需求

MCP-Auth Gateway 项目需求文档

1. 项目背景与目标

1.1 背景

Model Context Protocol（MCP）正在被用于让 LLM/Agent 以标准方式访问各种工具和 API，例如 GitHub、云 API、本地文件系统等。官方提供了 TypeScript SDK，支持快速实现 MCP Server / Client。

与此同时，业界开始讨论 **“基于 OAuth2.1 的 MCP 授权模式 + Gateway 中心化管控”**：

- GitGuardian 提出：在 MCP 场景下，推荐通过 Gateway 集中做 OAuth 授权、token 转换、审计，以适配 Agent 非确定性的调用模式。
- Curity 给出了「使用 MCP Authorization 暴露 OAuth 保护 API」的示例，实现了 OAuth JWT → 下游 API token 的转换和策略执行。

本项目的目标是实现一个 **MCP-Auth Gateway**：

- 对“上游 MCP Host（LLM 客户端）”暴露一个 MCP Server；
- 对“下游 MCP Server（工具）”充当 MCP Client；
- 中间统一接入 OAuth2.1/OIDC 授权服务，实现安全的 token 管理、请求签名、回调验签、撤销与审计。

1.2 业务目标

1. 提供一个「可直接对接现有 MCP 客户端」的网关，Host 只需要连 Gateway，而不直接连多个 MCP Server。
2. 通过 Gateway 统一使用 OAuth2.1/OIDC 授权下游 MCP Server，对接现有 IdP（如 Keycloak / Curity / Auth0 / 自建 OIDC Provider）。
3. 为每个工具调用提供：
    - 工具身份可验证（基于 Signed Manifest）；
    - 能力与 scope 可控；
    - 请求与回调有完整性保护（签名 / PoP）；

- 授权可撤销且全程可审计。

---

2. 技术选型与总体架构

2.1 技术栈要求

- 语言：**TypeScript（Node.js ⩾ 20）**
- 必选依赖：
  - `@modelcontextprotocol/sdk`：用于实现 MCP Server 和 MCP Client。
  - `openid-client`：Node.js OIDC/OAuth2.1 客户端库，用于与外部 IdP 交互。
  - `jose`：JWS/JWT/DPoP 相关签名与验签。
  - `axios` 或等价 HTTP 客户端。
  - 日志：`pino` 或 `winston` 等结构化日志库。

可选：

- 用 `docker-compose` 提供本地一键启动（含：Gateway 容器 + demo IdP，如 node-oidc-provider / Keycloak + demo MCP Server）。

2.2 架构概览

- 对上游：
  - Gateway 暴露 MCP Server 接口（stdio 或 HTTP/WebSocket 均可配置），Host 按文档连接即可。
- 对下游：
  - Gateway 通过 MCP Client SDK 连接若干 MCP Server（配置文件指定 endpoint、transport 等）。
- 对 OAuth：
  - Gateway 通过 `openid-client` 连接外部 IdP，支持 Authorization Code + PKCE、Client Credentials、token introspection / revocation。
- 对 Manifest Registry：
  - Gateway 从给定 URL 拉取工具的 **Signed Manifest（JWS/JWT 格式）**，校验签名并解析。

