

# TRAЕ (Meet>p)

9月24日 18:30-21:00

← TOKYO  
002



Shibuya Hikarie



Ship Faster with TRAE.

# TRAЕ Meetup@Tokyo

---

# Follow US

# TRAЕ 新たな流れ

# 検索エンジン → 大規模言語モデル

次世代機能により使い方が変化

A screenshot of a Stack Overflow question page. The question title is "How does the main thread transmit information to the child thread (How can I handle message in the run method? )?". It was asked 6 years, 10 months ago and viewed 146 times. The code snippet shown is:

```
    url = Uri.parse("https://api.stackexchange.com/2.2/questions?key=1519ced1eed94d249210840&filter=hasAcceptedAnswer");
    .build();
//将Request封装成call
Call call = okHttpClient.newCall(request);

//4, 执行call, 这个方法是异步请求数据
call.enqueue(new Callback() {
    @Override
    public void onFailure(Call call, IOException e) {
        //失败调用
        Log.e("MainActivity", "onFailure: " );
    }

    @Override
    public void onResponse(Call call, Response response) throws IOException
        //成功调用
        Log.e("MainActivity", "onResponse: " );

    //获取网络访问返回的字符串
    string0 = response.body().string();

    System.out.println("Asynchronous Request Output:ID="+Thread.currentThread().getId());
    Message message = new Message();
    message.obj = string0;
    message.what =TRANSMIT_DATA;

    dataHandler.sendMessage(message);
}
});
```

A screenshot of a ChatGPT interface. The user has asked "java 的并发线程调用怎么写". ChatGPT has responded with:

在 Java 里并发编程常见有几种方式：

1. 继承 Thread 类
2. 实现 Runnable 接口
3. 使用 Callable + Future (有返回值)
4. 使用线程池 ExecutorService (推荐)

我给你几个常用的写法示例。

**1. 继承 Thread**

```
java
public class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println("线程执行: " + Thread.currentThread().getName());
    }
}
```

**public static void main(String[] args) {**

```
MyThread t1 = new MyThread();
MyThread t2 = new MyThread();
t1.start();
t2.start();
```

# チャットボット → IDE 拡張

使い方が変わればツールのあり方も変わる

The image shows two side-by-side interfaces. On the left is the ChatGPT web application, and on the right is a Cloud IDE environment.

**ChatGPT Interface:**

- Header: ChatGPT
- Left sidebar:
  - New chat
  - Search chats
  - Library
  - Sora
  - GPTs
  - Projects (NEW)
- Main area:
  - Inquiry: java 的并发线程调用怎么写
  - Response:
    - 在 Java 里并发编程常见有几种方式:
    - 1. 继承 Thread 类
    - 2. 实现 Runnable 接口
    - 3. 使用 Callable + Future (有返回值)
    - 4. 使用线程池 ExecutorService (推荐)
  - Example code for "1. 继承 Thread":

```
java
public class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println("线程执行: " + Thread.currentThread().getName());
    }

    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        MyThread t2 = new MyThread();
        t1.start();
        t2.start();
    }
}
```
- Bottom bar:
  - + Ask anything
  - ChatGPT can make mistakes. Check important info.

**Cloud IDE Interface:**

- Header: 0520
- Left sidebar: EXPLORER (显示文件夹 0520)
- Right sidebar:
  - TRADE CHAT CLINE (⌘+I) ...
  - 体实现代码:
  - 方式1: 继承Thread类**

```
// 创建一个继承自Thread的子类
class MyThread extends Thread {
    @Override
    public void run() {
        // 线程执行的代码
        System.out.println("线程运行中: " + Thread.currentThread());
        // 执行任务...
    }
}

// 使用方式
public class ThreadDemo {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start(); // 启动线程
    }
}
```
  - 方式2: 实现Runnable接口**

```
'↑↓' to navigate input history, '←→' to insert a new line
```

# より便利に

The screenshot shows a Java development environment with the following details:

- EXPLORER View:** Shows a project named "0520" containing a file "test.java".
- Code Editor:** Displays the file "test.java" with the following content:

```
test.java
1 cmd-l to open Augment.
```
- Right Panel:** Shows a message about creating a "ThreadDemo.java" file to demonstrate Java thread implementation methods. It also displays a failed build status for "ThreadDemo.java".
- Code Completion:** A tooltip is shown with the following Java code example:

```
// 创建一个继承自Thread的子类
class MyThread extends Thread {
    @Override
    public void run() {
        // 线程执行的代码
        System.out.println("线程运行中: " + Thread.currentThread().getN
        // 执行任务...
    }
}

// 使用方式
public class ThreadDemo {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start(); // 启动线程
    }
}
```
- Bottom Status Bar:** Shows the status "Java: Ready" and "Cloud IDE" login information.

より正確に

The screenshot shows a Java IDE interface with two code editors and a sidebar.

**Code Editor 1 (Left): ThreadDemo.java**

```
1 // 创建一个继承自Thread的子类
2 class MyThread extends Thread {
3     @Override
4     public void run() {
5         // 线程执行的代码
6         System.out.println("线程运行中: " + Thread.currentThread().getName());
7         // 执行任务...
8     }
9 }
10
11 // 使用方式
12 public class ThreadDemo {
13     public static void main(String[] args) {
14         MyThread thread = new MyThread();
15         thread.start(); // 启动线程
16     }
17 }
```

**Code Editor 2 (Right): Java多线程调用实现方式**

Java中实现多线程有几种常见的方法，下面我将为您展示这些方法的具体实现代码：

**方式1：继承Thread类**

```
// 创建一个继承自Thread的子类
class MyThread extends Thread {
    @Override
    public void run() {
        // 线程执行的代码
        System.out.println("线程运行中: " + Thread.currentThread().getName());
        // 执行任务...
    }
}

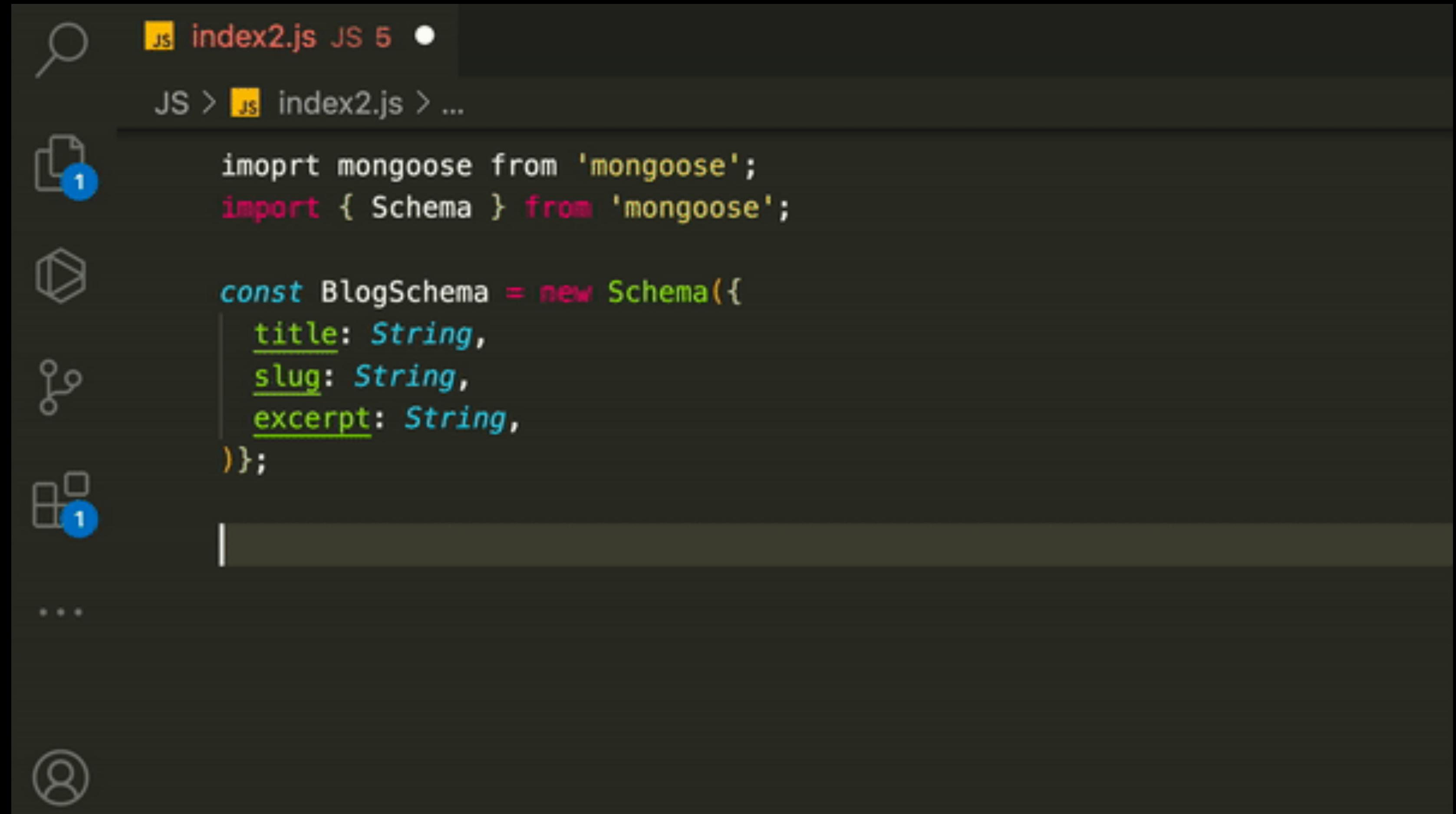
// 使用方式
public class ThreadDemo {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start(); // 启动线程
    }
}
```

**方式2：实现Runnable接口**

```
① ThreadDemo.java 12-13 ② ThreadDemo.java ③ 0520 #
```

Doubao-Seed-1.6 ^ ↑

より多くの  
シーンで



index2.js JS 5 ●

JS > index2.js > ...

