



# React Web Framework



2024 究竟怎样开发一个网页

- Library and Framework
- 渲染方式: CSR/SSR/SSG/ISR
- Next.js App Router 与 RSC
- Next.js 路由系统
- 全栈框架的后端能力
- Next.js 的 UX 优化
- SPA 模式下的 Framework

- **Library and Framework**
- 渲染方式: CSR/SSR/SSG/ISR
- Next.js App Router 与 RSC
- Next.js 路由系统
- 全栈框架的后端能力
- Next.js 的 UX 优化
- SPA 模式下的 Framework

# Library



React



Vue

# Framework



Next.js



Remix



Nuxt

# Library



- UI 渲染
- 状态管理

# Framework



- 路由管理
- 全栈能力
- UX 优化
- 开箱即用

- Library and Framework
- **渲染方式: CSR/SSR/SSG/ISR**
- Next.js App Router 与 RSC
- Next.js 路由系统
- 全栈框架的后端能力
- Next.js 的 UX 优化
- SPA 模式下的 Framework

# 常见渲染方式

**CSR**

Client-Side Rendering

**SSR**

Server-Side Rendering

**SSG**

Static Site Generation

**ISR**

Incremental Static Regeneration

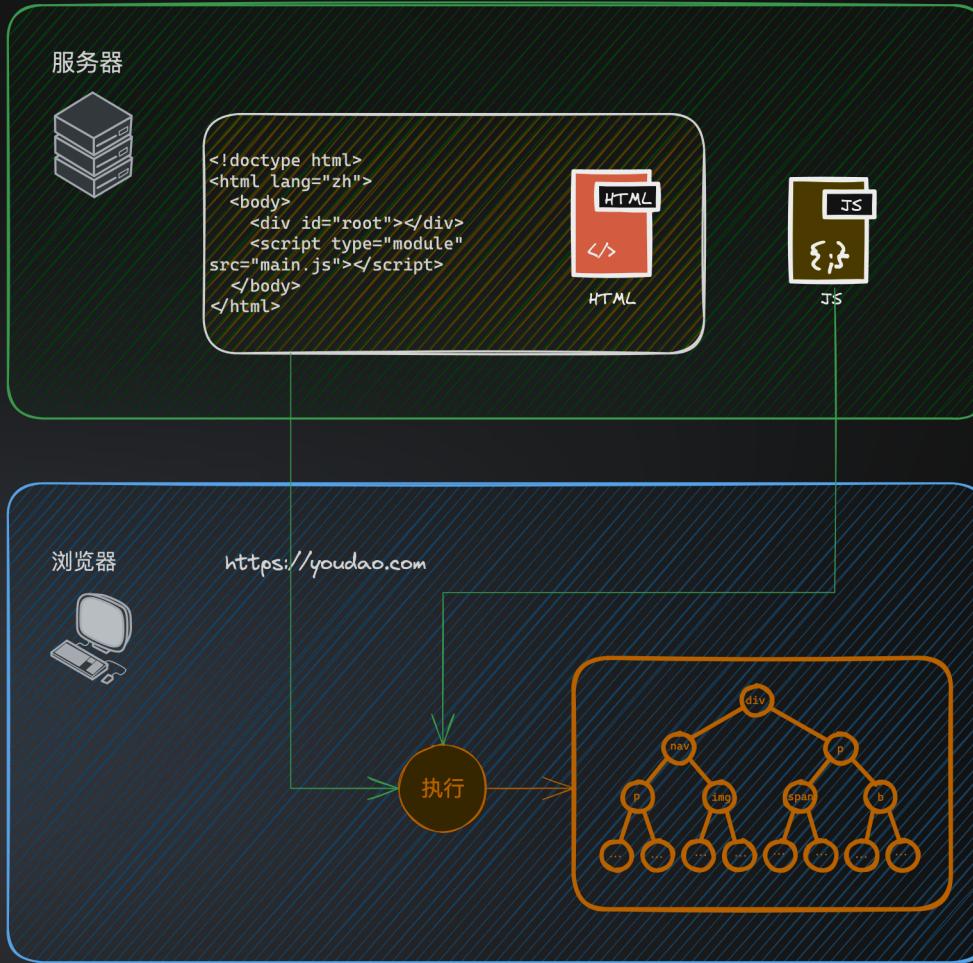
客户端渲染

服务端渲染

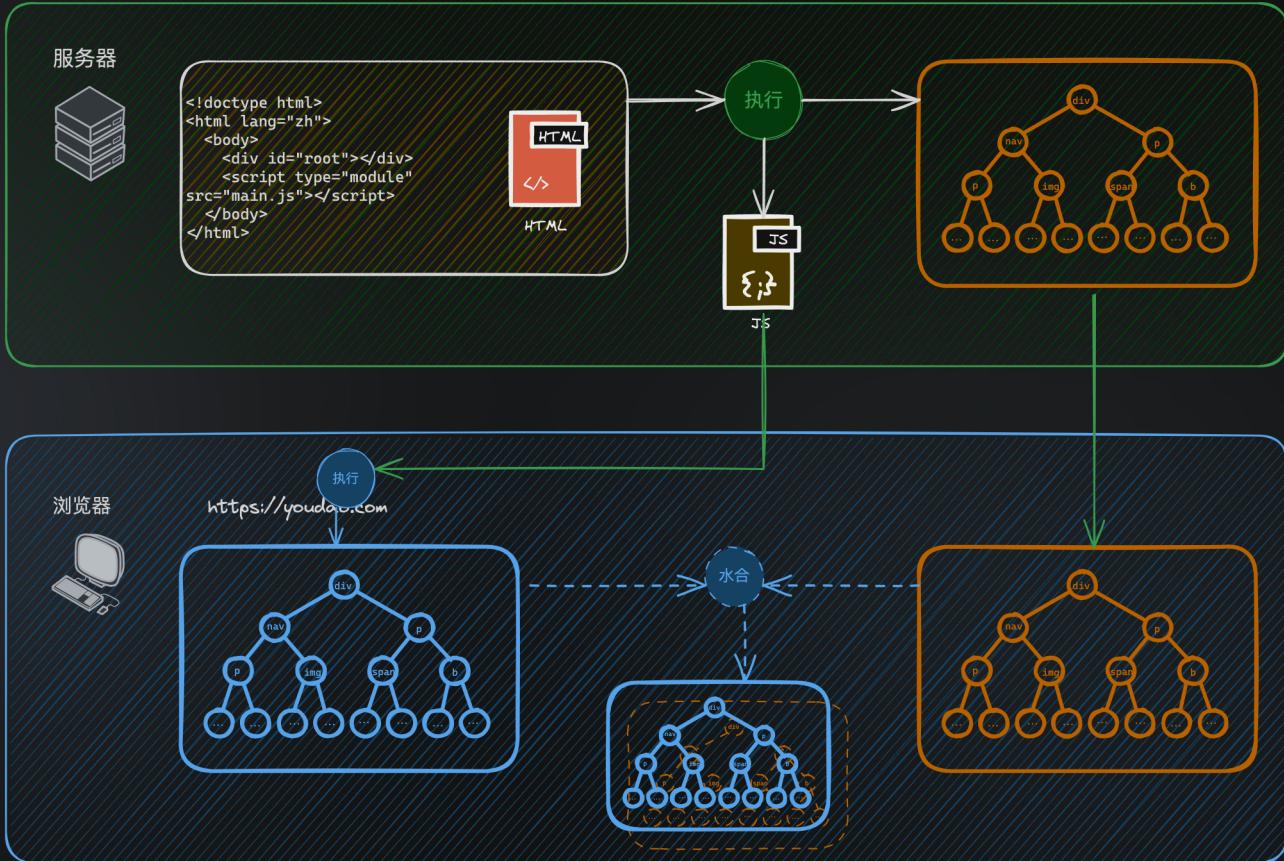
服务端生成

增量静态生成

