# Server Side Render

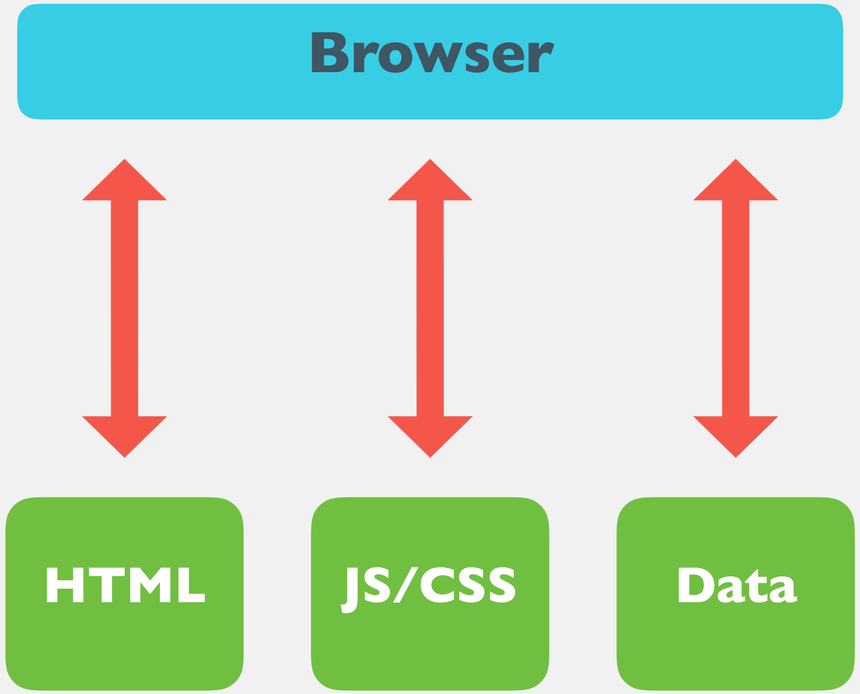
# 内容介绍

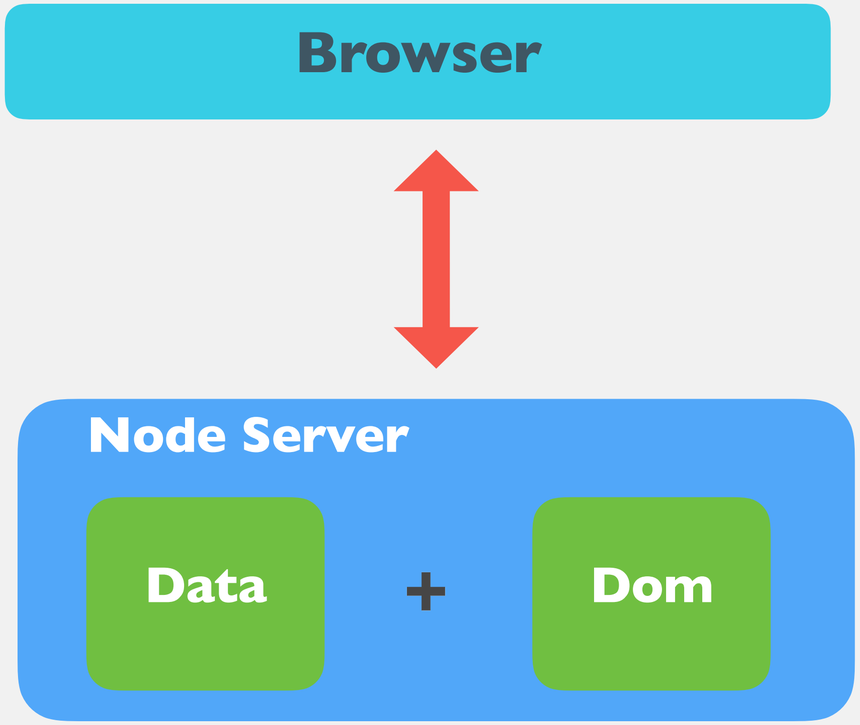
1. 什么是 SSR

2. SSR 的构建流程及架构