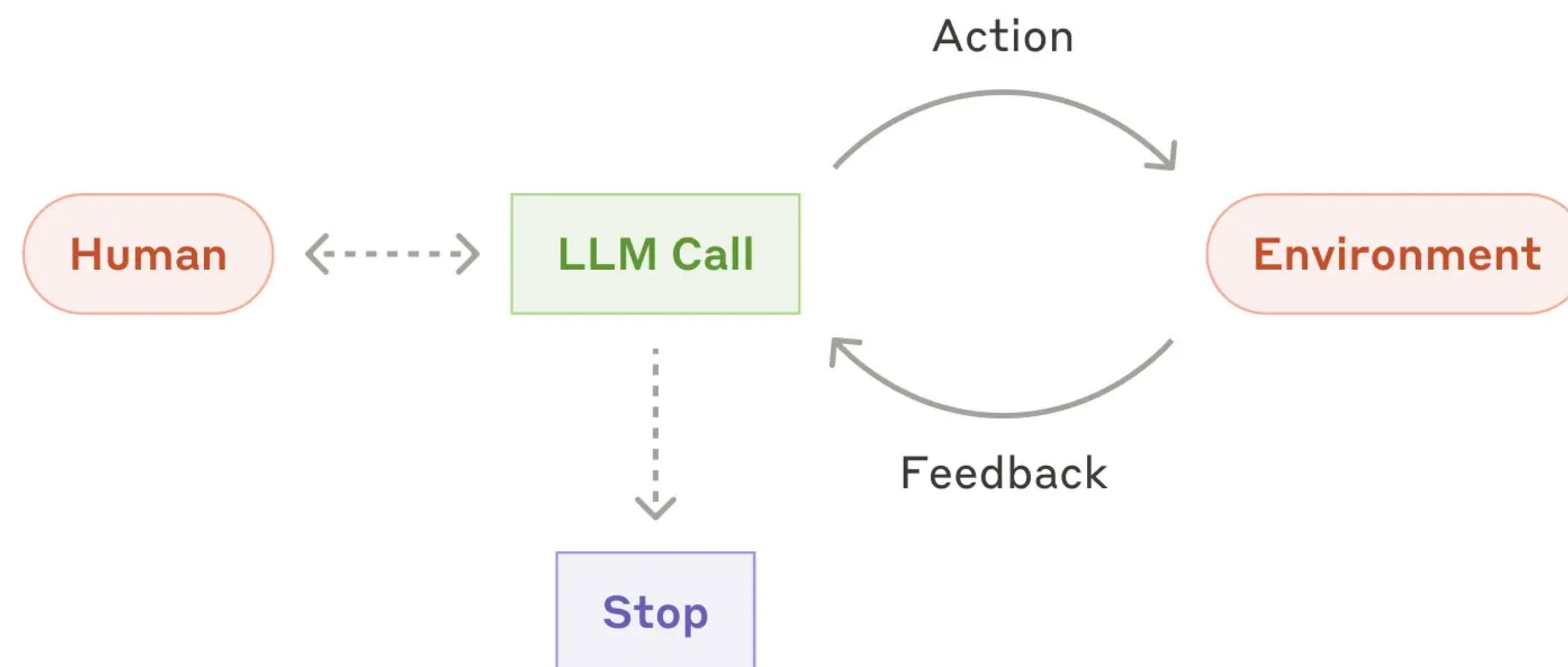
```
import mongoose from 'mongoose';
import { Schema } from 'mongoose';

const BlogSchema = new Schema({
  title: String,
  slug: String,
  excerpt: String,
});

...
```

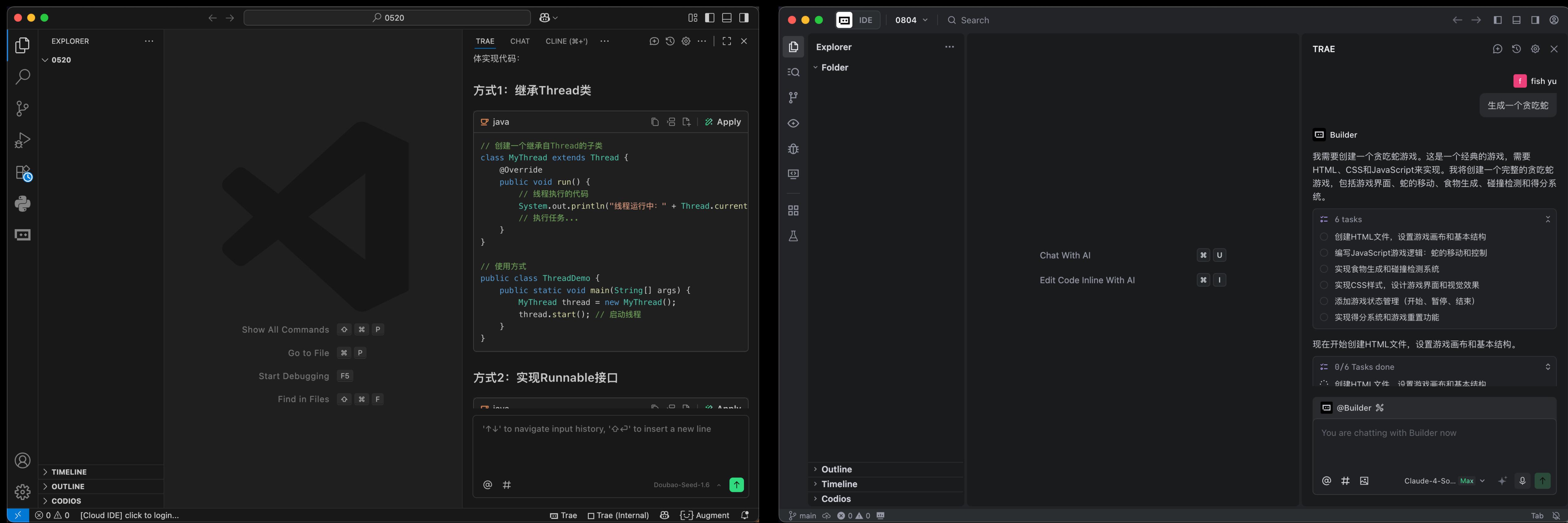
# チャット → エージェント

次世代機能により使い方が変化

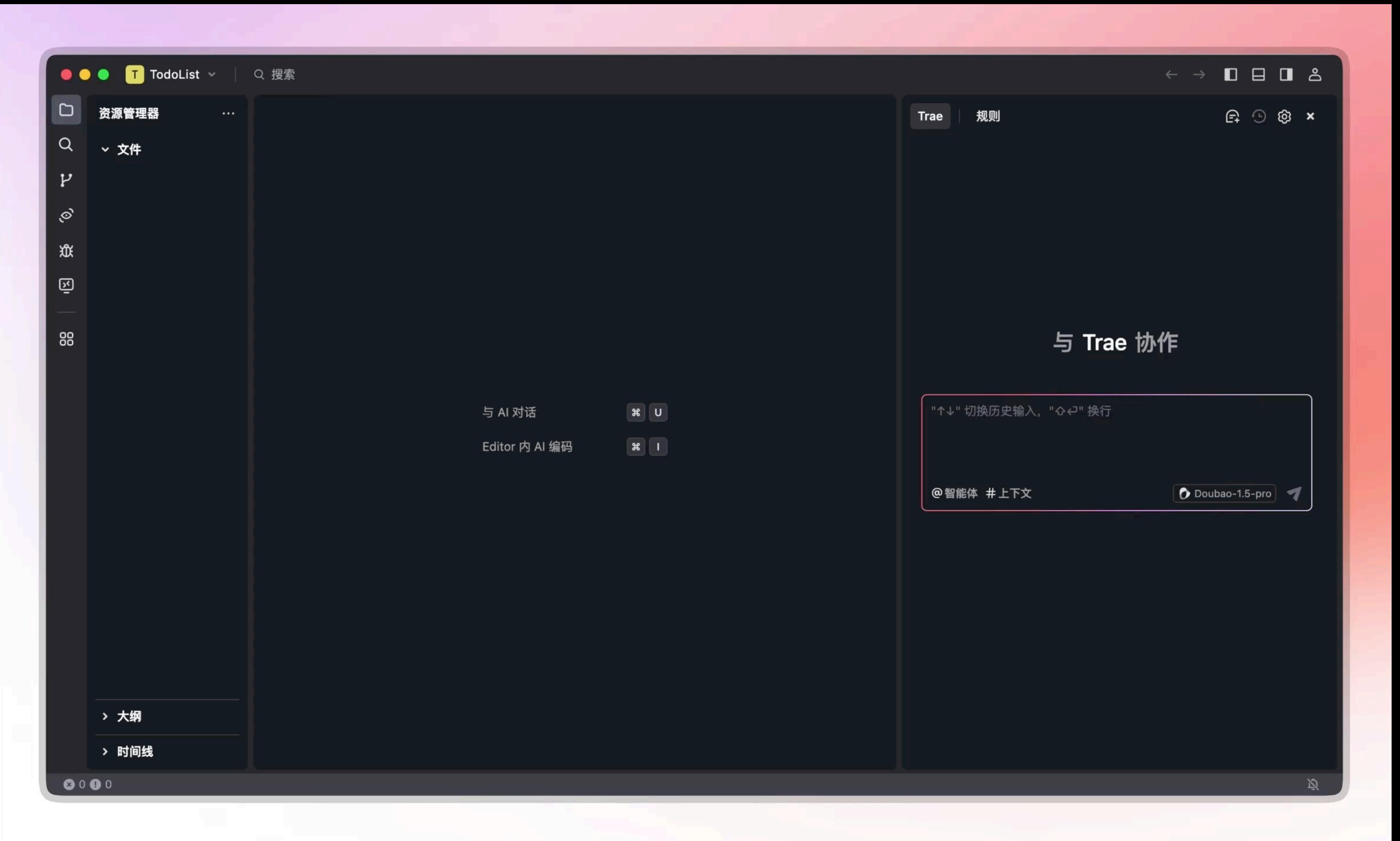


# IDE 拡張 → ネイティブ IDE

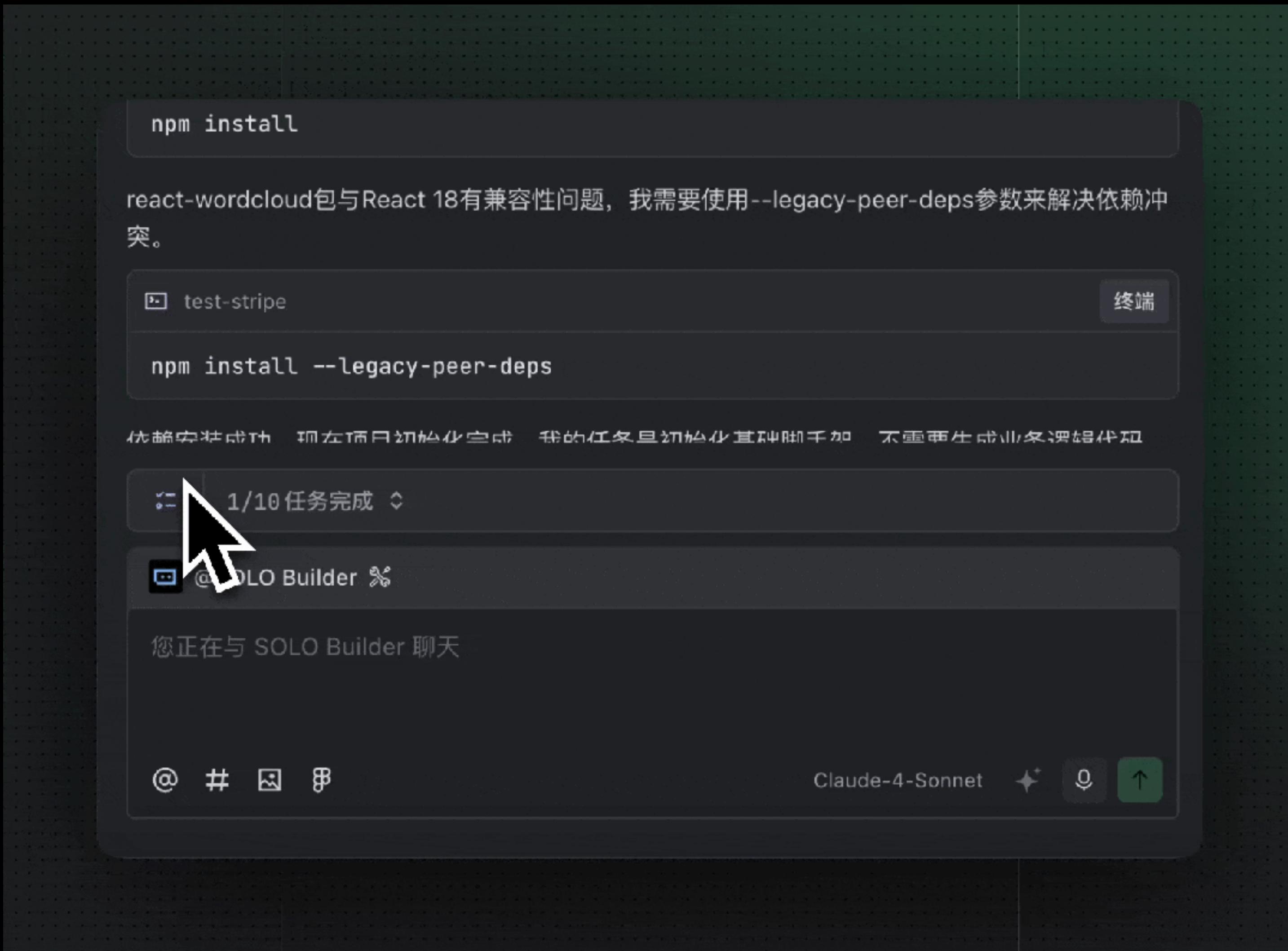
使い方が変わればツールのあり方も変わる



より自主的・  
自動的に



より自主的・  
自動的に



# コンテキスト・ウィンドウ

少量 → 大容量

(すぐに忘れる)

