

# 基于TypeScript的语言服务 为你的DSL带来丝滑的开发体验

吴登轲

百度 前端开发工程师



# 精彩继续！ 更多一线大厂前沿技术案例

📍 北京站

**ArchSummit**  
全球架构师峰会

时间：2023年3月17-18日

地点：北京·海航万豪酒店

扫码查看大会  
详情>>



📍 广州站

**QCon**  
全球软件开发大会

时间：2023年4月7-8日

地点：广州·翡翠希尔顿酒店

扫码查看大会  
详情>>



📍 上海站

**ArchSummit**  
全球架构师峰会

时间：2023年4月21-22日

地点：上海·宏安瑞士大酒店

扫码查看大会  
详情>>



# 大纲

---

- 背景介绍：前端的DSL与TypeScript语言服务，与MDX
- 解决方案：Volar插件，基于虚拟文件的方法
- 实现效果：用Volar实现MDX的语言服务带来丝滑体验
- 总结回顾：收益和展望

# 领域特定语言是什么

通用编程语言 (GPL)

多数领域问题

JavaScript、C++、Go、Rust等

领域特定语言 (DSL)

少数领域问题

# 前端业务中的DSL

- 前端领域的经典实践：GPL做控制，DSL做描述
- JavaScript做逻辑控制，HTML和CSS做文档结构和样式描述
- React、Vue：TypeScript，JSX
- 小程序：JavaScript，文档结构DSL
- .....

# 前端业务中的DSL的分类

- 外部DSL
- 嵌入式DSL
- 混合式DSL

# 外部DSL

- 不需要依赖JavaScript作为Host语言
- 例：HTML、CSS、GraphQL等

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
</body>
</html>
```

# 嵌入式DSL

- 常以JavaScript作为Host语言
- 表现为它的某个语用子集
- 例：React Hooks
- 广义：utils、组件库等

```
interface Props {  
  1 reference  
  foo: number  
}  
  
0 references  
export const Component : (props: Props) => any = (props: Props) : any => {  
  const [first, setfirst] = useState()  
  return <div>Comp: {props.foo}</div>  
}
```



# 混合式DSL

- 有自己独特的语法，并嵌入了其他现存语言
- 例：Vue SFC、MDX等

# MDX简介

- MDX = Markdown + Extension, 是典型的混合式DSL
- MD的基础上, 额外支持了JSX
- 轻松做到混排MD文本和可执行的组件

# MDX代码

```
import { Component } from './component'
export const meta = {
  title: 'Blog Post',
}
```

```
# Blog Post
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Ut ac lobortis velit.
```

```
...
css
@media (min-width: 400px) {
  border-color: #000;
}
...
```

```
<Component foo={2}>{/* This is a comment */}</Component>
```

## Blog Post

Lorem ipsum dolor sit amet, consectetur adipiscing **elit**. Ut ac lobortis **velit**.

```
@media (min-width: 400px) {
  border-color: #000;
}
```

Count: 1

Add

# 代码编辑方式的演进

- 记事本、TextArea
- 代码高亮 (MDX在此)
- 基于语法 (JavaScript曾在此)
- 基于语法和语义 (TypeScript在此)

# TypeScript的开发体验： 实时类型检查

编辑期就能够发现代码中的语义语法问题，缩短了问题发现—解决链路，有效提升研发效能

```
1 reference
interface Config {
  1 reference
  id: number;
}
const config: Config = {
  id: 1,
};
```

# TypeScript的开发体验： 实时代码补全

Completion, 通过静态类型检查, 提供基于类型的精准代码补全

```
0 references
interface Config {
  0 references
  type: 'plain' | 'json';
}
const config : | 'const' declarations must be initialized.
```

# TypeScript的开发体验： 实时代码补全

Completion, 属性包含undefined的情况自动补上?., 减少粗心错误