要求外包方基于以上架构画出一张体系架构图（含组件与流向）和一张 Host→Gateway→MCP Server→外部 API 的时序图，作为文档的一部分。

3. 项目结构要求

推荐的目录结构如下（外包方可以在保持职责清晰的前提下微调）：

```
src/
  server/
    gatewayMcpServer.ts     # 对 Host 暴露的 MCP Server 入口
    routing.ts              # initialize / tools.list / tools.call 分发与控制
  client/
    mcpClientManager.ts     # 下游 MCP Server 连接池
    mcpClientFactory.ts     # 创建具体 MCP Client
  manifest/
    registryClient.ts       # 从 Registry 拉取 Signed Manifest
    verifier.ts             # 验证 JWS 并解析 ToolManifest
    cache.ts                # Manifest 缓存
  auth/
    oauthClient.ts          # 与 IdP 的 OAuth2.1/OIDC 交互（openid-client）
    tokenStore.ts           # access/refresh token 安全缓存
    tokenTransformer.ts     # 长期 token → 短期下游 token（PoP 等）
    dpop.ts                 # DPoP / PoP 工具函数（预留接口）
  policy/
    engine.ts               # 调用前策略检查（scope/capability/HITL 标志）
    models.ts               # 领域模型（Tool、Capability、CallContext 等）
  http/
    callbackServer.ts       # OAuth 授权回调 & health check
    router.ts               # HTTP 路由（可选管理接口）
  security/
    requestSigner.ts        # 请求 canonicalization + 签名（Gateway → MCP）
    callbackVerifier.ts     # MCP Server 回调签名验签
  audit/
    logger.ts               # 结构化日志
    sink.ts                 # 输出到文件 / stdout / 将来可扩展到外部系统
  config/
    index.ts                # 加载配置（ENV / YAML）
    servers.yaml            # 下游 MCP Server & manifest URL 列表
    oauth.yaml              # IdP/客户端配置
main.ts                     # 程序入口：启动 MCP Server + HTTP 服务
```

要求：

- 每个模块职责单一，禁止出现「上帝文件」。

- 必须有基本单元测试（特别是 manifest 验签、OAuth 交互、request 签名/验签）。
- 需要基本集成测试（Host → Gateway → demo MCP Server）。

---

4. 功能需求（Functional Requirements）

FR-1：MCP Server 接口（对 Host）

FR-1.1 Gateway 需要实现标准 MCP Server 的最小接口集：

- `initialize`
- `tools/list`
- `tools/call`

FR-1.2 Gateway 作为 MCP Server 时，对 Host 的行为要求：

- `initialize`：返回 Gateway 自身信息 + 合并后的工具清单（来自下游 MCP Server 的 manifest）。
- `tools/list`：从 Manifest Cache 中取各工具元数据，按策略过滤后返回。
- `tools/call`：
  - 根据 `toolName` 找到对应下游 MCP Server & 工具；
  - 调用 Policy 引擎检查是否允许；
  - 检查是否存在可用 OAuth token，必要时触发授权流程（见 FR-3）；
  - 使用 Security/requestSigner 对请求签名并注入 token；
  - 通过 MCP Client 转发调用并返回响应。

FR-1.3 MCP 传输层：

- 要求至少支持一种传输模式：
  - stdio；或
  - HTTP/WebSocket（可优先实现 HTTP）。
- 传输层选型需要在 README 中说明，并提供 Host 连接示例（如 ChatGPT / AI SDK / 自定义 MCP Client）。

FR-2：Manifest Registry 接入与验证

FR-2.1 从配置文件 `servers.yaml` 中获取每个下游 MCP Server 的 `manifest_url`。

FR-2.2 `manifest/registryClient.ts` 负责通过 HTTP GET 拉取 Signed Manifest（JWS/JWT 字符串），并支持缓存与定期刷新（如 10 分钟/1 小时可配置）。

FR-2.3 `manifest/verifier.ts` 使用 `jose` 进行验签：

- 支持从固定 JWKS（本地配置或 IdP/Registry 提供 URL）获取签名公钥；
- 验证签名、`iss`、`aud`、`exp` 等基础字段。

FR-2.4 解析出统一的内部结构 `ToolManifest`：

```
Plain Text
interface ToolManifest {
  toolId: string;
  serverId: string;
  title: string;
  description?: string;
  capabilities: string[];       // 例如 ["read:issue", "write:issue"]
  oauthScopes: string[];        // 例如 ["repo:read", "repo:write"]
  resourceIndicator?: string;   // OAuth Resource Indicator，用于 token audience
  serverPublicKeyJwk: any;      // MCP Server 回调签名公钥
  // 可扩展字段，需文档说明
}
```

FR-2.5 Manifest Cache：

- 需要有简单的缓存模块，支持：
  - 按 `serverId` / `toolId` 查找；
  - 手动刷新（用于调试）；
  - 过期策略（TTL 配置）。

FR-3：OAuth2.1 / OIDC 集成

FR-3.1 使用 `openid-client` 对接外部 IdP：

- 配置文件 `oauth.yaml` 中至少包含：
  - `issuer`（授权服务器地址）；
  - `clientId` / `clientSecret`；
  - `redirectUri`；