(長期記憶)

# ツール数

少 → 多 (ネイティブ機能呼び出し)

(不安定)

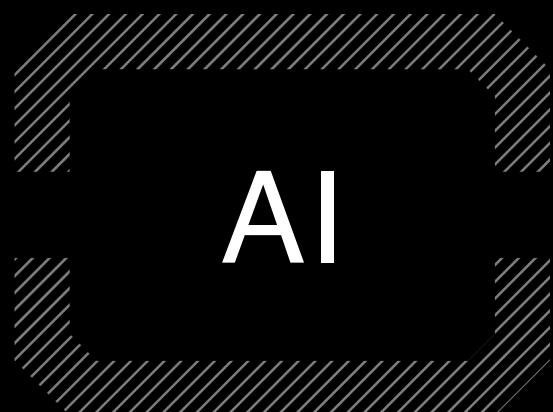
(安定)

ネイティブ IDE → ?

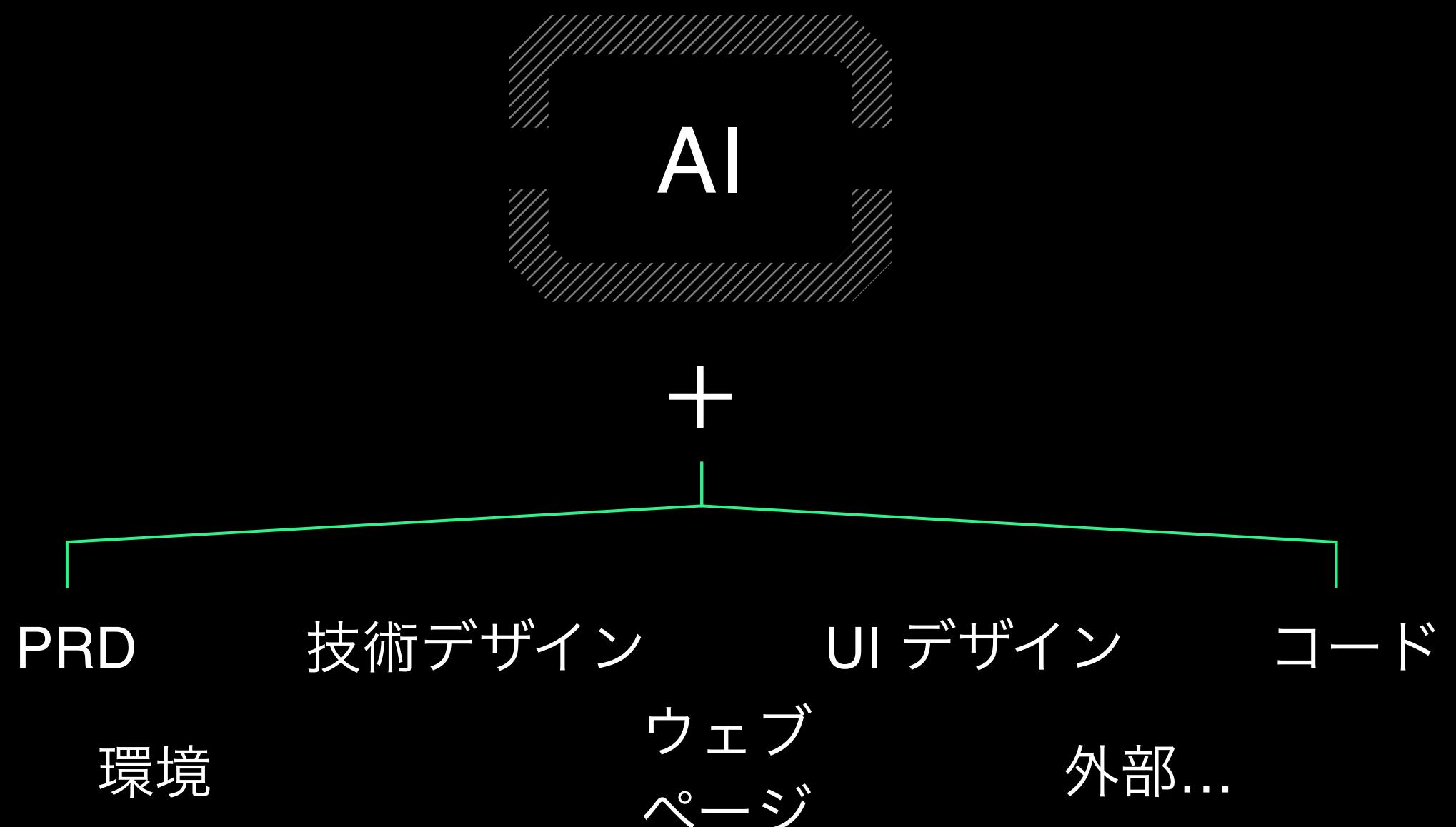
## IDE に AI を統合

IDE  
(コード/ファイル/フォルダー/  
コードベース)

