Dsh Cordis教程

Cordis [illegible]

English | [illegible]

Cordis [illegible] DeepSeek Harness [illegible] vendor [illegible] Cordis [illegible] vendor/README.md [illegible]

[illegible]

- [illegible] **Serv**[illegible] `inject` [illegible] `apply(ctx)` [illegible] `Service` [illegible] Cordis [illegible]
- [illegible] `ctx.<key>` [illegible] `ctx.tools` `ctx.llm` `ctx.sessions` [illegible] key [illegible]
- [illegible] **`inject`** [illegible]
- [illegible] TypeScr[illegible] `emit` `waterfall` `parallel` `serial` [illegible] `bail` [illegible] bail [illegible]
- [illegible] `ctx.effect()` [illegible] `ctx.on()` [illegible] reload [illegible] teardown [illegible]

[illegible]

[illegible]

[illegible]	[illegible] **await**[illegible]	[illegible]	[illegible]
`emit`	[illegible]	[illegible]	[illegible]
`waterfall`	[illegible]	[illegible]	[illegible]
`parallel`	[illegible]	[illegible]	[illegible]
`serial`	[illegible]	[illegible]	[illegible]
`bail`	[illegible]	[illegible] bail [illegible]	[illegible]

[illegible] `@mode` [illegible] harness [illegible]

Cordis Waterfall [illegible]

`ctx.waterfall` [illegible] `(...args, next)` [illegible] `next()` [illegible] `next()` [illegible] `next()` [illegible]

`prepend: true`

`next()`

Loader

`@deepseek-ai/cordis-plugin-include` `!!js` `ctx.serviceName` `config` `disabled`

`ctx.tools` `ctx.llm` `ctx.agents`

`ctx.effect()` Cordis teardown effect

Cordis 教程

English | 中文

Cordis 是 DeepSeek Harness 底层的插件框架：它是一个小型运行时，其中的每项能力，包括工具、LLM（大语言模型）适配器、文件访问乃至 agent loop（智能体循环）本身

本教程面向 agent 开发者。你不需要深入掌握 TypeScript：说明会解释可能陌生的语法，并且每一章都会给出确切命令和预期输出。

如果你想阅读精简的概念参考，请参阅 `cordis-surface` 区块，以及 is 核心 API 页面。

如果你要为 harness `cordis.yml` 挂载、在 Web UI 中驱动，而不是下面的插件开发器 ——请从

准备工作

你需要克隆本仓库并安装依赖；开发指南列出了前置条件。本教程不需要 API 密钥；所有示例均可在无密钥环境中运行。

```
git clone https://github.com/deepseek-ai/deepseek-harness.git
cd deepseek-harness
pnpm install
```

`tmp/` 用于存放临时目录，会被忽略，因此你在其中写入的任何内容都不会进入版本控制：

```
mkdir -p tmp/cordis-tutorial
cd tmp/cordis-tutorial
```

每一章都从该目录运行同一条命令：

```
node --import tsx ../../vendor/cordis/bin.js
```

这个单文件启动器（见 cordis/bin.js）创建 `Context`、挂载 Loade `./cordis.yml` 。目录中的内 `--import tsx` 标志配置它们，都 自你无需构建步骤即可运行 置所有的 TypeScr

章节

1. 你的第一个插件：插件是函数，由 loader 挂载。

2. 生命周期与 effect Cordis 管理的注册会在所属插件卸载时撤销。
3. [`ctx` [`inject` 依赖通过
4. 事件：类型化事件、广播分发和 waterfall（瀑布式事件）的短路行为。
5. [`cordis.yml` 中经过校验的配置，并在输入错误时明确报错。
6. 组合与 HMR 置为模块替换插件树，使用热重载，并诊断始终无法加载的插件。
7. 进入 harness 于真实的 harness 服务注册一个可由模型调用的工具。

