必须配合严格日志审计。

4.4 道具和背包管理接口

- **查询玩家背包** (`GET /api/player/{playerId}/inventory`):
- **功能：** 获取玩家所有背包道具列表。
- **响应：**

```
{
 "code": 0,
 "data": [
  { "itemId": 1001, "name": "强化石", "type": "材料", "quantity": 5, "expire": null, "bound": true },
  { "itemId": 1002, "name": "体力药水", "type": "消耗品", "quantity": 10, "expire": "2025-12-31", "bound": false },
  ...
 ]
}
```

- **权限：** ITEM_VIEW 或 INVENTORY_VIEW。
- **审计：** 不记录。

- **背包添加道具** (`POST /api/player/{playerId}/inventory`):

- **功能：** 给玩家新增一道具。请求JSON示例：`{ "itemId": 1002, "quantity": 5, "reason": "客服补偿" }`。
- **响应：** 成功返回code=0。后台在数据库增加或更新一条玩家物品记录（如果已存在则累加数量）。
- **权限：** ITEM_ADD/ITEM_SEND 权限。Agent以上。
- **审计：** 记录：“GM X 添加道具给玩家Y：道具1002 数量5，原因:客服补偿”。

- **背包道具修改** (`PUT /api/player/{playerId}/inventory/{itemId}`):

- **功能：** 修改玩家已有某道具的数量或属性。
- **请求：** JSON如`{ "quantity": 0 }` 可以用于减少数量或清零。
- **权限：** ITEM_EDIT 权限。
- **审计：** 若数量减少或变更明显，也需记录日志：“GM X 修改玩家Y道具1002数量: 10 -> 0”。

- **背包道具删除** (`DELETE /api/player/{playerId}/inventory/{itemId}`):

- **功能：** 将该玩家拥有的某道具全部移除。
- **权限：** ITEM_DELETE 权限（高权限）。
- **审计：** 记录：“GM X 删除玩家Y 道具1002 数量原有N”。

- **发送道具** (`POST /api/items/send`):

- **功能：** 面向“发物品”页面的接口，一次发送道具（或礼包）给一个玩家。
- **请求：** `{ "playerId": 1001, "itemId": 9001, "quantity": 1, "mailMsg": "补偿礼包" }`
- **响应：** 成功后`code=0`。后台实现可能是调用`inventory`添加接口，或者插入一条游戏内邮件记录到邮件表（如果有邮件系统）。

- **权限：** ITEM_SEND 权限。
- **审计：** 记录：“GM X 发送道具9001x1给玩家1001，邮件内容:'补偿礼包'”。

- **查询道具列表** (`GET /api/items`):

- **功能：** 获取游戏全部道具定义列表，可分页。
- **请求参数：** 支持`?page=1&pageSize=50`，`?type=装备`等过滤。
- **响应：**

```
{ "code": 0, "data": { "items": [ { "itemId":1001, "name":"强化石", "type":"材料", ... }, ... ], "total": 200 } }
```

- **权限：** ITEM_VIEW（多数GM都有）。
- **审计：** 不记录。

- **获取单个道具详情** (`GET /api/items/{itemId}`):

- **功能：** 查看道具配置详细信息。
- **权限：** ITEM_VIEW。
- **审计：** 不记录。

- **修改道具配置** (`PUT /api/items/{itemId}`):

- **功能：** 修改某道具的配置属性（如描述、掉落开关等）。
- **权限：** ITEM_CONFIG_EDIT 权限，仅Owner。
- **审计：** 记录修改内容，通常需要重启或通知服务器更新。

4.5 赛车管理接口

- **查询玩家赛车** (`GET /api/player/{playerId}/cars`):
- **功能：** 列出玩家拥有的赛车。
- **响应：**

```
{
  "code": 0,
  "data": [
    { "carId": 201, "name": "闪电跑车", "star": 3, "level": 5, "speed": 320, "acquiredAt": "2025-08-01" },
    ...
  ]
}
```

- **权限：** CAR_VIEW。
- **审计：** 不记录。

- **添加赛车** (`POST /api/player/{playerId}/cars`):

- **请求：** `{ "carId": 202, "star": 1, "level": 1 }` 向玩家添加指定车型的新赛车，初始星级和等级由请求指定或默认。
- **权限：** CAR_ADD。
- **审计：** 记录：“GM X 添加赛车202(1星1级)给玩家Y”。

- **修改赛车属性** (`PUT /api/player/{playerId}/cars/{carId}`):

- **功能：** 修改玩家某辆赛车的属性。例如升级星级或等级。
- **请求：** `{ "star": 4, "level": 1 }` 升星或降星。
- **权限：** CAR_EDIT。
- **审计：** 记录变更：“GM X 修改玩家Y赛车202星级:3->4”。

- **删除赛车** (`DELETE /api/player/{playerId}/cars/{carId}`):

- **功能：** 移除玩家的一辆赛车。
- **权限：** CAR_DELETE（仅Owner）。
- **审计：** 记录：“GM X 移除玩家Y赛车202”。

4.6 签到管理接口

- **查询玩家签到信息** (`GET /api/player/{playerId}/signin`):
- **功能：** 获取玩家当前签到状态及记录。
- **响应：**

```
{
 "code": 0,
 "data": {
  "month": "2025-09",
  "daysChecked": 10,
  "consecutiveDays": 5,
  "lastSignDate": "2025-09-18",
  "missedDays": [ "2025-09-05", "2025-09-12" ],
  "rewardsClaimed": [7] // 已领取累计奖励，比如7天奖励已领
 }
}
```

- **权限：** SIGN_VIEW。

- **审计：** 不记录。

- **补签** (`POST /api/player/{playerId}/signin/makeup`):

- **功能：** 为玩家补签指定日期。
- **请求：** `{ "date": "2025-09-05" , "reason": "系统bug补签"}` 补签9月5日。
- **权限：** SIGN_MAKEUP（Agent/Owner）。
- **审计：** 记录：“GM X 补签 玩家Y 2025-09-05 (原因: 系统bug补签)”。

- **备注：** 后端将更新玩家签到记录，将该日标记为已签，并可能触发补发当日奖励（如果玩家未领取）。

- **重置签到** (`POST /api/player/{playerId}/signin/reset`):

- **功能：** 重置玩家当月签到数据。
- **权限：** SIGN_RESET（Owner）。

- **审计：** 记录：“GM X 重置 玩家Y 2025-09月 签到记录”。

- **发放累计签到奖励** (`POST /api/player/{playerId}/signin/reward`):

- **功能：** 手动发放玩家未领取的累计签到奖励。
- **请求：** `{ "dayCount": 30 }` 比如发放累计签到30天的奖励。
- **权限：** SIGN_REWARD_GRANT（Agent/Owner）。
- **审计：** 记录奖励发放操作。

4.7 监控与服务器控制接口

- **查询在线人数** (`GET /api/monitor/online`):
- **功能：** 获取当前在线玩家数（可能包含各区服细分）。
- **响应：** `{ "code":0, "data": { "total": 1200, "byServer": { "s1":500, "s2":700 } } }`
- **权限：** MONITOR_VIEW（一般所有GM有）。