+



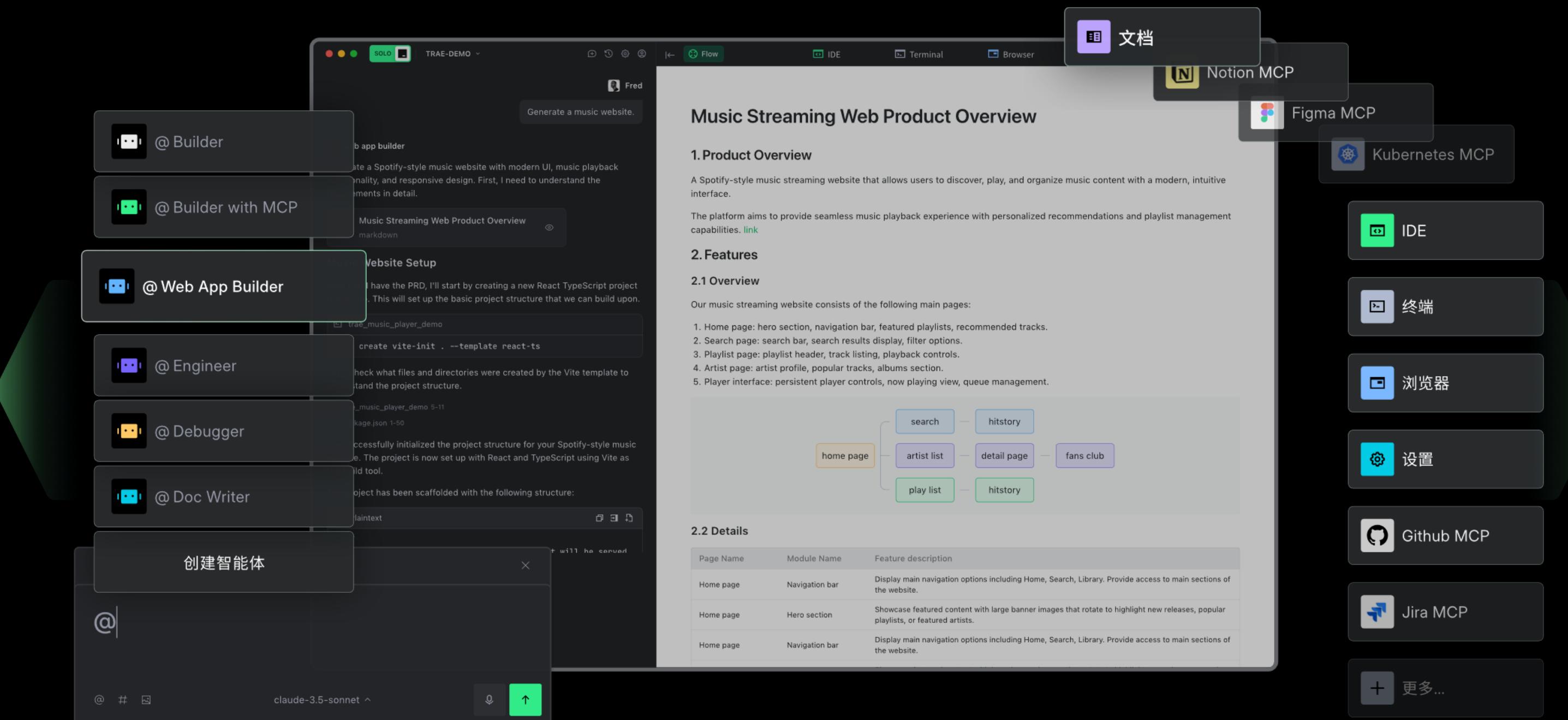
## AI にコンテキストを統合



# SOLO

AI 駆動型デザインパターンにより、人間が集中すべきポイントを変える

AI エージェント



コンテキスト

# シナリオベースによる ツールの再設計

实时跟随 编辑器 终端 浏览器 文档

查看变更 复制

## 1 Product Overview

公司内部采购系统后台，用于管理物料、供应商、合同、价格和财务参数的综合管理平台。

- 解决企业采购流程中的数据管理、价格控制和财务配置等核心问题，提供统一的后台管理界面。
- 目标是提升采购效率，规范采购流程，实现采购数据的集中化管理。

## 2. Core Features

### 2.1 User Roles

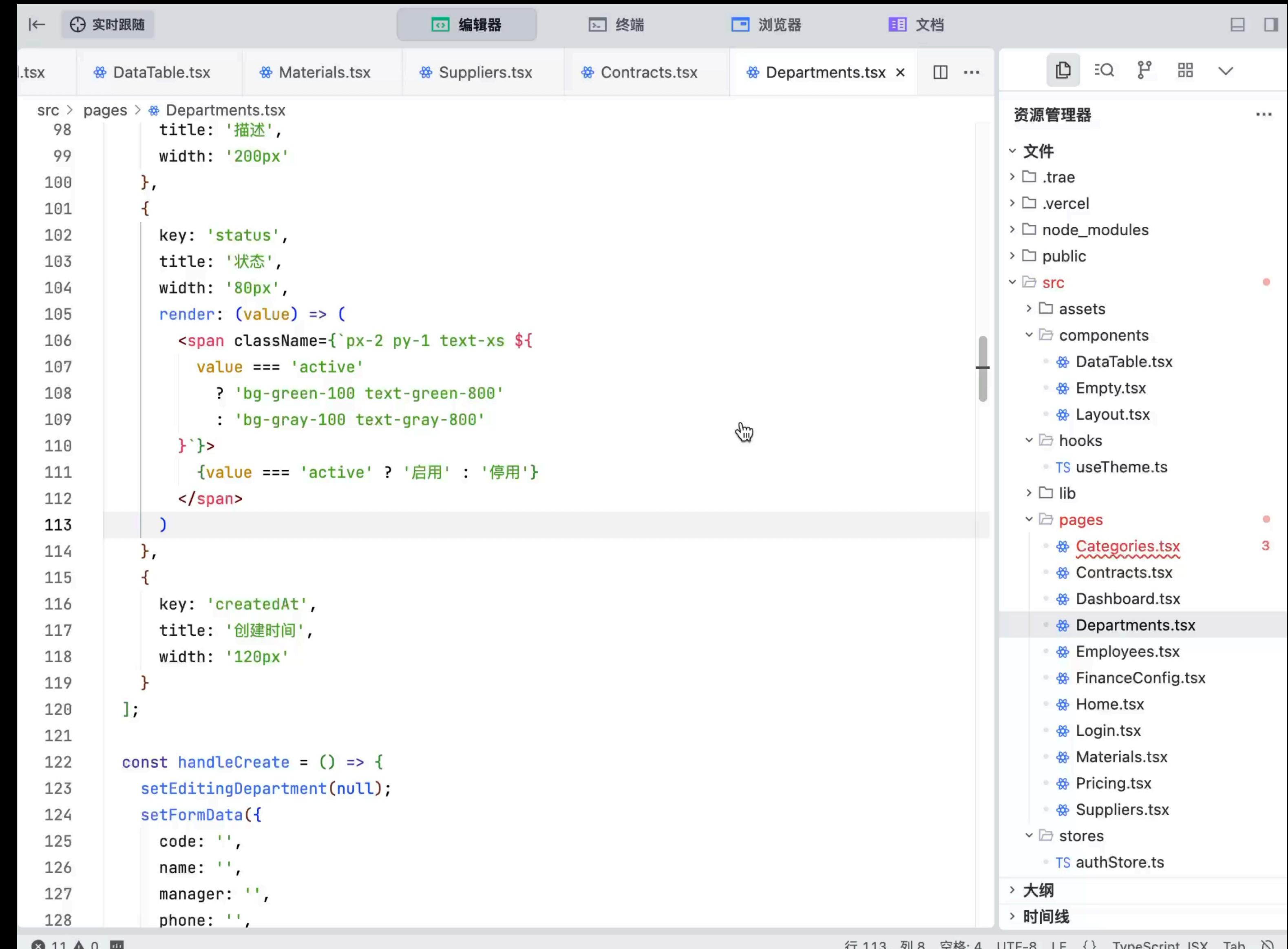
Role	Registration Method	Core Permissions
管理员	系统管理员分配	可访问所有功能模块，拥有完整的增删改查权限，可管理用户权限
普通用户	管理员创建账号	可访问所有功能模块，仅拥有查看权限，无法进行新增、编辑、删除操作

### 2.2 Feature Module

我们的采购系统后台包含以下主要页面：

- Dashboard页面：数据概览、统计图表、快捷操作入口。
- 合同管理页面：合同列表、合同详情、合同状态管理。
- 部门管理页面：部门信息维护、组织架构管理。
- 员工管理页面：员工信息管理、权限分配。
- 供应商管理页面：供应商档案、资质管理、评估记录。

# シナリオベースによる ツールの再設計



The screenshot shows a code editor interface with a dark theme. The main area displays a TypeScript file named `Departments.tsx`. The code defines a table structure with columns for description, status, created at, and handle create. The status column uses a conditional rendering approach with a ternary operator to determine the background color and text color based on the value ('active' or 'inactive'). The code editor includes tabs for other files like `DataTable.tsx`, `Materials.tsx`, `Suppliers.tsx`, `Contracts.tsx`, and `Departments.tsx`. A sidebar on the right shows a file tree under the `src` directory, with `Departments.tsx` highlighted. The bottom status bar indicates the current line (113) and column (8).

```
src > pages > Departments.tsx
  98     title: '描述',
  99     width: '200px'
100   },
101   {
102     key: 'status',
103     title: '状态',
104     width: '80px',
105     render: (value) => (
106       <span className={`${`px-2 py-1 text-xs ${value === 'active' ? 'bg-green-100 text-green-800' : 'bg-gray-100 text-gray-800`} ${value === 'active' ? 'font-weight-bold' : ''}`}>
107         {value === 'active' ? '启用' : '停用'}
108       </span>
109     )
110   },
111   {
112     key: 'createdAt',
113     title: '创建时间',
114     width: '120px'
115   }
116 ];
117
118 const handleCreate = () => {
119   setEditingDepartment(null);
120   setFormData({
121     code: '',
122     name: '',
123     manager: '',
124     phone: ''
```

# シナリオベースによる ツールの再設計

The screenshot shows a software interface for 'SOLO Builder' with the following components:

- Top Bar:** Shows the project name 'ordersys' and several tabs: 编辑器 (Editor), 终端 (Terminal), 浏览器 (Browser), and 文档 (Documentation).
- Chat Log:** A window titled 'SOLO Builder' with a message from 'Bonnie978': '采购系统后台需求文档.md 确认, 开始开发!' (Confirm purchase system backend requirements document, start development!). Below it, a message from 'SOLO Builder' says: '用户确认需求文档并要求开始开发。这是一个空项目, 需要先初始化项目环境, 然后基于需求文档开始编码实现采购系统后台。我将先设置项目基础架构。' (User confirms requirements document and requests to start development. This is an empty project, needs to initialize project environment first, then implement procurement system backend based on requirements document. I will set up the basic project structure first.)
- Terminal Logs:** A terminal window titled 'ordersys' showing the command: 'pnpm create vite-init@latest . --template react-ts'. Below it, another terminal window titled 'ordersys' shows the command: 'pnpm install'. The logs indicate the creation of a Vite project and the installation of dependencies.
- Browser Tab:** A tab labeled 'ordersys' showing a placeholder page with the text: '参考了1个上下文' (Referenced 1 context).
- Right Panel:** A terminal window titled 'node' showing the output of a Vite server: 'VITE v6.3.5 ready in 684 ms'. It also lists port conflicts and HMR updates for files like 'Dashboard.tsx', 'Layout.tsx', and 'index.css'.

# シナリオベースによる ツールの再設計

实时跟随

编辑器 终端 浏览器 文档

http://localhost:5175/dashboard

选择元素 部署

## 采购系统

### Dashboard

欢迎回来，系统管理员！这里是您的数据概览。

合同总数 156 +12%

供应商数量 89 +5%

物料种类 1,234 +8%

采购总额 ¥2,456,789 +15%

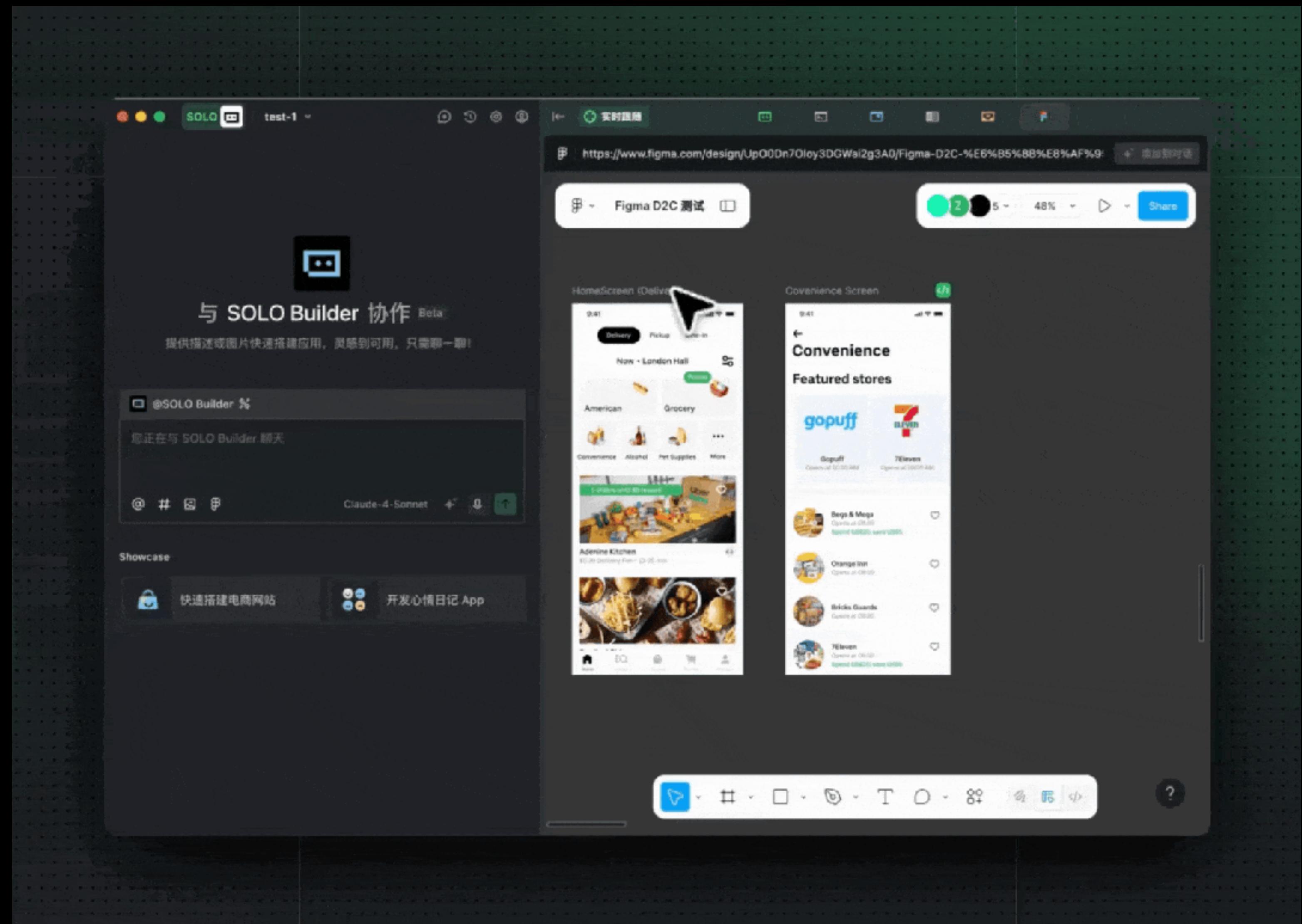
#### 合同状态分布



退出登录

当前无控制台错误

# シナリオベースによる ツールの再設計



# シナリオベースによる ツールの再設計

The screenshot shows the FLOW integration interface with a dark theme. At the top, there's a header bar with icons for back, forward, and other navigation. Below it, the main title is "Integrations". The interface is organized into several sections:

- Supabase Service**: A card for "Supabase" featuring its logo (a green lightning bolt), the text "For user authentication and data storage", and a "Connect" button.
- AI Services**: Three cards:
  - "OpenAI": Logo (a white swirl icon), text "A powerful conversational AI service for various tasks", and a "Configure" button.
  - "Anthropic": Logo (an "AI" icon), text "A conversational AI service focused on safety and usability", and a "Configure" button.
  - "Gemini": Logo (a blue starburst icon), text "An AI service by Google with search integration and multimodal capabilities", and a "Configure" button.
- Payment Service**: A card for "Stripe" featuring its logo (a blue 'S' icon), the text "Integrate Stripe to easily add payments and subscriptions to your product", and a "Configure" button.