TypeScript 说明

这些示例使用了普通现代 JavaScript 之外的三项 TypeScript 功能：

- `ctx: Context` `ctx` 具备 Cordis 上 `who: string` [`string[]` 表示字符串数组。
- **`import type { Context } from '@deepseek-ai/cordis'`** [`Context` 在插件文件会消除运行时类型注解使用
- `declare module '@deepseek-ai/cordis' { ... }`）会为 Cord `ctx.greeter` 加属性类型或事件名称。它不会生成任何运行时接线；插件必须另行提供服务或发出事

第 5 `interface` [`Schema<Config>` 这类泛型表示 schema 校验哪些对象字段。你可以直接照写这些声明；周围的正文会解释每项声明连接了什么。

Image not found or type unknown

1. 编写第一个插件

English | 中文

在本教程使用的 loader [`apply` rdis Cordis `apply`][`ctx` [对象，插件通过它注册自己贡献的所有内容。

编写插件

[`tmp/cordis-tutorial` [`hello.ts` :

```
import type { Context } from '@deepseek-ai/cordis'

export const name = 'hello'

export function apply(ctx: Context) {
  console.log('hello from my first plugin')
}
```

`name` 导出项是可选的显示元数据；它用于在诊断信息中标识插件。

组合应用

[`cordis.yml`]组装应用。创建

```
- name: './hello.ts'
```

该文件是一组 C `name` 配置项的列表，可以是相对 `inject` M[.启动 oa 文 er 的每个配置项。各项会并发启动，因此它们在列表中的位置不保证插件的加载先后；顺序由服务依赖（

运行

```
node --import tsx ../../vendor/cordis/bin.js
```

预期输出：

```
hello from my first plugin
```

[illegible]

1. [illegible] `Context` [illegible] **Loader** [illegible]
2. Loader [illegible] `cordis.yml` [illegible] `./hello.ts` [illegible]
3. Cordis [illegible] `apply(ctx)` [illegible]

`cordis.yml` [illegible] [`dsh`] [illegible] [base] [illegible] overlay [illegible]

[illegible]

[illegible] Cordis [illegible]

```
import { Service, type Context } from '@deepseek-ai/cordis'

// 1. Function plugin (what you just wrote).
export function apply(ctx: Context) {}

// 2. Object plugin: an object with an `apply` method.
export const objectPlugin = {
  name: 'object-plugin',
  apply(ctx: Context) {},
}

// 3. Class plugin: a Service subclass (covered in chapter 3).
export class MyService extends Service {
  constructor(ctx: Context) {
    super(ctx, 'myTutorialService')
  }
}
```

[[illegible] 3] [illegible]

[illegible]

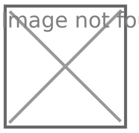
[illegible] `apply` [illegible]

```
export function apply(ctx: Context) {
  throw new Error('apply exploded')
}
```

[illegible]

[illegible] Cordis [illegible] logger [illegible] console [illegible]

[illegible] effect [illegible]



2. 生命周期与 effect

English | 中文

Cordis 插件可能因修改配置、热重载、显式资源释放或所 `ctx.effect()` Cordis API 建立的注册属于 effect，会在所属插件卸载时撤销；在这些 API 之外管理的资源必须包装在

Effect

对于 Cordis 尚未管理的资源 `ctx.effect()` watch dispose 资源释放函数）：

`lifecycle.ts` `tmp/cordis-tutorial` 中：

```
import type { Context } from '@deepseek-ai/cordis'

export const name = 'lifecycle-demo'

function heartbeat(ctx: Context) {
  console.log('heartbeat plugin loading')
  ctx.effect(() => {
    const timer = setInterval(() => console.log('tick'), 200)
    return () => {
      clearInterval(timer)
      console.log('heartbeat cleaned up')
    }
  })
}

export function apply(ctx: Context) {
  // Mount a child plugin and keep its fiber to dispose it later.
  const fiber = ctx.plugin(heartbeat)
  // The demo timer is itself an effect: if THIS plugin is unloaded first,
  // the pending callback is cancelled instead of firing on a dead app.
  ctx.effect(() => {
    const timer = setTimeout(async () => {
      await fiber.dispose()
      console.log('disposed')
```

```
      process.exit(0)
    }, 700)
    return () => clearTimeout(timer)
  })
}
```

