۸۰

سوال مصاحبه فرانت‌اند با توضیح اختصاصی

سلام دوستان! 👋

وقتی مشغول مصاحبه با شرکت‌ها برای شغل جدیدم بودم، لیستی از سوالات و نکته‌هایی که فکر می‌کردم مهم هستن و ممکنه مستقیم یا غیر مستقیم توی مصاحبه‌ها پرسیده بشه رو جایی ذخیره می‌کردم و قبل از هر مصاحبه‌ای اونها رو مرور می‌کردم. حالا قصد دارم همهٔ اون سوالات رو با توضیحات اختصاصی برای شما هم به اشتراک بذارم.

این نسخهٔ PDF مجموعه پست‌هایی هست که مدتی قبل توی وبسایت دیتی منتشر شد. امیدوارم استفاده کنین، توی کارِتون پیشرفت کنین و توی بهترین جاها فعالیت کنین.

قبل از ادامه یادی کنیم از صابر راستی کردار عزیز. امیدوارم با دعای من و شما روحش شاد و در آرامش باشه ✨💚🌹 همین فونت قشنگی که من با اون این سوالات رو نوشتم و شما هم با اون مشغول خوندن اونها هستین، حاصل کار صابرِ عزیز هست.

خب، بریم که سوالا رو بررسی کنیم 🔥

۱. چه تکنیک‌هایی برای بهینه‌سازی یک برنامهٔ فرانت‌اند می‌شناسین؟

برای بهینه‌سازی یک برنامهٔ فرانت‌اند تکنیک‌های زیادی وجود داره. بعضی از مهمترین اونها مثل:

- بهینه‌سازی فایل‌های برنامه (باندل) با تکنیک‌هایی مثل Tree shaking و Compression و Minification
- حذف Import های بلااستفاده
- جداسازی کدها و تقسیم اونها به فایل‌های کوچیک‌تر (Code Splitting) با استفاده از تکنیک‌هایی مثل Dynamic Import
- استفاده از Lazy Loading برای لود کردن محتوا فقط در شرایط نیاز
- استفاده از کامپوننت‌های Async هنگام استفاده از فریم‌ورک‌ها
- لود کردن کامپوننت‌ها فقط در شرایط نیاز (Conditional components) (loading
- مدیریت کردن درخواست‌های API با استفاده تکنیک‌هایی مثل AbortController

بیشتر بدانید

همه چیز از AbortController جاوااسکریپت

AbortController یک قابلیت کاربردی توی جاوااسکریپت هست که با اون خیلی راحت می‌تونیم یک یا چند عملیات Async رو در زمان لزوم متوقف کنیم

۲. فایل package-lock.json توی پروژه‌های فرانت‌اندی چیه و چه کاربردی داره؟

این فایل به صورت خودکار توسط ابزارهای مدیریت پکیج مثل npm و yarn تولید میشه و توی اون، ورژن دقیق پکیج‌هایی که توی برنامه استفاده شده ذخیره میشه. وقتی پروژهٔ ما توی محیط‌های دیگه Deploy میشه یا یک توسعه‌دهندهٔ جدید قراره روی اون کار کنه، وجود فایل `package-lock.json` هنگام اجرای دستور `npm install` این اطمینان رو به ما میده که دقیقاً همون ورژن از پکیج‌هایی برای ما نصب بشن که انتظار داریم. این کار باعث میشه که برنامهٔ ما توی محیط‌های مختلف رفتارهای یکسان و قابل پیش‌بینی داشته باشه.

برای اطلاعات کامل‌تر پست زیر رو ببینین:

بیشتر بدانید

فایل package-lock.json توی پروژه‌های جاوااسکریپتی چیه و چه کاربردی داره؟

توی برنامه‌های جاوااسکریپتی یک فایل کاربردی به اسم package-lock.json وجود داره که هدف اون حفظ سازگاری بین نسخه‌هایی از برنامه هست که توی جاهای مختلف نصب میشن

۳. چطوری می‌تونیم یک برنامهٔ فرانت‌اندی Maintainable داشته باشیم؟

برنامهٔ Maintainable برنامه‌ای هست که به راحتی و در هر زمانی می‌تونیم اون رو ویرایش کنیم، باگ‌های اون رو برطرف کنیم، توسعه بدیم و به اون توسعه‌دهنده‌های دیگه‌ای اضافه کنیم. چنین برنامه‌ای **ارزش نگهداری** داره یا به‌قول معروف Maintainable هست. برنامه‌ای که کمترین هزینه و بیشترین منفعت رو برای کسب و کارها به همراه داره.

موارد زیر کمک می‌کنن که یک برنامهٔ Maintainable داشته باشیم:

1. Best Practice ها و استانداردهای معروف و همگانی رو رعایت کنیم
2. از فریم‌ورک‌ها و کتابخونه‌های معروف استفاده کنیم
3. ابزارها و کتابخونه‌ها رو به‌طور مرتب بروزرسانی کنیم
4. به‌صورت ماژولار پروژه‌مون رو توسعه بدیم
5. از ابزارهای تست خودکار (Automated test) استفاده کنیم
6. از قسمت‌های مختلف برنامه و نحوهٔ کار با اونها مستندات تهیه کنیم
7. از قابلیت‌های ابزارهای Version Control استفاده کنیم

برای جزییات بیشتر می‌تونین پست زیر رو ببینین:

بیشتر بدانید

منظور از برنامهٔ Maintainable چیه و چطوری یک برنامهٔ فرانت‌اندی Maintainable داشته باشیم؟

روش‌هایی که کمک می‌کنن برنامه‌ای داشته باشیم که به‌راحتی و در هر زمانی میشه اون رو ویرایش کنیم، توسعه بدیم، باگ‌های اون رو برطرف کنیم و به راحتی به اون توسعه دهنده اضافه کنیم

۴. کاربرد Generic ها توی تایپ‌اسکریپت چیه؟

Generic ها کمک می‌کنن بتونیم توابع، کلاس‌ها و اینترفیس‌هایی داشته باشیم که

Reusability (قابلیت استفاده مجدد) بیشتری دارن. فرض کنیم تابعی داریم که پارامترهای اون محدود به نوع `number` هستن:

```
function getLastItem(items: number[]): number {
  return items[items.length - 1];
}
```

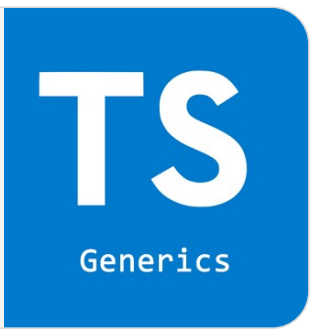
به این تابع نمی‌تونیم مقادیر `string` پاس بدیم که در نتیجه برای حل این مشکل باید اون تابع رو برای نوع `string` بازنویسی کنیم که این یعنی کدهای تکراری. Generic ها برای حل چنین مشکلاتی معرفی شدن. تابع بالا رو با استفاده از Generic می‌تونیم به این شکل بنویسیم:

```
function getLast<T>(items: T[]): T {
  return items[items.length - 1];
}
```

حالا این تابع می‌تونه برای هر نوعی استفاده بشه. کافیه موقع استفاده از اون نوع مد نظرمون رو مشخص کنیم:

```
getLast<number>([1, 2, 3]);       // typeof number
getLast<string>(["a", "b", "c"]); // typeof string
```

برای آشنایی بیشتر با Generic ها می‌تونین این پست رو ببینین:



بیشتر بدانید

آموزش Typescript به زبان ساده قسمت 10 - Generic ها

Generic ها در تایپ‌اسکریپت رو توی این پست به طور کامل بررسی میکنیم

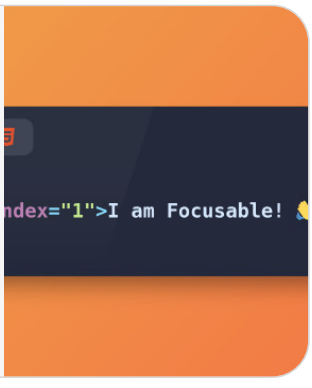
۵. اتریبیوت tabindex توی HTML چیه؟

این اتریبیوت برای راحتی جابجایی بین المنت‌های توی صفحه با استفاده از دکمهٔ Tab روی کیبورد به کار میره و اگه از اون به درستی استفاده کنیم می‌تونه باعث بهبود Accessibility و کارایی صفحهٔ ما بشه. بعضی از المنت‌ها مثل `input` و `button` به‌صورت خودکار این اتریبیوت رو دارن و به قول معروف Focusable هستن و به همین دلیل هست که وقتی توی یک فرم از دکمهٔ Tab استفاده می‌کنیم، می‌تونیم به راحتی بین ورودی‌های اون فرم جابجا بشیم بدون اینکه بصورت دستی و با ماوس اونها رو Focus کنیم.