ネイティブ IDE → ?

一緒に未来を探しにいきませんか？

The Real AI Engineer.



# TRAЕ Cue - コード補完の再定義

Jacob - TRAE テクニカルエキスパート

01

Cue の進化

02

典型的な使用  
シーンの紹介

03

主な課題

04

今後の展望

# Cue の進化



「Cue」とは？

「Cue」と「Tab」の違いは？

Cue (**Context UnderStanding Engine**) は、TRAE が提供する AI プログラミング支援ツールです。コード補完、複数行の編集、編集箇所予測、スマートインポート、スマートリネームなどの機能があります。

# 過去のマイルストーン

TRAECue は、コード補完から、その機能をより広範な「次のアクション予測（Next Action Prediction）」へと拡張されました。これは、TRAECue はコードを書くためだけのツールではなく、開発フローのさまざまな場面で活用できるようになったのです。

2024年6月	2024年9月	2025年6月	Now
MarsCode が中国国内で正式にリリース コード補完をサポート	マルチ編集機能のリリース カーソル周辺のコード編集とリライトをサポートし、単一ファイル内のジャンプ(nextline)を初期段階でサポート	Cue 1.0 のリリース コード補完と編集を单一のモデルに統合 大規模リライトに基づく次の編集箇所予測を実現	Cue 1.2 独立した Locator 方式の導入による、リポジトリ規模のクロスファイル編集箇所予測を実現

# 典型的な使用シーケンスの紹介



1

## コード補完

ユーザーの現在のカーソル位置に基づいて  
コード補完を `ghost-text` で表示

2

## マルチ編集

ユーザーの編集履歴に基づいてカーソル周辺  
の行に対する編集提案を `Diff + ポップアップ`  
で表示

3

## スマートリライト

コンテキストに基づいて入力内容をスマート  
リライト

4

## クロスファイルジャンプ

ファイルへのスマートジャンプと提案表示

5

## スマートインポート

スマートに認識して依存パッケージをイン  
ポート

6

## スマートリネーム

スマートに認識してエンティティ名をリライ  
ト

## コード補完

```

191 // Then calls function f and returns its result if o[F] contains value.
192 // Otherwise, Nil[T]() is returned.
193 //
194 // HINT: This function is similar to the Rust's std::option::Option.and_then
195 func Then[F, T any](o O[F], f func(F) O[T]) O[T] {
196     if !o.ok {
197         return Nil[T]()
198     }
199     return f(o.val)
200 }
201 
```

## スマートインポート

```

161 private toKey({ syncResource, profile }: IUserDataSyncResource): string {
162     return `${profile.id}:${resource}`;
163 }
164 //
165 //
166 // private readonly conflictsDisposables
167 private onDidChangeConflicts(conflicts: IUserDataSyncResourceConflicts[]) {
168     this.updateGlobalActivityBadge();
169     this.registerShowConflictsAction();
170     if (!this.userDataSyncEnablementService.isEnabled()) {
171         return;
172     }
173     if (conflicts.length) {
174         // Clear and dispose conflicts those were cleared
175         for (const [key, disposable] of this.conflictsDisposables.entries()) {
176             if (!conflicts.some(conflict => this.toKey(conflict) === key)) {
177                 this.conflictsDisposables.delete(key);
178             } else {
179                 disposable.dispose();
180             }
181         }
182     }
183 }
184 
```

## スマートリライト

```

56 // Of creates an optional value with type T from tuple (T, bool).
57 //
58 // NOTE: If the given bool is false, the value of T MUST be zero value of T,
59 // otherwise this will be an undefined behavior.
60 //
61 func Of[T any](v T, ok bool) O[T] {
62     return O[T]{v, ok}
63 }
64 
```

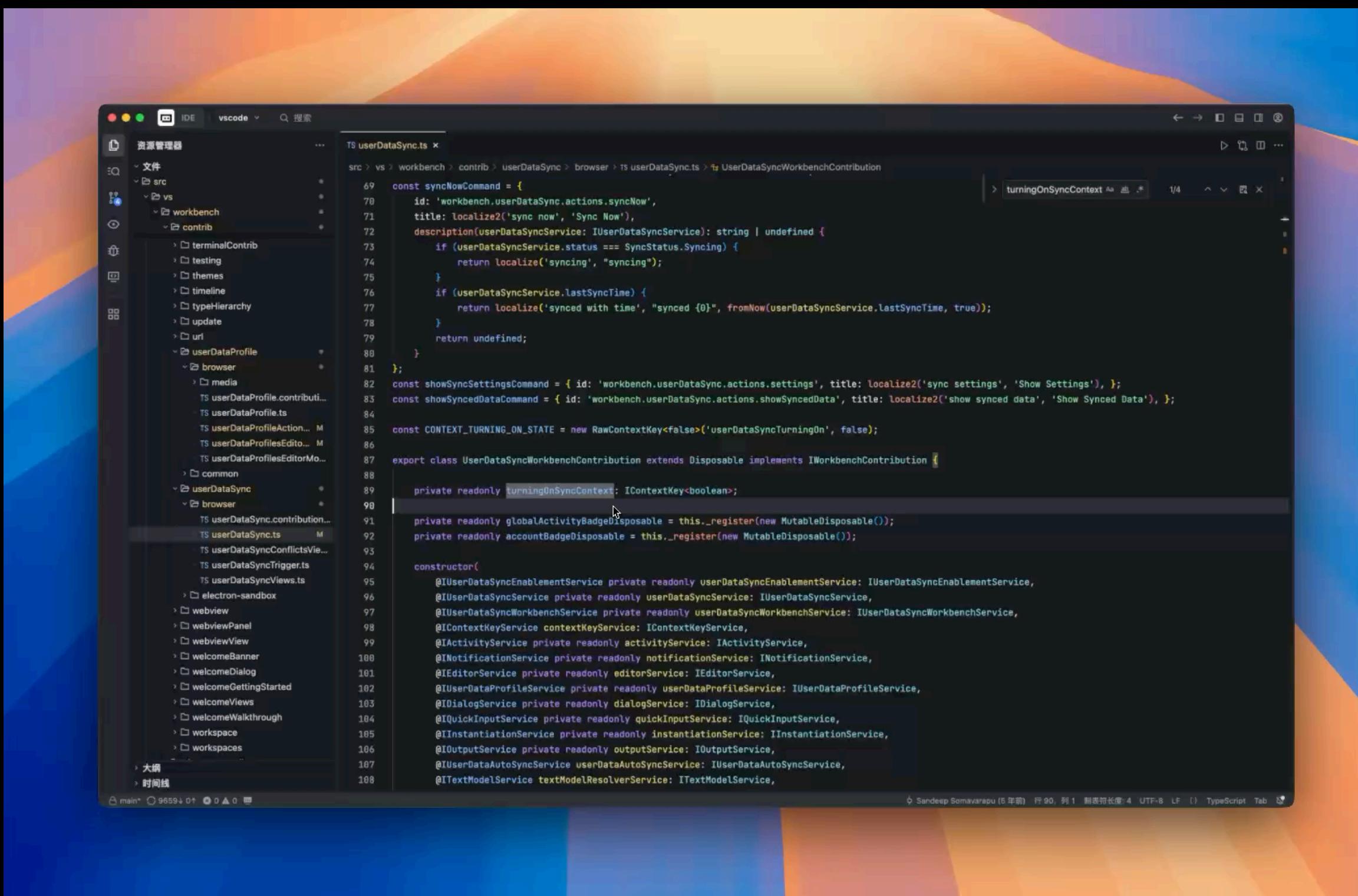
## スマートリネーム

```

192 func TryFilterMap[F, T any](s []F, f func(F) (T, error)) []T {
193     ret := make([]T, 0, len(s)/2)
194     for _int, vF := range s {
195         r, err := f(v)
196         if err != nil {
197             continue // ignored
198         }
199         ret = append(ret, r)
200     }
201     return ret
202 }
```

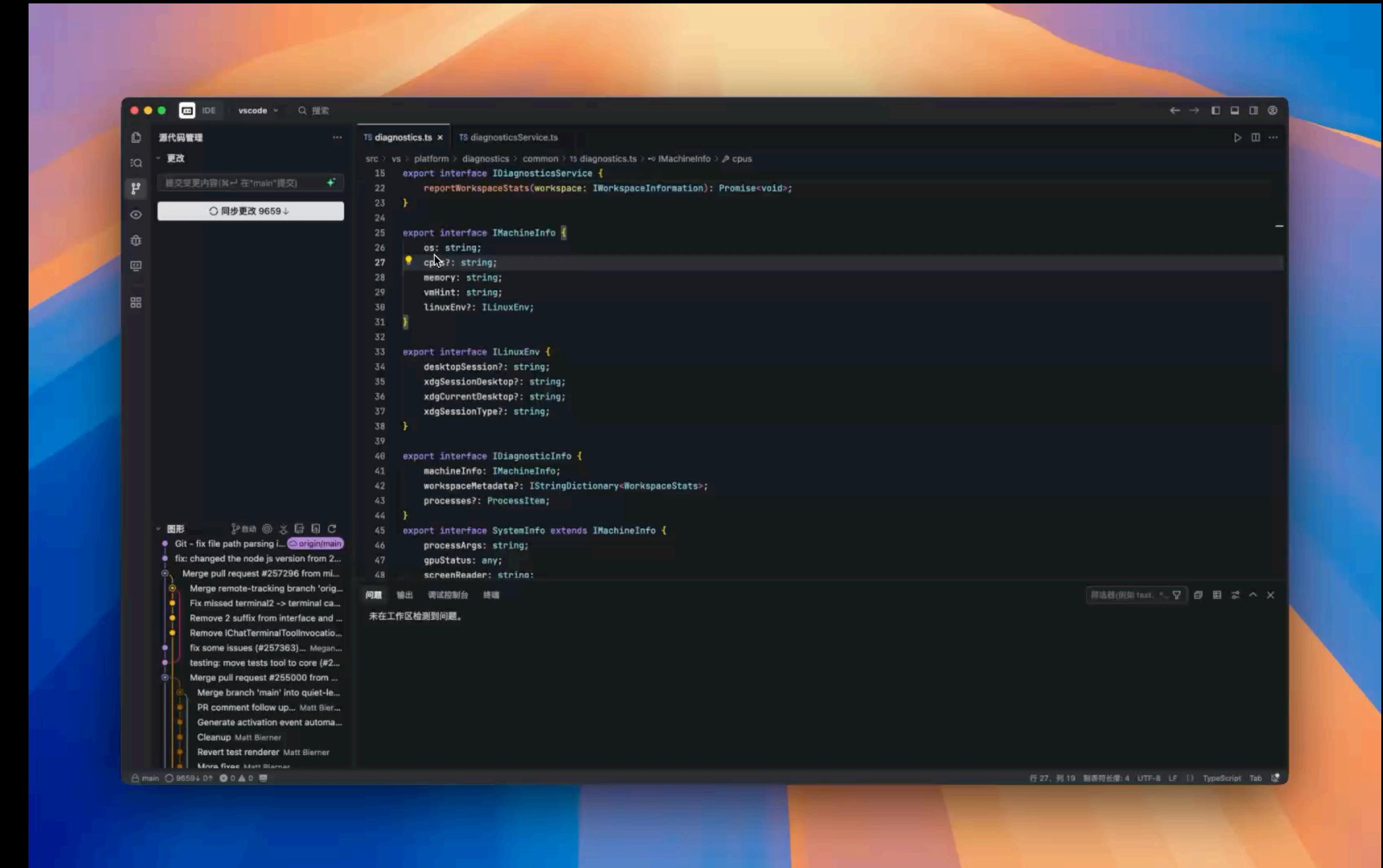
# リポジトリ全体でのクロスファイルジャンプ

## ファイル内ジャンプ



The screenshot shows the VS Code interface with the file `userDataSync.ts` open. The search bar at the top right has the text "turningOnSyncContext" entered. The code editor displays a portion of the `UserDataSyncWorkbenchContribution` class, specifically the `turningOnSyncContext` method. The sidebar on the left shows the project structure, including the `src` folder and its sub-directories like `vs`, `workbench`, and `contrib`.