[cordis.yml]

```
- name: './lifecycle.ts'
```

node --import tsx ../../vendor/cordis/bin.js

```
heartbeat plugin loading
tick
tick
tick
heartbeat cleaned up
disposed
```

- ctx.plugin(heartbeat) YA apply apply ctx.plugin({ apply(ctx) { /* ... */ } }) **fiber**
- effect disposer disposer
- fiber.dispose() disposer

Fiber

fiber

```
PENDING → LOADING → ACTIVE → UNLOADING → DISPOSED
                ↘ FAILED
```

- **PENDING** 3
- **LOADING / ACTIVE** apply
- **FAILED** apply
- **UNLOADING / DISPOSED** disposer

6 PENDING

已经属于 effect 的操作

[`ctx.effect()`，因为内置注册 API 本身已经是 effect：

- `ctx.on(event, listener)` 器会在卸载时移除（第
- `ctx.plugin(child)`：子插件会随父插件一同 dispose（资源释放）。
- 服务注册属于 `ctx.tools.register(...)` 等 harness 注册表也会把返回的第7 disposer 附着到调用插件上，因此会自动撤销（

对于 Cordis `ctx.effect()` 内获取它，并返回用于释放资源的 disposer。此后 Cordis 会在卸载期间调用该释放逻辑，热重载时也不例外。

有一项顺序注意事项异步 disposer 会并发运行。如果你的清理步骤必须按顺序执行，请把它们放在同一个 disposer 中，并在其中依次等待每步完成。

下一章：服务：插件如何共享功能。

Image not found or type unknown

3. 服务

English | 中文

[`ctx`提供的其他插件通过 `ctx.tools` `ctx.llm` [`ctx.agents` [`'tools'`]之类指定能力，而不导入其提供方，因此配置可以选择提供方，无需修改消费方。

提供服务

[`greeter.ts` [`tmp/cordis-tutorial` 中:

```
import { Service, type Context } from '@deepseek-ai/cordis'

declare module '@deepseek-ai/cordis' {
  interface Context {
    greeter: GreeterService
  }
}

export class GreeterService extends Service {
  constructor(ctx: Context) {
    super(ctx, 'greeter')
  }

  greet(who: string) {
    return `Hello, ${who}!`
  }
}

export const name = 'greeter'

export function apply(ctx: Context) {
  ctx.plugin(GreeterService)
}
```

两部分协同工作:

- `super(ctx, 'greeter')` [`greeter` [`ctx.greeter` [effect
- `declare module '@deepseek-ai/cordis'` [TypeScri `greeter` [`Context` [`ctx.greeter`

`Service` [`ctx.plugin(GreeterService)`

[inject [

[`consumer.ts` [

```
import type { Context } from '@deepseek-ai/cordis'

export const name = 'consumer'
export const inject = ['greeter']

export function apply(ctx: Context) {
  console.log(ctx.greeter.greet('world'))
}
```

`inject` [Cordis `apply` [`ctx.greeter` [`cordis.yml` [

[

```
- name: './greeter.ts'
- name: './consumer.ts'
```

```
Hello, world!
```

[`cordis.yml` [`./greeter.ts` [PENDING [6 [PENDING [fiber [Node [

[

`inject` [2 [effect [

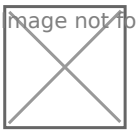
[`dsh-bash-local` [`shell` [`'shell'` [

[

inject [inject

```
export function apply(ctx: Context) {
  // undefined when no provider is loaded; the plugin still runs.
  const greeter = ctx.get('greeter')
  console.log(greeter?.greet('maybe') ?? 'no greeter available')
}
```