- **审计：** 不记录。

- **查询服务器性能** (`GET /api/monitor/system`):

- **功能：** 获取服务器CPU、内存等当前性能数据。
- **响应：** `{ "code":0, "data": { "cpu": 55.3, "memory": 68.1, "uptime": "5 days 4:33" } }`
- **权限：** MONITOR_VIEW。

- **审计：** 不记录。

- **查询近期在线趋势** (`GET /api/monitor/online_history?hours=1`):

- **功能：** 获取最近1小时的在线人数曲线数据。
- **响应：** 数据点数组，供前端图表绘制。

- **权限：** MONITOR_VIEW。

- **发送全服公告** (`POST /api/server/broadcast`):

- **请求：** `{ "message": "服务器将于10分钟后停机维护，请悉知。" }`
- **功能：** 触发游戏服务器发送公告给在线玩家。
- **权限：** SERVER_BROADCAST，Agent及以上。

- **审计：** 记录：“GM X 发送公告: 服务器将于10分钟后维护...”。

- **进入/结束维护模式** (`POST /api/server/maintenance`):

- **请求：** `{ "enable": true }` 或 false 切换维护状态。
- **功能：** 通知游戏服务器进入维护模式（停止新登录，开始踢人）或恢复正常。
- **权限：** SERVER_CONTROL，Owner。

- **审计：** 记录：“GM X 启动维护模式”或“结束维护模式”。

- **重启服务器** (`POST /api/server/restart`):

- **功能：** 远程执行服务器重启。可能需要后端通过SSH连接服务器执行脚本，或者在数据库设标志让服务器自休眠重启。
- **权限：** SERVER_CONTROL (Owner)。
- **审计：** 记录：“GM X 执行服务器重启”。

  安全注意： 所有服务器控制类接口必须非常慎重，建议增加额外校验，例如要求附带特别的验证码或二次确认（前端已做确认弹窗，但后端也可要求一个`confirm`参数或者多因子认证）。

4.8 审计日志接口

- **查询操作日志** (`GET /api/audit/logs`):
- **功能：** 提供根据筛选条件查询GM操作审计日志。
- **请求参数：** 支持多种过滤：
  - `?user=gmUsername` 按操作者筛选
  - `?action=PLAYER_BAN` 按操作类型筛选
  - `?targetPlayer=1001` 按目标玩家ID筛选
  - `?dateStart=2025-09-01&dateEnd=2025-09-30` 按时间范围筛选
  - 分页参数：`page, pageSize`
- **响应：**

```
{
 "code": 0,
 "data": {
   "logs": [
     { "id": 123, "user": "admin", "action": "PLAYER_EDIT", "target": "player:1001", "detail":
"level 10->15", "time": "2025-09-18 10:00:00" },
     { "id": 124, "user": "agent1", "action": "ITEM_SEND", "target": "player:1002 item:9001",
"detail": "sent item 9001x1", "time": "2025-09-18 10:05:00" },
     ...
   ],
   "total": 57
 }
}
```

- **权限：** AUDIT_VIEW 权限。通常Owner和部分监督人员有，Agent/Viewer可能也有只读权限视公司规定。
- **审计：** 查询日志本身可不再记录审计（以免递归膨胀），但可以记录查看日志的行为在安全日志中。

5. 数据库结构说明

本章节描述数据库设计，包括**玩家库（player）**、**用户库（user）**、**GM管理库（gm_admin）**中的相关表结构、字段含义、表间关系以及索引优化建议。所有字段名以英文为准，并在括号中说明中文含义。

5.1 玩家库 (player)

玩家库存放游戏内角色/玩家的数据。主要表设计如下：

- **players（玩家基本信息表）**：每条记录代表一个玩家角色（如果一账号对应一个角色，可以等同于账号）。
- 字段：
  - `player_id` (玩家ID) **int**: 主键，自增或由系统生成的唯一玩家标识。
  - `user_id` (账号ID) **int**: 关联用户库中用户的ID。如果玩家和账号一一对应，可相同，否则此字段为外键引用 user.users 表。
  - `nickname` (昵称) **varchar(100)**: 玩家角色名。
  - `level` (等级) **int**: 玩家当前等级。
  - `exp` (经验) **bigint**: 当前等级累计经验值。
  - `gold` (金币) **bigint**: 游戏币数量。
  - `diamond` (钻石) **bigint**: 付费货币数量。
  - `vip_level` (VIP等级) **int**: 贵族/VIP 等级。
  - `vip_exp` (VIP经验) **int**: VIP积分/经验累计值。
  - `status` (账号状态) **tinyint**: 状态标识（0正常，1封禁，2禁言...可以用位或枚举）。
  - `register_time` (注册时间) **datetime**: 角色创建时间。
  - `last_login_time` (最后登录时间) **datetime**: 最近一次登录游戏的时间。
  - `server_id` (服务器ID) **int**: 所在游戏服务器/区编号（如适用多服架构）。
- **主键：** `player_id`（聚簇索引）。
- **索引：**
  - 唯一索引 `idx_nickname_unique` 在 `nickname` 字段（若昵称要求唯一）。
  - 普通索引 `idx_user_id` 在 `user_id` 字段（用于通过账号查找玩家）。
  - 索引 `idx_status` 在 `status`（便于统计或筛选异常状态玩家）。
  - 可以有索引 `idx_last_login` 用于按最近登录排序检索活跃玩家等（视业务需要）。
- **表关系：** `players.user_id` 外键关联 user库的 `users.user_id`，表示哪个账号创建了该角色。若一账号一个角色，可一对一；若一账号多角色，则 user.users -> player.players 是一对多关系。

- **player_items（玩家道具表）**：记录玩家背包/仓库中的道具。

- 字段：
  - `player_id` (玩家ID) **int**: 对应玩家。
  - `item_id` (道具ID) **int**: 道具标识（对应道具列表中的定义ID）。
  - `quantity` (数量) **int**: 持有数量。
  - `expire_time` (过期时间) **datetime**: 道具过期时间，如果为空则表示永久或不适用。
  - `is_bound` (绑定标记) **tinyint**: 是否绑定（1绑定,0不绑定）。
  - (如果道具有唯一实例属性，比如装备有强化等级等，需要扩展字段或用单独装备表，这里简化为通用道具)
- **主键：** 可采用联合主键 `(player_id, item_id)`，这样一名玩家同一种道具只占一行。如果需要支持一玩家同种道具多条记录（比如不同绑定状态分别存），可以加上区分字段一起做主键。
- **索引：**
  - 主键已包含 `player_id`，因此按玩家查询自然快速。
  - 建议索引 `idx_item` 在 `item_id`（如需按道具ID统计全服持有量或查询玩家有没有特定道具）。
- **表关系：** 外键 `player_id` 引用 players 表。`item_id` 可以引用 game_config.items 表（若有道具定义表）。