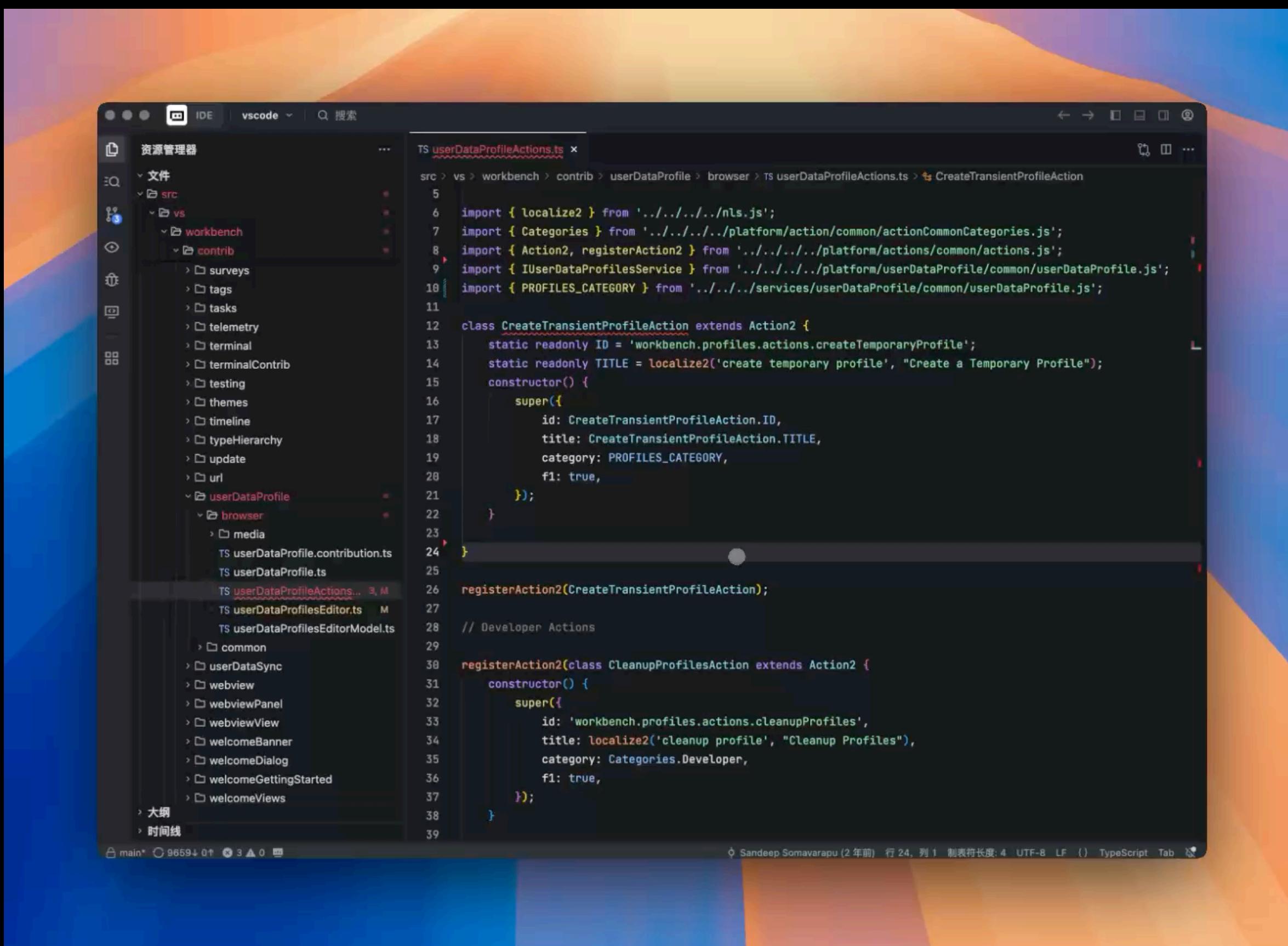
## クロスファイルジャンプ



The screenshot shows the VS Code interface with the file `diagnosticsService.ts` open. A search bar at the top right has the text "cpu" entered. A modal dialog in the center says "同步更改 9659". The code editor displays the `IDiagnosticsService` interface definition. The sidebar on the left shows the project structure, including the `src` folder and its sub-directories like `vs`, `platform`, and `diagnostics`.

# スマートインポート

TypeScript



```
src > vs > workbench > contrib > userDataProfile > browser > TS(userDataProfileActions.ts) > CreateTransientProfileAction
5
6 import { localize2 } from '../../../../../nls.js';
7 import { Categories } from '../../../../../platform/action/common/actionCommonCategories.js';
8 import { Action2, registerAction2 } from '../../../../../platform/actions/common/actions.js';
9 import { IUserDataProfilesService } from '../../../../../platform/userDataProfile/common/userDataProfile.js';
10 import { PROFILES_CATEGORY } from '../../../../../services/userDataProfile/common/userDataProfile.js';

11 class CreateTransientProfileAction extends Action2 {
12     static readonly ID = 'workbench.profiles.actions.createTemporaryProfile';
13     static readonly TITLE = localize2('create temporary profile', "Create a Temporary Profile");
14     constructor() {
15         super({
16             id: CreateTransientProfileAction.ID,
17             title: CreateTransientProfileAction.TITLE,
18             category: PROFILES_CATEGORY,
19             f1: true,
20         });
21     }
22 }

23 registerAction2(CreateTransientProfileAction);

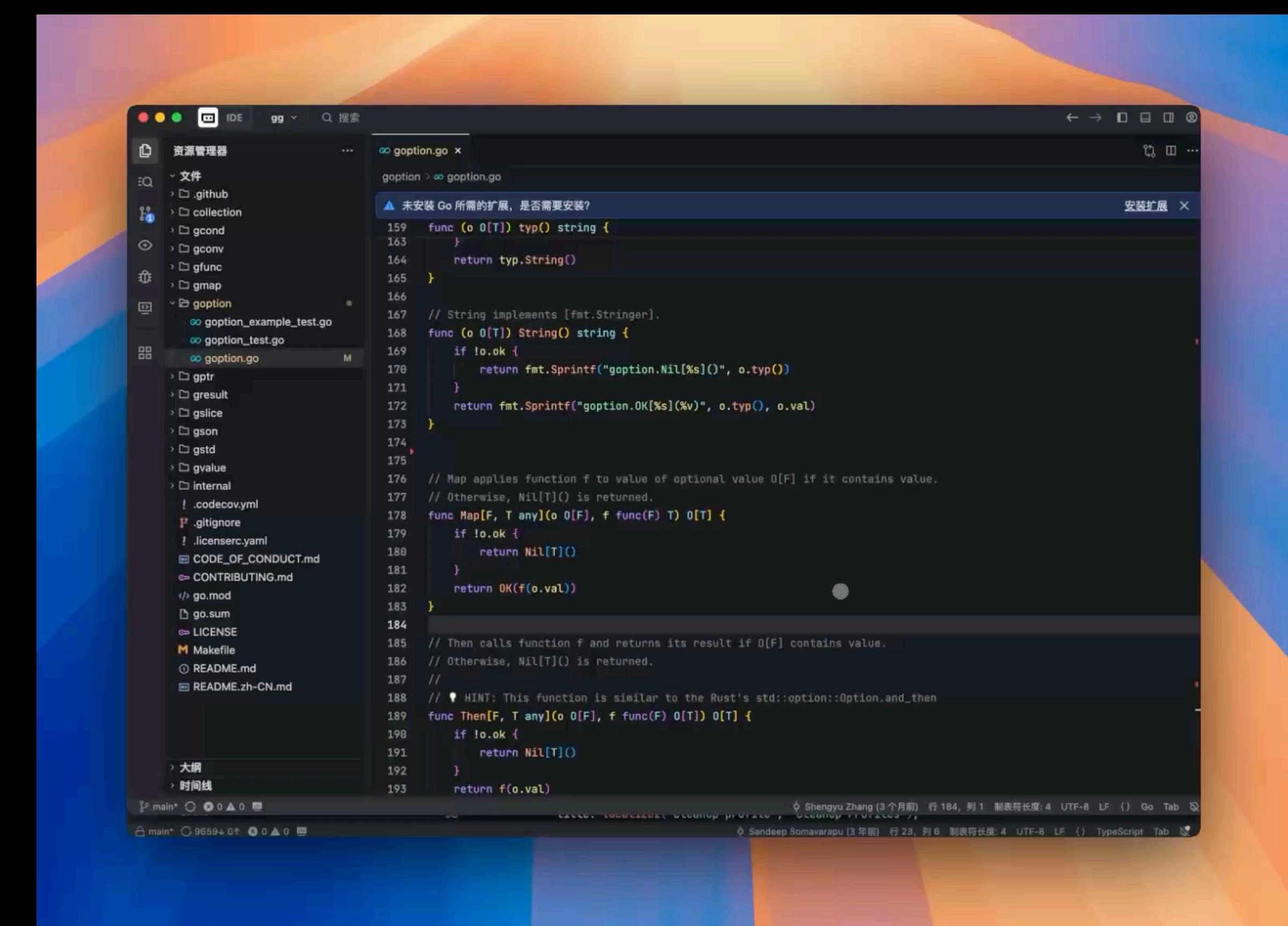
24

25 // Developer Actions

26 registerAction2(class CleanupProfilesAction extends Action2 {
27     constructor() {
28         super({
29             id: 'workbench.profiles.actions.cleanupProfiles',
30             title: localize2('cleanup profile', "Cleanup Profiles"),
31             category: Categories.Developer,
32             f1: true,
33         });
34     }
35 });

36 
```

