## next.js的SSR提前注入数据的方法（page router模式）

next.js提供的三种方法：getInitialProps、getServerSideProps、getStaticProps/getStaticPaths

1. getServerSideProps（SSR，服务端每次请求执行）

* **执行时机**：每次请求都会在服务端运行
* **适用场景**：需要实时数据（比如用户相关、搜索、订单详情等）
* **特点**：不会在客户端运行，数据对搜索引擎友好

1. getStaticProps（SSG，构建时运行）

* **执行时机**：**构建时**执行一次，生成静态 HTML
* **适用场景**：数据更新不频繁的页面（如博客、产品列表）
* **特点**：性能极佳，支持 revalidate 增量更新

1. getInitialProps（通用型，SSR + 客户端均可）

* **执行时机**：
* 第一次请求：服务端执行
* 页面跳转时：客户端执行
* **适用场景**：\_app.tsx 中需要数据注入（例如国际化、主题皮肤等）
* **特点**：已被官方推荐逐步替代（无法使用 SSG）

1. getStaticPaths（用于动态路由下的静态生成）

* **配合**：getStaticProps 使用，生成动态路径
* **适用场景**：动态路由（如博客详情页）静态生成多个页面

| **对比项** | **getStaticProps** | **getServerSideProps** | **getInitialProps** | **客户端请求 (useEffect+fetch)** |
| --- | --- | --- | --- | --- |
| ✅ **适用范围** | 所有静态可预渲染的页面 | 实时个性化页面（用户相关） | \_app.tsx, 页面组件 | 所有页面（客户端渲染） |
| ⏱️ **执行时机** | 构建时执行（Build 时） | 每次请求时服务端执行 | 首次 SSR 和客户端跳转时执行 | 用户访问页面后才执行 |
| 🧠 **客户端运行** | 否 | 否 | 是（页面跳转时） | 是 |
| 🔍 **SEO 支持** | ✅（静态 SEO 佳） | ✅（动态 SSR） | ✅（SSR 时） | ❌（需 JS 执行后才有内容） |
| 🔐 **加载敏感数据** | ❌（代码暴露） | ✅（服务器运行） | ✅ | ❌（容易暴露） |
| ⚙️ **用法复杂度** | 低 | 中 | 高（弃用趋势） | 低 |
| 🚀 **缓存/CDN 支持** | ✅（CDN 可缓存） | ❌（每次服务端请求） | ❌ | ❌ |
| 🧩 **中间件/拦截器支持** | ✅（可通过 Rewrites/Headers） | ✅（中间件支持） | 部分支持 | ❌ |
| 🌐 **支持** next export | ✅ | ❌ | ❌ | ✅ |

## 常用使用场景

| **使用场景** | **推荐方法** | **原因** |
| --- | --- | --- |
| 静态内容（博客、文档、商品列表） | getStaticProps | 性能高、支持 CDN 缓存 |
| 需要实时数据（订单详情、用户中心） | getServerSideProps | 每次请求都会取新数据 |
| 页面首次加载和客户端跳转都要获取数据 | getInitialProps（或用 AppRouter 模式 + layout.js） | 通用性强但复杂， |
| 加载语言包或皮肤配置（首次服务端获取） | getServerSideProps 或 \_app.tsx + getInitialProps | 提前注入数据用于初始化 |
| 页面数据频繁变更但不要求实时性（如价格表） | getStaticProps + revalidate | 静态生成 + 增量更新 |

## 实践建议

* **优先使用** getStaticProps：对性能和 SEO 友好，支持缓存，适合绝大多数非实时页面。
* **实时数据或敏感信息（如登录态）用** getServerSideProps。
* **全局预处理数据（如主题、国际化）建议** \_app.tsx + getInitialProps **或 Middleware 配合** cookies **设置全局上下文**。