 tools llm cordis-surface harness



4. 事件

English | 中文

服务支持直接调用；事件让插件无需知道有哪些插件正在监听，就能发出通知。harness 使用事件处理工具结果、模型请求和审批决定等交互。

声明、发出与监听

[stats.ts [tmp/cordis-tutorial 中。它是一项负责计数并在每次变化时发出通知的服务：

```
import { Service, type Context } from '@deepseek-ai/cordis'

declare module '@deepseek-ai/cordis' {
  interface Context {
    stats: StatsService
  }
  interface Events {
    'stats/report'(name: string, count: number): void
  }
}

export class StatsService extends Service {
  private counts = new Map<string, number>()

  constructor(ctx: Context) {
    super(ctx, 'stats')
  }

  bump(name: string) {
    const next = (this.counts.get(name) ?? 0) + 1
    this.counts.set(name, next)
    this.ctx.emit('stats/report', name, next)
  }
}

export const name = 'stats'
```

```
export function apply(ctx: Context) {
  ctx.plugin(StatsService)
}
```

`interface Events` 合并与`interface Context` [`ctx.emit`][`ctx.on`_`namespace/action` 命名约定让扁平的事件命名空间保持易读。

[`reporter.ts`:

```
import type { Context } from '@deepseek-ai/cordis'
import type {} from './stats.ts'

export const name = 'reporter'
export const inject = ['stats']

export function apply(ctx: Context) {
  ctx.on('stats/report', (name, count) => {
    console.log(`[stats] ${name} -> ${count}`)
  })
  ctx.stats.bump('tool_call')
  ctx.stats.bump('tool_call')
  ctx.stats.bump('prompt')
}
```

`import type {} from './stats.ts'` 行不会在运行时导入任何内容；它的作用是让 TypeScript 看到声明合并。组合并运行：

```
- name: './stats.ts'
- name: './reporter.ts'
```

```
[stats] tool_call -> 1
[stats] tool_call -> 2
[stats] prompt -> 1
```

[`ctx.on()` 属于 effec`removeListener`绝不需要手动维护

分发模式

`emit` 是 5 种分发模式之一。事件采用哪种模式是其约定的一部分，决定了监听器能否返回值、能否并发运行，以及能否彼此短路：

模式	调用	语义

emit	`ctx.emit(name, ...args)`	□□□□□□□□□□□□□□ promise □□□
parallel	`await ctx.parallel(name, ...args)`	□□□□□□□□□□□□□□□□
serial	`await ctx.serial(name, ...args)`	□ `null` / `false` / `undefined` □□□□□□□□□□□□□□□
bail	`ctx.bail(name, ...args)`	serial □□□□□□
waterfall□□□□□□□	`ctx.waterfall(name, ...args, next)`	□□□□□□□□□□

□□ harness□□□□□□□□□□□□□□□□

waterfall□□□□□

waterfall □ `next()` □□□□continuation□ `next()` □ `next()` □□□□□□□□□□□ `waterfall-demo.ts` □□□□□□□□□□□□□□□□□

```
import type { Context } from '@deepseek-ai/cordis'

declare module '@deepseek-ai/cordis' {
  interface Events {
    'demo/transform'(input: string, next: () => Promise<string>): Promise<string>
  }
}

export const name = 'waterfall-demo'

export function apply(ctx: Context) {
  // Listener 1: wrap the downstream result.
  ctx.on('demo/transform', async (input, next) => {
    const downstream = await next()
    return downstream.toUpperCase()
  })

  // Listener 2: short-circuit when it owns the decision.
  ctx.on('demo/transform', async (input, next) => {
    if (input.includes('blocked')) return '** blocked **'
    return next()
  })

  void (async () => {
    console.log(await ctx.waterfall('demo/transform', 'hello', async () => 'hello'))
```

```
    console.log(await ctx.waterfall('demo/transform', 'blocked words', async () => 'blocked words'))
  })()
}
```

[cordis.yml]

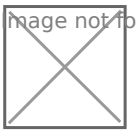
```
HELLO
** BLOCKED **
```

[next()] [blocked] [next()] [ctx.waterfall] 1

wa next() [next()] waterfall

harness [waterfa agent/request] [approval/request]

[cordis.yml]



5. [illegible]

English | [illegible]

`cordis.yml` [illegible] Cor`config`[illegible]`apply`[illegible]

[illegible]