نحوهٔ استفاده از این اتریبیوت به صورت زیر هست:

```
<div tabindex="0">I'm focusable!</div>
```

المنت `div` با استفاده از دکمهٔ Tab کیبورد قابل انتخاب شدن هست. البته نحوهٔ استفاده از این اتریبیوت اهمیت زیادی داره که پیشنهاد می‌کنم پست زیر رو بخونین:



بیشتر بدانید

بایدها و نبایدهای tabindex توی HTML

توی HTML یک اتریبیوت به اسم tabindex داریم که کمک می‌کنه بتونیم با استفاده از کیبورد بین المنت‌های صفحه جابجا بشیم. این اتریبیوت خاصیت‌ها و باید و نبایدهایی داره که توی این پست می‌خوایم با اون آشنا بشیم

۶. آیا جاوااسکریپت یک زبان Parallel هست یا Concurrent؟

هرچند به جاوااسکریپت مدرن قابلیت‌هایی مثل Web Workers اضافه شده که توی بعضی از کاربردهای خاص می‌تونه شبیه به یک زبان Parallel عمل کنه، اما جاوااسکریپت به دلیل اینکه Single Thread هست، یک Call Stack داره و بنابراین ذاتاً یک زبان Concurrent هست.

بیشتر بدانیم

Parallelism چیه؟

وقتی می‌گیم یک زبان Parallel هست یعنی این زبان می‌تونه چندین کار رو به صورت هم‌زمان و مجزا شروع و پردازش کنه. فرض کنیم می‌خوایم پردازشی روی یک مجموعه اطلاعات عظیم انجام بدیم. با یک زبان Parallel می‌تونیم این اطلاعات رو به قسمت‌های کوچیک‌تر تقسیم و هر بخش رو به صورت مجزا پردازش کنیم که باعث میشه با سرعت بیشتری به نتیجه برسیم. یک زبان Parallel می‌تونه چندین تسک رو به صورت کاملاً همزمان شروع و پردازش کنه.

Concurrency چیه؟

وقتی می‌گیم یک زبان Concurrent هست، یعنی اون زبان می‌تونه توی یک لحظه چندین تسک رو مدیریت و پردازش کنه. یک نمونهٔ اون می‌تونه اجرای ۵ درخواست Ajax باشه. از لحاظ فنی، توی زبانی مثل جاوااسکریپت این ۵ درخواست در یک واحد از زمان قابل شروع شدن نیستن. اما بعد از شروع شدن

می‌تونن به صورت همزمان پردازش بشن.

۷. چه زمانی از Tuple توی تایپ‌اسکریپت استفاده نکنیم؟

با Tuple توی تایپ‌اسکریپت می‌تونیم یک تایپ Array-like (شبه آرایه) داشته باشیم که توی اون طول آرایه و نوع دقیق اعضا مشخص شده. تایپی که توی خط ۱ کد زیر می‌بینین یک Tuple هست که همونطور که می‌بینیم سینتکس عجیب غریبی نداره:

```
1  type Name = [string, string];
2
3  const john: Name = ["John", "Doe"];
```

متغیر `john` که از تایپ `Name` استفاده کرده، حتماً باید یک آرایه با طول ۲ باشه که عضو اول و دوم اون از نوع `string` هستن. خاصیت Tuple عدم انعطاف‌پذیری اون هست و استفاده نادرست از اون ممکنه باعث بروز مشکلاتی توی برنامه بشه.

مثال بالا یک نمونهٔ استفاده نادرست از Tuple هست. چون برای مثال از `Name` برای نام و نام خانوادگی‌هایی که بیشتر از ۲ کلمه دارن نمی‌تونیم استفاده کنیم. برای این کار باید تایپ `Name` رو دستکاری کنیم که ممکنه باعث بروز باگ توی قسمت‌های دیگهٔ برنامه بشه. همچنین توی بعضی از فرهنگ‌ها ابتدا نام خانوادگی ذکر میشه و بعد نام کوچک. بنابراین اینجا وقتی از Tuple استفاده می‌کنیم دقیقاً نمی‌دونیم `john[0]` نام کوچک شخص هست یا نام خانوادگی.

پس به‌طور کلی، بهتره از Tuple زمانی استفاده کنیم که مطمئن هستیم طول و نوع اعضای آرایهٔ ما هیچوقت تغییر نمی‌کنه. مثلاً ذخیره کردن طول و عرض جغرافیایی:

```
1  type Coordinates = [number, number];
2
3  const c1: Coordinates = [46.111341, -151.005893];
4  const c2: Coordinates = [27.092156, 51.302929];
```

۸. توی تایپ‌اسکریپت کلمه‌کلیدی declare چکار

می‌کنه؟

اگه توی یک فایل تایپ‌اسکریپتی با یک مقدار مثل یک تایپ، آبجکت و ... سر و کار داریم که برای تایپ‌اسکریپت ناشناخته و غیر قابل دسترس هست، کامپایلر به ما خطا میده که این مقدار تعریف نشده. برای مثال توی برنامه یک متغیر گلوبال داریم و می‌خوایم از اون توی یک فایل تایپ‌اسکریپتی استفاده کنیم:

```
// myModule.ts

console.log(myGlobalVariable); // Error: Cannot find name 'myGlobalVariable'.
```

توی این مثال در شرایطی که ما می‌دونیم چنین مقداری وجود داره و اگه برنامه باندل بشه اون مقدار قابل دسترس خواهد بود، کامپایلر به ما خطا داد که این مقدار وجود نداره. برای رفع چنین خطاهایی قابلیتی به اسم `declare` به کارمون میاد.

با استفاده از کلمه‌کلیدی `declare` می‌تونیم به تایپ‌اسکریپت بگیم که: «با اینکه می‌دونم چنین مقداری برای تو ناشناخته هست، لطفاً طوری رفتار کن که انگار می‌شناسیش. لطفاً به من اعتماد کن.»

نحوهٔ استفاده از کلمه‌کلیدی `declare` به این صورت هست:

```
declare var myGlobalVariable: string;

console.log(myGlobalVariable);
```

با این کار، دیگه خطایی از سمت کامپایلر تایپ‌اسکریپت نمی‌گیریم. دقت کنیم declare فقط برای رفع خطاهای هنگام توسعه به‌کار میره و زمانی که برنامه به‌طور واقعی اجرا میشه اگه مقداری واقعاً وجود نداشته باشه خطا می‌گیریم. بنابراین `declare` توی فاز Runtime تأثیری نداره.

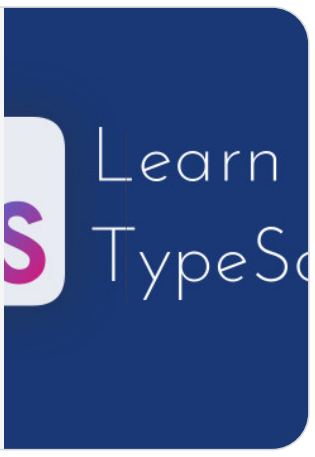
۹. چند تا از تایپ‌های پرکاربرد تایپ‌اسکریپت رو نام ببرین

تایپ‌اسکریپت علاوه بر اینکه اجازه میده تایپ‌های مد نظر خودمون رو بسازیم، چند تایپ کاربردی رو ارائه میده که با اونها می‌تونیم تایپ‌های موجود رو دستکاری و به یک حالت دیگه تبدیلشون کنیم. به این تایپ‌ها که بصورت گلوبال در دسترس

هستن می‌گیم Utility Types. موارد زیر چند تا از پرکاربردترین این تایپ‌ها هستن:

- `Partial<Type>`
- `Required<Type>`
- `Record<Keys, Type>`
- `Omit<Type, Keys>`
- `ReturnType<Type>`

برای آشنایی کامل با این تایپ‌ها می‌تونین پست زیر رو ببینین:



بیشتر بدانید

آشنایی با Utility Type های پرکاربرد تایپ‌اسکریپت - قسمت اول

تایپ‌اسکریپت با ارائه دادن یک قابلیت کاربردی به اسم Utility Types کمک می‌کنه خیلی راحت‌تر و سریع‌تر تایپ‌های دلخواهمون رو بسازیم که توی این قسمت‌ها می‌خوایم با اونها آشنا بشیم.

۱۰. دستور Git زیر چکار می‌کنه؟

```
1  git commit --amend
```

فلگ `--amend` دو کاربرد داره:

۱. اگه در حال حاضر تغییراتی رو commit کردیم، اما یک سری فایل از این کامیت جا موندن، می‌تونیم بعد از Stage کردن اونها (دستور `git add`) از دستور بالا استفاده کنیم تا فایل‌های جدید به آخرین کامیت اضافه بشن.

۲. اگه قصد داریم متن پیام آخرین کامیت رو تغییر بدیم، می‌تونیم از دستور بالا استفاده کنیم. با این کار ادیتور پیشفرض برای ما باز میشه و می‌تونیم متن قبلی رو اصلاح کنیم.

. . .