- **player_cars（玩家赛车表）**：记录玩家拥有的赛车。

- 字段：
  - `player_id` (玩家ID) **int**: 拥有者。
  - `car_id` (赛车模板ID) **int**: 表示赛车类型的ID。
  - `instance_id` (赛车实例ID) **int**: 主键，自增，每获得一辆赛车产生一个唯一ID（如果需要区分同类型的多辆车）。
  - `star_level` (星级) **int**: 当前星级/品质。
  - `level` (等级) **int**: 当前改装等级。
  - `speed` (速度属性) **int**: （示例属性）当前速度或性能值。
  - `acquired_time` (获得时间) **datetime**: 获得的日期时间。
  - `status` (状态) **tinyint**: 状态标识（0正常，1已回收等）。
- **主键：** `instance_id`（每辆车唯一的记录ID）。
- **索引：**
  - 索引 `idx_player_id` 在 `player_id`（玩家查其所有赛车）。
  - 索引 `idx_car_type` 在 `car_id`（可按车型统计）。
- **表关系：** `player_id` 外键关联 players 表。`car_id` 可关联 game_config.cars 定义表（如果有）。

- **player_signin（玩家签到表）**：记录玩家每日签到状态。

- 字段：
  - `player_id` (玩家ID) **int**: 主键/外键。
  - `month` (月份) **char(7)**: 如 "2025-09"，表示记录哪一月的签到情况。可以与player_id组成主键，因为每月一条记录。
  - `days_checked` (当月已签天数) **int**.
  - `consecutive_days` (连续签到天数) **int**.
  - `last_sign_date` (最近签到日期) **date**.
  - `missed_days` (漏签天列表) **varchar(100)**: 存储未签到的日期列表（如用逗号分隔字符串"05,12"表示5号和12号未签)。或者不存这个，由days_checked和last_sign_date推算。
  - `rewards_claimed` (已领累计奖励标记) **varchar(50)**: 记录玩家已领取过的阶段奖励，例如"7,30"表示7天和30天奖励已领取。存储为字符串或位字段。
- **主键：** (`player_id`, `month`) 联合主键，每玩家每月一条。
- **索引：**
  - 主键已足够按玩家查。
  - 可索引month单列以便统计某月整体签到率等，但非必要。
- **表关系：** `player_id` 外键关联 players 表。

（玩家库可能还有诸如邮件表player_mail、好友表等，这里仅列与GM工具功能相关的表）

索引与性能： - 玩家库各表以玩家ID为主要查询键，索引设计围绕玩家ID进行，以快速定位单个玩家的数据。如上players表主键即玩家ID，子表均有player_id索引。 - 在玩家很多的情况下（百万级），查询单个玩家数据因为有索引支持一般较快。需要注意的是**模糊搜索昵称**可能成为瓶颈，在用户量大时需要考虑建立 `nickname` 的前缀索引或使用ElasticSearch辅助搜索。如果nickname字段用LIKE检索，索引只有在前缀匹配时生效。 - 对于审计和统计需要的字段（如last_login_time），可以配合业务使用合适的索引。 - 事务处理：如果GM工具直接修改玩家数据（如改金币），要确保与游戏服务器可能同时修改数据的冲突。可能需要DB层加锁或让游戏端配合检查。这属于并发控制范畴，不在表结构中体现，但需在操作流程上注意。

5.2 用户库 (user)

用户库主要存储游戏账号的登录信息和全局属性。常见表：

- **users（用户账号表）**：
- 字段：
  - `user_id` (用户ID) **int**: 主键，游戏账号ID。
  - `username` (登录名) **varchar(100)**: 唯一登录账号。
  - `password_hash` (密码哈希) **varchar(255)**: 加密存储的密码。
  - `salt` (盐) **varchar(50)**: （如使用盐则存储）
  - `email` (邮箱) **varchar(100)**: （可选）绑定的邮箱。
  - `phone` (手机号) **varchar(20)**: （可选）绑定的手机。
  - `reg_time` (注册时间) **datetime**: 账号注册时间。
  - `last_login_time` **datetime**: 最近登录时间。
  - `status` (账号状态) **tinyint**: 0正常，1封禁，2需要验证 等。玩家被封禁可在这里标记（游戏服务器会查此状态）。
  - `channel` (渠道) **varchar(50)**: 账号注册的渠道标识（如来自渠道服代号）。
  - `extra` (扩展信息) **json/text**: 其他附加信息（如实名验证信息等）。
- **主键：** user_id。
- **索引：**
  - 唯一索引 `idx_username` 在username。
  - 索引 `idx_phone`, `idx_email` （如允许通过这些查找登录）。
  - 索引 `idx_status_channel` 在(status, channel)以支持按渠道统计封号等。
- **关系：** user_id被player.players引用。1个user可能对应1个players，也可能对应多个角色（视游戏机制，一般手游是一对一）。
- (其他user库表：如用户登录日志、充值记录等，但不在本GM工具直接管理范围)

索引与优化: - 登录和获取账号信息以username为主，确保username唯一索引，查询登录时能快速匹配。 - 封禁玩家时，需要更新 user.status 字段，同时players.status 也可以冗余存一份（或者游戏逻辑统一只看一个库的状态）。根据实现选择一方为准。 - user库在GM工具中主要用于查询账号->角色映射和更改封禁状态、重置密码。相应操作通过主键或唯一键进行，性能易保障。 - 对于批量封禁或解封，可以利用WHERE条件一次更新多个user.status，但这需谨慎（大批量操作需审计）。

5.3 GM管理库 (gm_admin)

GM工具自身的管理数据存储，包含GM用户账号、权限配置和审计日志等表：

- **gm_users（GM用户表）**：
- 字段：
  - `gm_user_id` (GM用户ID) **int**: 主键。
  - `username` (用户名) **varchar(50)**: GM登录用户名，要求唯一。
  - `password_hash` (密码哈希) **varchar(255)**: 登录密码的哈希值。
  - `name` (姓名) **varchar(50)**: GM真实姓名或昵称（用于显示）。
  - `role_id` (角色ID) **int**: 引用 gm_roles.role_id，表示此用户的角色。
  - `status` (状态) **tinyint**: 0正常，1禁用。禁用的账号不能登录。
  - `last_login_time` **datetime**: 上次登录时间。
  - `last_login_ip` **varchar(45)**: 上次登录IP地址（IPv6兼容）。
  - `created_at` **datetime**: 账号创建时间。
  - `updated_at` **datetime**: 信息更新时间。

- **索引：**
  - 唯一索引 `idx_username` 在username（确保登录名唯一）。
  - 索引 `idx_role` 在role_id（方便按角色筛选）。
  - 索引 `idx_status` 在status。
- **关系：** `role_id` 外键关联 gm_roles.role_id。一个GM用户只能有一个主角色。如果需要多角色，可取消role_id字段，改为gm_user_roles映射表。

- **gm_roles（角色表）：**

- 字段：
  - `role_id` **int**: 主键。
  - `role_name` **varchar(50)**: 角色名称（如 "OWNER","AGENT","VIEWER","CUSTOM1"...）。
  - `description` **varchar(100)**: 描述。
  - `is_system` **tinyint**: 是否系统预设角色（1是系统内置，可能不允许删除）。
- **数据示例：**
  - 1, "OWNER", "超级管理员", 1
  - 2, "AGENT", "普通GM", 1
  - 3, "VIEWER", "观察者", 1
  - （自定义角色从ID4起）
- **关系：** gm_users 多对一到 gm_roles； gm_roles 通过 gm_role_perm 关联权限。