```
interface Config {  
    0 references  
    value?: string;  
}  
declare const config: Config | undefined;  
console.log();
```



# TypeScript的开发体验：快速代码重构

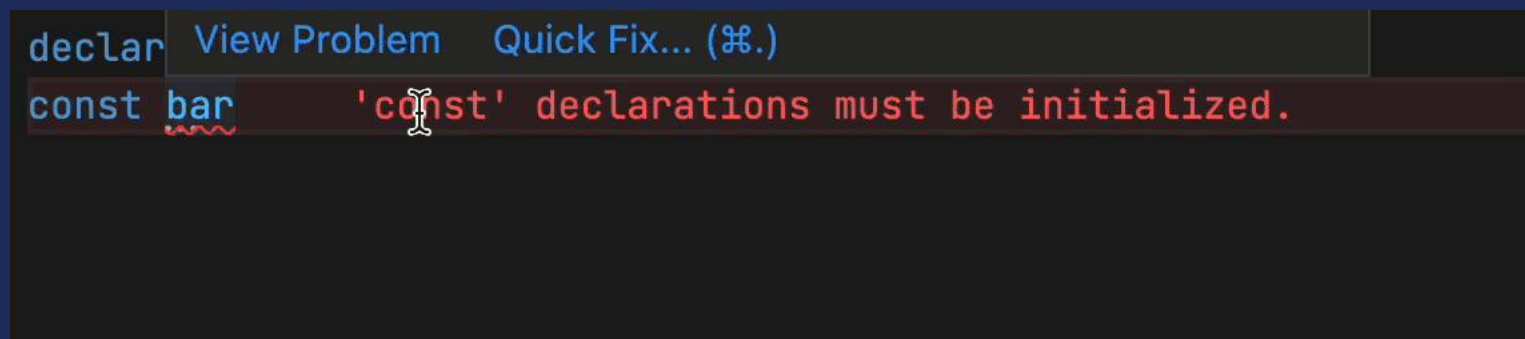
CodeAction, 自动生成类型标注, 组件移动等功能, 大大减少了手工操作

```
function foo() {  
  ⚡ if (1 > 2) {  
    return {bar: 'test'};  
  } else {  
    return {bar: 1};  
  }  
}
```



# TypeScript的开发体验： 嵌入类型提示

InlayHint, 在编辑器中嵌入变量、表达式的类型



The screenshot shows a code editor with a TypeScript declaration. The first line is `declare` followed by a tooltip that says "View Problem" and "Quick Fix... (⌘.)". The second line is `const bar`, where `bar` has a red squiggly underline. A tooltip points to the `const` keyword, displaying the error message: "'const' declarations must be initialized.".