Resources:

- https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/tabindex

۱۱. منظور از Mobile-first Strategy چیه؟

Mobile-first Strategy یک روش و یک استراتژی طراحی صفحات وب (UI و UX) هست که توی اون همونطور که از عنوان یعنی Mobile-first مشخصه، دستگاه‌های موبایلی به عنوان اولویت اصلی برای طراحی صفحات قرار می‌گیرن و بعد پلتفرم‌های دیگه مثل دسکتاپ.

توی روش‌های قدیم، معمولاً صفحات ابتدا برای پلتفرم‌های دستکاپ طراحی می‌شد و بعد در صورت نیاز برای پلتفرم‌های موبایلی. اما با گسترش اینترنت و توجه بیشتر مردم به دستگاه‌های موبایلی، چیزی که اهمیت بیشتری پیدا کرده رابط کاربری توی این دستگاه‌ها هست و به همین خاطر خیلی از کسب و کارها دستگاه‌های موبایلی رو به عنوان اولویت اصلی قرار میدن. به این استراتژی طراحی می‌گیم Mobile-first Strategy.

۱۲. Garbage Collection توی جاوااسکریپت چیه؟

Garbage Collection به پروسهٔ آزادسازی خودکار حافظه گفته میشه. توی این پروسه، جاوااسکریپت به طور خودکار مقادیری که دیگه توی برنامه مورد استفاده قرار نمی‌گیرن رو از حافظه پاک می‌کنه تا فضای حافظه آزاد بشه.

مثلاً متغیرهایی که توی Scope های داخلی یا توابع تعریف می‌کنیم، بعد از اینکه کار اون Scope یا تابع تموم بشه، اون متغیرها هم از حافظه پاک میشن. اما برای مثال متغیرهای گلوبال که همیشه در دسترس هستن، توسط Garbage Collector حذف نمیشن و تا آخر توی حافظه حضور دارن.

```
1  var iAmGlobal = "Hi";
2
3  function run() {
4    const iAmLocal = "Hi";
5    // ...
6  }
7
8  run();
```

توی این کد، وقتی کار تابع `run` به پایان می‌رسه، متغیری که داخل این تابع هست هم از بین میره و از حافظه پاک میشه. اما متغیر خط ۱ تا پایان کار برنامه قابل دسترس هست و Garbage Collector کاری با اون نداره.

۱۳. متدهای HEAD و OPTION چه تفاوت‌هایی با

هم دارن؟

توی مبحث RESTful API در کنار متدهای پرکاربردی مثل GET و POST، دو متد با کاربرد خاص وجود داره به اسم‌های HEAD و OPTION.

متد HEAD: وقتی توی یک درخواست HTTP از متد HEAD استفاده می‌کنیم، به معنی هست که از Response فقط به اطلاعات Header احتیاج داریم. بنابراین توی Response این درخواست، body وجود نخواهد داشت.

متد OPTION: اگه می‌خوایم اطلاعاتی کلی درباره قوانین و نحوهٔ تعامل با API مد نظر (مثل متدهای HTTP قابل استفاده و یا مجوز CORS) داشته باشیم، از این متد استفاده می‌کنیم. شاید دقت کرده باشین که مرورگر هنگام بررسی CORS، ابتدا یک درخواست با متد OPTION به آدرس مد نظر میزنه تا CORS رو بررسی کنه.

۱۴. چند تا از Best Practice های تست‌نویسی رو می‌شناسین؟

موارد زیر چند Best Practice برای تست کردن هر برنامه‌ای هست:

۱. خوانایی تست‌ها: تست‌های ما باید اونقدر واضح باشن که وقتی یک توسعه‌دهندهٔ دیگه اونها رو می‌بینه بتونه به آسونی متوجه اونها بشه.

۲. عنوان تست‌ها: همونطور که می‌دونیم تست‌ها معمولاً توی محیط ترمینال اجرا میشن و وقتی یک تست Fail میشه باید دقیقاً بدونیم چه تستی و چرا Fail شده. برای همین هنگام نام‌گذاری تست‌ها باید مشخص کنیم که: چه چیزی قراره تست بشه؟ تحت چه شرایطی؟ خروجی مورد انتظار چی هست؟

```
// 1. The unit under test
describe('Products Service', () => {
  describe('Add new product', () => {
    // 2. scenario and 3. expectation
    it('When no price is specified, then the product
status is pending approval', () => {
      const newProduct = new ProductService().add(...);

expect(newProduct.status).to.equal('pendingApproval');
    });
  });
});
```

۳. الگوی AAA: این الگو میگه بهتره تست‌هامون رو به ۳ قسمت مجزا تقسیم کنیم:

۱. Arrange: کدهایی که مخصوص راه‌اندازی و آماده‌سازی تست هست

۲. Act: کدهایی که مخصوص اجرای تست هست

۳. Assert: کدهایی که مخصوص بررسی و مقایسهٔ خروجی تست با چیزی که انتظار داریم هست

```
describe("Customer classifier", () => {
  test("When customer spent more than $500, should be classified as premium", () => {
    //Arrange
    const customerToClassify = { spent: 505, joined: new Date(), id: 1 };
    const DBStub = sinon.stub(dataAccess, "getCustomer").reply({ id: 1, classification: "regular" });

    //Act
    const receivedClassification = customerClassifier.classifyCustomer(customerToClassify);

    //Assert
    expect(receivedClassification).toMatch("premium");
  });
});
```

۴. تست کردن فقط متدهای Public: وقتی API ها و یا متدهای Public رو تست می‌کنیم، مطمئن می‌شیم که متدهای درونی و Private هم تست میشن. بنابراین نیازی به تست‌نویسی مجزا برای اونها نیست.

۵. استفاده از ابزارهای تولید دیتای Fake: بجای استفاده از کلماتی مثل Foo و Bar و لورم ایپسوم، بهتره از کتابخونه‌هایی استفاده کنیم که مخصوص تولید کردن دیتای Fake هستن تا بتونیم اطلاعاتی متنوع‌تر و واقعی‌تر داشته باشیم.

برای مشاهدهٔ لیست کامل می‌تونین این ریپازیتوری رو ببینین.

۱۵. دستور git cherry-pick چکار می‌کنه؟

فرض کنیم توی یک ریپازیتوری مشغول کار روی یک برنچ هستیم. از طرفی برنچ‌های دیگه‌ای هم وجود داره که توسط خودمون یا افراد دیگه داره روی اونها کار میشه. حالا می‌خوایم تغییراتی که توی یکی از برنچ‌ها وجود داره رو اضافه کنیم به برنچ خودمون. شاید به ذهنمون برسه که از دستور `git merge` استفاده کنیم. اما همونطور که می‌دونیم استفاده از این دستور کل برنچ مد نظر به همراه کامیت‌های اون رو با برنچ ما ادغام می‌کنه. گاهی اوقات ما فقط به یک کامیت

خاص احتیاج داریم که به برنچ خودمون اضافه کنیم. برای این‌کار دستور `git cherry-pick` به کارمون میاد.

نحوهٔ استفاده از دستور `git cherry-pick` به این صورت هست:

```
git cherry-pick <commit hash>
```

برای استفاده از این دستور، باید Hash کامیت مد نظرمون رو بدونیم. با اجرای دستور بالا تغییراتی که توی کامیت مد نظر وجود داره به برنچ ما اضافه میشه. دقت کنیم که این دستور یک کامیت جدید با یک Hash جدید برامون می‌سازه.

۱۶. هدف از User Agent توی مرورگرها چیه؟

User Agent یک رشته هست که مرورگر اون رو توی هدر درخواست‌های HTTP به سرور ارسال می‌کنه و حاوی اطلاعاتی مثل اسم و ورژن مرورگر، سیستم عامل، نوع دستگاه و ویژگی‌هایی مثل رزولوشن صفحه نمایش هست. با این کار، سرور می‌تونه پاسخ مناسب‌تری به این درخواست بده. مثلاً اگه کاربری با استفاده از موبایل می‌خواد از وبسایت ما دیدن کنه، توی سرور با بررسی کردن User Agent می‌تونیم ورژن موبایلی برنامه‌مون رو به اون کاربر ارائه بدیم.

User Agent به این صورت توی درخواست دیده میشه:

```
GET /example.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.1
Safari/537.36
```

نکته‌ای که باید در نظر داشته باشیم اینه که این رشته می‌تونه توسط ابزارهایی مثل اکستنشن‌های مرورگر دستکاری بشه. بنابراین محل مناسبی برای پاس دادن اطلاعات امنیتی و حساس نیست.

۱۷. Node و Element توی DOM چه تفاوت‌هایی با هم دارن؟

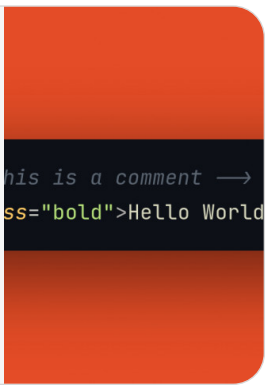
همونطور که می‌دونیم یک صفحهٔ HTML از اجزای مختلفی مثل تگ‌ها، متن‌ها، کامنت‌ها و ... تشکیل میشه. برای دسترسی به این اجزا و کار کردن با اونها، مرورگر برای ما آبجکت DOM که یک رابط بین HTML و جاوااسکریپت هست رو می‌سازه. با DOM می‌تونیم به همهٔ این اجزا دسترسی داشته باشیم و تغییراتی توی صفحه بدیم. به این اجزایی که توی DOM وجود دارن به طور کلی گفته میشه **Node**. مثلاً

متن توی یک تگ `p` ، کامنت‌ها و خودِ المنت‌ها همگی نوعی Node هستن. و همچنین اعضایی مثل `div` و `img` یک نوع خاص از Node هستن که بهشون گفته میشه **Element**.