- **gm_permissions（权限点表）：**

- 字段：
  - `perm_code` **varchar(50)**: 主键/唯一键，权限代码，例如"PLAYER_VIEW", "ITEM_SEND"。
  - `perm_name` **varchar(50)**: 权限名称，如 "查看玩家", "发放道具"。
  - `category` **varchar(50)**: 权限类别（模块），如 "玩家管理", "道具管理" 用于界面分组。
  - `description` **varchar(200)**: 详细描述。
- **数据：** 预先插入系统支持的所有权限列表。perm_code作为主要标识，用于代码判断也用于界面显示。使用字符串而非ID便于直接在代码和配置中引用。
- **关系：** 通过gm_role_perm与角色关联多对多。

- **gm_role_perm（角色-权限关联表）：**

- 字段：
  - `role_id` **int**: 角色ID。
  - `perm_code` **varchar(50)**: 权限代码。
- **主键：** 复合主键(`role_id` , `perm_code`)。
- **说明：** 表中每一行赋予一个角色一种权限。Owner角色会有多条记录涵盖所有权限。Viewer则只有查看类权限等。
- **查询优化：** 按role_id查询一个角色权限集，需要通过索引快速获取：建议主键或独立索引都可满足role_id筛选的需要。
- **关系：** 外键 role_id -> gm_roles.role_id, 外键 perm_code -> gm_permissions.perm_code。

- **gm_audit_log（审计日志表）：**

- 字段：
  - `log_id` **bigint**: 主键，自增ID。

    - `gm_user_id` **int**: 操作人GM用户ID。
    - `action` **varchar(50)**: 操作类型/代码，例如 "PLAYER_EDIT", "ITEM_SEND" 等，可以与权限代码相对应。
    - `target_type` **varchar(20)**: 目标类型，如"player","item","server"等。
    - `target_id` **varchar(100)**: 目标对象标识。如玩家ID、道具ID或服务器ID等。如果涉及多个目标，可存组合字符串或JSON。
    - `detail` **text**: 操作详情描述。可以是结构化JSON或文本，记录变化的具体内容。例如 "level 30->40, gold 1000->0" 或 "sent item 1001 x 5 to player 2002"。
    - `timestamp` **datetime**: 操作时间。
    - `ip` **varchar(45)**: 操作者IP地址。
- **索引：**
    - 索引 `idx_user_time` 在(gm_user_id, timestamp)用于按人员筛选时间范围。
    - 索引 `idx_target_time` 在(target_type, target_id, timestamp)用于按目标对象查相关日志。
- **关系：** gm_user_id 外键到 gm_users.gm_user_id。这样日志里可以join拿到GM用户名显示。
- **(可选) gm_config（系统配置表）**：
- 用于存储系统可配置参数，例如UI schema定义、功能开关等。
- 字段示例: `config_key` varchar PK, `config_value` text (JSON or string).
- 例如 `("ui_schema_inventory", "<JSON文本>")` 存储背包页面UI Schema结构，以便动态加载；或者 `("feature_item_send_enabled", "true")` 控制某功能开关。
- GM工具启动时可加载这些配置，动态调整界面和功能。

表关系概览: - gm_users -> gm_roles 是多对一（每个用户一个角色）。 - gm_roles -> gm_permissions 是多对多（通过gm_role_perm）。 - gm_audit_log -> gm_users 多对一（日志记录关联执行操作的GM账号）。 - gm_permissions 列出所有可能权限，不直接关联 GM用户，必须通过角色赋予。

索引与优化: - gm_users: 经常按username查找用于登录，必须唯一索引；按role分组查也常用；日志join按user_id索引也重要。 - gm_role_perm: 频繁用于检查某角色有无某权限，业务上通常会在登录时将用户的权限加载到缓存避免频繁查询。但仍应有良好索引，role_id为查询条件，应索引它（主键包含已具备）。 - gm_audit_log: 此表预计增长较快，应规划存档策略（比如只保留最近半年或按需归档）。索引按user和target提供多样查询支持。可考虑建立时间分区以提高查询和删除性能。 - **查询优化案例：** 例如前端权限矩阵需要获取全部角色权限，可以一次JOIN取出或分别查询：

```
SELECT r.role_name, p.perm_code
FROM gm_role_perm rp
JOIN gm_roles r ON rp.role_id = r.role_id
WHERE r.role_name = 'AGENT';
```

有索引的话很快返回Agent拥有的权限列表。 - **安全考虑：** 密码存储使用哈希+盐保证安全，GM用户表不要存明文密码。审计日志表可能会非常大，查询时可以按日期范围限制并分页。

6. UI 构建说明

本章节说明前端UI各组件的构建方式、组件接口 (props)、使用示例，以及如何通过权限裁剪规则和UI Schema来实现动态界面配置。

前端使用React框架，组件化开发，每个页面由多个可复用组件组成。我们在设计组件时遵循**单一职责**和**高度可配置**原则，使组件能通过props调整行为，甚至通过**UI　Schema**数据驱动渲染。下面介绍关键组件和UI构建方式：

6.1 通用组件与属性

- **表单组件(Form)及表单项**: 封装了一组输入控件和标签的布局，用于页面上的数据输入。
- **Props 示例**: `<Form onSubmit={handleSave} initialValues={playerData}> ... </Form>`
  - `onSubmit`: 提交时的回调函数。
  - `initialValues`: 初始化表单各字段的值对象。
- **子组件**: `<Form.Item label="昵称" name="nickname" component={Input} disabled={!canEdit} />`
  - label: 字段显示名。
  - name: 对应数据字段名。
  - component: 实际使用的输入组件（如 Input, Select 等）。
  - disabled: 控制输入是否可用。
- **使用示例**: 在玩家资料编辑区域，用一系列 `<Form.Item>` 放在 `<Form>` 中。可通过权限判断将某些Item的disabled设为true或隐藏整个Item。

- 表单也支持**Schema驱动**：可以定义一个 JSON schema 列表描述各表单项，遍历生成表单结构。见UI Schema部分示例。

- **输入组件**:

- **Input** (单行输入框): Props 常用有 `value`, `onChange`, `placeholder`, `type` 等。用于文本、数字等简单输入。我们可能基于Tailwind封装一个 Input 组件，统一样式。
  - 用例: 玩家ID输入框, 昵称编辑框等。
- **Select** (下拉选择): Props: `options` (选项数组), `value`, `onChange`, `disabled`。
  - 用于选择固定集合值，如状态(正常/封禁)切换，角色选择等。
  - 可封装为**PlayerSelector**或**ItemSelector**等带搜索功能的下拉。
- **InputNumber** (数字输入): Props: `min`, `max`, `step`, `value`, `onChange` 等。用于数量、等级输入等，保证只输入数字范围。
- **TextArea** (多行文本框): Props: `rows`, `value`, `onChange`, `maxLength` 等。用于备注、邮件内容输入。

- **Switch/Checkbox**: 用于布尔值切换，如账号状态（启用/禁用），权限勾选等。

  - `<Switch checked={value} onChange={toggle} />` 视觉上是开关。
  - `<Checkbox checked={permGranted} onChange={...}>玩家编辑</Checkbox>` 用于权限矩阵的单元。

