通用 React Hooks 库技术评审

一、项目背景

> @h/h-hooks 一个更通用、更高效、更好管理及扩展的通用 React Hooks 库

二、项目目标

📌 一、输出 N 个 Hooks，可供使用

二、优先接入 Saas 组内部业务项目，减少项目重复代码，减小项目体积

三、UT 覆盖率 100%，用例通过率 100%，构建时间 5min 以内

四、改造 N 个 Hooks，使其场景容性更强，功能得到扩展（optional）

三、项目拆分

> 带 * 为优先做的

Question

💡 **一、已有项目通用 Hooks 整理**

二、改造 Hooks 目标更通用、更适用

> Q1：小程序端通用 Hooks 改造会有问题（Taro 独有的 Hooks，如：useDidHide、useDidShow等，Taro 是否有专门 Hooks 包）
>
> Q2：每个项目都有 utils 文件夹下有些通用函数可能更适合以 Hooks 的方式使用

三、Hooks 的 API 格式与命名、入库规则

四、是否需要二次封装其他 Hooks 库

五、依赖选择、管理

> Q3：Hooks 是否依赖其他东西(redux，Taro，h-toolbox)
>
> 例如像 useDvaState(react-redux) 这样的需要一并整理嘛？useObservable(mobx)

为了通用性，考虑 hooks 依赖以及业务的相关性，依赖作为参数引入或作为其peerDeps

> Q4：h-hooks 项目依赖 ahooks 等其他三方库怎么解决?
>
> 按文件引入，最好不引入整个包

六、ESLint、UT

> Q5：Lint?
>
> ESLint、Style Lint 直接引入 Fe-lint

七、包构建、发布以及版本管理

> Q6：spartan-cli/webpack(vite?)

spartan-cli 太老，webpack 体量太大，考虑 vite 去打包（调研）

> Q7：发布 hnpm 的流程

参考一下 h-rest 的发包流程

任务八：README 维护

> TODO：可以考虑将 README 写成类似 aHooks 官网标准 demo 的形式

四、已有项目整理分工

> 针对不同端分类：一个库引用不同文件夹下内容
>
> 参考 ahooks 目录结构

目录结构：

XML

```
.
├── __test__
├── docs
├── packages
│   ├── common
│   │   ├── hooks
│   │   └── utils
│   ├── mp
│   │   ├── hooks
│   │   └── utils
│   └── web
│       ├── hooks
│       └── utils
├── types
└── utils
├── README.md
├── index.html
├── package.json
├── tsconfig.json
├── tsconfig.node.json
├── vite.config.ts
└── yarn.lock
```

（点击下方链接可跳转到对应表格）

已有项目整理分工表格

五、Hooks CheckList

Hooks 改造：

1、补充UT、TS约束、注释

2、兼容性、功能扩展

checkList 加入流程

1、待启动：hooks 加入 checkList

2、进行中：补充 checkList 内容，同时放入 gitLab 项目

3、已入库：添加至 hooks 库中 hooks，TS、UT 完善、注释

4、改造完成：hooks 完成兼容性或功能扩展改造

（点击下方链接可跳转到对应表格）

Hooks Checklist表格

六、是否需要二次封装其他 Hooks 库

社区：react-use

据说版本迭代特别快，用在项目可能更新版本，代价比较大

阿里：umi-hooks、ahooks

https://zhuanlan.zhihu.com/p/149860703?from_voters_page=true

umi-hooks 是 ahooks 的前身

ahooks 据说是阿里团队为了 OKR 把 umi-hooks 给改造了，并加入了一些 API 规范

二次封装其他 Hooks 库

优点：其他 Hooks 库封装集成在 h/h-hooks 中。基础 hooks 齐全，能很大程度拓展 hooks 的使用场景

缺点：不利于我们的 hooks 开发，可能会出现类似 @h/antd 的问题，升级版本影响面太大，局限于版本

other：收集第三方 hooks 库优秀 hooks，从源码提取入库

结论：允许多个 hooks 库进入业务项目，(同时引用和 h-hooks 不强依赖)

七、Hooks 的 API 格式与命名、入库规则

约定什么样的 Hooks 可以加入项目

> 命名规则：用途 > 使用场景，不应携带与业务相关的内容
>
> 有参数类型约束(TS)、一个规范的 Hooks example
>
> 入库规则：Hooks 需要达到什么程度才能入库通用 Hooks，要考虑的因素：
>
> 1) 各个项目使用频率高，（状态库相关 hooks，考虑改造成一个通用的）
>
> 2) 不过分依赖项目的私有内容、业务
>
> ps: 如果各个项目使用频率高且对项目业务有一定依赖，是否可以入库?
>
> 3）按需引用 Hooks

TypeScript

```tsx
// hooks example，一起改造
// 后决定大家第一个 pr，一起看

import { useRef, useLayoutEffect, DependencyList } from 'react'

/**
 * auto scroll to top
 * @param {array} deps dependencies which trigger scrolling
 * @param {string} selector scrollable element selector
 * @example
 * ```tsx
 * useScrollToTop([page, status], '.module-data-content')
 * ```
 */
export const useScrollToTop = (deps: DependencyList = [], selector = ''): void => {
  const container = useRef<HTMLElement | null>(null)
  useLayoutEffect(() => {
    if (!container.current) {
      const child = document.querySelector(selector)
      container.current = child?.parentElement as HTMLElement
    }
    if (container.current?.scrollTop === 0) return
    container.current.scrollTop = 0
  }, [...deps])
}
```

八、依赖管理

这里列举项目要引入的依赖、依赖类型。及引入的原因（如：某个 hooks 依赖：peerDeps）

	A	B	C
1	**依赖**	**依赖类型**	**引入原因**
2	vite	devDeps	用于项目打包
3	@vitejs/plugin-react	devDeps	vite react plugin
4	react	devDeps/peerDeps	
5	@types/react	devDeps	
6	typescript	devDeps	
7	fast-glob	devDeps	读取文件夹
8	@types/node	deps	
9			
10			

九、基建

使用 vite + ts-morgh 打包

使用 fe-lint + git husky 完成 lint

使用

十、进度跟进

> 双周会议，周四

会议纪要

十二、初步分工

> 第一阶段：收集已有项目，并分类加入 checklist
>
> 范围：先 Saas 组内前端项目
>
> 第二阶段：基础建设
>
> 第三阶段：checklist 入库
>
> 第四阶段：Hooks 落地业务项目
>
> 第五阶段：入库 Hooks 改造

第一阶段：

- 收集 checkList 文档
- 预估项目指标
- 项目模板构建

第二阶段：

代码规范、UT 配置、包构建、发布、版本管理

- 构建，发布 npm、版本管理 @杨海啸 @王亚
- ESLint、Style Lint、UT jest @白亚鹏 @汪莹

。。。