بیشتر بدانید

Node و Element توی دام (DOM) چه تفاوتی دارن؟

هنگام توسعهٔ برنامه‌های فرانت‌اند حتماً واژه‌های Node و Element رو دیدیم که گاهی اوقات فکر می‌کنیم یکی هستن و بجای همدیگه استفاده میشن. اما باید بدونیم که این دو تفاوت‌هایی دارن که توی این قسمت اونها بررسی می‌کنیم.



۱۸. توی جاوااسکریپت Syntax Error و Type Error چه تفاوت‌هایی با هم دارن؟

Syntax Error: اگه ساختار ظاهری و قواعد نوشتاری جاوااسکریپت رو رعایت نکنیم، مفسر جاوااسکریپت بهمون Syntax Error برمی‌گردونه. کد زیر شامل یک Syntax Error هست:

```
function add(x, y) {
  return x + y;

add(2, 5);
```

اینجا توی خط ۳ براکت پایانی تابع `add` حذف شده. بنابراین مفسر قبل از اینکه کدهای ما رو به طور واقعی اجرا کنه بهمون چنین خطایی رو برمی‌گردونه:

```
SyntaxError: missing } after function body
```

Type Error: این خطا رو زمانی می‌گیریم که قصد اجرای یک عملیات غیر منتظره روی یک مقدار رو داریم. برای مثال فراخونی یک متد از `null` یا مقدار مجدد نسبت دادن به یک متغیر `const` . موارد زیر همگی نمونهٔ Type Error توی جاوااسکریپت هستن:

```
// 1. TypeError: a is not a function
const a = 1;
a();


// 2. TypeError: null has no properties
null.toUpperCase();


// 3. TypeError: invalid assignment to const 'x'
const x = 1;
x = 2;
```

بیشتر بدانیم

نوع دیگه‌ای از خطا توی جاوااسکریپت وجود داره به اسم Reference Error که اون رو زمانی می‌گیریم که می‌خوایم به یک مقدار غیر قابل دسترسی یا تعریف‌نشده دسترسی پیدا کنیم. مثلاً:

```
// ReferenceError: y is not defined
console.log(y);

{
  const x = 10;
}
// ReferenceError: x is not defined
console.log(x);
```

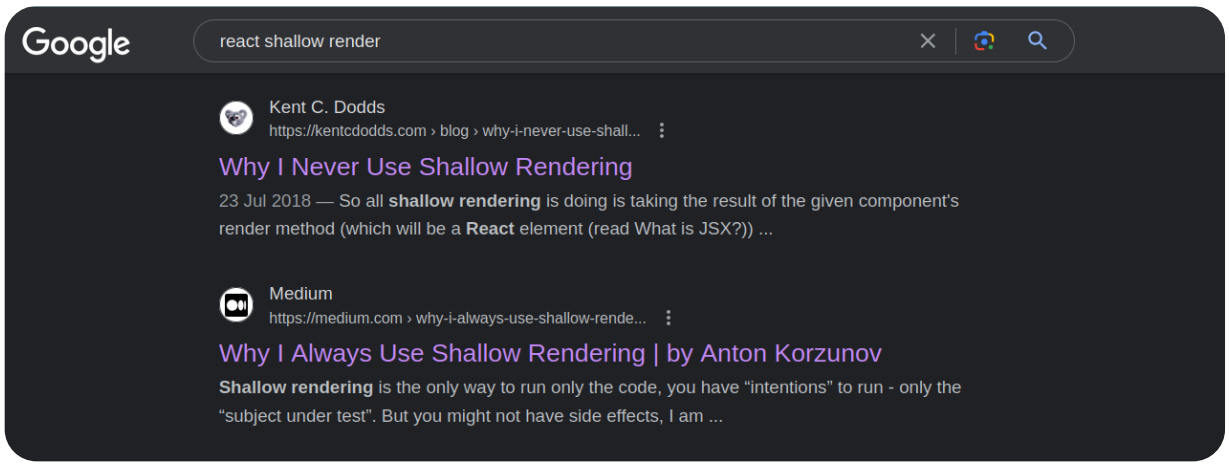
۱۹. DNS چیه؟

DNS مخفف Domain Name System هست و وظیفهٔ اون ترجمه و تبدیل کردن اسم دامنه‌ها (مثلاً google.com) به آدرس IP قابل خوندن برای کامپیوترهاست (مثلاً 4.2.2.4). وقتی برای مثال توی مرورگر آدرس google.com رو باز می‌کنیم، ابتدا این آدرس باید ترجمه و تبدیل بشه به IP آدرس مربوطه تا بتونیم اون آدرس رو لود کنیم و ببینیم. این تبدیل، وظیفه DNS هست. در واقع DNS مثل اپ Contact توی گوشی‌ها عمل می‌کنه که توی اون هر اسمی به یک شماره تلفن نسبت داده شده.

۲۰. چرا از Shallow Rendering توی تست‌نویسی استفاده کنیم؟

Shallow Rendering یک تکنیک تست‌نویسی هست که بیشتر توی فریم‌ورک‌های

Component-Based مثل ویو و ری‌اکت دیده میشه. وقتی می‌خوایم یک کامپوننت رو با تکنیک Shallow Rendering تست کنیم، هنگام تست فقط خود کامپوننت رندر میشه و کامپوننت‌های داخلی اون کامپوننت رندر نمیشن تا توجه و تمرکز روی خود کامپوننت والد باشه. با این کار، تست کردن راحت‌تر، سریع‌تر و دقیق‌تر خواهد بود. اما باید بدونیم که استفاده از Shallow Rendering همیشه خوب نیست.



از معایب Shallow Rendering اینه که:

- ممکنه رفتارهای اون کامپوننت رو توی محیط‌های واقعی نادیده بگیریم
- اگه کامپوننت به کامپوننت‌های داخلی وابستگی داشته باشه ممکنه حتی باعث سخت‌تر شدن تست بشه
- باعث نادیده گرفته‌شدن خیلی از تعامل‌های بین کامپوننت والد و فرزندها میشه که باعث میشه تست کمتر قابل اعتماد باشه

پس بهتره از این تکنیک هم مثل هر چیز دیگه‌ای با آگاهی استفاده کنیم.

. . .

Resources:

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS
- https://stackoverflow.com/questions/9339429/what-does-cherry-picking-a-commit-with-git-...
- https://www.freecodecamp.org/news/type-error-vs-reference-error-javascript/
- https://stackoverflow.com/questions/46881630/what-is-shallow-rendering-in-jest-unit-tests-i...

۲۱. هنگام باندل کردن برنامه چه چیزهایی مانع Tree-Shaking میشه؟

ابتدا یک تعریف از Tree-shaking داشته باشیم. Tree-shaking به عملیات حذف کردن کدهای اضافی و بلااستفاده توسط باندلر هنگام باندل کردن گفته میشه.

توی هر برنامه‌ای همیشه یک سری کدها (مثل توابع) وجود داره که هیچوقت استفاده نمیشن و بنابراین حضور اونها توی باندل نهایی نتیجه‌ای جز افزایش سایز باندل نداره. بنابراین با حذف کردن اونها می‌تونیم سایز باندل رو کاهش بدیم تا سرعت و عملکرد برنامه بهتره بشه. اما چه کدهایی کاندیدای حذف شدن هستن؟ باندلر چه‌جوری تشخیص میده که چه کدی رو باید حذف کنه و چه کدی رو نه؟

قوانینی وجود داره که اگه اونها رو رعایت نکنیم، باعث میشه باندلر نتونه Tree-shaking انجام بده:

۱. Import کردن همهٔ ماژول

وقتی برای استفاده از قسمتی از یک ماژول، همهٔ اون رو Import می‌کنیم، باندلر نمی‌تونه قسمت‌های بلااستفادهٔ اون رو حذف کنه:

```
1  import * as utils form 'module';
2
3  utils.add(2, 5);
```

توی کد خط ۱ ما داریم همهٔ ماژول رو Import می‌کنیم. اینجا باندلر نمی‌تونه تشخیص بده که ما به کدوم عضو احتیاج داریم. بنابراین همهٔ اعضای اون ماژول توی باندل نهایی حضور دارن؛ حتی اگه فقط یک عضو از اون آبجکت رو فراخونی کنیم. پس بهتره فقط اعضایی رو Import کنیم که بهشون احتیاج داریم:

```
1  import { add } form 'module';
2
3  add(2, 5);
```

۲. Import کردن ماژول به صورت Dynamic

وقتی از Dynamic Import برای Import کردن ماژول‌ها استفاده می‌کنیم، باندلر نمی‌تونه تشخیص بده که کدوم عضو ماژول قراره استفاده بشه. بنابراین همهٔ اعضای این ماژول توی باندل نهایی حضور دارن. پس اگه Tree-shaking برامون

مهمتر هست، باید از روش Static Import (روش بالا) استفاده کنیم.