- **表格组件(Table)**:

- 用于显示列表数据（背包道具、玩家列表等）。可使用现有库（如 Ant Design Table）或自己封装。
- **Props**: `columns` (列定义数组), `dataSource` (数据数组), `pagination` (分页配置), `onChange` (分页/排序变化回调) 等。
- **列定义**: 每列定义对象包含 `title` 列名, `dataIndex` 对应数据字段键, `render` 自定义渲染函数, `filters` 筛选选项, `sorter` 排序等属性。
- **使用示例**: 背包页面:

```
const columns = [
 { title: '道具ID', dataIndex: 'itemId' },
```

```
  { title: '道具名称', dataIndex: 'name' },
  { title: '数量', dataIndex: 'quantity', sorter: (a,b)=>a.quantity-b.quantity },
  ...
  { title: '操作', render: (_, record) => (
     <>
       { hasPerm('ITEM_EDIT') && <a onClick={()=>editItem(record)}>修改</a> }
       { hasPerm('ITEM_DELETE') && <a onClick={()=>deleteItem(record)} style={{marginLeft:8}}
>删除</a> }
     </>
   )
  }
];
return <Table columns={columns} dataSource={items} rowKey="itemId" />;
```

通过条件渲染在操作列嵌入权限控制，灵活隐藏按钮。

- 表格组件在UI　　Schema中也可以用抽象定义，如道具列表页面用schema列出列配置（参考3.2节UI Schema示例），组件根据schema自动生成columns。

- **按钮(Button)**:

- **Props**: `onClick`, `type` (样式类型，如primary/secondary), `disabled`.
- **使用**: 用于提交表单、执行操作。结合Tailwind，可有不同颜色表示主次操作或危险操作。
- **权限控制**: Disabled或隐藏通过条件判断：

```
<Button onClick={save} disabled={!hasPerm('PLAYER_EDIT')}>保存修改</Button>
```

或者完全不渲染如先前例子所示。

- **确认对话**: 对于高危按钮，可以封装一个 ConfirmButton，点击时弹确认模态框再执行实际操作。

  - ConfirmButton Props: `onConfirm`, `confirmMessage`, 其他同Button。
  - 在UI Schema actions配置里，可有`"confirm": "确认要删除吗？"`，组件渲染时识别这个属性自动处理确认交互。

- **对话框(Modal/Dialog)**:

- 用于二次确认和子表单（如编辑道具、添加用户）。
- **Props**: `visible`, `onClose`, `title`.
- **使用**:

```
<Modal visible={showEditModal} onClose={()=>setShowEditModal(false)} title="修改道具">
 <EditItemForm item={currentItem} onSubmit={handleSaveItem}/>
</Modal>
```

可以复用Form组件内部。

- 权限上，对话框本身一般由按钮触发，按钮已控制权限。因此对话框内部组件权限可与页面相同逻辑或简单信任调用者权限即可。

- **布局组件**:

- **Card**: 用于包裹一组信息成为独立视觉块。Props: `title`, children。监控页可能用 Card 包裹每个指标。
- **Grid/Flex**: Tailwind 提供实用的栅格/弹性布局类。不需要特殊组件，用div配合Tailwind类名实现页面布局。
- **Tabs**: 有时一个页面可能用标签页区分子内容（比如玩家资料页可以有“基本信息”“道具”“车辆”Tab）。React可以用状态管理当前选中Tab，或使用UI库的 Tabs 组件。
- **Menu/Sidebar**: 整个应用框架有菜单栏/侧边栏导航。每个菜单项对应一个路由页面。菜单组件Props包括菜单项列表、当前选中项等，且菜单项应根据权限过滤（例如Viewer不显示权限管理菜单）。

6.2 权限裁剪规则的实现

前端通过全局的权限列表（当前GM用户拥有的权限代码数组）来控制组件的显示和行为。具体规则有：

- **菜单级**：导航菜单配置中，每个页面路由项包含所需权限标记。例如：

```
const menuItems = [
 { key: 'playerProfile', label: '玩家资料', icon: 'UserIcon', requiredPerm: 'PLAYER_VIEW' },
 { key: 'sendItem', label: '发放物品', icon: 'GiftIcon', requiredPerm: 'ITEM_SEND' },
 ...
];
```

在渲染菜单时，我们过滤掉当前用户没有 `requiredPerm` 的项，使其不可见。这保证用户看不到无权访问的页面。

- **页面路由级**：在React路由配置处，也可以加一层校验，进入某路由时检查权限，不通过则重定向。例如定义一个 ProtectedRoute 组件，在内部判断：

```
<Route path="/send-item" element={
 hasPerm('ITEM_SEND') ? <SendItemPage/> : <Navigate to="/unauthorized"/>
} />
```

- **组件级**: 通过封装**权限容器**组件:

```
const Permit = ({perm, children}) => {
 return hasPerm(perm) ? children : null;
};
```

用法：

```
<Permit perm="PLAYER_EDIT">
 <Button onClick={...}>保存修改</Button>
</Permit>
```

这样内部内容仅在有权限时渲染，无需每次手写条件判断。这在JSX中简化权限控制表达。

也可以进一步封装**DisabledIfNoPerm**:

```
const DisabledIfNoPerm = ({perm, children}) => {
 return React.cloneElement(children, { disabled: !hasPerm(perm) });
};
```

用于输入框等需根据权限只读/禁用的场景：

```
<DisabledIfNoPerm perm="PLAYER_EDIT">
 <Input value={nickname} onChange={...} />
</DisabledIfNoPerm>
```

这样如果没有权限则子Input会被赋予disabled属性。

- **后端兜底**: 前端隐藏只是避免误操作，但并不绝对安全，所以所有接口在后端均验证权限。例如用户调用`POST /api/player/1001`修改等级时，后端会根据其token识别身份，检查是否有PLAYER_EDIT权限，无则返回错误即使前端试图绕过UI发请求。这是最后防线。

6.3 UI Schema 与动态界面

UI Schema是一种用数据结构（通常JSON）描述UI布局和元素的方法。本工具支持UI Schema定义，使部分界面可以通过配置驱动，无需改代码即可增减字段或调整权限控制。实现思路：

- **Schema结构**: 典型的UI Schema包含页面级配置（如页面标题、布局类型）和组件树描述。常见字段：
- `component`: 要渲染的组件类型名称（需在代码中有映射，比如 "Input" -> 实际 Input 组件）。
- `props`: 一个对象，包含传递给组件的属性。
- `children` 或 `fields`: 子组件数组（用于容器类组件，或表单的字段列表）。
- `field` / `name`: 如果是表单项，绑定的数据字段名。
- `label`: 标签文本。
- `visibleFor` 或 `permission`: 用于权限控制，可为数组或单个权限代码，表示此组件只有当用户具备这些权限才可见。或者用 `hiddenFor` 表示对特定角色隐藏。
- 其他：如校验规则`validation`, 占位符`placeholder`等均可写入props或专有字段。