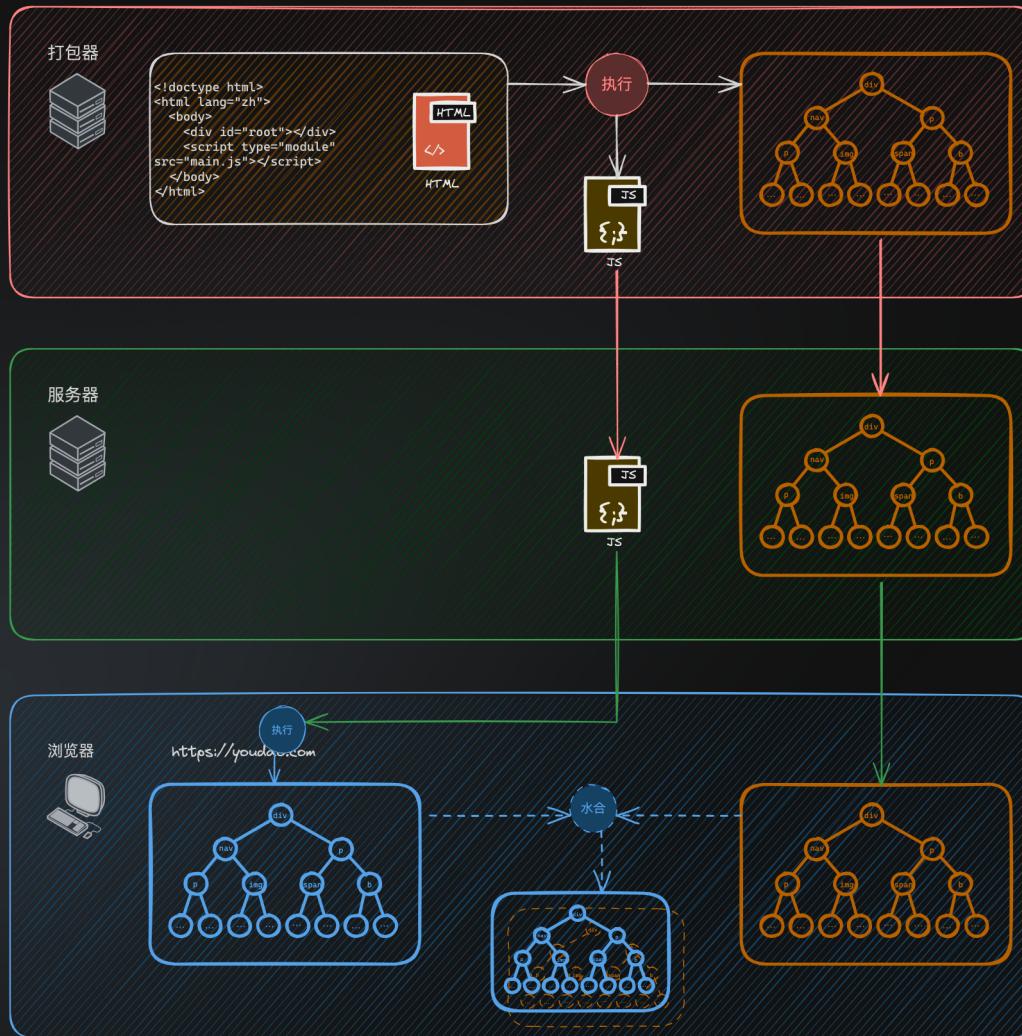
# CSR



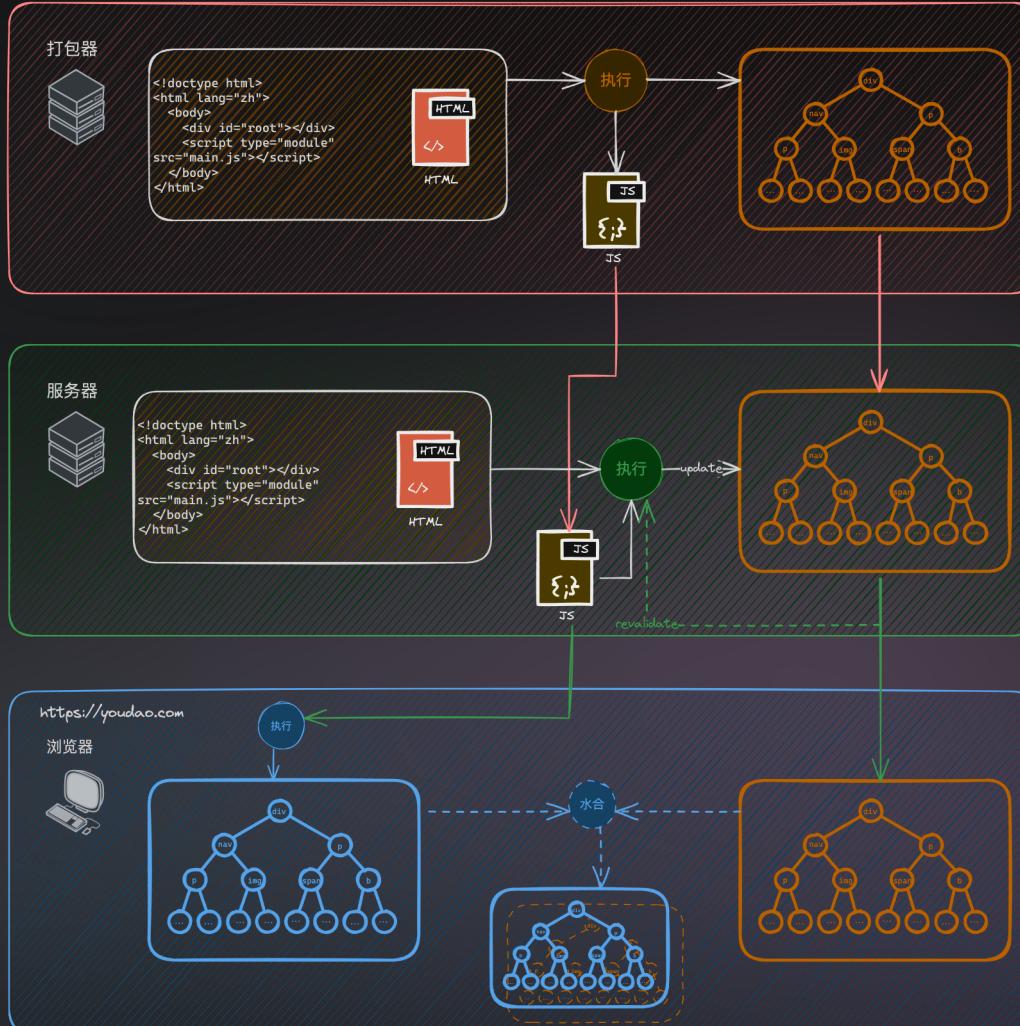
# SSR



# SSG

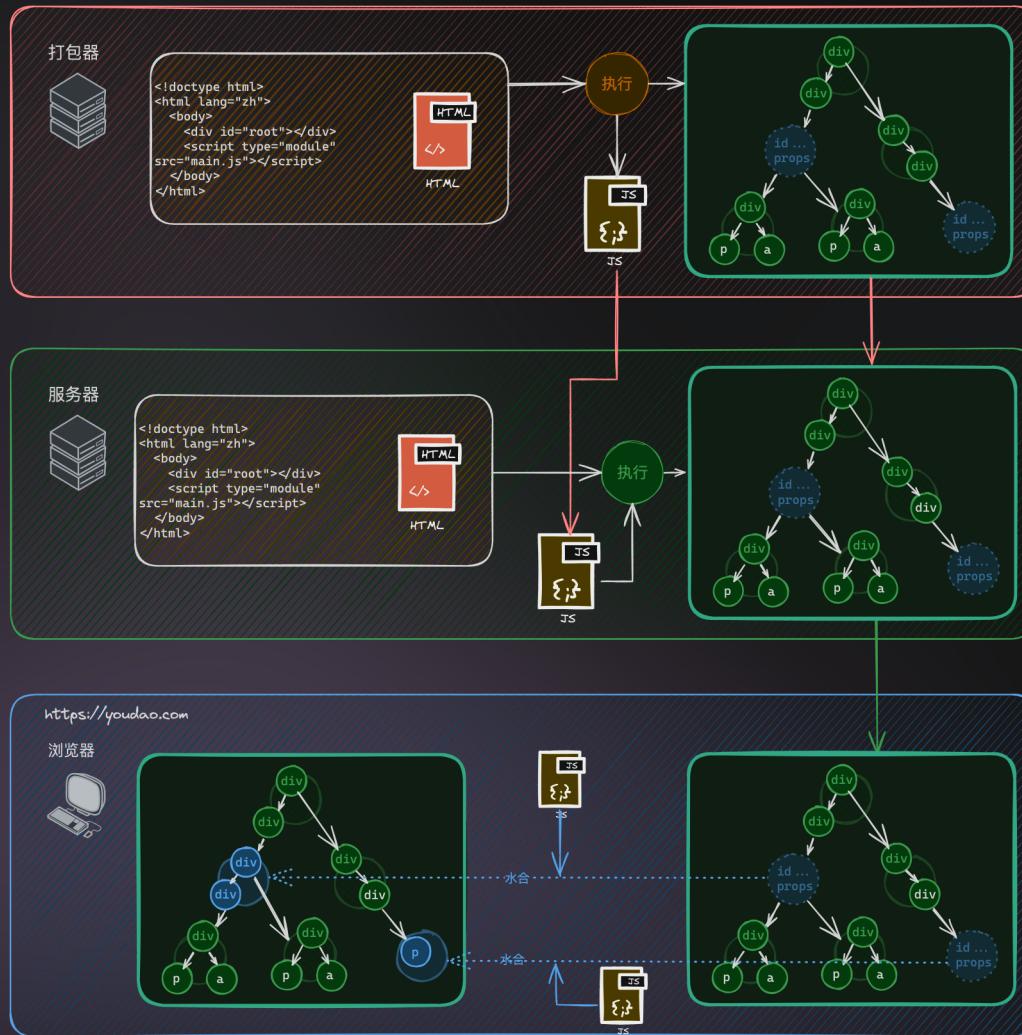


# ISR



- Library and Framework
- 渲染方式: CSR/SSR/SSG/ISG
- **RSC 与 Next.js App Router**
- Next.js 路由系统
- 全栈框架的后端能力
- Next.js 的 UX 优化
- SPA 模式下的 Framework

# RSC



## 静态渲染

Static Rendering (Default)

```
async function Article({ id }) {  
  const content = await getArticle(id);  
  return <p>{content}</p>  
}
```

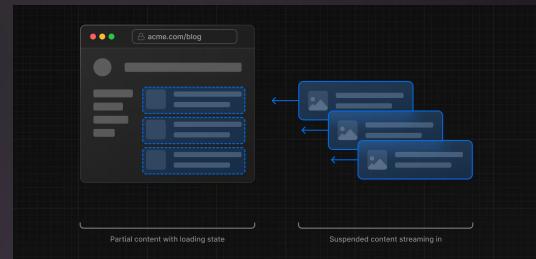
## 动态渲染

Dynamic Rendering

```
async function UserInfo({ id }) {  
  const id = new URLSearchParams(window.location.search).get('id');  
  const user = await getUserInfo(id);  
  return <p>{user}</p>  
}
```

## 流式渲染

Streaming Rendering



- Library and Framework
- 渲染方式: CSR/SSR/SSG/ISR
- RSC 与 Next.js App Router
- **Next.js 路由系统**
- 全栈框架的后端能力
- Next.js 的 UX 优化
- SPA 模式下的 Framework