Go



```
func (o O[T]) typ() string {
    if !o.ok {
        return typ.String()
    }
}

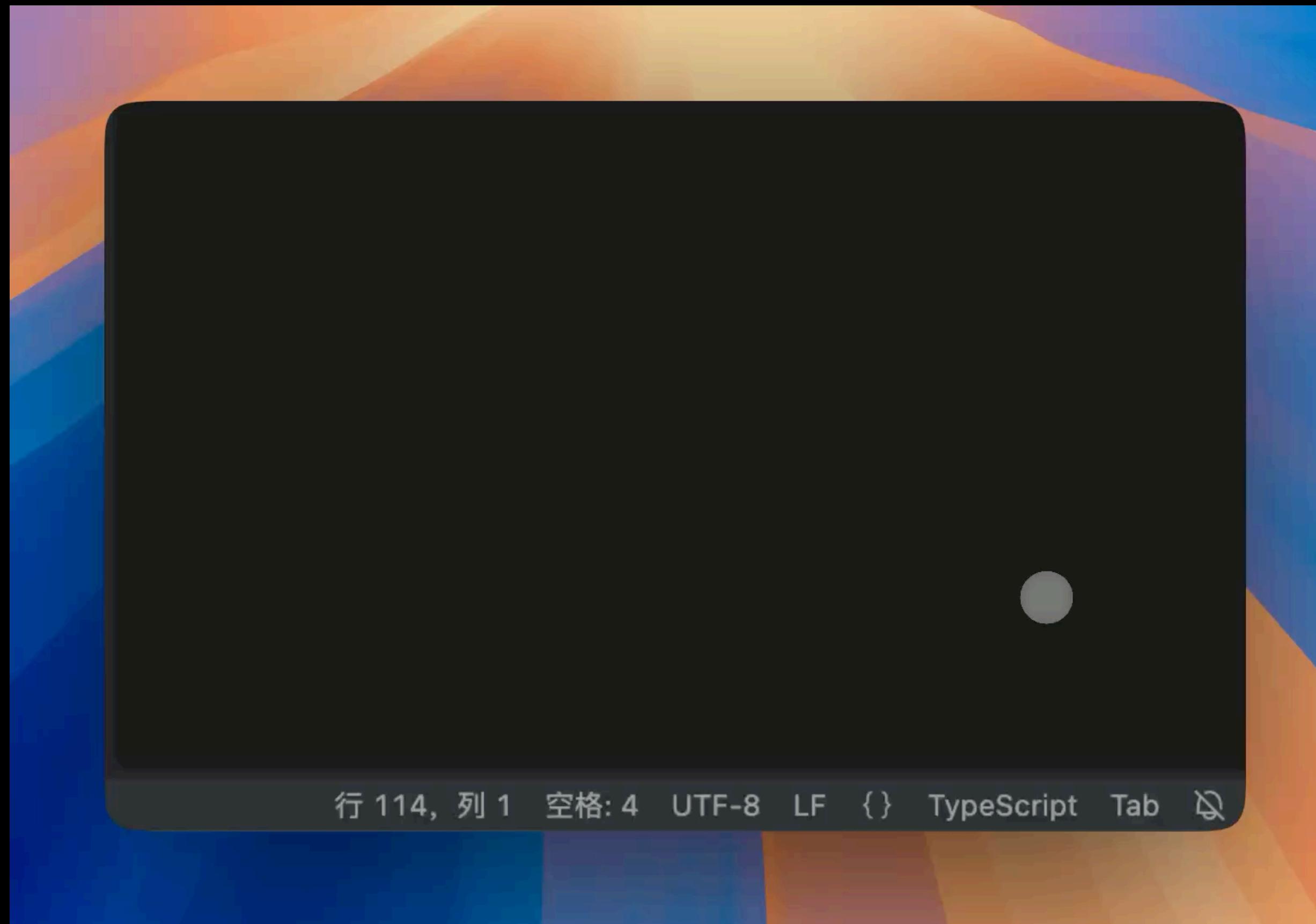
// String implements [fmt.Stringer].
func (o O[T]) String() string {
    if !o.ok {
        return fmt.Sprintf("gooption.Nil[%s]()", o.typ())
    }
    return fmt.Sprintf("gooption.OK[%s](%v)", o.typ(), o.val)
}

// Map applies function f to value of optional value O[F] if it contains value.
// Otherwise, Nil[T]() is returned.
func Map[F, T any](o O[F], f func(F) T) O[T] {
    if !o.ok {
        return Nil[T]()
    }
    return OK(f(o.val))
}

// Then calls function f and returns its result if O[F] contains value.
// Otherwise, Nil[T]() is returned.
func Then[F, T any](o O[F], f func(F) O[T]) O[T] {
    if !o.ok {
        return Nil[T]()
    }
    return f(o.val)
}
```

## 実用的なヒント

Snooze



ツールバー

A screenshot of the tool bar in the code editor. The 'Tab' button is highlighted with a green outline, and a tooltip above it says '接受' (Accept). Another tooltip below it says '接受行' (Accept Line) with a small icon. The code editor window shows a snippet of TypeScript code:

```
112
113
114 public getPlanets() {
115     return this.planets;
}
```

The status bar at the bottom of the code editor shows the current file type as 'TypeScript'.

# 実用的なヒント

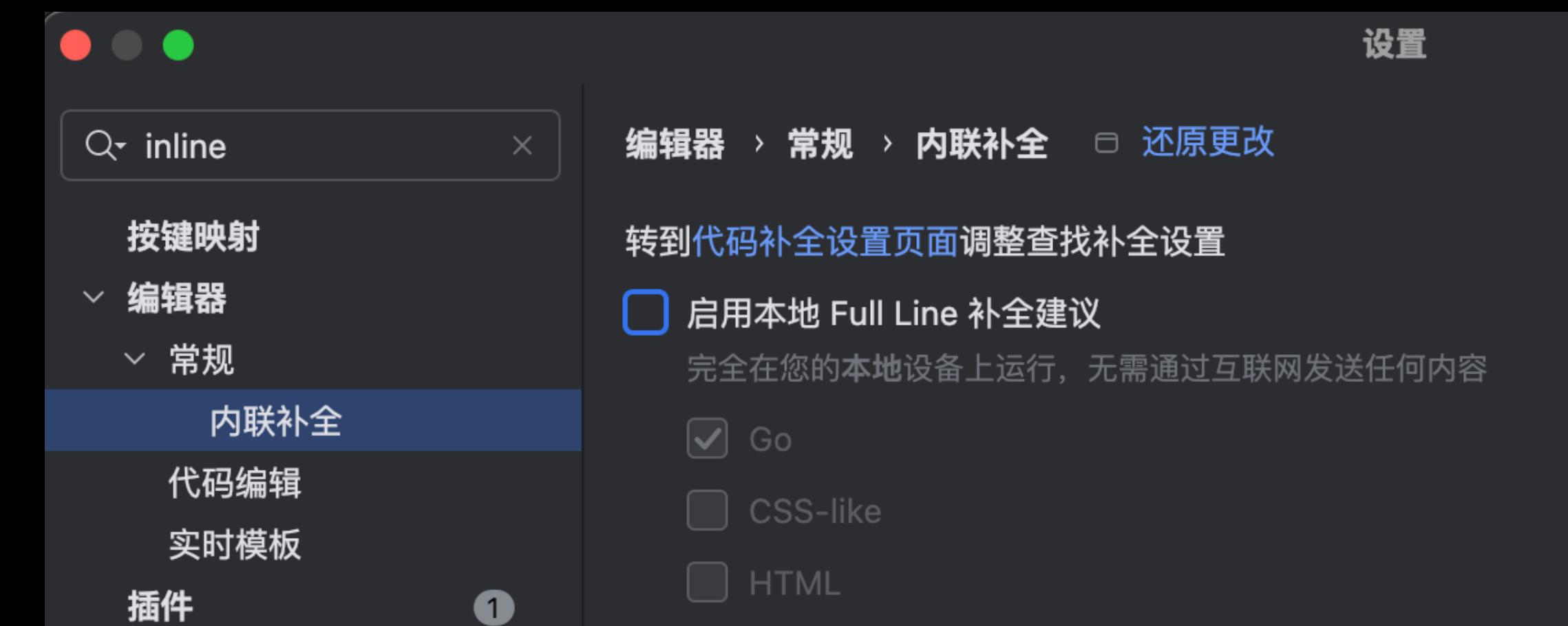
## Cue と LSP のショートカットキーの競合

```

TS userDataProfileActions.ts
src > vs > workbench > contrib > userDataProfile > browser > TS userDataProfileActions.ts > CreateTransientProfileAction > run
  9 import { ServicesAccessor } from '../../../../../platform/instantiation/common/instantiation.js';
10 import { IUserDataProfilesService } from '../../../../../platform/userDataProfile/common/userDataProfile.js';
11 import { IUserDataProfileManagementService, PROFILES_CATEGORY } from '../../../../../services/userDataProfile/common/
    userDataProfile.js';
12
13 class CreateTransientProfileAction extends Action2 {
14   static readonly ID = 'workbench.profiles.actions.createTemporaryProfile';
15   static readonly TITLE = localize2('create temporary profile', "Create a Temporary Profile");
16   constructor() {
17     super({
18       id: CreateTransientProfileAction.ID,
19       title: CreateTransientProfileAction.TITLE,
20       category: PROFILES_CATEGORY,
21       f1: true,
22     });
23   }
24
25   async run(accessor: ServicesAccessor) {
26     return accessor.get(IUserDataProfileManagementService).createAndEnterTransientProfile();
27   }
28 }
29
30 registerAction2(CreateTransientProfileAction);
31
32 // Developer Actions
33
34 registerAction2(class CleanupProfilesAction extends Action2 {
35   constructor() {
36     super({
37       id: 'workbench.profiles.actions.cleanupProfiles',
38       title: localize2('cleanup profile', "Cleanup Profiles"),
39       category: Categories.Developer,
40       f1: true,
41     });
42   }
43
44   async run(accessor: ServicesAccessor) {
45     return accessor.get(IUserDataProfilesService).cleanUp();
46   }
47 });
48

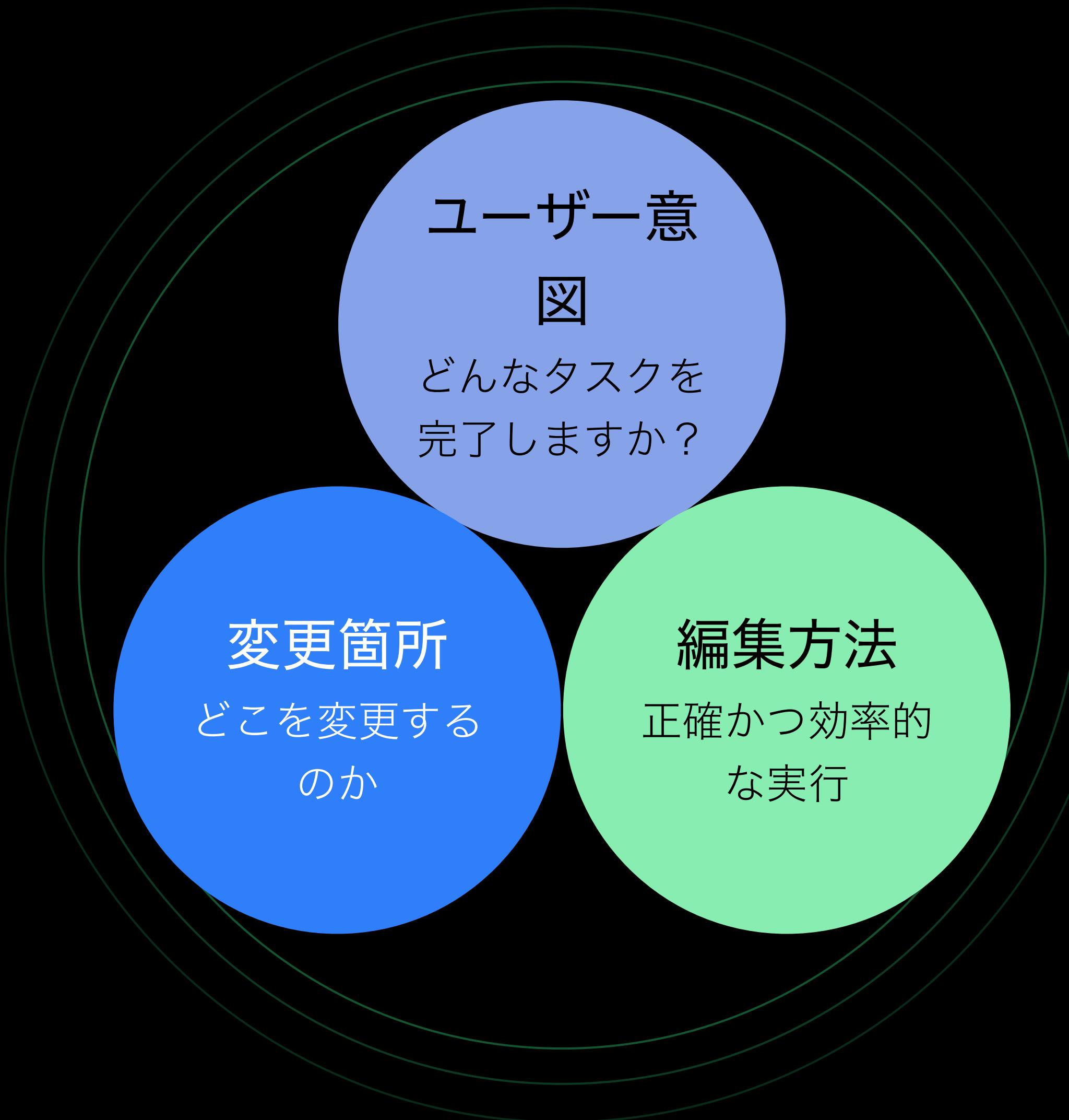
```

