40. Next.js 的 SSR, SSG, ISR 有什么区别?

主题：Next.js 渲染策略 (SSR, SSG, ISR)

Q1:
请简要解释一下 Next.js 中的三种主要页面渲染策略：SSR, SSG, 和 ISR。

A1:

- **SSR (Server-Side Rendering / 服务器端渲染)**：在**每次**收到页面请求时，服务器会动态生成该页面的完整 HTML 并发送给客户端。
- **SSG (Static Site Generation / 静态站点生成)**：在**构建时 (build time)**，就为每个页面预先生成一个静态的 HTML 文件。这些文件可以被部署到 CDN，用户请求时直接获取。
- **ISR (Incremental Static Regeneration / 增量静态再生)**：这是 SSG 的增强版。它在构建时生成静态页面，但允许在用户请求后，根据设定的策略**按需**、**增量地**在后台重新生成特定页面，而无需重新构建整个站点。

Q2:
`getServerSideProps` 和 `getStaticProps` 这两个 Next.js 函数的核心用途是什么？它们的执行时机有何不同?

A2:

- `getServerSideProps`:
  - **用途**: 用于实现 SSR (服务器端渲染)。它在页面加载前获取数据，并将数据作为 props 传递给页面组件。
  - **执行时机**: 在**每次**页面被请求时，于**服务器端**执行。
- `getStaticProps`:
  - **用途**: 用于实现 SSG (静态站点生成) 和 ISR (增量静态再生)。它在构建时获取数据以预渲染页面。
  - **执行时机**:
    - 在 SSG 模式下，它在**构建时 (build time)** 于服务器端执行。
    - 在 ISR 模式下，它除了在构建时执行外，还会在后台根据 `revalidate` 的配置，在请求到达后被触发执行。

Q3:
请详细描述一下一个采用 SSR 策略的页面，从用户发出请求到页面变得可交互的完整工作流程。

A3:
SSR 的工作流程如下：

1. **用户请求**: 用户的浏览器向服务器发送一个页面请求。
2. **服务器处理**: 服务器接收请求，并执行该页面的 `getServerSideProps` 函数，以获取动态数据（例如，从数据库或 API）。
3. **生成 HTML**: 服务器使用获取到的数据和页面组件，动态渲染出完整的 HTML 内容。
4. **发送给客户端**: 服务器将生成的完整 HTML 发送回用户的浏览器。浏览器接收到后可以立即解析和显示内容，因此 First Contentful Paint (FCP) 很快。
5. **客户端激活 (Hydration)**: HTML 加载完毕后，浏览器会下载并执行相关的 JavaScript 代码。React 接管静态的 HTML，并将其“激活”成一个功能完备、可交互的单页面应用 (SPA)，例如附加事件监听器等。

Q4:
在使用 SSG 时，如果你的页面使用了动态路由（例如 `pages/posts/[id].js`），你需要额外使用哪个函数？它的作用是什么？

A4:
当为动态路由页面使用 SSG 时，需要额外使用 `getStaticPaths` 函数。

- **作用**: `getStaticPaths` 的作用是告诉 Next.js 在构建时需要为这个动态路由生成哪些具体的路径（即哪些 `id`）。它会返回一个包含 `paths` 数组的对象，每个 `path` 都定义了一个需要预渲染的页面的 `params`。
- **例如**: 对于博客文章 `/posts/[id]`， `getStaticPaths` 会返回类似 `{ paths: [{ params: { id: '1' } }, { params: { id: '2' } }], fallback: ... }` 的结果，Next.js 就会在构建时为 `/posts/1` 和 `/posts/2` 这两个页面调用 `getStaticProps` 并生成对应的 HTML 文件。

Q5:
ISR 是如何实现“增量静态再生”的？请解释其核心机制，特别是 `revalidate` 参数的作用。

A5:
ISR 的核心机制基于 "stale-while-revalidate" 策略，通过在 `getStaticProps` 中返回 `revalidate` 参数来实现。

1. **初始构建**: 和 SSG 一样，在构建时生成页面的静态 HTML。
2. **首次请求**: 当用户请求页面时，服务器会立即返回已缓存的静态 HTML 文件，保证了极快的加载速度 (TTFB)。
3. **触发再生**: 如果距离上次生成页面的时间超过了 `revalidate` 设定的秒数，这次请求会触发一次**后台**的页面重新生成。

4. **提供旧页面**: 在后台重新生成期间，用户看到的**仍然是旧的、缓存的**静态页面，保证了服务的可用性。
5. **后台更新**: Next.js 在后台悄悄执行 `getStaticProps`，获取新数据，并生成新的页面 HTML。
6. **替换缓存**: 如果后台生成成功，新的 HTML 文件会替换掉旧的缓存。
7. **提供新页面**: 在缓存被更新后，**下一次**对该页面的请求将会得到更新后的内容。

`revalidate` 参数的作用就是定义一个时间窗口（单位为秒），告诉 Next.js 在此窗口过后，可以尝试重新生成页面。

Q6:
请从数据新鲜度、性能（首字节时间 TTFB）、服务器负载和 SEO 这几个维度，对比一下 SSR, SSG 和 ISR 的优缺点。

A6:

特性	SSR (服务器端渲染)	SSG (静态站点生成)	ISR (增量静态再生)
数据新鲜度	**实时**：每次请求都获取最新数据。	**构建时快照**：数据是构建时的状态，更新需重新构建。	**接近实时**：数据在 revalidate 间隔内可能是旧的，之后会更新。
性能 (TTFB)	**可能较慢**：因为服务器每次都需要计算和生成 HTML。	**非常快**：直接从 CDN 提供预先生成的静态文件。	**非常快**：和 SSG 一样，直接提供缓存的静态文件。
服务器负载	**较高**：每次请求都需要服务器端计算。	**非常低**：服务器仅作为文件主机，压力极小。	**低到中等**：平时负载低，仅在需要按需再生时有计算负载。
SEO	**良好**：搜索引擎可以直接抓取到完整的 HTML 内容。	**非常好**：内容完整，且加载速度极快，对 SEO 非常有利。	**非常好**：兼具 SSG 的所有 SEO 优点。

Q7:
在实际项目开发中，你会如何为以下场景选择最合适的渲染策略？请说明理由。

1. 公司官网的“关于我们”页面。
2. 一个新闻网站的文章列表页，新闻大约每小时更新几次。
3. 一个电商网站的用户个人订单中心页面。

A7:

1. **“关于我们”页面**:
   - **选择**: **SSG (静态站点生成)**。
   - **理由**: 这类页面的内容非常稳定，几乎不发生变化。使用 SSG 可以在构建时生成一个静态 HTML，获得最佳的加载性能和最低的服务器成本，同时 SEO 效果也最好。没有必要为这种页面增加实时计算的开销。
2. **新闻网站的文章列表页**:
   - **选择**: **ISR (增量静态再生)**。
   - **理由**: 新闻内容需要相对频繁地更新，但又不是绝对的实时。完全使用 SSG 会导致更新不及时（需要频繁构建），而完全使用 SSR 则可能对服务器造成较大压力且牺牲了静态页面的速度优势。ISR 是完美的平衡点：它可以提供静态页面的秒开体验，同时通过设置一个合理的 `revalidate` 值（例如 `revalidate: 60`），保证页面能在一两分钟内自动更新到最新内容，且无需手动部署。
3. **用户个人订单中心页面**:
   - **选择**: **SSR (服务器端渲染)** 或 **客户端渲染 (CSR) 配合 SSG Shell**。
   - **理由**:
     - 内容是高度个性化且必须是实时的。每个用户看到的订单都不同，且数据必须是账户的最新状态。
     - 页面通常需要身份验证。SSR 允许在服务器端通过 `getServerSideProps` 访问请求的 `context`（如 cookies），从而进行用户身份验证并获取该用户的私有数据。SSG/ISR 因为是在构建时或为所有用户共享生成，无法处理这种个性化的私有内容。
     - （备选方案）也可以使用 SSG 生成一个页面的“骨架”或“外壳”(Shell)，然后用客户端渲染(CSR)的方式，在页面加载后通过 `useEffect` 请求用户的私有数据来填充页面。

Q8:
面试官可能会问：“是不是应该总是优先选择 SSR，因为它能保证数据总是最新的？”你会如何回答这个问题，并指出这种想法可能忽略了哪些权衡（trade-offs）？

A8:
这种说法是不准确的。“看场景说话”是选择渲染策略的关键，不应该绝对地说某一种最好。虽然 SSR 保证了数据的实时性，但它也带来了显著的权衡（trade-offs），在许多场景下并非最佳选择。

我将从以下几点进行回应：

1. **性能与成本**: SSR 对服务器的计算资源消耗更大，因为每次请求都需要在服务器上运行代码、获取数据并渲染 HTML。这会导致更高的服务器成本和运维压力。相比之下，SSG 和 ISR 将计算前置到构建时或按需进行，用户请求时只需提供静态文件，服务器负载极低，响应速度（TTFB）也更快。

2. **并非所有页面都需要实时数据**: 网站中的很多页面，如博客文章、产品文档、公司介绍等，内容更新频率很低。为这些页面使用 SSR 是不必要的性能浪费，SSG 或 ISR 是更合适的选择。
3. **用户体验**: 对于非个性化内容，SSG/ISR 能够部署在 CDN 的边缘节点上，用户可以从最近的节点获取内容，实现极速加载，这对于用户体验和留存至关重要。而 SSR 的响应时间则受限于服务器的处理能力和地理位置。
4. **“静态优先 (Static First)”原则**: 在 Next.js 的实践中，我们通常遵循“静态优先”的原则。优先考虑能否用 SSG 实现。如果 SSG 的构建时间或数据更新问题成为瓶颈，再考虑 ISR。只有当页面内容必须是实时、高度个性化且无法提前生成时，才最后选择 SSR。

因此，选择 SSR 仅仅是为了追求“数据最新”而忽略了性能、成本和具体业务场景，是一种常见的误区。正确的做法是根据页面的数据特性和业务需求，综合权衡后做出最合理的选择。