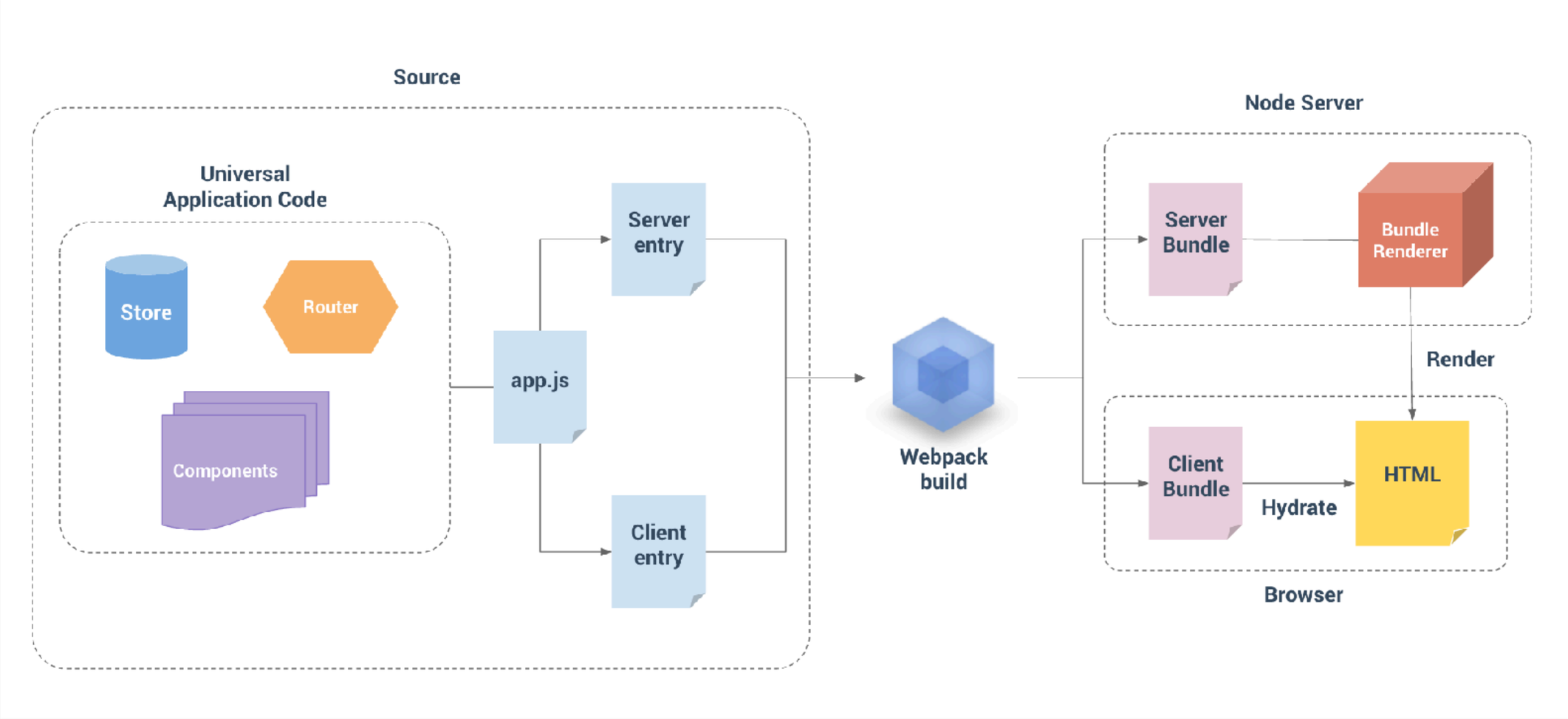
3. 演示一个 SSR 的 Demo

# 什么是服务端渲染

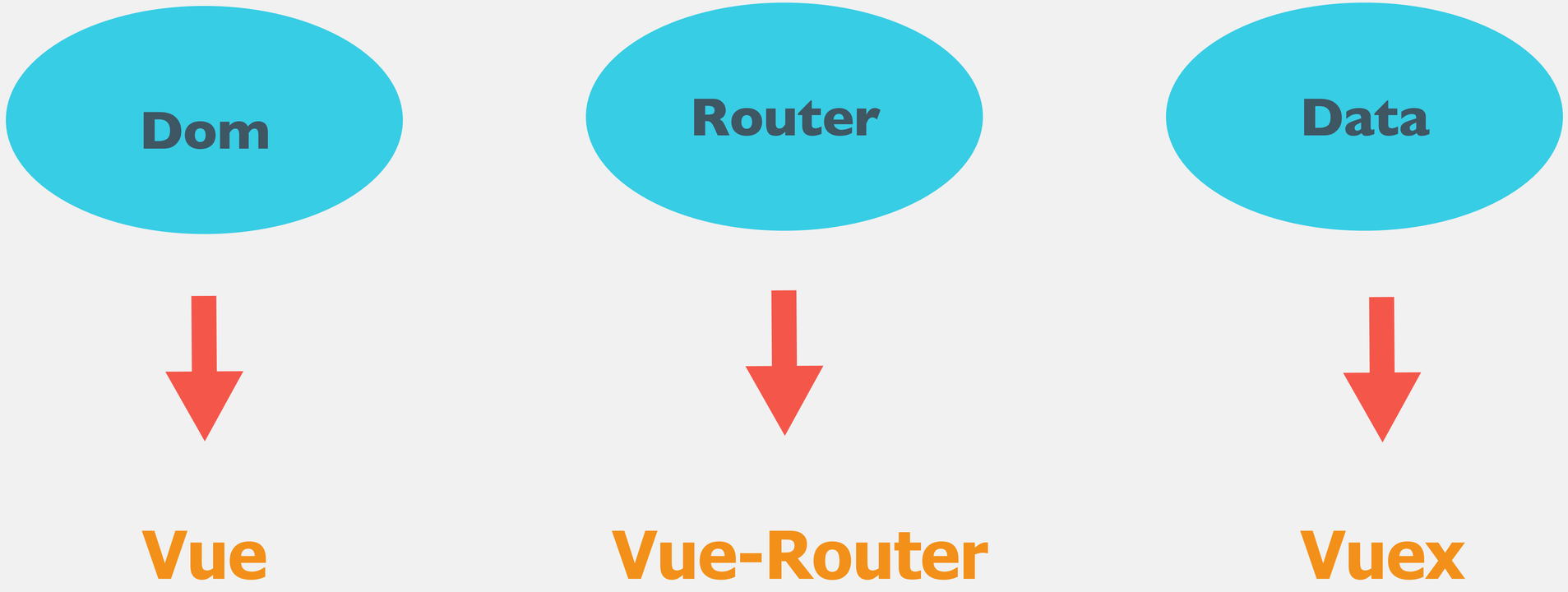


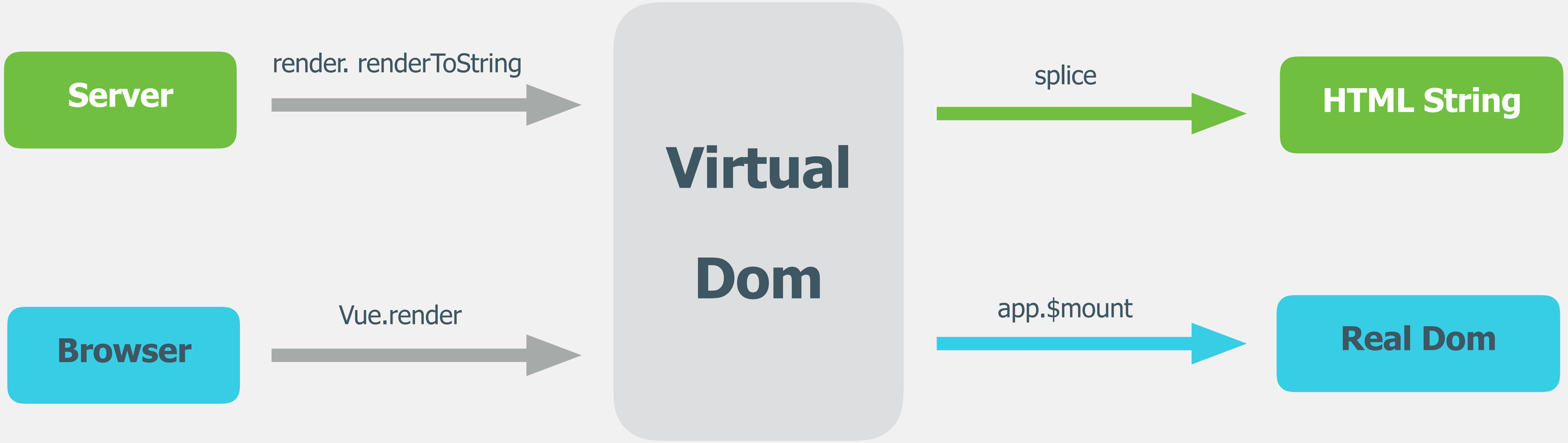
# Vue SSR的实现流程图



# 服务端渲染三大要素



# DOM 一致性


Server
render. renderToString
Virtual
Dom
splice
HTML String
Browser
Vue.render
app.$mount
Real Dom