## JetBrains Full Line Completion の競合



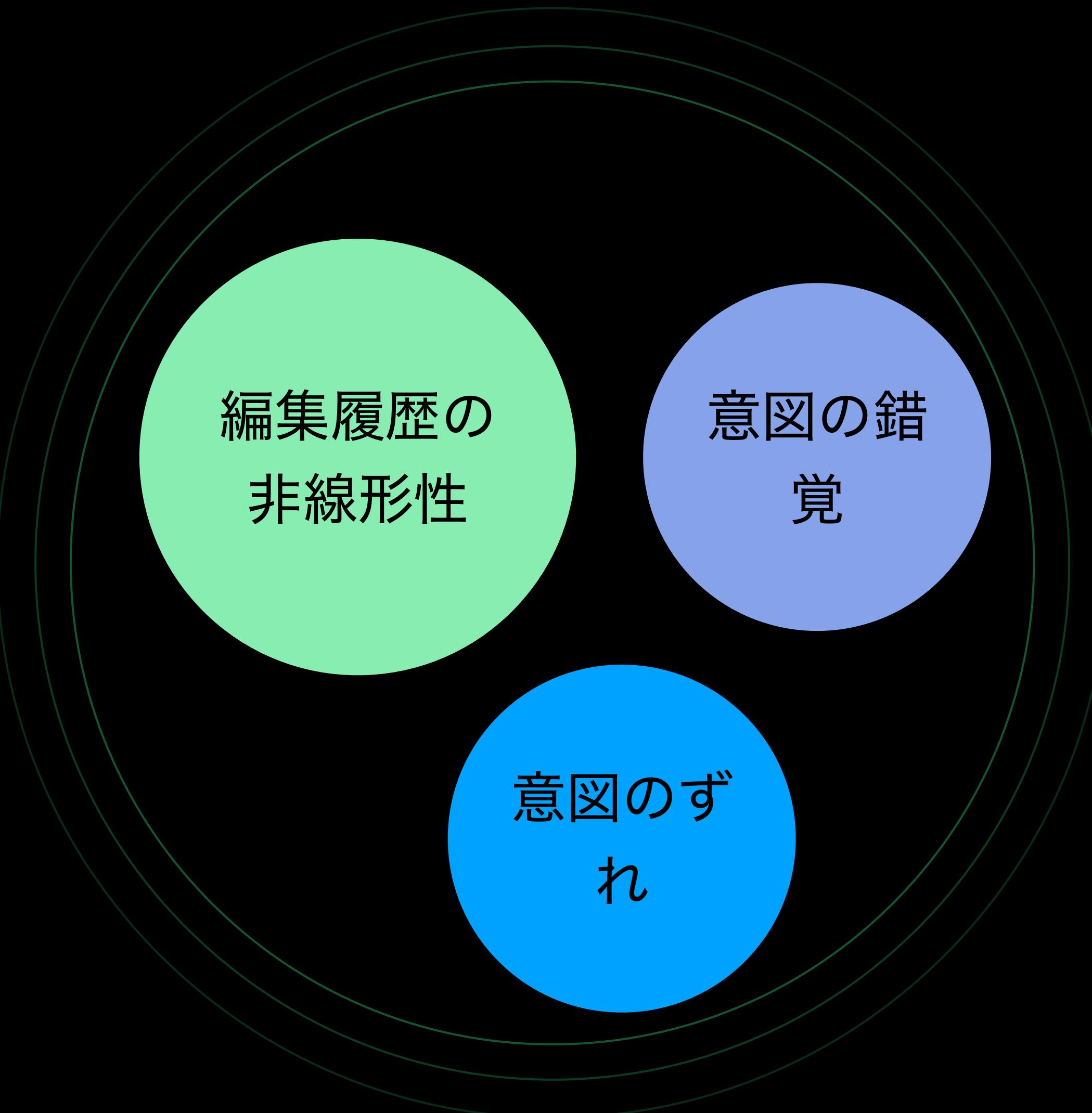
# 主な課題

TRAECue の技術アーキテクチャは、高度なAIと最適化技術を組み合わせることで、リアルタイム応答と直感的なユーザ一体験を実現しています。これらの技術的詳細を理解することで、TRAECue の強みと限界をより深く理解することができます。



## 3つの課題

Cue の構築の3つの主な研究課題：ユーザー意図の理解、変更箇所の特定、編集方法の決定



# ユーザー意図の理解

ユーザーがどんなタスクを完了しようとしているのかを特定する  
**非線形性な編集履歴**

非線形な開発ワークフローは、変更履歴を複雑にし、開発者の本来の意図を誤って推定させる原因となる。

## 意図の錯覚

モデルは、ユーザーが最近行った編集に直接関係のない提案をしてしまう傾向があり、その結果、あまり役に立たない提案が表示されることがある。

## 意図のずれ

不適切な Diff ブロックの分割粒度は、実際のコーディングプロセスとは異なるバイアスを生じさせてしまう。たとえば、モデルが最近変更した部分を編集しないよう学習してしまうことがある。しかし、その部分こそさらに修正が必要な場合もある。

# 変更箇所の特定

どこを変更するか決定

拡張可能なローカリゼーションメカニズム

非常に速い速度

高い精度と再現率

## 拡張性

大規模なコードベースを過剰なリソースを消費せずに効率的に処理でき、専用の `retrieve model` を設計

## 速度

超高速処理で高度なインタラクティブの使用パターンをサポート。ユーザーが新しい変更を加えた際に即座に提案

## 高い精度と再現率

関連する場所を正確に特定し、不要な提案でユーザーを煩わせず、重要な提案を見逃さない。カーソル周辺のコードを優先的に処理

## カーソル以外の複雑な編集

専用の diff フォーマットを設計。モデルが複雑な編集を簡潔に表現し、生成される token 数を最小限に抑え、大規模ファイルを効率的な処理を実現。これにより、遅延が数秒から数百ミリ秒に短縮

## リポジトリのコンテキスト認識

強力な検索拡張生成 (RAG) インフラストラクチャを利用して、Cue にコードベース固有のコンテキストを追加。コードベースの関連部分を検索することで、モデルはプロジェクト固有のコーディング標準に準拠し、カスタム API と正しく連携する提案が可能

# 編集方法の決定

## カーソル以外の複雑な編集

既存のモデルは、ユーザーのカーソル付近への単純な挿入以上の大規模変更が苦手

## 遅延制約

これらの編集の生成は高速であり、大幅な遅延なしにリアルタイムの提案を可能にする必要がある

## リポジトリのコンテキスト認識

提案内容はプロジェクトのコーディング標準や規約に沿い、カスタム API を正しく使用する必要がある。そのため、モデルはコードベース全体のコンテキストを理解している必要がある。

## 正確な意図理解

### 編集履歴のシミュレーション分析

直近の編集履歴からユーザー意図を推測。コミットメッセージ、初期コミット状態、最終コミット状態を分析して、一般的な開発者の行動を反映した現実的な編集シナリオをシミュレートする関連アルゴリズムを開発

## 効率的な位置認識

### Locator ソリューション

専門的にトレーニングされた Locator モデル：更新が必要なコード位置を迅速に特定するために使用される

## 高速かつ正確な 編集生成

### 斬新な Diff モデリング

斬新な Diff デコードソリューション：複雑な編集を効率的に表現し、遅延を数秒から数百ミリ秒に短縮  
検索拡張生成（RAG）：リアルタイムのコードベースインデックスに基づくコンテンツ認識編集

# 今後の展望

TRAECueは、新しいAI支援コーディングのパラダイムです。従来のコード補完ツールを超越し、最近普及しているVibe Codingとも異なります。独立してコード全体を生成するのではなく、開発者がコードを作成するプロセスで連携し、インテリジェントな支援を提供します。

## コアとなる価値提案

開発効率の  
向上

予測可能、反復  
可能

ユーザーコ  
ントロール  
の維持

方向性と品質の  
把握

認知負荷の  
軽減

低レベルな詳細  
の処理

グローバル  
リポジトリ  
の理解

リポジトリ規模  
のコンテキスト  
認識

一貫性の保  
証

コーディングパ  
ターンとスタイ  
ルへの適応

# プロダクトの価値と未来

## プロ開発者の効率向上

コード補完の域を超えて、インテリジェントなコード編集、低遅延応答、高精度な提案を提供することで、プロ開発者のコーディング効率を飛躍的に向上させます。

## 高い採用率とユーザーからの評価

多数のユーザーから高い評価を得ており、開発効率の向上における Cue の実用性を実証しています。

## 高レバレッジ5%に焦点を当てる

VibeCoding 開発モードでは、Cue は人間プログラマーの投入を必要とする5%のコードに高いレバレッジ効果をもたらし、人間と機械の協働を実現します。

## プロダクト体験の継続的な最適化

より低いエンドツーエンド遅延、よりユーザーフレンドリーなインターラクション、より的確な編集提案。

## 完全な PR の提案生成

より正確なユーザーコーディング意図認識、単一ステップに限定されない、より大規模で完全な Pull Request の提案生成をカバーします。

## 高性能モデル構造の探索

拡散モデルや MoE などのより効率的なコード編集シナリオでの使用を検討し、より広範囲で高速な生成をサポートします。

「Cue」で、思いのままに

使えば使うほど、提案が正確になり、コーディングが速くなる

More use it, more accurate, and more efficient

## 「Cue」をぜひお試しください

TRAЕ ユーザーグループに参加、または WeChat 公式アカウントをフォローして、TRAЕ の最新アップデート情報を入手してください

The Real AI Engineer.

# Yuta の Part

# Billy の Fellow Program