- **Schema解析**: 前端实现一个通用的Schema渲染器，递归解析JSON生成React元素。例如:

```
function renderSchema(node) {
 if (!node) return null;
 if (node.permission && !hasPerm(node.permission)) {
  return null; // 当前用户无权限则不渲染此组件
 }
 const Comp = componentsMap[node.component]; // 从映射获取实际组件类
 const children = node.children?.map(renderSchema);
 return <Comp {...node.props}>{children}</Comp>;
}
```

对于表单类组件，还需将fields解析为Form.Item等特殊处理。

- **Schema存储与加载**：Schema可以存在前端代码里作为常量，或者存储在后端（数据库gm_config表或JSON文件），这样当需要调整UI时，可以修改配置数据而无需重新部署前端。此项目支持UI Schema结构，但在初期可以将schema定义写在前端代码中，等需要灵活配置时再迁移到后端管理。

- **UI Schema 示例**：以**发物品页面**为例，前文已经给出过JSON示例，这里再以更贴近实现的形式展示并说明：

```
{
 "page": "send_item",
 "title": "发放物品",
 "layout": "vertical",
 "fields": [
  {
   "component": "PlayerSelector",
   "field": "player_id",
   "label": "玩家账号",
   "props": { "placeholder": "输入玩家ID或昵称" },
   "required": true
  },
  {
   "component": "ItemSelector",
   "field": "item_id",
   "label": "道具",
   "props": {},
   "required": true
  },
  {
   "component": "NumberInput",
   "field": "quantity",
   "label": "数量",
   "props": { "min": 1, "max": 9999 },
   "required": true
  },
  {
   "component": "TextArea",
   "field": "mail_msg",
   "label": "邮件内容",
   "props": { "rows": 3, "placeholder": "可选：输入发送附言" }
  }
 ],
 "actions": [
  {
   "component": "Button",
   "props": { "type": "primary", "text": "发送" },
   "onClick": "submit",
   "confirm": "确认发送物品？",
   "permission": "ITEM_SEND"
  },
  {
   "component": "Button",
```

```
    "props": { "text": "重置" },
    "onClick": "reset"
   }
  ]
}
```

- `fields` 数组定义了4个表单字段，各自有组件类型和属性。渲染引擎看到 `PlayerSelector` 会用我们封装的玩家选择组件，赋予 placeholder 等。
- 每个字段有 `field` 用于绑定数据名，在表单submit时收集这个键的值。
- `required: true` 可以用于自动添加校验（如必填提示）。
- `actions` 定义了两个按钮。第一个按钮有 `permission: ITEM_SEND`，渲染时会检查权限，无权限则不显示/禁用。`onClick: "submit"` 这种可以约定为提交表单操作（内部会调用API发送道具）。`confirm` 字段会触发二次确认。
- 这个Schema可以保存在数据库，使得假如以后我们想增加一个字段比如选服务器（针对多服情况），可以直接修改配置，无需改代码逻辑（前端解析新字段即可）。

- **Schema扩展**: 当需要扩展一个全新页面或模块时，我们可以：

- 定义新页面路由和菜单项，指定其schema key。
- 在后端或配置中添加该页面的UI Schema定义（字段、操作等）。
- 前端渲染器无需改动就能根据Schema生成UI，前提是所用组件类型都已有实现。如果有全新组件，也需要实现并注册到组件映射。
- 如此，在运维需求变化时，可以通过调整配置快速上线新表单或新功能界面（配合后端提供的新API）。

6.4 组件使用示例

综合以上，举几个实际页面片段的实现示例，以展示组件props和权限控制：

- **玩家资料页面**（片段）：

```
function PlayerProfilePage() {
 const { player, loading } = usePlayerData(playerId);
 const canEdit = hasPerm('PLAYER_EDIT');
 if (loading) return <Spinner/>;
 return (
  <div className="p-4">
   <h2 className="text-xl font-bold">玩家资料</h2>
   <Form initialValues={player} onSubmit={handleSave}>
    <Form.Item label="玩家ID">
     <span>{player.playerId}</span> {/* 只读 */}
    </Form.Item>
    <Form.Item label="昵称">
     <Input name="nickname" defaultValue={player.nickname} disabled={!canEdit}/>
    </Form.Item>
    <Form.Item label="等级">
     <InputNumber name="level" defaultValue={player.level} min={1} disabled={!canEdit}/>
    </Form.Item>
    <Form.Item label="金币">
```

```
        <InputNumber name="gold" defaultValue={player.gold} min={0} disabled={!canEdit}/>
      </Form.Item>
      <Form.Item label="钻石">
        <InputNumber name="diamond" defaultValue={player.diamond} min={0} disabled={!
canEdit}/>
      </Form.Item>
      <Form.Item label="账号状态">
       { hasPerm('PLAYER_BAN') ? (
          <Select name="status" defaultValue={player.status}>
            <Option value="0">正常</Option>
            <Option value="1">封禁</Option>
            <Option value="2">禁言</Option>
          </Select>
        ) : (
          <span>{ player.status===0 ? '正常' : '封禁' }</span>
        )
       }
      </Form.Item>
      <div className="mt-4">
       { canEdit && <Button type="primary" htmlType="submit">保存修改</Button> }
       { hasPerm('PLAYER_BAN') && player.status===0 &&
          <Button type="danger" onClick={banPlayer}>封停账户</Button> }
       { hasPerm('PLAYER_BAN') && player.status===1 &&
          <Button onClick={unbanPlayer}>解封账户</Button> }
      </div>
    </Form>
  </div>
 );
}
```

在这个例子中，通过 `hasPerm` 控制了许多元素的渲染和属性：

- 如果没有编辑权限，相关Input/NumberInput都disabled，且保存按钮不显示。
- 如果没有封禁权限，账号状态不可更改，只显示文本。
- 封停/解封按钮根据权限和当前状态决定显示哪个。
- **权限矩阵组件**（片段）：

```
function RolePermMatrix({ roles, permissions }) {
 const [permMap, setPermMap] = useState(initFromRoles(roles, permissions));
 const handleToggle = (roleId, permCode, value) => {
  setPermMap({ ...permMap, [roleId]: { ...permMap[roleId], [permCode]: value } });
 };
 return (
  <table className="min-w-full table-fixed border">
   <thead>
    <tr>
     <th className="w-40">权限</th>
     { roles.map(r => <th key={r.id}>{r.name}</th>) }
    </tr>
```

```
    </thead>
    <tbody>
     { permissions.map(perm => (
      <tr key={perm.code}>
       <td>{perm.desc}</td>
       { roles.map(role => (
        <td key={role.id}>
         <Checkbox
          checked={permMap[role.id][perm.code] || false}
          onChange={e => handleToggle(role.id, perm.code, e.target.checked)}
          disabled={!hasPerm('ADMIN_MANAGE') || role.isSystem && role.name==='OWNER'}
          // 可能禁止修改OWNER的关键权限, 或禁止非Owner用户操作整个矩阵
         />
        </td>
       )) }
      </tr>
     )) }
    </tbody>
   </table>
  );
}
```

- 此组件接收 roles 和 permissions 列表，通过双重循环输出checkbox矩阵。
- `disabled` 属性根据权限控制：只有有ADMIN_MANAGE权限才可勾选；此外例子里假设我们不允许通过界面修改Owner角色的权限(`role.isSystem && role.name==='OWNER'` 返回true则禁用checkbox)，以防自己去掉自己权限。
- 保存矩阵调整需要在外层页面提供“保存”按钮，点击汇总permMap差异调用API更新角色权限。