```
declare  
const bar    'const' declarations must be initialized.
```

# TypeScript语言服务

- TypeScript 约等于 JavaScript + 静态类型系统
- TS语言服务基于静态类型系统为编辑器提供了上面提到的各种功能

# 语言服务和LSP

- 语言服务（Language Service）就是对编辑器提供语言支持，提供刚刚提到的开发体验的程序
- 语言服务协议（Language Server Protocol，简称LSP）是一种开源的语言服务器协定，各个编辑器只要支持LSP就能够简单接入语言服务

# 使用MDX进行开发的挑战

- 挑战：无语言服务，刀耕火种
- 只有语法高亮，但是没有完善的语言服务支持
- 开发者在写作MDX、利用MDX进行组件开发的过程中容易犯错
- 并非MDX独有的问题，以MDX为例进行研究

# 无语言服务进行MDX开发是个什么体验

- MDX没有语言服务，开发体验约等于在记事本中编辑有高亮的代码
- 从头写一个MDX语言服务？

# 从头写一个语言服务?

- MDX包括独有语法，也嵌入了其他语言
- 需要对接TypeScript、CSS等多个语言服务的各个操作
- 细节太多，成本太高，且很难实现丝滑流畅的开发体验

是否可以使用现成的TypeScript语言服务，  
为MDX构建一个语言服务，并获得丝滑流畅的开发体验？

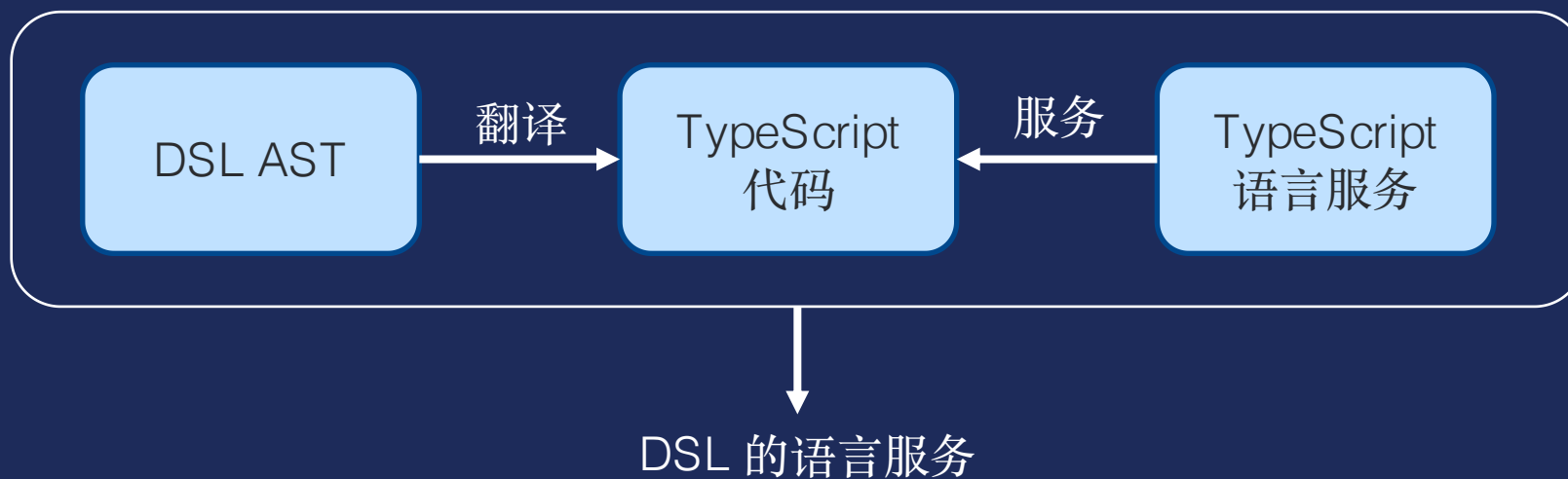
# DSL翻译到TS以实现语言服务

总体思路：翻译 + 借用TS语言服务

- 外部DSL：翻译
- 嵌入式DSL：本身是TS，无需额外实现
- 混合式DSL：翻译 & 根据需要分派给现成的其他语言服务



# 总体结构



# 下一步

- MDX是混合式DSL
- 目标：翻译到TypeScript并将相关代码分派给现成的其他DSL语言服务

# 大纲

---

- 背景介绍：前端的DSL与TypeScript语言服务，与MDX
- 解决方案：Volar插件，基于虚拟文件的方法
- 实现效果：用Volar实现MDX的语言服务带来丝滑体验
- 总结回顾：收益和展望

# Volar介绍

- 一个源自Vue社区，最初是为Vue提供TypeScript语言服务的一个插件
- 目前已经衍生出Volar.js，目标是为不同的语言提供嵌入服务
- Volar语言服务插件：TypeScript、CSS、HTML.....
- 只需关注DSL的翻译过程，桥接到插件上即可

# Volar的原理图



# 用Volar实现MDX的语言服务

1. 确定需要实现的MDX特性
2. 翻译到TypeScript，并分派语言服务
3. 生成Source Map。将MDX的range和代码段的range建立对应关系
4. 将生成的Source Map和构造出的代码段交给Volar，使用虚拟文件进行处理

# 确定要实现的MDX特性

- import/export语句
- JSX片段
- 嵌入其他语言代码块

```
import { Component } from './component'  
export const meta = {  
  title: 'Blog Post',  
}
```

```
# Blog Post
```

```
Lorem ipsum dolor sit amet,  
consectetur adipiscing **elit**.  
Ut ac lobortis <b>velit</b>.
```

```
...  
CSS  
@media (min-width: 400px) {  
  border-color: #000;  
}  
...
```