[illegible]`config-demo.ts`[illegible]`tmp/cordis-tutorial` [illegible]

```
import type { Context } from '@deepseek-ai/cordis'
import Schema from '@deepseek-ai/schemastery'

export const name = 'config-demo'

export interface Config {
  greeting: string
  targets: string[]
}

export const Config: Schema<Config> = Schema.object({
  greeting: Schema.string().default('Hello'),
  targets: Schema.array(String).default(['world']),
})

export function apply(ctx: Context, config: Config) {
  for (const target of config.targets) {
    console.log(`${config.greeting}, ${target}!`)
  }
}
```

[illegible]`Config` [illegible] TypeScript [illegible] schema [illegible] Schemastery [illegible] Config schema [illegible] Cordis [illegible] Standard Schema [illegible]`Config` [illegible]

[illegible]

```
- name: './config-demo.ts'
  config:
    targets: ['alpha', 'beta']
```

[illegible]

```
Hello, alpha!
Hello, beta!
```

[illegible] `greeting` [illegible] schem `apply` [illegible]

[illegible]

[illegible]

```
- name: './config-demo.ts'
  config:
    targets: 'not-an-array'
```

```
ValidationError: invalid config:
  - $.targets expected array but got not-an-array (at targets)
```

[illegible] fiber [illegible] FAILED [illegible] 1 [illegible] schema [illegible]

[illegible]

[illegible] lo `!!js` r [illegible]

```
- name: './config-demo.ts'
  config:
    greeting: !!js process.env.DEMO_GREETING ?? 'Hello'
```

`!!js` [illegible] `config` [illegible] `disabled` `disabled: !!js ...` [illegible] `name` `id` `inject` [illegible] loader [illegible]

[illegible] HM `cordis.yml` [illegible]

Image not found or type unknown

6. HMR

[English](#) |

`cordis.yml`

Cordis

Cordis `name` `config`

```yaml
- id: greeter         # stable identity for this entry
  name: './greeter.ts'
- id: consumer
  name: './consumer.ts'
  disabled: true      # keep the entry, skip mounting it
```

`id` Cordis loa `disabled: true` Cordis Cordis PENDING

`isolate` `shell` [Cordis](#)

#

effect `@deepseek-ai/cordis-plugin-hmr`

`tmp/cordis-tutorial` `cordis.yml`

```yaml
- id: logger
  name: '@deepseek-ai/cordis-plugin-logger-console'
- id: timer
  name: '@deepseek-ai/cordis-plugin-timer'
- id: hmr
  name: '@deepseek-ai/cordis-plugin-hmr'
  config:
    root: ['.']
- id: hello
```

```
  name: './hello.ts'
```

[illegible] HMR [illegible] `inject` [illegible] `timer` [illegible] `@deepseek-ai/cordis-plugin-timer` [illegible] PENDING [illegible]

HMR [illegible] Loader [illegible] Node [illegible] loader [illegible] tsx [illegible] Cordis [illegible]

```
node --import tsx ../../vendor/cordis/bin.js
```

[illegible] `hello.ts` [illegible]

```
hello from my first plugin
2026-07-22 15:44:36 [I] hmr watching [ '.' ]
2026-07-22 15:44:39 [I] hmr reload plugin at hello.ts
hello from my EDITED plugin
```

[illegible] `apply` [illegible] `cordis.yml` [illegible] `id` [illegible] Cordis [illegible] `id` [illegible] Cordis [illegible] Cordis [illegible] id

[illegible]

[illegible] `inject` [illegible] PENDING [illegible]

[illegible] `diagnose.ts` [illegible]

```
import { FiberState, type Context } from '@deepseek-ai/cordis'


export const name = 'diagnose'


export function apply(ctx: Context) {
  setTimeout(() => {
    for (const runtime of ctx.registry.values()) {
      for (const fiber of runtime.fibers) {
        if (fiber.state === FiberState.PENDING) {
          console.log(`${fiber.name} is PENDING — a required service is missing`)
        }
      }
    }
  }, 500)
}
```