6.5 UI Schema 数据示例

为更清晰说明UI Schema驱动界面，这里再给出一个**玩家资料页面**的schema数据简化示例，以及**监控仪表盘页面**的schema示例：

- **玩家资料页面 Schema:**

```
{
 "page": "player_profile",
 "title": "玩家资料",
 "fields": [
  { "component": "Text", "label": "玩家ID", "value": "{{player.playerId}}" },
  { "component": "Text", "label": "昵称", "value": "{{player.nickname}}" },
  { "component": "InputNumber", "label": "等级", "field": "level", "props": { "min": 1 },
"permission": "PLAYER_EDIT" },
  { "component": "InputNumber", "label": "金币", "field": "gold", "props": { "min": 0 },
"permission": "PLAYER_EDIT" },
  { "component": "InputNumber", "label": "钻石", "field": "diamond", "props": { "min": 0 },
"permission": "PLAYER_EDIT" },
  { "component": "Select", "label": "账号状态", "field": "status",
   "props": { "options": [ {"0":"正常"}, {"1":"封禁"}, {"2":"禁言"} ] },
   "permission": "PLAYER_BAN"
```

```
    }
  ],
  "actions": [
    { "component": "Button", "label": "保存修改", "onClick": "submit", "permission":
"PLAYER_EDIT" },
    { "component": "Button", "label": "封停账户", "onClick": "banPlayer", "permission":
"PLAYER_BAN", "visibleWhen": "{{player.status == 0}}" },
    { "component": "Button", "label": "解封账户", "onClick": "unbanPlayer", "permission":
"PLAYER_BAN", "visibleWhen": "{{player.status == 1}}" }
  ]
}
```

- 此schema展示了如何用`permission`字段控制字段/按钮的呈现编辑能力，以及`visibleWhen`作为一个简单的条件渲染（基于当前数据状态）。
- 双花括号`{{ }}`内表示可以在渲染时动态替换的数据字段，比如`{{player.status}}`代表当前玩家状态。
- `Text`组件表示纯文本展示。
- **监控仪表盘页面 Schema:**

```
{
  "page": "dashboard",
  "title": "监控仪表盘",
  "layout": "grid",
  "gridProps": { "cols": 2, "gap": 4 },
  "elements": [
    { "component": "Card", "props": { "title": "当前在线" }, "children": [
        { "component": "Statistic", "props": { "value": "{{stats.online.total}}", "suffix": "人" } },
        { "component": "AreaChart", "props": { "data": "{{stats.online.history}}", "xField": "time",
"yField": "count" } }
      ]
    },
    { "component": "Card", "props": { "title": "服务器性能" }, "children": [
        { "component": "Gauge", "label": "CPU 使用率", "props": { "value": "{{stats.cpu}}", "max":
100 } },
        { "component": "Gauge", "label": "内存使用率", "props": { "value": "{{stats.memory}}",
"max": 100 } },
        { "component": "Text", "value": "运行时间: {{stats.uptime}}" }
      ]
    },
    { "component": "Card", "props": { "title": "公告与控制" }, "permission":
"SERVER_CONTROL", "children": [
        { "component": "Button", "label": "发送公告", "onClick": "openBroadcastModal",
"permission": "SERVER_BROADCAST" },
        { "component": "Button", "label": "重启服务器", "onClick": "confirmRestart",
"permission": "SERVER_CONTROL", "confirm": "确认重启服务器？" },
        { "component": "Switch", "label": "维护模式", "field": "maintenance", "onChange":
"toggleMaintenance", "permission": "SERVER_CONTROL" }
      ]
```

```
  }
 ]
}
```

- 采用了网格布局2列（cols:2）。第一列Card显示在线人数统计及图表，第二列显示CPU/内存。
- 第三个Card“公告与控制”整个卡片设置了`permission: SERVER_CONTROL`，表示只有有该权限才整个显示，否则连标题都不出现。
- 卡片内的按钮各自也有权限标记，一个需要BROADCAST权限（发送公告），一个需要CONTROL权限（重启）。维护模式Switch也需要CONTROL权限。
- `Statistic`, `AreaChart`, `Gauge` 这些组件假定我们已有封装（或来自UI库），它们通过props接受数据绘制。比如Statistic显示数字，AreaChart绘制曲线。
- `{{stats.online.total}}` 表示从绑定的数据中取在线总人数。假定前端通过调用 `/api/monitor/online` 等接口获取数据后，将结果存在context里，使schema可以访问 `stats` 对象。

通过以上schema例子可以看出，UI Schema能有效将界面结构、数据绑定、权限控制解耦出来。开发人员或者高级GM可以编辑这些Schema配置文件，增删界面元素或调整权限要求，而开发人员只需确保有对应组件和数据接口即可。这给未来功能扩展提供了极大便利。

7. 配置与扩展

本章节说明系统如何支持**勾选式配置**和**组件级控制**，以及如何扩展新功能。

7.1 功能勾选配置

“勾选式配置”指的是通过在管理界面勾选选项或修改配置文件/表的数据，即可启用/禁用某些功能模块、页面、字段或操作，无需修改程序代码。这主要体现在：

- **权限勾选**：权限设置页面允许对每个角色进行权限点的勾选开关，这实际上控制了对应功能在UI和API层的开启/关闭。例如取消Agent的“发放道具”权限勾选，前端菜单立刻不显示“发物品”页面，后端也会拒绝Agent再调用相关接口。通过**修改数据库gm_role_perm表**并刷新缓存，即可生效。
- **UI元素勾选**：有些特定功能可以做成可配置开关。例如想暂时关闭“重置密码”功能，可以在配置中为该按钮的权限设置一个特殊值或在gm_config添加一项，前端读取后不显示按钮。更简洁的是利用已有权限，如果不想任何人用就不给任何角色赋予该权限。这样达到了通过权限勾选来关闭功能的目的。
- **模块开关**：如果需要在某环境下彻底隐藏某模块（比如测试阶段不开放VIP管理给普通GM），可在配置中增加**Feature Flag**。实现上，可以在gm_config表加入 `feature_vip_enabled = false`，前端收到后不渲染贵族页面菜单，后端在路由上也可根据该值决定是否注册相关接口（或接口返回未启用提示）。通过修改这一配置即可动态控制模块开关。
- **UI Schema动态调整**：由于UI Schema本身就是数据驱动UI，调整schema等价于调整界面。比如在玩家资料页想新增显示一个“称号”字段而不想改前端代码，只需：
- 后端players表已有`title`字段储存玩家称号。
- 在UI Schema的 player_profile.fields 列表中插入：

```
{ "component": "Text", "label": "称号", "value": "{{player.title}}" }
```

- 前端重新加载schema配置，就会多出这一行显示玩家称号。 同理，要移除某显示，也可以从schema配置删掉对应节点。
- **组件级控制**：甚至可以做到每个小组件的属性都配置。例如决定背包表格是否显示“删除”按钮，就在schema里删除那段action或通过角色权限控制它。这意味着系统的行为可以由运营团队通过配置调整，而不必每次找开发修改。

