**【前端技术方案】semi-icons 懒加载优化处理**

**1. 需求概述**

目前厂商平台使用 semi 版本是 v1，semi icon 的懒加载开启碰到一些问题。

**1.1 术语解释**

**2. 功能范围**

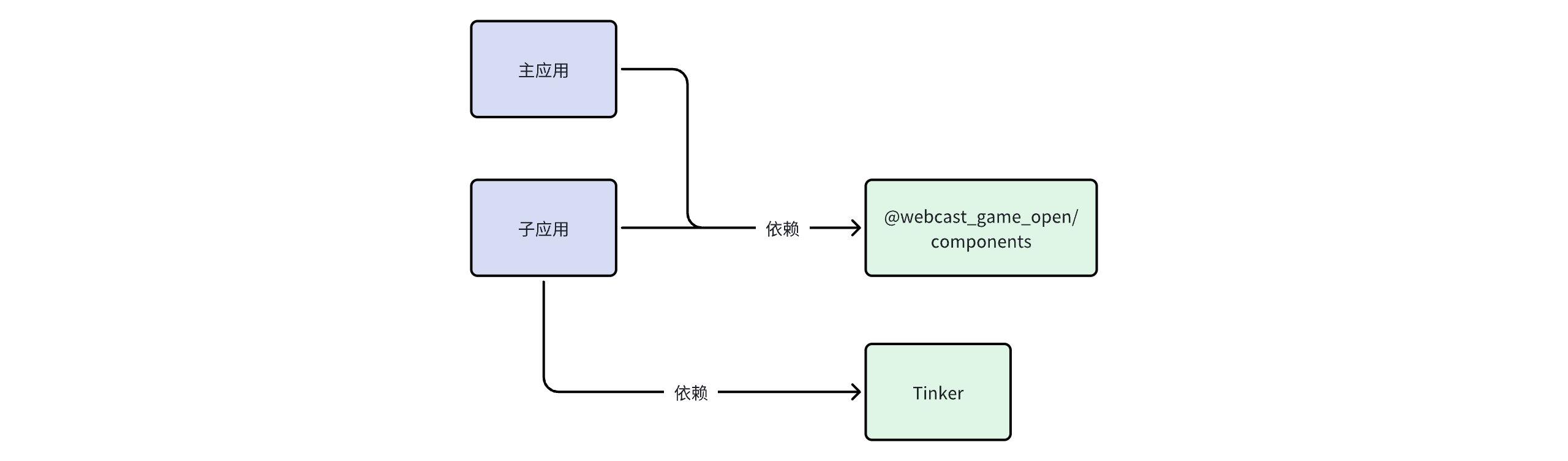
**2.1 本方案实现的功能**

厂商平台开启 semi icon 懒加载

**2.2 本方案涉及的模块（影响范围）**

厂商平台所有模块

**2.3 外部依赖**



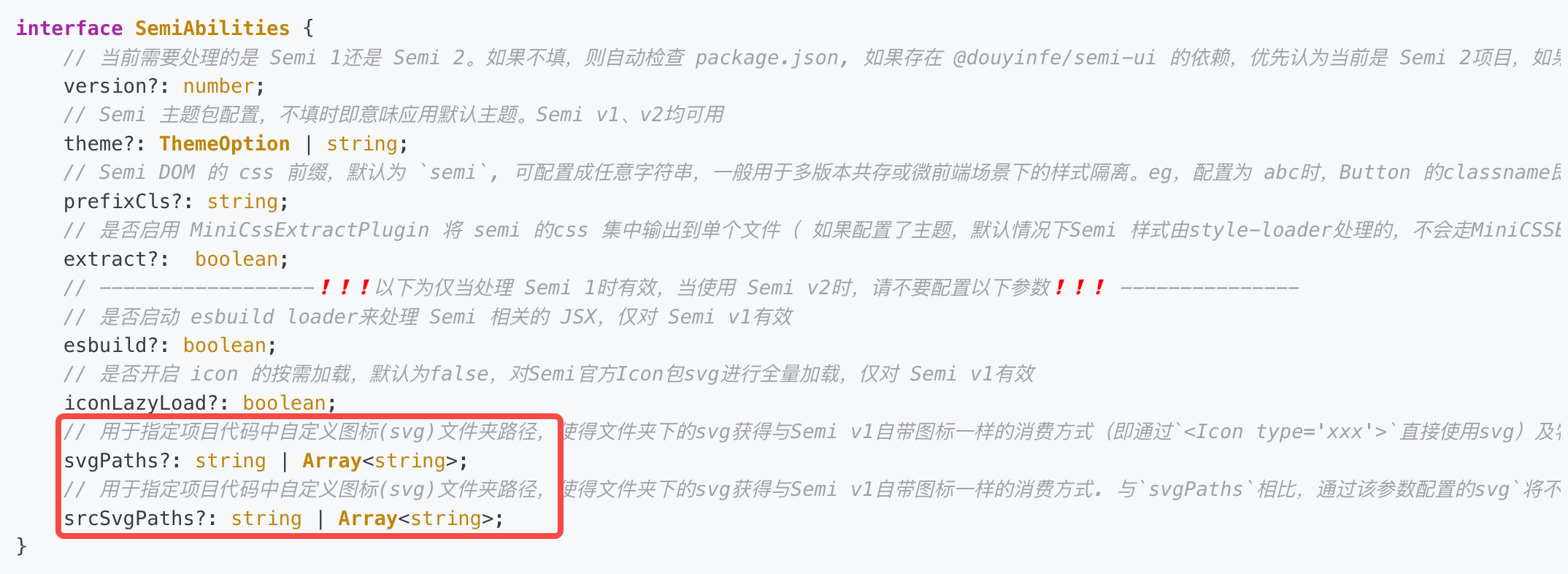
**3. 方案设计**

**~~去除 semi 将 svg 插入 document 的操作~~**

1. semi icon 的原理是使用 svg use，通过 id 使用插入在 document 的[svg symbol](https://developer.mozilla.org/zh-CN/docs/Web/SVG/Element/symbol)，类似于通过 id 获取 dom 节点操作。

|  |  |
| --- | --- |
|  |  |

1. 因此使用 semi icon 前，需要将要使用的 svg 插入到 document，这一步操作由 semi 插件[@ies/eden-plugin-semi-webpack](https://bnpm.bytedance.net/package/@ies/eden-plugin-semi-webpack)完成。通过配置 svg 路径，在代码编译阶段，将 svg 插入 document。



使用自定义 icon，需要配置 srcSvgPaths，例如:

|  |
| --- |
| TypeScript srcSvgPaths: [/libs\/components\/common\/icons/], |

并且插件会默认配置 semi icon 路径，使得这些路径下的 svg 可以插入 document。**开启懒加载后，只会按需插入。**

|  |
| --- |
| TypeScript DEFAULT\_SEMI\_SVG\_PATH\_REGS: [/@ies\/semi-ui\/icons/i, /@ies\/semi-ui-react\/icons/i, /@ies\/semi-icons/i, /@ies\/semi-icons-.+/i], DEFAULT\_SEMI\_SRC\_SVG\_PATH\_REGS: [/@ies\/semi-illustrations/i], |

1. ~~为了优化需要将插入 svg 操作去掉，通过插件源码可以发现更改路径配置可以实现。~~

|  |
| --- |
| TypeScript svgPaths: () => [] |

**处理 Tinker icon 无法显示问题**

子应用开启 semi icon 懒加载后，Tinker 中 <Icon type="xxx" /> 组件使用会导致无法显示。

因为是依赖，所以只能通过 babel 插件预编译做静态处理，需要两步：

1. 插入将对应的 svg import
2. 将 <Icon type="xxx" /> 改成 <Icon type="semi-icons-xxx" />

|  |
| --- |
| JavaScript /\*\*  \* 将 tinker-ui 中 <Icon type="xxx" /> 转为 <Icon type="semi-icons-xxx" />，并插入 svg import  \*/  module.exports = function ({ types: t, template }) {  const buildImport = template(`  import %%importName%% from %%source%%;  `);  let root;  let iconMap = {};  const PREFIX = 'semi-icons-';   return {  visitor: {  Program(path) {  root = path;  },  CallExpression(path) {  const { callee, arguments: args } = path.node;  if (  (callee &&  callee.object &&  callee.property &&  callee.object.name !