API Gateway

English | 中文

本文是 Typert API Gateway 的当前状态参考。它描述业务服务如何声明一元 Remote 方法、构建如何 `/api` [illegible] 它们可 [illegible] 使用同一

编程模型

`@Remote` `@RemoteScope` 选择对 Client 开放的方 `ctx.remote` [illegible] 的 Client 类型或运行时贡献，也不能通过

`@Remote` 表示调用根 Host Context 中注册的 Cordis 服务 `TypertLookupMap` [illegible] `ctx.typert.lookups` `Agent` [illegible] Host `agent`，生成的 w `agentId`，Gateway 在调用业务方法前将 id 解 `ctx.typert.lookups.configure()` 覆盖某个 lookup key 的解析策略，而 symbol。

`@RemoteScope(key)` `ctx.typert.contexts` 把 identity 解析为一个作用域 Context `Agent` [illegible] 取得服务并调用方法。它适用于方法本身依赖作用域

`TypertRemoteService`，让 Cordis 服务 key 与默认 Remote name `readonly typertRemote = bindTypertRemote(this, serviceKey)`；两种方式都会留下可检查的公开 binding，不依赖编译器向构造函数注入 symbol。

```
import type { Agent } from '@deepseek-ai/dsh-agent'
import { TypertRemoteService, Remote, RemoteScope } from '@deepseek-ai/dsh-typert-protocol'
import type { Context } from '@deepseek-ai/cordis'

export interface CreateGoalRequest {
  objective: string
}

export interface CreateGoalResult {
  accepted: boolean
}

export class GoalService extends TypertRemoteService {
  constructor(ctx: Context) {
    super(ctx, 'goals')
  }

  @Remote('create')
```

```
  createForClient(
    agent: Agent,
    request: CreateGoalRequest,
    signal: AbortSignal,
  ): CreateGoalResult {
    signal.throwIfAborted()
    return this.create(agent, request)
  }

  @RemoteScope('agent', 'current')
  currentForClient(): CreateGoalResult {
    return { accepted: true }
  }

  private create(_agent: Agent, request: CreateGoalRequest): CreateGoalResult {
    return { accepted: request.objective.length > 0 }
  }
}
```

Remote 方法可以同步返回或返回 Pro`signal: AbortSignal` `args` 最后一个参数 `AbortSignal` 个可选的

Client 使用普通对象上的具体函数，不使用 J`ctx.remote.<namespace>` `agentCtx.remote.<namespace>` 。每个 namespace 都是注册为 `remote.<namespace>` 的可追踪 Cordis 子服务；Client ass`ctx.remote.$mount()` 挂载贡献，最后一个方法撤回后该 namespace 随即卸载。依赖 `ctx.remote.<namespace>` `agentCtx.remote.<namespace>` `inject` `remote` `remote.<namespace>` ；只负责挂载 contribution 的 assembly，以及不调用该 namespace 的上层运行 `@Remote` `TypertContextMap` 使用相同 wire identi identity `@RemoteScope` 只生成作用域调用接口。

```
import type { SessionId } from '@deepseek-ai/dsh-session/types'
import type { AgentContext } from '@deepseek-ai/dsh-api-session-controller/client'
import type { Context } from '@deepseek-ai/cordis'
import type {} from '@deepseek-ai/dsh-api-remotes/client'

export const inject = ['remote', 'remote.goals']

declare const ctx: Context
declare const agentCtx: AgentContext
declare const agentId: SessionId

await ctx.remote.goals.create(agentId, { objective: 'ship it' })
await agentCtx.remote.goals.create({ objective: 'ship it' })
```

Client [@deepseek-ai/dsh-api-remotes [/remote] [ctx.remote.$mount()] Host Remo
Gateway Remote JS

api-remotes [ctx.remote React Client Host Remote

	@deepseek-ai/dsh-typert-protocol	decorator Gateway binding Cordis
	@deepseek-ai/dsh-typert-generator	Host ts.Program Remote lookup Context Host H
Host	@deepseek-ai/dsh-typert-registry Loader	Host s ctx.typert lookup Context
Host	@deepseek-ai/dsh-api-session-controller	Agent/Session Typert lo
Host	@deepseek-ai/dsh-api-gateway	[ctx.typertGateway Remote endpoint Context Cordis
Client	@deepseek-ai/dsh-api-gateway/client	[ctx.remote [remote.<namespace> Connection
Client	@deepseek-ai/dsh-api-remotes/client	[/remote
	@deepseek-ai/dsh-client-connection	RPC carrier /api HTTP envelope bridge