۳. استفاده از ماژول-سیستم‌هایی غیر از ES Modules

وقتی باندلر با ماژول‌هایی مواجه میشه که به روشی غیر از ES Modules مثل Common JS یا AMD استفاده شدن، این امکان وجود داره که نتونه بدرستی Tree shaking انجام بده. چنین ماژول‌هایی برخلاف ماژول‌های ES، ساختار و رفتار Dynamic (مثل مورد بالا) دارن و بنابراین کار رو برای باندلر برای پیدا کردن کد بلااستفاده سخت می‌کنن.

۴. ماژولی که Side Effect داره

اگه توی یک ماژول اعضایی داریم که Side Effect دارن، باندلر نمی‌تونه تصمیم بگیره کدوم عضو کاندیدای حذف از باندل هست. برای مثال توی این ماژول یک Side Effect داریم:

```
export const greet = (name) => {
  return "Hello";
};

console.log("This is a side effect");
```

هر چند بعضی از باندلرها با تکنیک‌هایی می‌تونن سایدافکت رو تشخیص بدن و Tree Shaking انجام بدن، داشتن ماژولی که سایدافکت داره نقض‌کنندهٔ برنامه‌نویسی ماژولار هست.

۲۲. برنامه‌نویسی Functional چه مزایایی داره؟

برنامه‌نویسی Functional با داشتن قواعد و اصولی مثل Immutability و Pure functions مزایای زیر رو به همراه داره:

- تست کردن راحت‌تر برنامه به دلیل عدم وجود سایدافکت
- توسعهٔ ماژولار و قابلیت استفاده مجدد از ماژول‌ها (Reusability) به دلیل استفاده از Pure functions
- سادگی، خوانایی بالاتر و پیش‌بینی پذیری کدها
- کمک به داشتن یک برنامهٔ Maintainable

۲۳. چه زمانی اتریبیوت aria-label توی HTML استفاده کنیم؟

کاربرد این اتریبیوت بیشتر برای مبحث Accessibility و افرادی که از تکنولوژی‌های کمکی استفاده می‌کنن هست. توی هر صفحه‌ای ممکنه المنت‌هایی داشته باشیم که وقتی توسط یک ابزار کمکی مثل Screen Reader پردازش میشن، هیچ اطلاعاتی رو به کاربر منتقل نمی‌کنن. بنابراین کاربر ممکنه نتونه به درستی هدف اون المنت رو متوجه بشه. مثلاً آیکنی که به این صورت لینک شده:

```
<a href="/next">
  <svg class="icon"> ... </svg>
</a>
```

Screen Reader وقتی با این المنت مواجه میشه، نمی‌تونه اطلاعات مناسبی دربارهٔ وظیفهٔ این المنت به کاربر منتقل کنه که در نتیجه Accessibility و کاربردپذیری اون صفحه پایین میاد. کاربرد `aria-label` اینجا مشخص میشه. با استفاده از این اتریبیوت می‌تونیم کاربرد چنین المنت‌هایی رو برای ابزارهای کمکی مشخص کنیم:

```
<a href="/next" aria-label="Go to the next page">
  <svg class="icon"> ... </svg>
</a>
```

توی کد بالا با توضیحاتی که برای مقدار `aria-label` نوشتیم، ابزارهای کمکی می‌تونن منظور و هدف این المنت رو به کاربرها منتقل کنن. این اتریبیوت کاربرد مشابهی با تگ `label` که فقط برای Input های فرم استفاده میشه داره.

۲۴. چه زمانی از تگ توی HTML استفاده کنیم؟

معمولاً توی صفحه‌های وب برای هر عکس ورژن و سایزهای متفاوتی وجود داره و ما گاهی اوقات می‌خوایم هنگام طراحی صفحه‌های Responsive بر اساس سایزهای مختلف صفحه سایز متفاوتی از عکس رو نمایش بدیم. تگ `<picture>` این امکان رو میده که بر اساس شرایط مختلف، عکس متفاوتی نمایش بدیم تا از هدر رفت منابع و پهنای‌باند جلوگیری بشه:

```
<picture>
  <!-- if window width >= 900px -->
  <source media="(min-width: 900px)" srcset="img-
large.jpg">

  <!-- if window width >= 600px -->
  <source media="(min-width: 600px)" srcset="img-
medium.jpg">

  <!-- default -->
  <img src="img-small.jpg">
</picture>
```

توی این تگ برای هر ورژن از تصویر یک تگ `source` قرار می‌دیم و مشخص می‌کنیم که این تصویر توی چه شرایطی باید نمایش داده بشه. انتهای اون هم یک تگ `img` هم می‌ذاریم که زمانی نمایش داده میشه که هیچکدوم از شرایط برقرار نباشه. مرورگر به صورت خودکار تصمیم می‌گیره که کدوم از تصاویر باید دانلود و نمایش داده بشه.

۲۵. دستور git tag چکار می‌کنه؟

اگه مشغول توسعهٔ نرم‌افزاری هستیم و این توسعه به یک نقطهٔ خاص و قابل توجهی رسیده، با استفاده از دستور `git tag` می‌تونیم برای آخرین کامیت یک اسم و برچسب بامعنا در نظر بگیریم؛ با این هدف که این کامیت نقطهٔ مهمی برای ما هست. با این کار می‌تونیم به شکل راحت‌تری مراحل مهم برنامه‌مون رو شناسایی و پیدا کنیم. بدون این ویژگی باید توی انبوهی از Hash ها دنبال کامیت مد نظر باشیم. بنابراین، `git tag` یک اسم بامعنا برای یک Hash کامیت بی‌معنا مثل `abcdef12` هست.

نحوهٔ استفاده از این دستور به این صورت هست:

```
git tag [TAG NAME]
```

۲۶. دستور git pull و git fetch چه فرقی با هم دارن؟

دستور `git pull` در واقع ترکیبی از دو دستور `git fetch` و `git merge`

هست. یعنی وقتی `git pull` رو اجرا می‌کنیم، در واقع داریم اون دو دستور رو اجرا می‌کنیم. دستور `git fetch` برای گرفتن آخرین تغییرات (کامیت‌ها، برنچ‌ها و تگ‌ها) از سرور استفاده میشه. وقتی این دستور رو می‌زنیم، اطلاعات دریافت میشن اما Merge صورت نمی‌گیره و بنابراین کدهایی که داریم روی اونها کار می‌کنیم تحت تاثیر قرار نمی‌گیرن. پس برای مرج شدن باید بعد از Fetch به صورت دستی از `git merge` استفاده کنیم. استفاده از دستور `git fetch` برای زمانی مناسب هست که می‌خوایم قبل از مرج شدن تغییرات رو بررسی کنیم.

۲۷. توی مرورگر رویدادهای load و DOMContentLoaded چه فرقی با هم دارن؟

توی وب از زمانی که مرورگر شروع می‌کنه به دریافت اطلاعات وبسایت و تا زمانی که صفحه به طور کامل آمادهٔ تعامل بشه، رویدادهای مختلفی رخ میده. این دو رویداد با اینکه شبیه به هم هستن، کاربردهای منحصر به فرد خودشون رو دارن:

رویداد `DOMContentLoaded`

این رویداد زمانی رخ میده که مرورگر به طور کامل HTML رو دریافت کرده و ساختار اون و آبجکت DOM رو به طور ساخته باشه. توی این مرحله می‌تونیم با DOM تعامل داشته باشیم و اون رو دستکاری کنیم. باید دقت کنیم که رخ دادن این رویداد وابسته به این نیست که ریسورس‌هایی مثل تصاویر، اسکریپت‌ها و استایل‌های خارجی حتماً لود شده باشن. بنابراین این رویداد تضمین نمی‌کنه که صفحه به طور کامل آمادهٔ تعامل باشه. نحوهٔ استفاده از این رویداد به این صورت هست:

```
1  document.addEventListener('DOMContentLoaded', (event) =>
2  {
3    // DOM is ready and accessible
4    // Manipulate DOM elements
   });
```

رویداد `load`

این رویداد زمانی رخ میده که DOM به‌طور کامل ساخته شده باشه و اطلاعات صفحه از جمله ریسورس‌های خارجی همگی لود و پردازش شده باشن و کاربر آمادهٔ تعامل با صفحه باشه. نحوهٔ استفاده از رویداد `load` به این صورت هست:

```
window.addEventListener('load', (event) => {
  // Page and its resources are ready
});
```