== 'React' &&  callee.property.name !== 'createElement') ||  !Array.isArray(args)  ) {  return;  }   if (!args[0] || args[0].name !== 'Icon' || !args[1] || args[1].type !== 'ObjectExpression') {  return;  }   const propArg = args[1];  const typeIdx = propArg.properties.findIndex(prop => prop.key.name === 'type');  const typeProp = propArg.properties[typeIdx];  if (!typeProp) {  return;  }   // console.log('typeProp', typeProp);   // 动态渲染 Icon 情况，type 拼接前缀 semi-icons-  if (t.isMemberExpression(typeProp.value)) {  const quasis = [t.templateElement({ raw: PREFIX }, false), t.templateElement({ raw: '' }, true)];  const templateLiteral = t.templateLiteral(quasis, [typeProp.value]);  // console.log('templateLiteral', templateLiteral);  propArg.properties[typeIdx].value = templateLiteral;  return;  }   // type='xxx'  if (t.isStringLiteral(typeProp.value)) {  const iconName = typeProp.value.value;  // 避免重复处理  if (iconName.startsWith(PREFIX)) {  return;  }   // 插入 svg import  if (!iconMap[iconName]) {  iconMap[iconName] = 1;  const importIconName = `${iconName.replace(/-(\w)/g, (\_, c) => c.toUpperCase())}SemiIcon`;   const importIconDeclaration = buildImport({  importName: t.identifier(importIconName),  source: t.stringLiteral(`@ies/semi-icons/${PREFIX}${iconName}.svg`),  });  root.unshiftContainer('body', importIconDeclaration);  }   // type="xxx" -> type="semi-icons-xxx"  propArg.properties[typeIdx].value = t.stringLiteral(`${PREFIX}${iconName}`);  return;  }  },  },  }; }; |

**结论**

只需要处理 Tinker icon 显示问题，整体影响不大。