# 思考一个问题

```
<!DOCTYPE html>
<html lang="zh_CN">
  <head>
    <title>{{ title }}</title>
    <meta charset="utf-8"/>
    <meta name="mobile-web-app-capable" content="yes"/>
    <meta http-equiv="X-UA-Compatible" content="IE=edge, chrome=1"/>
    <meta name="renderer" content="webkit"/>
    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-
    <link rel="shortcut icon" sizes="48x48" href="/public/favicon.ico"/>
    <meta name="theme-color" content="#f60"/>
  </head>
  <body>
    <!--vue-ssr-outlet-->
  </body>
</html>
```

# Router 一致性

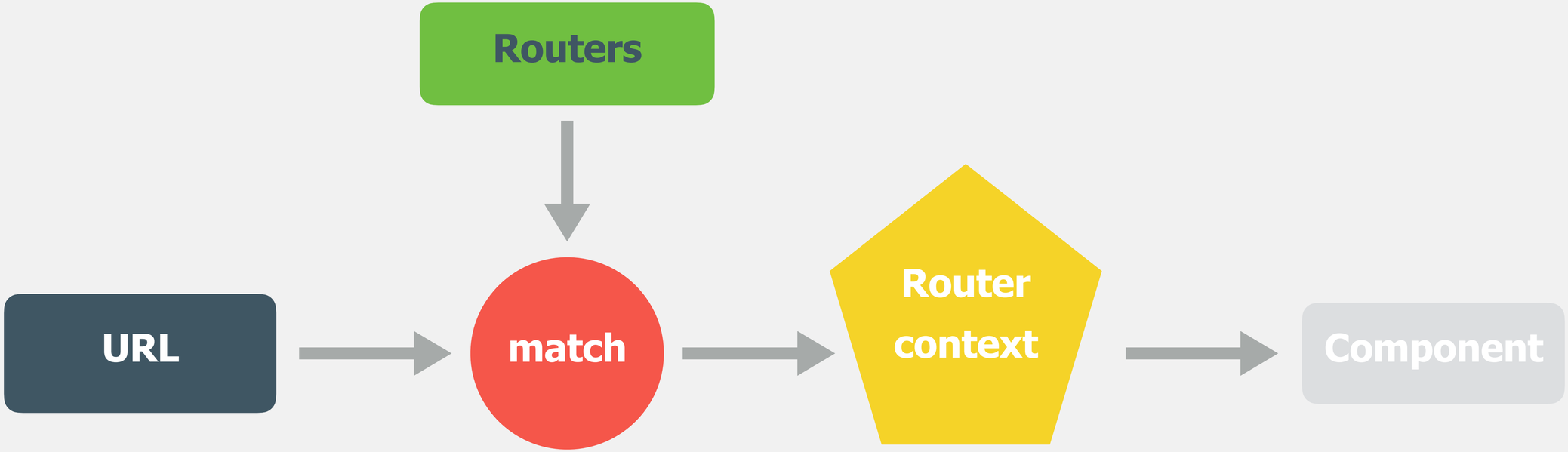
```
export function createRouter () {
    return new Router({
        mode: 'history',
        scrollBehavior: () => ({ y: 0 }),
        routes: [
            { path: '/home', component: Home },
            { path: '/join', component: Join },
            { path: '/activity', Activity },
            { path: '/mobile', Mobile },
            { path: '/', redirect: '/home' }
        ]
    })
}
```