# 定义路由



```
// app/page.js
export default function Page() {
  return <h1>Hello, Next.js!</h1>
}
```

# 动态路由

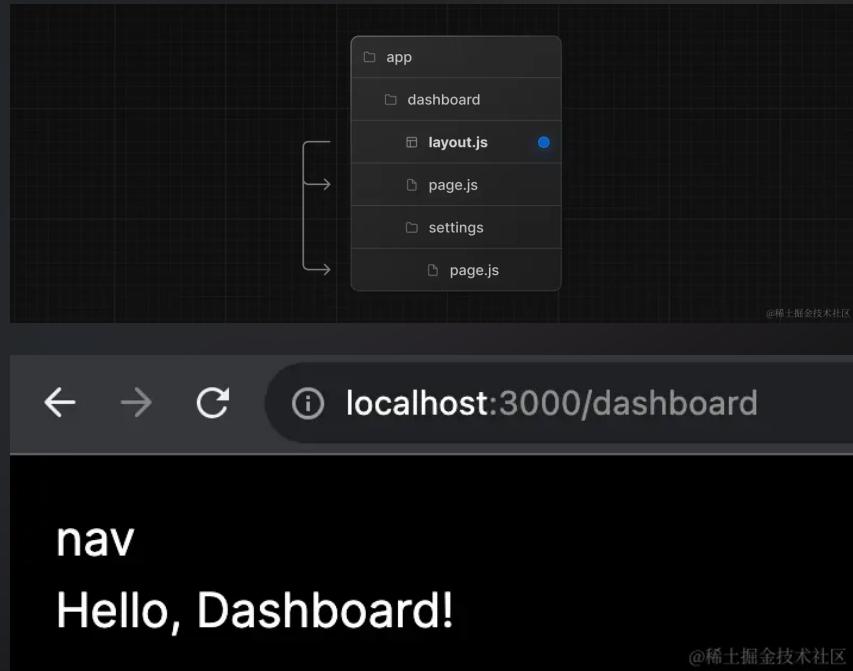
```
// app/blog/[slug]/page.tsx
export default function Page({ params }: { params: { slug: string } }) {
  return <div>My Post: {params.slug}</div>
}
```

Route	Example Url	params
app/blog/[slug]/page.js	/blog/a	{ slug: 'a' }
app/blog/[slug]/page.js	/blog/b	{ slug: 'b' }
app/blog/[slug]/page.js	/blog/c	{ slug: 'c' }

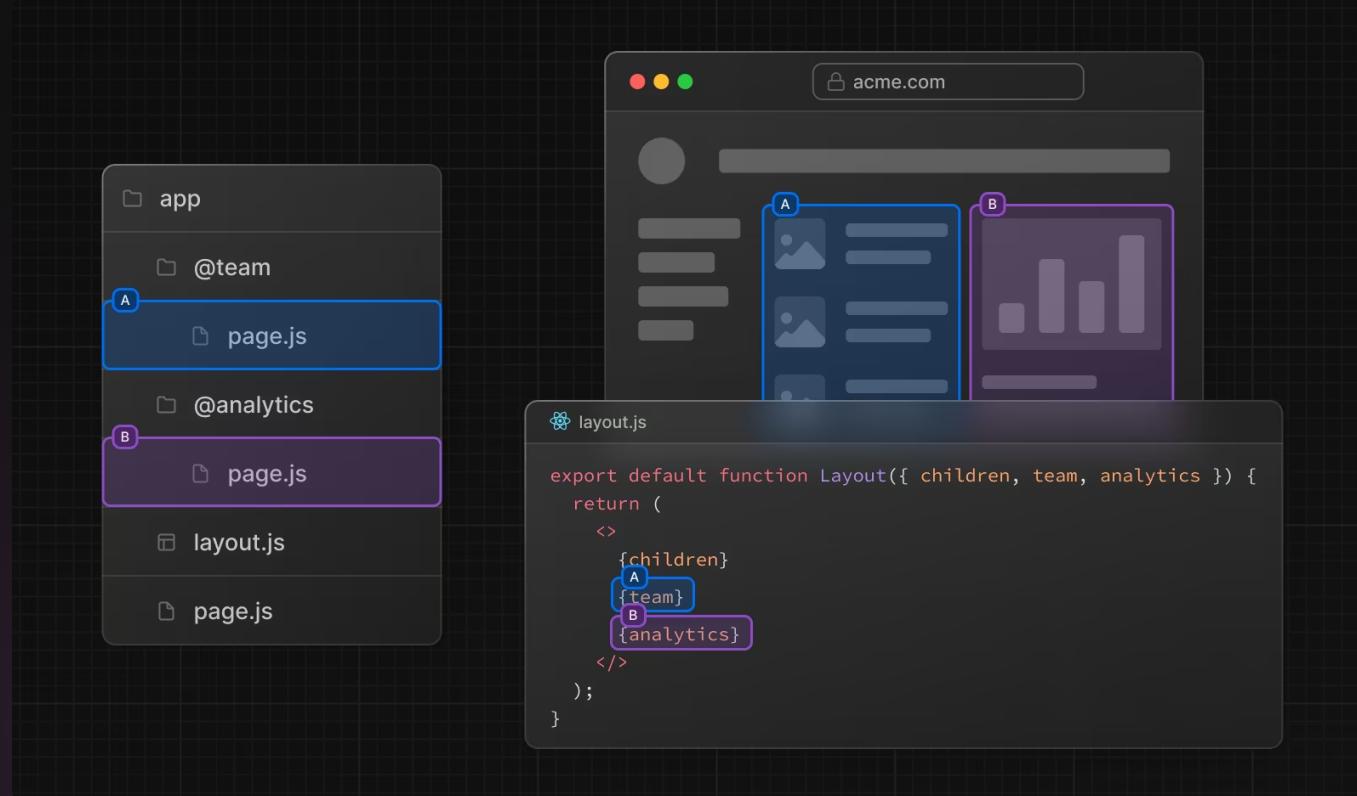
# 布局

```
// app/dashboard/layout.js
export default function DashboardLayout({
  children,
}) {
  return (
    <section>
      <nav>nav</nav>
      {children}
    </section>
  )
}
```

```
// app/dashboard/page.js
export default function Page() {
  return <h1>Hello, Dashboard!</h1>
}
```