۲۸. چند تا از Best Practice های Accessibility رو می‌شناسین؟

- مطمئن بشیم که محتوای دکمه‌ها و لینک‌ها یکتا و با معنا هستن. برای مثال لینکی که متن اون «جزییات بیشتر درباره محصول XYZ» هست، مفهوم و معنای بهتری در مقایسه با متن «جزییات بیشتر» داره و کمتر باعث سردرگمی میشه
- از اتریبیوت `lang` روی تگ `<html>` صفحه استفاده کنیم تا برای مرورگر و ابزارهای کمکی زبان مد نظر رو مشخص کنیم
- از اتریبیوت `title` روی المنت‌ها به منظور پیاده‌سازی Tooltip استفاده نکنیم. به دلیل اینکه این اتریبیوت برای کاربران موبایلی، کاربران وابسته به Screen reader و یا هنگام استفاده از کیبورد کارایی خاصی ندارن
- از المنت‌های مخفی ولی Focusable استفاده نکنیم (اطلاعات بیشتر درباره Tabindex)
- برای همهٔ تصاویر از اتریبیوت `alt` استفاده کنیم. برای بعضی از تصاویر که معنای خاصی ندارن (مثل آیکن‌ها) از `alt` خالی استفاده کنیم
- توی یک صفحه HTML بیشتر از یک تگ `h1` استفاده نکنیم
- برای ساختن لینک (مخصوصاً توی SPA ها) از تگ `<a>` استفاده کنیم؛ نه از `button` و یا المنت‌های دیگه
- از Autoplay برای ویدئوها و آهنگ‌ها استفاده نکنیم

۲۹. فایل d.ts تو پروژه‌های تایپ‌اسکریپتی چیه؟

این فایل یک جای اختصاصی برای نوشتن Type Declaration هست. همونطور که می‌دونیم، میشه Type Declaration ها رو به‌صورت Inline توی فایل‌های تایپ‌اسکریپت (`.ts`) هم نوشت. اما اگه تعداد Type Declaration ها زیاد بشه بهتره اونها رو منتقل کنیم به یک فایل مجزا. چنین کاری توی تایپ‌اسکریپت با ساختن فایل `.d.ts` انجام می‌گیره. توی این فایل فقط می‌تونیم Type Declaration بنویسیم و کدهای دیگه‌ای از تایپ‌اسکریپت معتبر نیستن.

۳۰. منظور از Stateless بودن REST چیه؟

همونطور که می‌دونیم REST یک معماری مبتنی بر HTTP و شامل قوانینی هست برای ساختن وب‌سرویس‌ها. و یک برنامهٔ RESTful به برنامه‌ای گفته میشه که از این قوانین تبعیت می‌کنه. یکی از قوانین برنامهٔ رِست‌فول اینه که هر درخواستی که از کلاینت به سمت سرور زده میشه، باید توی خودش همهٔ اطلاعاتی که سرور برای دادنِ پاسخ مناسب لازم داره رو داشته باشه. به عبارت دیگه، هر درخواست باید کاملاً مستقل از درخواست‌های دیگه عمل کنه و سرور برای فهمیدن یک درخواست نباید وابسته به درخواست‌های دیگه باشه.

وابستگی درخواست‌ها زمانی به وجود میاد که سرور اطلاعاتی از یک درخواست رو ذخیره کرده تا برای درخواست‌های بعدی از اون استفاده کنه و اینجاست که قانون Stateless بودن نقض میشه. برای مثال توی یک برنامه‌ای که می‌خواد از قوانین رست تبعیت کنه می‌خوایم نظر ارسال کنیم. اینجا هر درخواست توی خودش علاوه بر متن نظر، باید اطلاعات مربوط به شناسایی و احراز هویت کاربر (مثلاً توکن JWT) رو همیشه به همراه داشته باشه. توی این برنامه قانون رست زمانی نقض میشه که سرور اطلاعات مربوط به احراز هویت رو (مثلاً به وسیلهٔ قابلیت Session ها) توی خودش ذخیره کنه تا برای درخواست‌های بعدی دیگه نیازی به ارسال اطلاعات احراز هویت نباشه.

بیشتر بدانیم

State چیه؟🤔

منظور از State اطلاعاتی هست که توی برنامه در جریانه. مثل اطلاعات کاربری، تم، زبان و ... .

. . .

Resources:

- https://www.smashingmagazine.com/2021/05/tree-shaking-reference-guide/
- https://betterprogramming.pub/dynamic-import-and-tree-shaking-in-javascript-ddc2f3cd69f
- https://stackoverflow.com/questions/22039910/what-is-aria-label-and-how-should-i-use-it

۳۱. Snapshot Test چیه و چه مزایا و معایبی داره؟

Snapshot Test یک تکنیک تست‌کردن برنامه‌های فرانت‌اند هست که توی اون بررسی میشه که آیا یک کامپوننت یا قسمتی از ظاهر برنامه در طول توسعهٔ اون ثابت و بدون تغییر باقی می‌مونه یا نه. معنی تحت‌اللفظی کلمهٔ Snapshot به معنی یک عکس لحظه‌ای هست. توی این تکنیک:

۱. ابتدا کامپوننت رندر میشه
۲. یک Snapshot در صورت عدم وجود اون، تهیه و ذخیره میشه
۳. توی تست‌های بعدی، از UI مد نظر یک Snapshot تهیه میشه و با Snapshot ابتدایی مقایسه و در صورت عدم تطابق، تست Fail میشه
۴. در صورت نیاز می‌تونیم Snapshot ابتدایی رو آپدیت کنیم

از مزایای Snapshot Test اینه که می‌تونیم یک کامپوننت بزرگ و پیچیده رو بدون نوشتن کدها و تست‌های زیاد تست کنیم. زمانی این تست Fail میشه که تغییری توی ظاهر اون کامپوننت صورت گرفته باشه.
از معایب این روش تست اینه که دقیقاً مشخص نیست چه ویژگی‌ای از کامپوننت داره تست میشه. بنابراین اگه قصد داریم نحوهٔ کارایی و منطق این کامپوننت رو تست کنیم، این روش کارایی چندانی نداره. پس بهتره فقط زمانی از این تکنیک استفاده کنیم که پایدار بودن ظاهر یک کامپوننت (مثلاً دکمه یا ظاهر یک فرم) برامون اهمیت داره.

۳۲. توی جاوااسکریپت Same-origin policy چیه؟

Same-origin policy یک قابلیت امنیتی توی مرورگرها هست که مانع دستکاری و دسترسی بدون مجوز جاوااسکریپت به ریسورس‌هایی میشه که خارج از دامنه (Origin) فعلی توی صفحه حضور دارن. برای مثال اگه دامنهٔ فعلی برنامهٔ ما a.com باشه و فرض کنیم می‌خوایم یک آی‌فریم از b.com رو لود کنیم، مرورگر با استفاده از قابلیت Same-origin policy مانع دستکاری و دسترسی بدون مجوز جاوااسکریپت به محتوای لود شده توی آی‌فریم میشه.

منظور از Origin همون آدرسی هست که ریسورس لود میشه و Same-origin یعنی دو آدرسی که با هم برابر هستن. مرورگر با مقایسه کردن معیارهایی مثل Domain و Scheme و Port از URL تصمیم می‌گیره که آیا دو آدرس با هم برابر هستن یا نه.

برای مثال هیچ‌کدوم از آدرس‌های زیر با هم برابر نیستن و بنابراین **Cross-origin** در نظر گرفته میشن:

```
https://anotherdomain.com

http://example.com
https://example.com
https://subdomain.example.com

https://example.com:8080
https://example.com:1234
```

اما آدرس‌های زیر Same-origin در نظر گرفته میشن:

```
https://example.com
https://example.com/foo
https://example.com/foo/bar
```

اگه قصد داریم از یک ریسورس Cross-origin رو لود کنیم، سرور باید حتماً هدر CORS رو ست کرده باشه.