```
<Component foo={2}>{/* This is a comment */}</Component>  
  
<Component foo={444}>{/* This is a comment */}</Component>  
  
<Component foo={123}>{/* This is a comment */}</Component>
```

# MDX到TS的翻译

```
[...importNodes, ...exportNodes, ...jsxNodes].forEach((n) => {  
  const start = getNodeStartOffset(n)  
  const end = getNodeEndOffset(n)  
  const nodeText = this.snapshot.getText(start, end)  
  
  const oldContentLength = contents.length  
  contents += nodeText  
  const newContentLength = contents.length  
  contents += ';'   
  
  mappings.push({  
    sourceRange: [start, end],  
    generatedRange: [oldContentLength, newContentLength],  
    data: FileRangeCapabilities.full,  
  })  
})
```



# 代码块中语言服务的递归嵌入

- MDX中可以使用``语法来包含代码块，这些代码块同样需要有语言服务，我们递归地提供语言服务
- 实现：CSS、TypeScript

```
visit(tree, isCodeBlockNode, (node: CodeBlock) => {  
  if (['ts', 'css'].includes(node.lang)) {  
    localMappings.push({  
      sourceRange: [...],  
      generatedRange: [...],  
      data: FileRangeCapabilities.full,  
    })  
    this.embeddedFiles.push(...)  
  }  
})
```

```
const plugin: LanguageServerPlugin = () => ({
  extraFileExtensions: [
    {
      extension: 'mdx',
      isMixedContent: true,
      scriptKind: ScriptKind.Deferred,
    },
  ],
  getLanguageModules() {
    return [MdxLanguageModule]
  },
  getLanguageServicePlugins() {
    return [createCssPlugin(), createTypeScriptPlugin()]
  },
})

startLanguageServer(createConnection(), plugin)
```

# 将构造出来的TS代码交给Volar

```
this.embeddedFiles.push({
  fileName: `${this.fileName}.virtual.tsx`,
  kind: FileKind.TypeScriptHostFile,
  snapshot: {
    getText: (s, e) => contents.substring(s, e),
    getLength: () => contents.length,
    getChangeRange: () => undefined,
  },
  mappings,
  capabilities: {
    ...FileCapabilities.full,
    documentFormatting: false,
  },
  embeddedFiles: [],
})
```

# 大纲

---

- 背景介绍：前端的DSL与TypeScript语言服务，与MDX
- 解决方案：Volar插件，基于虚拟文件的方法
- 实现效果：用Volar实现MDX的语言服务带来丝滑体验
- 总结回顾：收益和展望

# TypeScript语句

```
# This is a MDX file
```

```
export c
```

# JSX组件的类型检查

```
# This is a MDX file

import {Component} from './component'

<Component|
```

# 代码块的递归语言服务：CSS

```
# This is a MDX file
```

```
```\nCSS\n| \n```\n
```



# 代码块的递归语言服务：TS

```
# This is a MDX file
```

```
💡`ts
```



# 大纲

---

- 背景介绍：前端的DSL与TypeScript语言服务，与MDX
- 解决方案：Volar插件，基于虚拟文件的方法
- 实现效果：用Volar实现MDX的语言服务带来丝滑体验
- 总结回顾：收益和展望

# 收益

- 以MDX为例，我们以极低的成本为DSL提供了语言服务
- 将DSL开发从刀耕火种拯救出来，带来了丝滑的开发体验
- 大大提升了开发体验和研发效率

# DSL翻译到TS的一般方法

- MDX的TS翻译比较简单。对于更加复杂的外部DSL，怎么翻译？
- 不可避免地需要一些编译器设计的知识，因为涉及DSL处理
- 可采用语法制导翻译（Syntax-Directed Translation）方法

# 局限性

- DSL翻译到TypeScript的方法适用于各种常见的DSL
- 不适用于类型系统无法通过TypeScript类型检查器实现的DSL
  - 例如：依赖类型（Dependent Type）

# 总结

- 我们可以通过“借用”TypeScript的成熟语言服务的方式，大大提升DSL的开发体验
- 这种适配方法也适用于其他绝大多数DSL，从而得到一个相对完备的语言服务，获得丝滑的开发体验

# 其他案例

- volarjs/angular-language-tools
- volarjs/svelte-language-tools



# 免费领 24 张 IT 职业技能图谱

涵盖领域

架构、后端、前端、云计算、大数据、  
机器学习、运维等



扫码免费领取







# THANKS