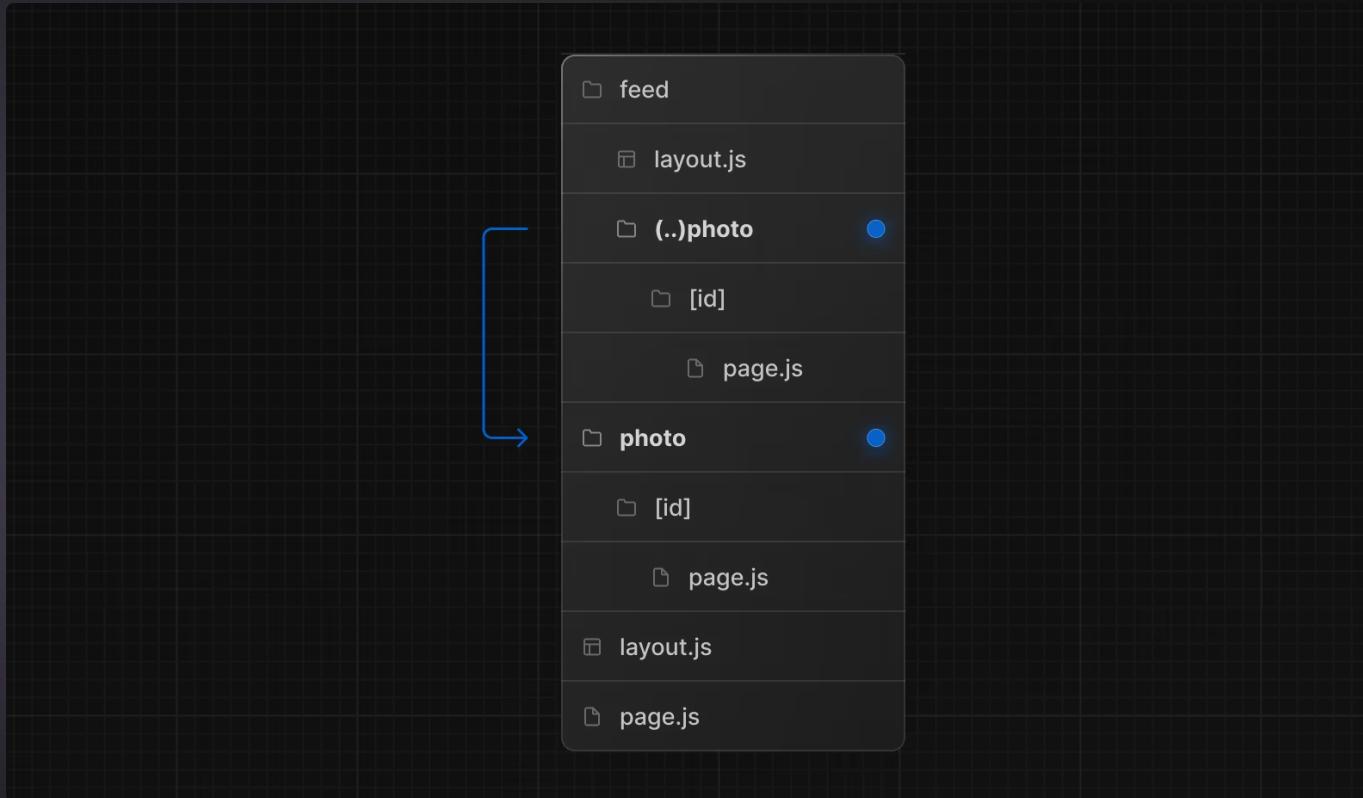


# 平行路由 (插槽)



# 拦截路由

Dribbble

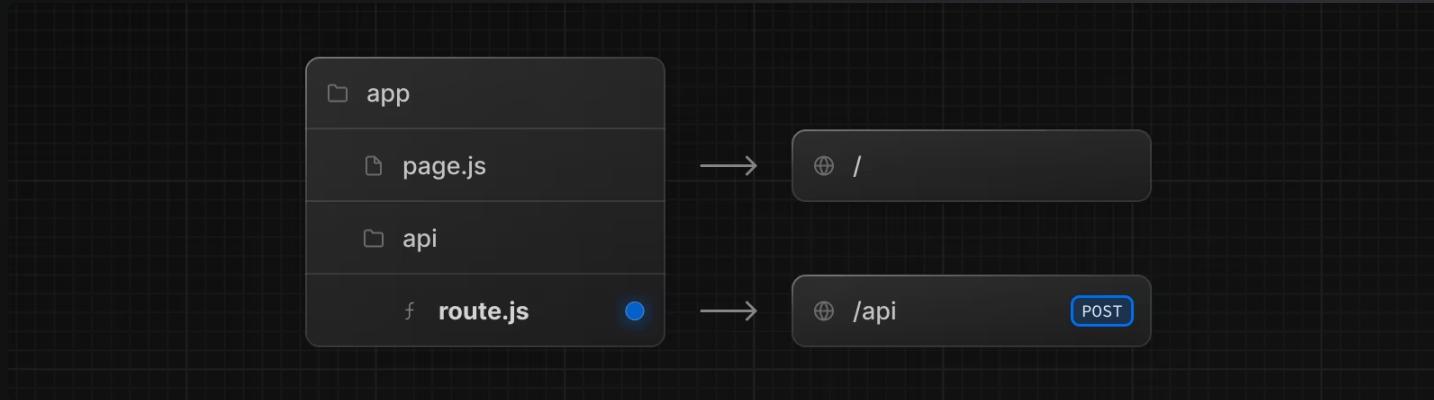


- Library and Framework
- 渲染方式: CSR/SSR/SSG/ISR
- RSC 与 Next.js App Router
- Next.js 路由系统
- **全栈框架的后端能力**
- Next.js 的 UX 优化
- SPA 模式下的 Framework

# Next.js 的后端能力

- 路由处理程序
- 服务端组件
- Server Actions
- Remix 的处理方式

# 路由处理程序



```
import { NextRequest, NextResponse } from "next/server"

export async function POST(req: NextRequest) {
  const { startTime, duration, type } = await req.json()
  return NextResponse.json({
    success: true,
    message: "create pomodoro success",
  })
}
```

# 服务端组件

```
export default async function Page() {
  const md = fs.readFileSync(`.${process.cwd()}/README.md`, "utf8")
  const html = marked(md)

  return <div dangerouslySetInnerHTML={{ __html: html }} />
}
```

# Server Actions

```
'use client'

export default function Page() {
  async function createInvoice(formData: FormData) {
    'use server'

    const rawFormData = {
      customerId: formData.get('customerId'),
      amount: formData.get('amount'),
      status: formData.get('status'),
    }

    supabase.from('invoices').insert(rawFormData)
  }

  return <form action={createInvoice}> ... </form>
}
```

# Remix 的处理方式

```
export async function action({
  request,
}: ActionFunctionArgs) {
  const body = await request.formData();
  const todo = await fakeCreateTodo({
    title: body.get("title"),
  });
  return redirect(`~/todos/${todo.id}`);
}

export async function loader() {
  return json(await fakeGetTodos());
}

export default function Todos() {
  const data = useLoaderData<typeof loader>();
  return (
    <div>
      <TodoList todos={data} />
      <Form method="post">
        <input type="text" name="title" />
        <button type="submit">Create Todo</button>
      </Form>
    </div>
  );
}
```