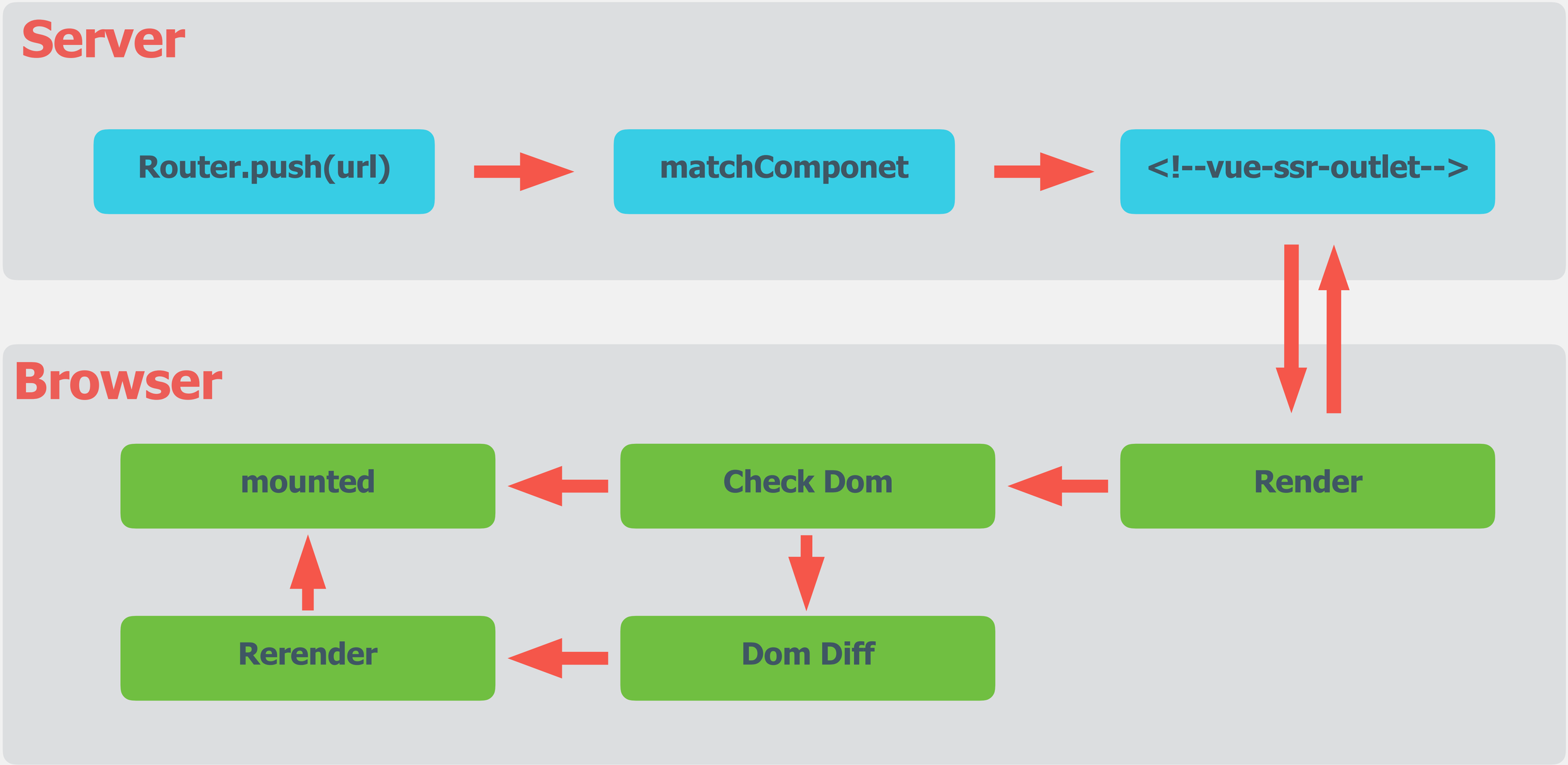
创建 **Router** 实例

```
import Vue from 'vue'
import App from './App.vue'
import { createRouter } from './router'
export function createApp () {
    const router = createRouter()
    const app = new Vue({
        router,
        render: h => h(App)
    })
    return { app, router }
}
```