۳۳. توی جاوااسکریپت Map و WeakMap چه فرقی با هم دارن؟

Map و WeakMap توی جاوااسکریپت Data Structure اختصاصی هستن که برای نگهداری مجموعه‌ای از اطلاعات به صورت Key/value استفاده میشن. مجموعهٔ WeakMap برای کاربردهای خاص و بیشتر به منظور مصرف بهینهٔ حافظه معرفی شده و شباهت زیادی به مجموعهٔ Map داره ولی با این تفاوت‌ها:

- کلیدهای اعضای مجموعه WeakMap باید آبجکت یا `Symbol` باشن (در مقایسه با Map که کلید از هر نوعی می‌تونست باشه)
- اعضای مجموعه WeakMap قابل پیمایش (Iterable) نیستن و نمی‌تونن مثل Map توی حلقهٔ `for...of` یا `forEach` قرار بگیرن
- مجموعهٔ WeakMap بر خلاف Map پراپرتی‌ای به اسم `size` و یا متدهایی مثل `clear` و `forEach` و `entries` رو نداره

برای آشنایی کامل با این دو مجموعه می‌تونین این پست رو ببینین:

بیشتر بدانید

همه چیز از آبجکت‌های Map و WeakMap توی جاوااسکریپت

جاوااسکریپت برای مدیریت کردن اطلاعات برنامه علاوه‌بر آبجکت‌های معمولی، Data Structure های کاربردی دیگه‌ای هم داره که توی این پست می‌خوایم با ۲ از اونها آشنا بشیم

۳۴. تکنیک Currying رو توی جاوااسکریپت پیاده‌سازی کنین

Currying یک تکنیک نوشتن توابع هست که بیشتر توی برنامه‌نویسی Functional استفاده میشه. با این تکنیک می‌تونیم بجای داشتن یک تابع با چندین پارامتر، چند تابع (تو در تو) با یک پارامتر داشته باشیم. چنین تابعی رو در نظر بگیرین:

```
function logger(level, message) {
  console.log(`${level}: ${message}`);
}
```

برای استفاده از اون باید چنین کدی بنویسیم:

```
logger('warning', 'This is a warning message');
logger('warning', 'This is another warning message');

logger('info', 'This is an info message');
logger('info', 'This is another info message');
```

همونطور که می‌بینیم کدهای تکراری (Code duplication) داریم. برای داشتن هر Log همیشه باید همهٔ پارامترهای اون تابع رو مشخص کنیم. اما با استفاده از تکنیک Currying می‌تونیم این کدهای تکراری رو حذف کنیم و البته کدهایی خواناتر با قابلیت استفاده مجدد داشته باشیم.

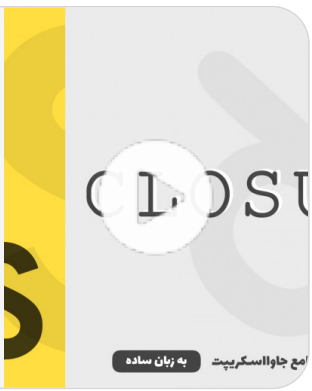
برای اعمال تکنیک Currying می‌تونیم تابع `logger` رو به این صورت بازنویسی کنیم:

```
function logger(level) {
  return function (message) {
    console.log(`${level}: ${message}`);
  }
}
```

همونطور که می‌بینیم به‌سادگی با استفاده از کلوژرها تونستیم یک تابع با چند پارامتر رو تبدیل کنیم به چند تابع با یک پارامتر. مزیت این روش هنگام استفاده کردن از اون تابع به چشم میاد:

```
1  const infoLog = logger('info');
2
3  infoLog('This is an info message #1');
4  infoLog('This is an info message #2');
5  infoLog('This is an info message #3');
6  infoLog('This is an info message #4');
7  infoLog('This is an info message #5');
```

همونطور که می‌بینیم برای Log گرفتن فقط یک بار `level` رو مشخص کردیم (خط ۱). برای درک نحوهٔ کارایی این روش توی جاوااسکریپت پیشنهاد می‌کنم پست زیر رو ببینین:



بیشتر بدانید

[ویدئو] کلوژرها Closure | آموزش جاوااسکریپت به زبان ساده

کلوژرها یکی از پرکاربردترین تکنیک‌ها توی جاوااسکریپت هستن و نکته‌های ظریفی دارن که اونها رو توی این قسمت بررسی می‌کنیم

۳۵. چند اصل دنیای نرم‌افزار رو می‌شناسید؟

برای داشتن یک برنامهٔ موفق و قابل توسعه توی دنیای نرم‌افزار همیشه باید یک سری اصول رو رعایت کنیم. مثل:

Separation of Concerns

این اصل میگه که یک برنامه باید طوری به قسمت‌های مجزا تقسیم‌بندی بشه که هر قسمت یک مسئولیت مشخص و قابل فهم داره. این اصل کمک می‌کنه که برنامه‌ای خواناتر و با قابلیت توسعهٔ بیشتری داشته باشیم.

DRY (Don't Repeat Yourself)

طبق این اصل باید تا جایی که میشه از نوشتن کدهای تکراری خودداری کنیم و اعضای برنامه مثل توابع و ماژول‌ها رو طوری بنویسیم که بیشترین Reusability (قابلیت استفاده مجدد) رو داشته باشن. مثل استفاده از تکنیک Currying (سوال قبل). نکته‌ای که باید در نظر داشته باشیم اینه که Reusability بیش از حد هم ممکنه باعث گنگ شدن و عدم شفافیتِ کارایی اعضای برنامه بشه. مثل نوشتن تابع

و یا ماژول همه کاره. پس بهتره این اصل رو تا جایی باید رعایت کنیم که کارایی یک عضو (مثل تابع، ماژول) مشخص و قابل فهم باشه.

SOLID

SOLID یک کلمهٔ مخفف برای ۵ اصل مهم توی دنیای نرم‌افزار هست. این اصول بهمون کمک می‌کنن یک برنامهٔ خوانا و قابل توسعه داشته باشیم که توی اون هر عضو، رفتار و خروجی قابل پیش‌بینی داره و که با حداقل وابستگی در کنار هم فعالیت می‌کنن. برای آشنایی با این اصول می‌تونین پست زیر رو ببینین:

بیشتر بدانید

اصول SOLID به زبان ساده

اصول SOLID به پنج قسمت قسمت تقسیم میشه که از مهمترین اصول توی برنامه‌نویسی شی‌گرا به حساب میان

YAGNI (You Ain't Gonna Need It):

این اصل بهمون میگه که تا جایی که می‌تونیم از نوشتن ویژگی‌هایی که در لحظهٔ حال بدردمون نمی‌خوره خودداری کنیم. به بیان دیگه، کدهامون رو با ذهنیت «شاید بعداً بدرد بخوره» ننویسیم و زمانی اونها رو پیاده‌سازی کنیم که واقعاً نیاز هست. این کار کمک می‌کنه برنامهٔ ساده‌تر و قابل پیش‌بینی داشته باشیم.

KISS (Keep It Simple, Stupid):

طبق این اصل، بهتره همیشه سادگی رو به پیچیدگی ترجیح بدیم. چون سادگی همیشه تاثیرگذارتر هست. طبق این اصل، باید کدهامون رو با نهایت سادگی و خوانایی بنویسیم تا برای همه قابل فهم و توسعه باشه.

۳۶. منظور از درخواست‌های Idempotent توی درخواست‌های HTTP چیه؟

یک درخواست Idempotent به درخواستی گفته میشه که وقتی یک یا چند بار ارسال میشه، تغییراتی که توی سرور به‌وجود میاره و همچنین پاسخ اون کاملاً قابل پیش‌بینی باشه. به عبارت دیگه، یک درخواست Idempotent درخواستی هست که Side effect نداره.

برای مثال اگه ۱۰ درخواست با متد GET به آدرس example.com/posts زده بشه، باید همه پاسخ‌ها یکسان و قابل پیش‌بینی باشه. و یا برای مثال وقتی قصد داریم با

متد POST یا DELETE یک ریسورس رو آپدیت یا حذف کنیم، رفتاری که توی درخواست دهم انتظار داریم، باید مشابه رفتار اولین درخواست باشه.

۳۷. توی یک صفحه وب برای دانلود ریسورس‌ها چه زمانی از preload و چه زمانی از prefetch استفاده کنیم؟

وقتی توی یک صفحه برای لود کردن یک ریسورس از `<link rel="preload">` استفاده می‌کنیم، یعنی به مرورگر می‌گیم که قصد داریم از این ریسورس خیلی زود توی ادامهٔ برنامه استفاده کنیم. پس لطفاً اون رو برای من با **بالاترین اولویت** دانلود کن

```
<link rel="preload" href="/font.woff" as="font" />
```

با توجه به کد بالا، ریسورس مد نظر حتی شاید زودتر از حالت عادی دانلود بشه. پس بهتره از `preload` زمانی استفاده کنیم که واقعاً نیاز داریم یه ریسورس زودتر دانلود بشه و مطمئن هستیم که حتماً توی ادامهٔ برنامه مورد استفاده قرار می‌گیره.

با `prefetch` می‌تونیم یک ریسورس که احتمال می‌دیم توی ادامهٔ فعالیت‌های برنامه مورد استفاده قرار بگیره رو با اولویت **پایین** دانلود و کش کنیم:

```
<link rel="prefetch" href="/prism.js" as="script" />
```

با این کار به مرورگر می‌گیم که لطفاً ابتدا بقیه ریسورس‌های با اولویت بالاتر رو دانلود کن و بعد اگه صلاح دیدی این ریسورس رو دانلود کن، چون بعداً ممکنه بهش احتیاج داشته باشم و می‌خوام زودتر قابل دسترس باشه.

