41. 什么是 Hydration？React 18 对它做了哪些优化？

基础概念与定义

Q1: 什么是服务端渲染 (SSR)？它主要解决了什么问题？
A1: 服务端渲染（SSR）是指在服务器端（如 Node.js 环境）将 React 组件渲染成完整的 HTML 字符串，然后将这份 HTML 直接发送给浏览器的技术。它主要解决了两个问题：

1. **更快的首屏内容展现 (FCP)**：浏览器接收到 HTML后可以立即渲染页面，无需等待 JS 加载和执行。
2. **更好的搜索引擎优化 (SEO)**：搜索引擎爬虫可以直接解析返回的完整 HTML 内容。

Q2: 什么是 Hydration (注水/水合)？它在 SSR 流程中扮演什么角色？
A2: Hydration 是一个“激活”过程，指的是在浏览器端，React 接管由服务器渲染的静态 HTML。它会为这份静态 HTML 附加事件监听器（如 `onClick`）和组件状态，最终将页面从一个静态的“空壳”转变为一个功能完整、可交互的单页应用。它的角色是让静态页面“活过来”。

Q3: 请简述一下传统 Hydration 的核心步骤。
A3: 传统 Hydration 的核心步骤如下：

1. **接收 HTML**：浏览器从服务器获取 SSR 生成的 HTML 骨架并立即展示。
2. **下载 JavaScript**：浏览器开始下载所有必需的 JS 文件，包括 React 库和应用代码。
3. **执行 Hydration**：所有 JS 下载完成后，React 在内存中构建虚拟 DOM，然后将其与浏览器中的真实 DOM 进行匹配，并在此过程中挂载事件监听器。

传统 Hydration 的痛点

Q4: 传统 Hydration (React 18 之前) 有哪些主要的性能痛点？
A4: 传统 Hydration 主要有两个性能痛点：

1. **全有或全无 (All-or-Nothing)**：必须等待页面上**所有**组件的 JS 代码都下载完毕后，才能对**整个**应用开始 Hydration。一个加载慢的组件会阻塞所有组件的交互。
2. **同步阻塞 (Synchronous Blocking)**：Hydration 过程本身是同步且一气呵成的，它会长时间占用浏览器主线程。在此期间，页面无法响应任何用户交互（如点击、输入），导致页面看起来像“卡死”了，这会显著增加可交互时间（TTI）。

Q5: 如何理解“全有或全无 (All-or-Nothing)”的 Hydration 模式？请举例说明其负面影响。
A5: “全有或全无”意味着 React 必须将整个应用作为一个整体进行 Hydration。例如，一个页

面由页头（Header）、文章（Article）和评论区（Comments）三个组件构成。假设评论区组件因为功能复杂，其 JS 文件加载需要 5 秒，而页头和文章的 JS 只需要 1 秒。即使用户在 1 秒后就看到了整个页面，并尝试点击页头中的菜单，页面也无法响应。用户必须等待最慢的评论区组件加载完成后，整个页面才能一起被 Hydration，从而变得可交互。

React 18 的优化方案

Q6: React 18 为了解决传统 Hydration 的痛点，引入了哪两个核心新特性?
A6: React 18 引入了两个相辅相成的核心特性来解决 Hydration 痛点：

1. **流式服务端渲染 (Streaming SSR)**：通常与 `<Suspense>` 结合使用，允许服务器分块、流式地将 HTML 发送到浏览器。
2. **选择性注水 (Selective Hydration)**：允许 React 对已准备好的内容进行分块、有优先级的注水，而无需等待所有代码加载完成。

Q7: 什么是流式服务端渲染 (Streaming SSR)？它是如何工作的?
A7: 流式 SSR 是 React 18 的一项新功能。开发者可以用 `<Suspense>` 将应用中加载较慢的部分（如需要请求数据的组件）包裹起来。在服务端，服务器会先发送一个包含主要结构的 HTML “外壳”，对于被 `<Suspense>` 包裹的慢组件，则先发送一个加载状态的 UI（fallback）作为占位符。当服务器准备好这部分慢组件的内容后，再通过同一个 HTTP 连接，将这部分 HTML “流式”地发送给客户端，并附带脚本将其替换到对应的占位符位置。

Q8: 什么是选择性注水 (Selective Hydration)？它相比传统 Hydration 最大的优势是什么?
A8: 选择性注水是指 Hydration 不再是一个一次性的、宏大的任务。即使某些组件的 JS 代码还没有下载完成，React 也可以开始对那些已经可见、并且代码已就绪的组件进行“选择性”的注水。
其最大的优势是：**Hydration 过程可以响应用户的交互**。例如，如果用户在一个还未完全注水的页面上点击了某个按钮，React 18 会检测到这次交互，并**优先**对用户正在交互的组件进行注水，使其能立刻响应，而不是像过去那样被阻塞。

Q9: 在 React 18 中，如果用户点击一个组件时，该组件的 Hydration 恰好还未开始，这个点击事件会丢失吗？React 是如何处理的?
A9: 不会丢失。React 18 具备事件重放（Event Replay）机制来处理这种情况。当用户点击一个尚未注水的组件时，React 会在文档的根节点上记录这个事件。然后，等到目标组件完成注水后，React 会自动将刚才记录的事件“重放”到该组件上。这确保了用户的早期交互不会丢失，提供了更流畅的体验。

综合与应用

Q10: 请深入解释一下，React 18 的并发特性 (Concurrent Features) 是如何优化 Hydration 过程的?

A10: 我们可以从三个层面来回答这个问题：

1. **指出传统痛点**：首先，传统 Hydration 是“全有或全无”且“同步阻塞”的。它必须等所有 JS 下载完才能开始，并且一旦开始就会阻塞主线程，直到整个应用注水完毕，导致页面长时间无响应（TTI 过长）。
2. **引入并发作为基础**：React 18 的核心是并发特性，它使得渲染任务变得**可中断和可恢复**。这是所有优化的基础。因为任务可被拆分和中断，Hydration 就不再是一个不可分割的宏任务。
3. **连接新特性与解决方案**：
   - 基于并发能力，**流式 SSR** 可以先发送无需等待数据的 HTML 骨架，让用户更快看到内容。慢组件则后续以流的方式发送。
   - 同样基于并发能力，**选择性注水**得以实现。React 不再需要一次性注水整个应用。它可以根据代码的加载情况和用户的交互，有策略地、分批次地进行注水。如果检测到用户交互，它会中断当前的注水任务，优先为用户交互的组件注水，从而保证了页面的响应性。通过事件重放机制，还能确保早期交互不丢失。
   - **总结**：因此，并发特性通过支持流式 SSR 和选择性注水，将一个宏大、阻塞的 Hydration 过程，转变为一个可中断、有优先级、非阻塞的精细化过程，从根本上解决了传统 Hydration 的痛点，极大地提升了用户体验和页面的可交互性。