注入 **router** 到根 **Vue** 实例

# Router 一致性

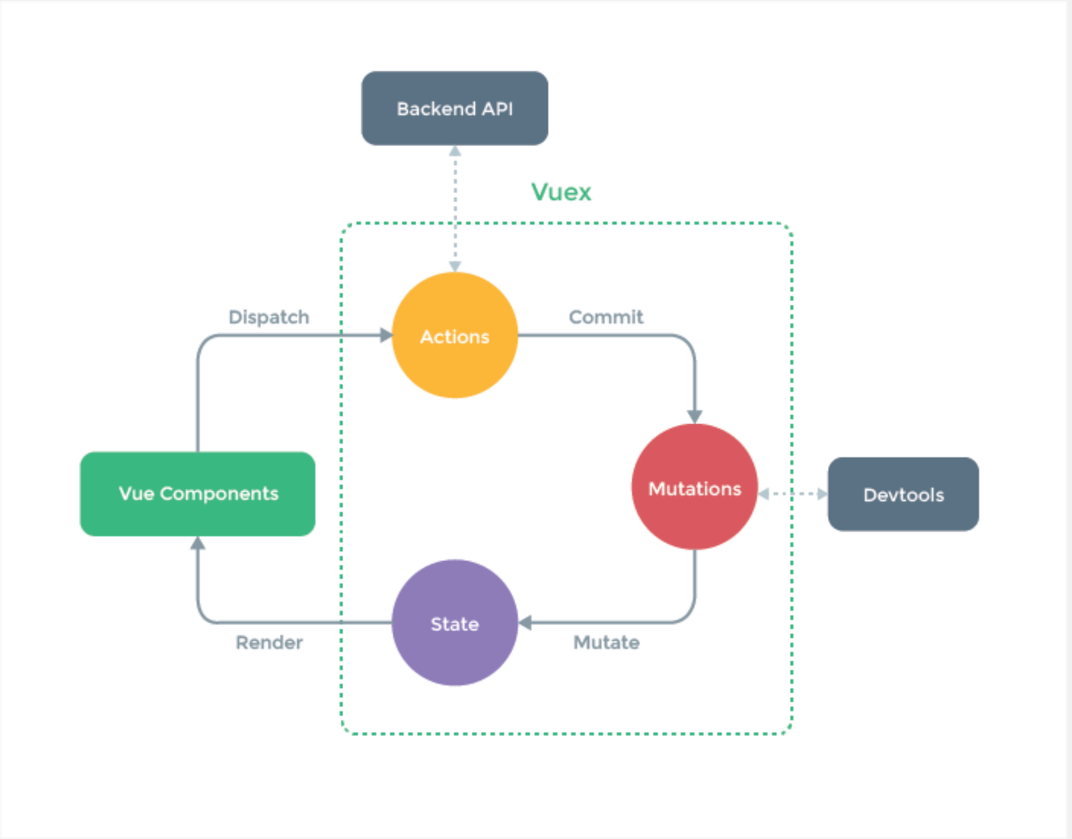



Server
Router.push(url)
matchComponet
<!--vue-ssr-outlet-->
Browser
mounted
Check Dom
Render
Rerender
Dom Diff

# Data 一致性

**Vuex 单一数据对象**

**用于保存页面的所有状态**



# Data 一致性

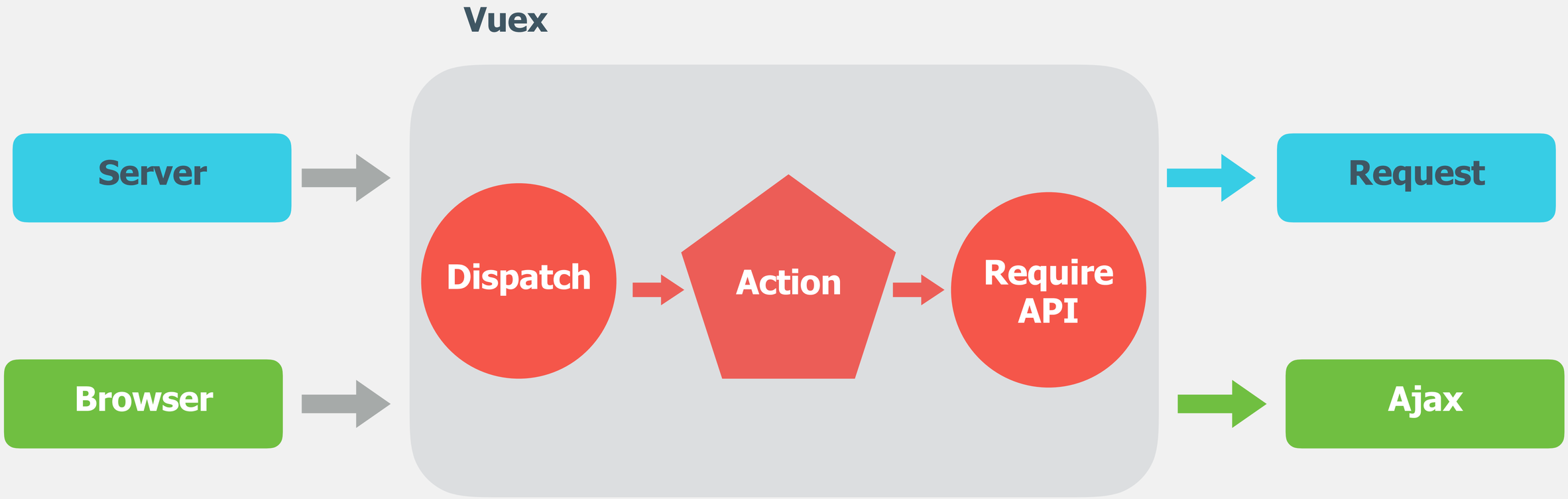
```
export function createStore () {
    const store = new Vuex.Store({
        state,
        actions,
        mutations,
        getters,
        modules: {
            match,
            user
        },
        plugins: [persistence],
        strict: process.env.NODE_ENV !== 'production'
    })
    return store
}
```