- 支持的 scope 前缀或模板。

FR-3.2 授权模式：

- 必须支持：**Authorization Code + PKCE**（用户级授权）；
- 建议支持：**Client Credentials**（服务级/机器级授权），用于不涉及用户隐私的场景。

FR-3.3 Resource Indicators：

- 在授权请求和 token 请求中，如果 IdP 支持 Resource Indicators，则按 `ToolManifest.resourceIndicator` 传入，使 access token 与特定 MCP Server 绑定。

FR-3.4 TokenStore：

- 用简单可插拔接口实现：可以先用内存 Map，接口设计要允许后续替换为 Redis 等。
- 以 `sessionId + serverId + userId` 为 key 存储 tokenSet（access_token/refresh_token/exp 等）。
- 不允许直接把 token 传入 LLM Prompt 或暴露给上游 Host。

FR-3.5 Token 生命周期策略：

- access token TTL 应可配置（默认 5—15 分钟）；
- 若有 refresh token，则需实现 refresh 流程，自动在调用前刷新过期 token；
- 提供集中 revoke 能力（FR-6）。

FR-4：策略引擎（Policy Engine）

（注意：此版本不要求复杂 Profile，只需实现基础规则，保留扩展点即可。）

FR-4.1 `policy/engine.ts` 输入：

- 当前调用上下文 `CallContext`（包含 toolId, serverId, 用户/会话标识、调用类型等）；
- 解析后的 `ToolManifest`；
- 当前有效 token 中的 scope 信息。

FR-4.2 输出：

- `decision: "allow" | "deny"`；
- `requireHITL: boolean`（是否建议 Host 做一次 HITL 确认）。

FR-4.3 最小规则集示例（必须实现）：

- 未在配置中显式允许的 MCP Server / 工具 → `deny`；

- 用户未获取对应 token 或 token 无效 → `deny` 或触发授权流程；
- 调用类型为明显写操作（比如参数中 `action = delete`）时，`requireHITL = true`；
- scope 不包含 manifest 声明的最低要求时 → `deny`。

FR-4.4 策略可配置：

- 在 `config` 下提供简单策略配置文件，允许管理员按 tool/server 调整策略。

FR-5：请求保护（Request Protection）

FR-5.1 Canonicalization：

- `requestSigner.ts` 需要对下游 HTTP 请求构造一个 canonical 表示（例如 `{method, url,path, query, selected headers, body_hash}`）。

FR-5.2 请求签名：

- 使用 `jose` 生成一个短时效 JWS，包含：
  - canonical 请求摘要；
  - `iat/exp`；
  - `jti`（防重放）。
- 将此签名作为：
  - MCP 请求中的 `mcp_authz.request_signature` 字段；或
  - HTTP header（例如 `X-MCP-Request-Signature`）。

FR-5.3 PoP/DPoP（可以实现简化版接口）：

- 预留接口用于未来集成 DPoP / mTLS：
  - 至少提供一个 `generateDpopProof(token, request)` 函数签名，目前可以返回占位值；
  - 在代码结构中保留调用点，使未来可以无侵入升级为 PoP token。

FR-5.4 故障处理：

- 任意签名失败 / 过期 / 验签失败时，应返回明确错误码和日志（不得默默降级为无签名调用）。

FR-6：回调验证与撤销（Callback & Revocation）

FR-6.1 OAuth 授权回调：

- 在 `http/callbackServer.ts` 中实现 `/oauth/callback` 路由：
  - 验证 `state` / `code`；
  - 调用 `oauthClient.exchangeCode` 获取 tokenSet；
  - 写入 `tokenStore`；
  - 返回简单 HTML 提示“授权成功，可以关闭此窗口”。

FR-6.2 MCP Server 回调验签（如支持异步/流式）：

- 约定：下游 MCP Server 在 manifest 中提供 `serverPublicKeyJwk`，并对重要回调（如工具执行结果）签名；
- Gateway 在 `callbackVerifier.ts` 中：
  - 使用 manifest 公钥验证签名；
  - 校验 nonce/timestamp 防重放；
  - 验证成功后才将回调结果注入 MCP 会话。

FR-6.3 授权撤销：

- 提供内部接口 `authz.revoke(sessionId, serverId, userId?)`：
  - Gateway 调用 IdP 的 revocation endpoint；
  - 本地 tokenStore 删除对应 token；
  - 后续调用检测到缺失 token 时需重新授权。
- 可选：提供一个 HTTP 管理 API `/admin/revoke`，通过简单鉴权调用上述内部接口。