7.2 新功能扩展指南

当游戏加入新玩法或有新的GM需求时，我们需要扩展GM工具功能模块。凭借模块化设计和配置驱动，本系统扩展相对容易：

1. **后端扩展接口**：根据新需求设计新的REST接口。遵循已有风格，确定路由、请求响应结构。例如游戏新增“宠物”系统，则：
2. 在后端添加 PetModule（或在现有 PlayerModule 中扩展）实现 `GET /api/player/{id}/pets` 等接口。
3. 在 gm_permissions 表加入相应权限项，如 PET_VIEW, PET_EDIT 等。
4. 在 gm_role_perm 为需要的角色赋予这些新权限。
5. 后端实现完成后部署，使之可用。
6. **数据库调整**: 如果新功能需要数据库支持（比如新增 pet 表），先在游戏数据库增加表/字段。GM工具这边，可能需要在玩家库增加 pet 表结构说明，同步更新ER关系。如果影响现有查询（比如玩家删除时要连宠物一起删除），要更新相关代码。但这些一般是游戏服的职责，GM工具主要读写这些新表，所以更多是**确保有权限访问**。
7. 在 gm_admin.gm_permissions 插入新权限。
8. 若有配置表如 gm_config 需要记录一些默认值，可以更新之。
9. **前端界面新增**：
10. 创建新页面组件或使用Schema配置：
    - 最小代价方案：利用 UI Schema 定义一个页面，无需编译前端。例如我们可以在gm_config的 UI schema里写一个 "宠物管理" 页面定义，只要前端渲染器识别component类型并有基本组件（表格、表单、按钮）就能呈现。此方法适合简单的增删改查UI。如果界面复杂交互多，可能还是写React组件更灵活。
    - 写 React 组件方案：在 `pages/` 新建 PetPage.jsx，实现UI。可以参考类似的模块写法（比如 Inventory.jsx结构类似）。然后在导航菜单配置加上 { key:'pets', label:'宠物管理', requiredPerm:'PET_VIEW' }。确保菜单只给有PET_VIEW权限的人显示。
11. 无论哪种方式，都要保证权限判断：菜单 `requiredPerm`, 页面访问控制 ProtectedRoute，页面内操作按钮使用hasPerm等。
12. UI Schema方式下，也需要在permissions页面给角色赋权。如果忘记赋予PET_VIEW，则没人能看到新页面，需注意。
13. **测试与验证**：
14. 赋予测试GM账号新功能的权限，登录前端检查界面是否出现，操作流程是否通。
15. 验证权限限制：用无权限账号验证看不到或不能操作。
16. 验证审计日志是否记录。
17. **文档更新**:
18. 在说明文档（如本文件）中补充新模块的描述、接口、权限说明，保持文档完整性。
19. 更新权限矩阵UI，使新权限项出现在权限设置界面可配置。

扩展例子: 假如要增加“全服邮件群发”功能（GM可以给所有玩家发送邮件）： - 后端：新增接口 `POST /api/broadcast/mail`，接收邮件标题、内容等，并批量写入player_mail表。定义权限 MAIL_BROADCAST。 - 数据库：利用现有邮件表，不需要新表。如果没有则设计mail表。 - 前端：在公告与控制Card中，添加一个按钮：

```
{ hasPerm('MAIL_BROADCAST') && <Button onClick={openMassMailModal}>群发邮件</Button> }
```

并实现一个表单Modal输入邮件内容。或者通过UI Schema在dashboard的actions里插入定义。 - 权限：给Owner或特定角色赋MAIL_BROADCAST权限。 - 审计：接口实现里记录日志 “GM X 群发邮件 给所有玩家 标题 Y”。

通过以上步骤，一个新功能可以比较迅速地整合进系统。

7.3 组件级控制与重用

为了保持代码整洁和减少重复开发，我们提倡**组件重用**和**配置扩展**：

- **组件库**: 将常用元素抽象为组件库（utils/components目录下）。例如：
- PlayerSelector组件可以在发物品、背包等多个页面使用，统一从 `/api/player/search` 获取数据，支持根据ID/昵称搜索。
- ItemSelector组件同理，用于发物品表单和其他涉及道具选择的地方。
- ConfirmButton, Permit 等权限相关组件可以在全局用，以避免每处写繁琐逻辑。
- Table组件可包装一层使其支持通过schema定义列，从而各页面表格渲染逻辑通用。
- **样式统一**: Tailwind CSS用class实现统一样式，但我们也可封装一些带预设样式的组件，如 `<PrimaryButton>` 相当于 `<Button className="bg-blue-600 text-white ...">` 等，方便更改整体风格。
- **权限扩展**: 当组件行为与权限强相关时，可以内置处理。例如我们的 Form.Item 可以扩展一个prop `perm`，指定需要权限才能编辑：

  ```
  <Form.Item label="钻石" name="diamond" perm="PLAYER_EDIT">
    <InputNumber min={0} />
  </Form.Item>
  ```

  让Form组件上下文知道当前用户权限，渲染时如果没有PLAYER_EDIT，则自动将此Item下的输入框disabled。这种组件级别的封装让开发更简单，不用每次在JSX外判断权限。
- **配置扩展**: gm_config表或配置文件可以设计层次结构，例如 UI Schema、Feature Flag、UI主题 等。前端在初始化时会fetch这些配置，存入context。例如:

  ```
  const { features, uiSchemas } = useConfig();
  ```

  通过features判断if/else逻辑，通过uiSchemas渲染动态页面部分。

7.4 维护与升级

为确保系统长期可维护和扩展，在设计上需考虑：

- **升级技术栈**: .NET和React都会有新版本，尽量使用LTS版本并保持依赖可升级。例如.NET未来升级到9, React到19，要确保代码遵循标准做法，减少与版本强耦合。Tauri 框架也要跟进版本以利用安全更新。

- **多语言/国际化**: 目前界面文字都是中文。如果需要支持英文或其他语言，可以将文案抽取为配置或使用i18n框架。UI Schema中的label等也可以按所选语言动态切换。
- **多服务器/多游戏支持**: 目前系统针对单一游戏的多个服务器。如将来需要一个GM工具管理多个不同游戏产品，可以通过**分库**或**前端增加产品选择**实现。数据库结构上不同游戏用不同player库，gm_admin可以通用但加字段区分产品。前端菜单可以按选择的游戏上下文加载对应schema和接口路径。
- **安全**: 扩展功能时始终牢记安全，新的接口和页面都要**最小权限**原则。有些接口即使赋权也可能被滥用（如批量发奖励），可以考虑增加额外确认、限速、防重复提交等措施。
- **日志监控**: GM工具自身应该被监控，如接口异常、前端错误日志，应该有记录。可以扩展一个GM工具的监控页，显示内部错误日志或第三方服务状态（比如邮件服务状态）。

通过以上详尽的说明，我们实现了对GM管理工具的全面设计：包含目标和用途、技术架构、UI/UX细节、API规范、数据库结构以及配置扩展方法。此文档可直接提供给开发团队或投放Claude等编程助手用于代码生成。请在实际实现中根据本说明仔细验证每个步骤，确保系统按设计要求工作。