创建 Vuex 实例

```
export function createApp (ssrContext) {
    const store = createStore()
    const router = createRouter()
    const i18n = createI18n()
    sync(store, router)
    const app = new Vue({
        router,
        store,
        i18n,
        ssrContext,
        render: h => h(App)
    })
    return { app, router, store }
}
```

注入 Store 到根 Vue 实例

# Data 一致性




同构的过程中，如何去加载不能的 API 方法

# Data 一致性

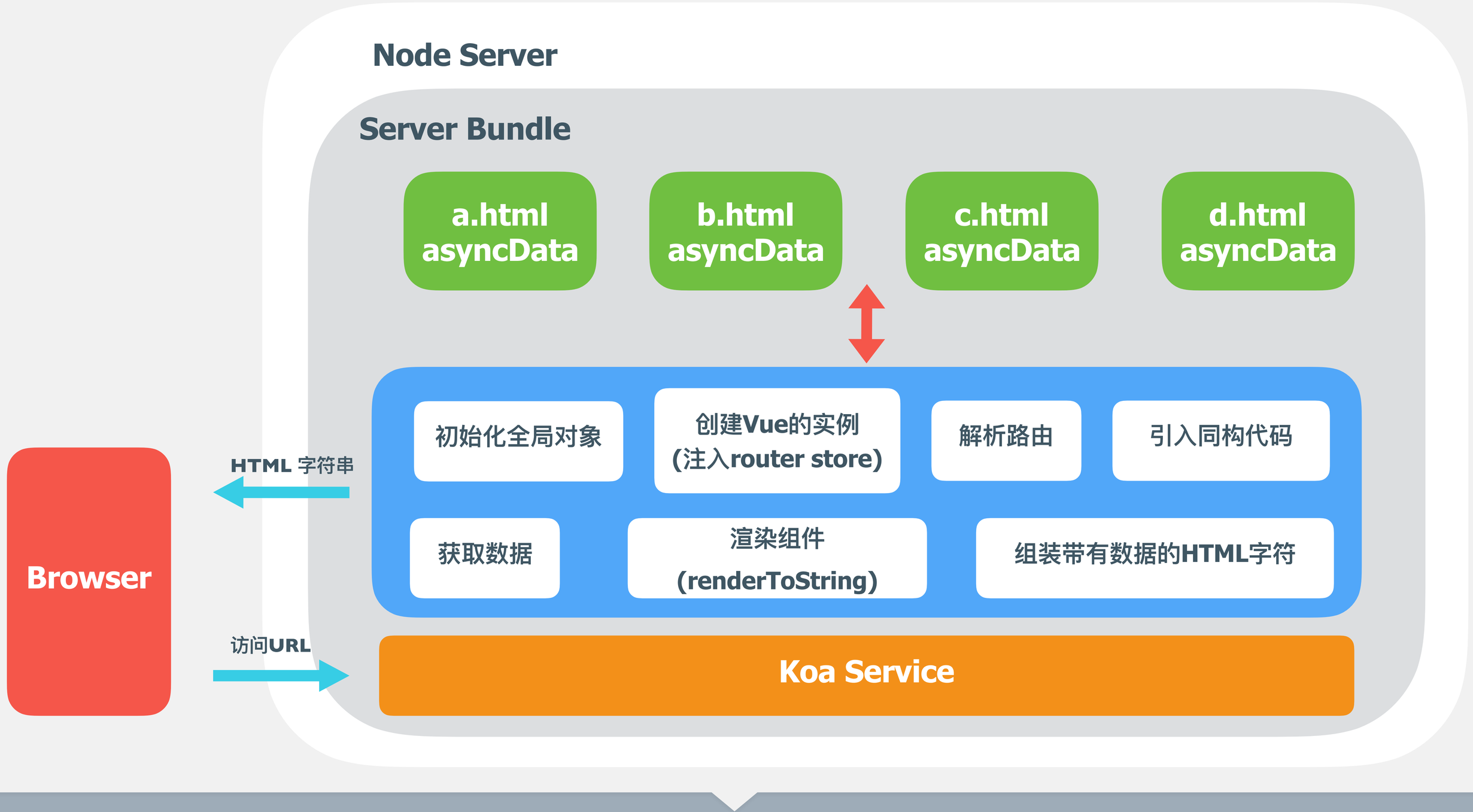
Server Data Prefetch

asyncData

window._INITIAL_STATE

Client Replace Data

```
Store.replace(
    window._INITIAL_STATE
)
```


Node Server
Server Bundle
a.html
asyncData
b.html
asyncData
c.html
asyncData
d.html
asyncData
初始化全局对象
创建Vue的实例
(注入router store)
解析路由
引入同构代码
获取数据
渲染组件
(renderToString)
组装带有数据的HTML字符
Browser
HTML 字符串
访问URL
Koa Service

# SSR 项目结构

```
├── build/                  # build 脚本
├── src/                    # Vue 核心业务