FR-7：审计与日志（Audit & Logging）

FR-7.1 结构化日志内容至少包括：

- 时间戳；
- Host 标识 / sessionId / userId（如可获取）；
- toolId / serverId；
- 调用类型（read/write 等）；
- OAuth token 的 meta（不要记录 token 本身，仅记录 scope/exp/iss 等）；
- Policy 决策结果（allow/deny/requireHITL）；
- 错误码 / 异常信息。

FR-7.2 输出：

- 默认输出到 stdout（方便容器采集）；
- 可配置输出到文件。

FR-7.3 要求至少有两类日志级别：info / error（可增加 debug）。

FR-8：配置与部署

FR-8.1 所有敏感配置（clientSecret、JWKS 等）需通过环境变量或独立密钥文件管理，不允许硬编码在仓库中。

FR-8.2 配置文件：

- `servers.yaml`：列出所有下游 MCP Server；
- `oauth.yaml`：IdP 配置；
- 若使用外部 Registry/JWKS，需在 `config/index.ts` 明确加载逻辑。

FR-8.3 部署样例：

- 提供一个 `docker-compose.yml`，包含：
  - Gateway 服务；
  - 一个 demo IdP（可选 node-oidc-provider 或 Keycloak）；
  - 一个开源 MCP Server（可选 Curity 的 MCP Authorization example 或简单 demo MCP Server）。
- README 中给出从零启动的步骤说明。

---

5. 非功能性需求（NFR）

NFR-1：代码质量

- 所有 TypeScript 源码需通过 `tsc --noEmit` 和 ESLint（可选 Prettier）；
- 关键模块需有单元测试，整体测试覆盖率建议 ⩾ 60%（可不强制，但需要覆盖所有安全关键路径）。

NFR-2：性能与可扩展性（最低可用标准）

- 在单实例下能稳定支撑 50 并发会话、每秒 10 次工具调用（主要是为了 PoC 阶段演示，不需要重度优化）；
- 所有外部请求（Registry/OAuth/MCP Server）要有超时与重试策略。

NFR-3：安全性

- 不在日志中打印 access_token/refresh_token 等敏感信息；
- 对所有 HTTP 接口做基本输入校验，防止明显注入/路径遍历；
- 除 `/health` 和 `/oauth/callback` 外，其余 HTTP 管理接口需有简单鉴权（API key 或 Basic auth 即可）。

---

6. 交付物与验收标准

6.1 交付物

1. 完整代码仓库（TypeScript 源码、test、配置样例、脚本）。
2. README 文档，至少包括：
   - 架构说明；
   - 运行环境要求；
   - 如何启动 demo（docker-compose 或本地启动步骤）；
   - 如何配置接入一个实际 MCP Server + IdP。
3. 简要设计文档（可以是 Markdown），包含：
   - 模块划分；
   - 关键时序图（授权、调用、回调、撤销）；
   - 数据结构说明（ToolManifest、CallContext 等）。

6.2 验收场景（至少）

1. **基础连通性**
   - 启动 Gateway + demo MCP Server + demo IdP；
   - 使用官方 MCP Client 示例或 AI SDK 连接 Gateway，能获取 tools 列表并调用一个只读工具。

2. OAuth 授权流程

- 调用需要用户授权的工具时，Gateway 返回一个含授权 URL 的错误/提示（具体形式可约定）；
- 用户在浏览器完成授权后，Gateway 成功缓存 token，再次调用时无需重新授权；
- 在 IdP 管理页面撤销授权后，Gateway 对后续调用返回相应错误，并记录审计日志。

3. 策略与签名

- 对不在 manifest / 未授权 scope 内的操作，Gateway 拒绝调用；
- 关闭/篡改请求签名后，下游 MCP Server 能检测到并拒绝（若 demo MCP Server 不支持，可用 fake 检查替代，但需要保留接口）。

4. 撤销与审计

- 通过管理接口触发 `authz.revoke`，Gateway 后续不再使用原 token；
- 审计日志包含调用记录、错误记录，能明确回溯某次调用的决策原因。