API Gateway Host dispatcher Client Remote endp ts.Program Host Client Context Clie
Host Gateway

[build:lib:host build:lib:client [build:web Host lib tsc -b tsconfig.host.json [tsdown --env.DSH_BUILD_FACE host
Typert generator Host Project Reference tsdown Ho ts.Program te Clien tsc -b
tsconfig.client.json [tsdown --env.DSH_BUILD_FACE client Remote Client Typert

tsdown work lib/types tsc JavaScript Clien DSH_BUILD_FACE s
Node loader browser bundle

api/remotes api/gateway api/session-controller [api/workspace-controller [client/connection TypeScript fac
api/remotes Client project Host ts /remote aggrega tsconfig.host.json [tsconfig.client.json api-
remotes [clientBundle(..., { hostPhase: true }) Host Host tsdown Client tsdown browser
lookup [@deepseek-ai/dsh-api-session-controller [api-remotes]

[lib/]

typert.host.js	Host Loader	Host face schema
typert.host.d.ts	Host	Host face
typert.remote-client.js	api-remotes	[TypertRemoteContribution cod
typert.remote-client.d.ts	Client	TypertRemoteNamespaceMap TypertRemoteScopeMap Client-safe
typert.remote-client.d.ts.map		Host Remote

[./typert] Host Loade ./remote] Host-for-Client export

Remote Client wire ctx.remote.goals.create [@Remote] Host declaration-m Client [.d.ts]

Remote TypertLookupMap lookup Context JSON Typ

Remote Connecti /api] Client Remo connection.rpc.call('/api', '<namespace>/<method>', { args }, signal) HTTP carrier POST /api/<namespace>/<method> payload args

Connection HTTP bridge /api] FetchHandler Typert Gateway SRC mar 404Connection RPC id envelope Gateway Remote Connection carrier

Gateway [args wire lookup Context binding

lookup [register() configure() r Host e gateway/lookup-unavailable Controller [agent] [session resolver live Agent session lookup Agent Session ownership f RemoteError session/not-found [session/agent-busy]Gateway wire gateway/internal]

Client Host endpoint SRC

SRC

Host [node --import tsx/esm] Typert de TypertRemoteService [bindTypertRemote() binding Gateway ts.Program [remoteMethods()

SRC 回退从运行中函数`parameter``agent` l`session``agentId` `sessionId` wire 字段并在 Host 解析对象；其他参数只检查值是否为无循环、无特殊 prot
`@RemoteScope` 直接使用已注册 Host Context 提供方的 wire 字段。SRC 不读取 TypeScript 类型，不生成 Zod schema，不推断可选参数，也不支

SRC 只解决 Host 源码进程的分发问题。Client 不会从运行中的 Host 发现 decorator，Client Remote 也拒绝挂载`lib/typert.remote-client.*`

开发模式

Web `pnpm run build` 准备当前 Host、Client 与 Web 产物，然后在两个终端中分别运行源码 Host 和 Client plugin watcher：

```
pnpm dsh web

pnpm run dev:web
```

`dsh` 通过 tsx 启动 Host 源码，所以 H`dev:web` S`dsh.client` 声明的 Clie`lib/client.js`，它不会分析 Host decorator，也不会生成 Remote

只修改 Remote 方法实现体而不改变约定时，无需重新生成 Typert 文件。新增或删除 decorator、修改导出名、namespace、参数、返回值、lookup、Context

```
pnpm run build:lib
```

运行中的 Client watch`pnpm run build:lib:host`删除运 Hos`pnpm run build:lib:client` 完成 Client 侧；干净工作树不能跳过 Host 阶段。仅重新编
`pnpm run typecheck` 会执行 Host lib 阶段后再运行 Client tsc，CI 与发布构建也使用同一顺序。

边界

Remote 只处理有单个请求与单个结果的一元方法调用。会话事件流、分页、增量 reduce、projection 和实体子流需要独立的数据协议与注册模型；即使它们复用 Connection，也

API `remotes → gateway → connection → webserver` 组织。BFF 与 Typert R`packages/api`；Connection 与 WebServe
`packages/client/connection` `packages/host/webserver`。需要流式或浏览器原生响应的功能注册精确的 Connection Fetch 路由，而不定义 Rem

lookup 策略按 ke`agent``session` 参数共享冷恢复行为。只接受 live 对象需要显式的逐参数或逐 endpoint 策略，而这种策略并不存在；不能通过业务方法内部猜测对象是否来自恢

版本号 #1
由一说春秋 创建于 3 九月 2026 17:08:27
由一说春秋 更新于 3 九月 2026 17:36:27