│   ├── api/                # 接入微服务的基础 API
│   ├── assets/             # 静态文件
│   ├── components/         # 组件
│   ├── config/             # 配置文件
│   ├── lang/               # 语言包
│   ├── router/             # 路由
│   ├── store/              # Vuex 状态管理
│   ├── utils/              # 通用的工具类函数
│   ├── view/               # 各个页面
│   ├── app.js              # Vue 对象初始化
│   ├── App.vue             # Vue 的框架页面
│   ├── entry-client.js     # 前端业务
│   ├── entry-server.js     # 服务端业务
│   └── index.template.html # HTML 模版
├── server.js               # koa 服务，处理请求
└── package.json
```

## 基础设施

api/

utils/

lang/

config/

## 表现层

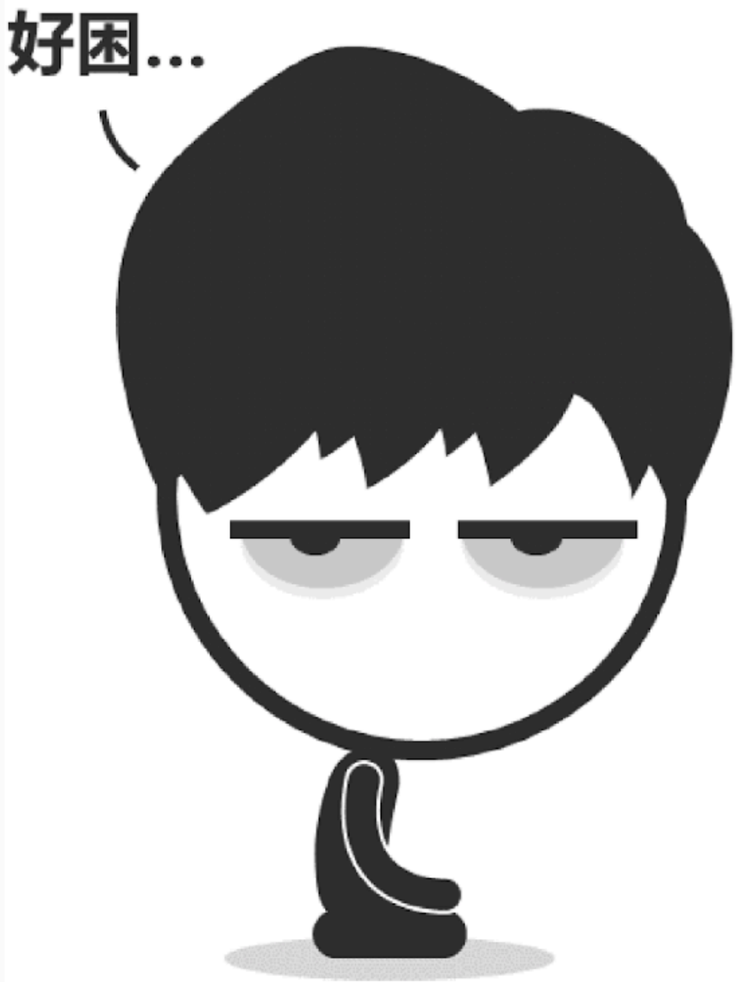
state/

router/

component/

view/

*Vue* 下具体的交互展示层业务




# Talk is cheap
# Show me the code

# 提问

1. 同构是如何避免服务端和客户端交叉请求引起的状态污染?

2. 你们觉得 SSR 应该怎么使用?

# 下集预告

1. 如何做缓存?
2. 如何初始化用户数据?
3. 踩过的 SSR 的坑。。。

# THE END!