[illegible] `needs-timer.ts` [illegible]

```
import type { Context } from '@deepseek-ai/cordis'

export const name = 'needs-timer'
export const inject = ['timer']

export function apply(ctx: Context) {
  console.log('needs-timer loaded')
}
```

```
- name: './needs-timer.ts'
- name: './diagnose.ts'
```

[`node --import tsx ../../vendor/cordis/bin.js`, 按 Ctrl-C 停止)：

```
needs-timer is PENDING — a required service is missing
```

`inject: ['timer']` [`- name: '@deepseek-ai/cordis-plugin-timer'` 后，插件就会加载。如果插件既不执行任何操作，也不报告任何内容，请检查其 fiber 状态。不加 P

进入 harness 相同模式用于真实的 harness 服务。

7. 进入 harness

English | 中文

本章会向 harne`tools` 服务注册一个可由模型调用的工具，通过 harness 工具流水线执行它，并观察结果事件。整个示例无需密钥，也不会调用模型。

工具插件

[`greet-tool.ts` [`tmp/cordis-tutorial` 中:

```
import type { Context } from '@deepseek-ai/cordis'
import { brandString } from '@deepseek-ai/dsh-brand'
import { defineTool } from '@deepseek-ai/dsh-tools'
import type { ToolCallId } from '@deepseek-ai/dsh-llm'

export const name = 'greet-tool'
export const inject = ['tools']

export function apply(ctx: Context) {
  ctx.tools.register(defineTool({
    name: 'greet',
    description: 'Greet the named person.',
    parameters: {
      name: { type: 'string', required: true, description: 'Who to greet' },
    },
    output: {
      schema: { type: 'string' },
      render: (_args, value) => [{ type: 'text', text: value }],
    },
    async execute(args) {
      return `Hello, ${args.name}!`
    },
  }))

  // Drive one call through the real execution pipeline, standing in for
  // the model. ToolCallId brands the correlation id a provider would issue.
```

```
  void (async () => {
    const result = await ctx.tools.execute({
      callId: brandString<ToolCallId>('demo-1'),
      name: 'greet',
      arguments: { name: 'Cordis' },
      signal: new AbortController().signal,
    })
    console.log('tool replied:', JSON.stringify(result.content))
  })()
}
```

inject: ['tools'] 3 ctx.tools.register(...) dispose 2 defineTool parameters JSO args h execute output.schema output.render Native renderer

tool-logger.ts tools/result

```
import type { Context } from '@deepseek-ai/cordis'
import type {} from '@deepseek-ai/dsh-tools'

export const name = 'tool-logger'
export const inject = ['tools']

export function apply(ctx: Context) {
  ctx.on('tools/result', (exec, result) => {
    const text = result.content
      .map(block => (block.type === 'text' ? block.text : ''))
      .join('')
    console.log(`[tool-logger] ${exec.name} -> ${text}`)
  })
}
```

import type {} from '@deepseek-ai/dsh-tools' 'tools/result' payload stats.ts 4


```
- name: '@deepseek-ai/dsh-system-prompt'
- name: '@deepseek-ai/dsh-tools'
- name: './tool-logger.ts'
- name: './greet-tool.ts'
```

`@deepseek-ai/dsh-tools` [`systemPrompt` 服务，因为工具需要 [illegible] schema，所以组合中也要列出该服务的提供方。缺少提供方时，工具插件会像

```
node --import tsx ../../vendor/cordis/bin.js
```

```
[tool-logger] greet -> Hello, Cordis!
tool replied: [{"type":"text","text":"Hello, Cordis!"}]
```

logger `tools/result` [`execute` [illegible]回的 promise 兑现之前。两个插件都不知道另一个插件存在，它们由注册表服务和事件连接。

从这里走向完整 agent（智能体）

真实 agent 就是这套组合再加上更多插件：[illegible] `--patch` [illegible] `greet-tool.ts` 即可。

后续可以阅读：

- [`defineTool`，包括呈现和更丰富的 schema。
- 三层 [illegible]arness 如何组织可替换能力。
- [`cordis-surface` 区块：可以注入和监听的所有内容，各在其所属页面上。
- 架构：这些插件所处的系统地图。

Image not found or type unknown