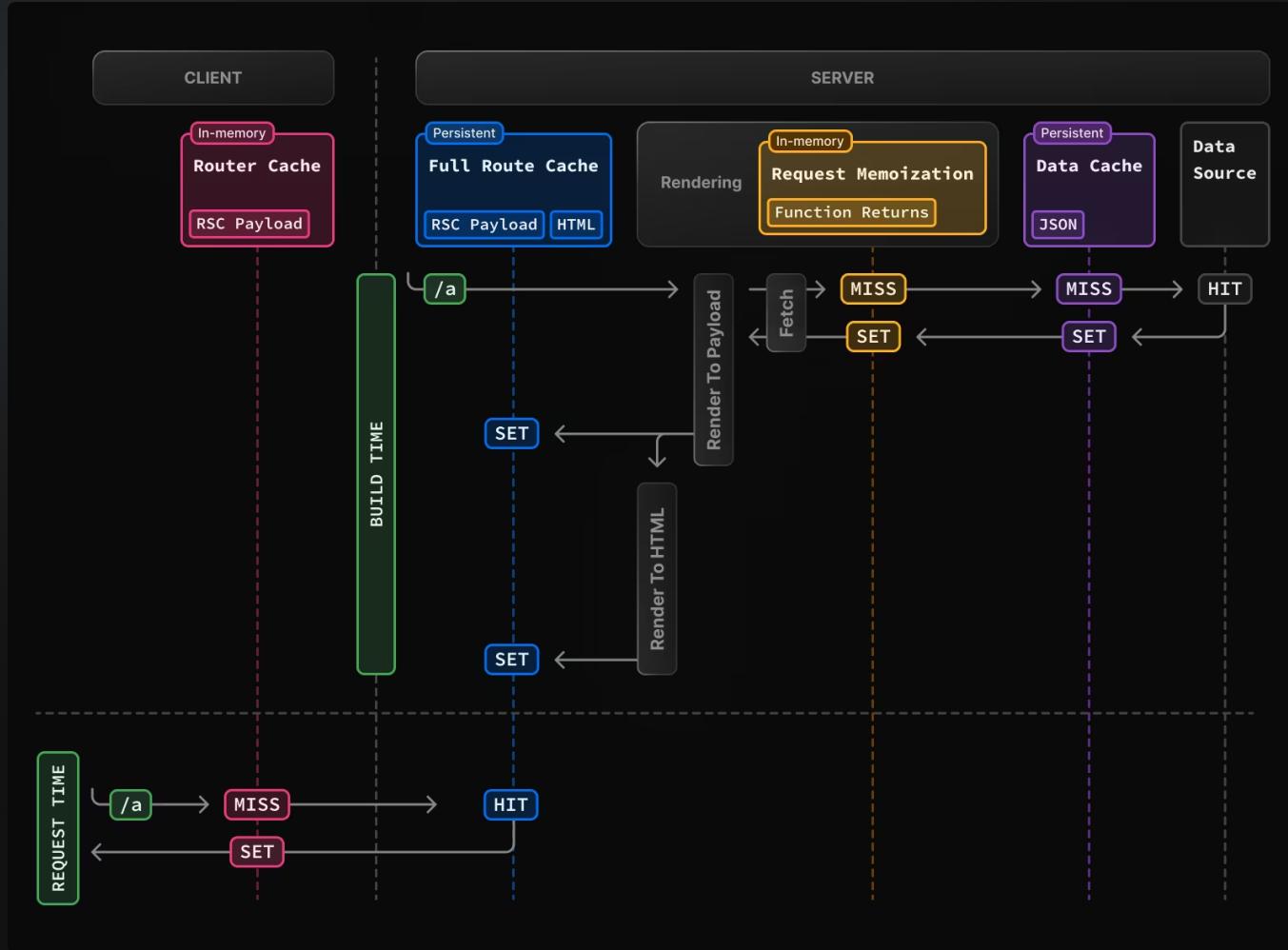
- Library and Framework
- 渲染方式: CSR/SSR/SSG/ISR
- RSC 与 Next.js App Router
- Next.js 路由系统
- 全栈框架的后端能力
- **Next.js 的 UX 优化**
- SPA 模式下的 Framework

# Next.js 的 UX 优化

- Cache
- 定制组件 Link Img

# Cache

机制	缓存内容	存储地方	目的	期间
请求记忆 (Request Memoization)	函数返回值	服务端	在 React 组件树中复用数据	每个请求的生命周期
数据缓存 (Data Cache)	数据	服务端	跨用户请求和部署复用数据	持久 (可重新验证)
完整路由缓存 (Full Route Cache)	HTML 和 RSC payload	服务端	降低渲染成本、提高性能	持久 (可重新验证)
路由缓存 (Router Cache)	RSC payload	客户端	减少导航时的服务端请求	用户会话或基于时间



# RSC

## 1. 性能优化

- 减小客户端包体积
- 更快的首屏渲染

## 2. SEO 友好

## 3. 数据获取更快更安全

- 服务端获取数据更安全
- 更快看到数据

## 4. 代码更易维护

- 客户端与服务端代码复用
- 逻辑集中

# 定制组件

- Link
- Image
- Metadata
- ...

- Library and Framework
- 渲染方式: CSR/SSR/SSG/ISR
- RSC 与 Next.js App Router
- Next.js 路由系统
- 全栈框架的后端能力
- Next.js 的 UX 优化
- **SPA 模式下的 Framework**

# Next.js 静态导出

## 1. 服务端组件

next build 时生成静态 HTML

## 2. 客户端组件

使用 API 获取数据

## 3. 路由处理器

只支持 GET，执行并生成静态的 HTML json 等文件

## 4. 不支持的功能

Cookies、Headers、Redirects、Middleware、ISR 等

## 5. 部署

任何静态资源服务器，例如 Nginx

# 谢谢！

Slides on [slides.haydenhayden.com](http://slides.haydenhayden.com)

---