برای آشنایی بیشتر با این تکنیک‌ها این پست رو ببینید:

بیشتر بدانید

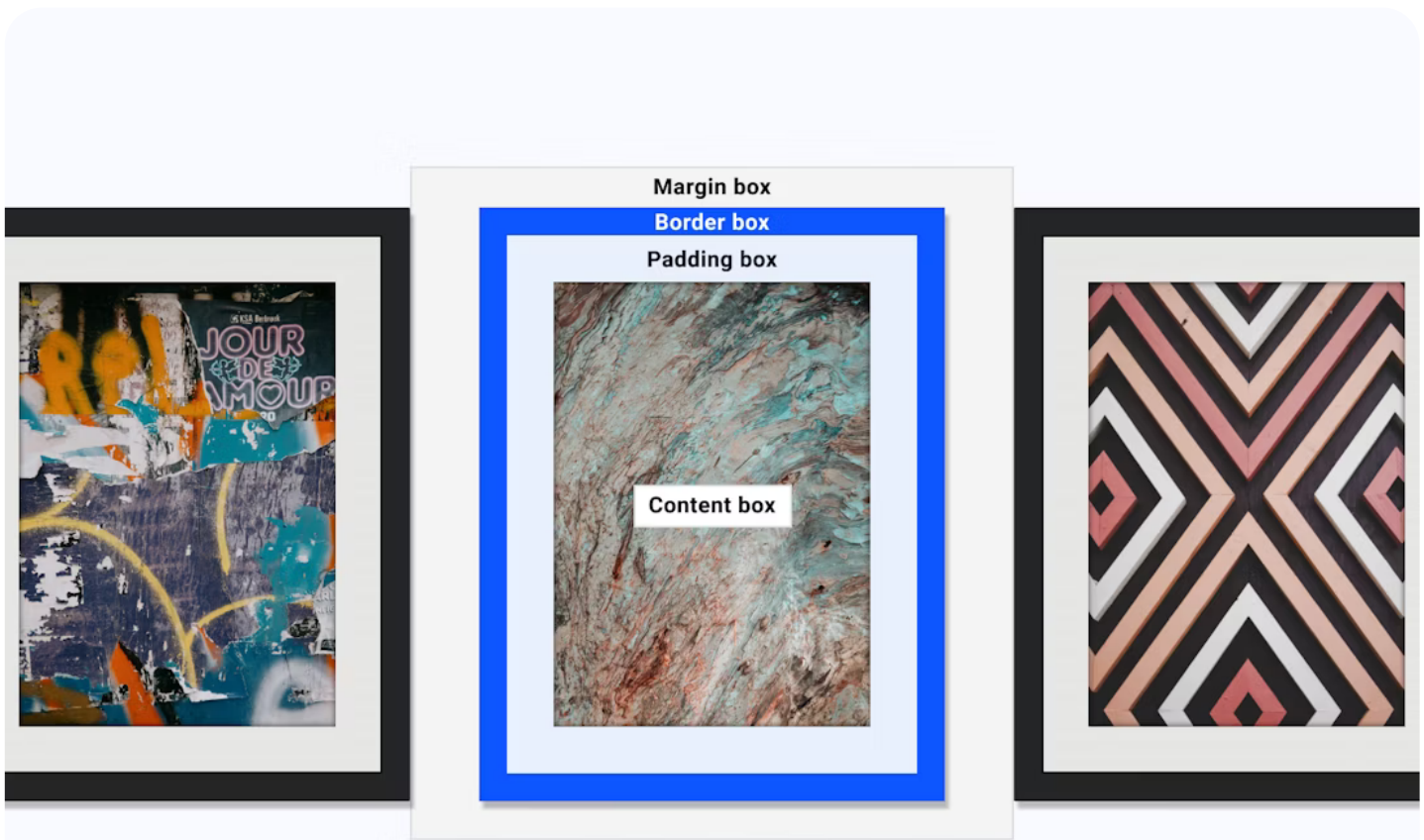
افزایش سرعت برنامه با Preload و Preconnect و چند تکنیک دیگه

با تکنیک‌ها و قابلیت‌هایی آشنا می‌شیم که کمک می‌کنن یک برنامهٔ سریع‌تر و کاربر پسندتر داشته باشیم

۳۸. منظور از Box Model توی CSS چیه؟

Box Model یک مفهوم و طرح فرضی برای تشخیص نحوهٔ قرارگیری یک المنت داخل صفحه هست که توی اون هر المنت توسط یک محفظه (Box) فرضی محصور میشه و معیارهایی مثل Margin و Padding و Border و محتوای المنت روی اندازهٔ این محفظه تأثیرگذار هست.

یک مثال دنیای واقعی از این مفهوم، عکس زیر هست:



https://web.dev/learn/css/box-model

همونطور که می‌بینیم معیارهایی مثل Margin و Padding و Border روی اندازهٔ نهایی هر عکس و همچنین نحوهٔ قرارگیری اون در کنار بقیه عکس‌ها روی دیوار تأثیرگذار هست. توی عکس بالا، به اون طرح فرضی اطراف عکس وسط گفته میشه Box model.

۳۹. منظور از توی صفحه‌های HTML چیه؟

معمولاً ابتدای سورس صفحه‌های HTML چنین کدی رو می‌بینیم:

```
<!DOCTYPE html>
<html>
<head>
  ...
```

`<!DOCTYPE html>` در واقع یک تگ HTML نیست. بلکه یک دستور هست و به مرورگر میگه نوع این سند HTML ورژن 5 هست. بنابراین مرورگر با این صفحه مثل

یک صفحه وب با استانداردهای HTML5 رفتار می‌کنه. بدون حضور این دستور، مرورگر ممکنه چنین استانداردهایی رو رعایت نکنه و به قول معروف وارد حالت Quirks Mode میشه که در نتیجه خروجی غیر منتظره‌ای نمایش داده میشه.

۴۰. هدف از اتریبیوت‌های data-* توی HTML چی هست؟

`data-*` اتریبیوت‌های دلخواهی هستن که روی تگ‌های HTML قرار می‌گیرن تا اطلاعات اضافه و بیشتری رو درباره اون تگ منتقل کنن. برای مثال اگه قصد داریم دو اتریبیوت `userid` و `price` رو به یک المنت اضافه کنیم از روش زیر استفاده می‌کنیم:

```
<div data-userid="123" data-price="49.99">
  ...
</div>
```

می‌تونستیم بدون `data-` هم چنین اتریبیوت‌هایی رو به المنت اضافه کنیم. اما این روش استانداردی نیست و ممکنه باعث تداخل با بعضی از اتریبیوت‌ها و ویژگی‌های مرورگرها بشه. همچنین هر اتریبیوتی که بدون `data-` شروع میشه به این معنی هست که اون اتریبیوت جزء استانداردهای HTML هست. مثل اتریبیوت `name` روی `<input>` ها. پس وقتی برای یک اتریبیوت دلخواه از `data-` استفاده نمی‌کنیم، علاوه‌بر اینکه اعتبار صفحه پایین میاد، ممکنه باعث بروز رفتارهای غیر قابل پیش‌بینی توی مرورگر بشه.

نکته‌ای که باید در نظر داشته باشیم اینه که اطلاعاتی که توی اتریبیوت‌های `data-*` ذخیره می‌کنیم ممکنه توسط موتورهای جستجو و ابزارهای Assistive نادیده گرفته بشن. پس بهتره از این اتریبیوت‌ها زمانی استفاده کنیم که می‌خوایم از مقدار اونها توی جاوااسکریپت استفاده کنیم (که با گسترش استفاده از فریم‌ورک‌ها، چنین اتریبیوت‌هایی کمتر استفاده میشن.)

. . .

Resources:

- https://web.dev/same-origin-policy

۴۱. Type و Interface توی تایپ‌اسکریپت چه تفاوت‌هایی با هم دارن؟

این دو ویژگی شباهت‌های زیادی به هم دارن و توی بیشتر موارد می‌تونن به جای همدیگه استفاده بشن. اما تفاوت‌های زیر اونها رو از هم دیگه متمایز می‌کنه:

کار با مقادیر Primitive

برای داشتن یک تایپ برای یک مقدار Primitive (مثل رشته و عدد) فقط می‌تونیم از Type استفاده کنیم. به عبارت دیگه، از Interface فقط می‌تونیم برای مقادیر آبجکتی استفاده کنیم

نحوهٔ ادغام کردن تایپ‌ها و اینترفیس‌ها

ما توی یک برنامه می‌تونیم چند اینترفیس با اسم‌های یکسان داشته باشیم که در این صورت اینترفیس‌ها با هم ادغام میشن. اما این کار برای تایپ‌ها شدنی نیست و ما خطا می‌گیریم.

```ts
interface Animal {
  run(): any;
}

// ok
interface Animal {
  eat(): any;
}

type Person = {
  name: string;
}

// Error: Duplicate identifier 'Person'
type Person = {
  age: number;
}
```

نحوهٔ توسعه دادن

هم اینترفیس‌ها و هم تایپ‌ها می‌تونن Extend بشن. تفاوت توی نحوهٔ پیاده‌سازی هست. برای توسعه‌دادن یک اینترفیس از کلمه‌کلیدی `extends` استفاده می‌کنیم:

```
// Extending an Interface
interface Employee extends Person {
  age: number;
}

// Extending a Type
type Person = {
  name: string;
}

type Employee = Person & { age: number }
```

قابلیت Implement شدن توسط کلاس‌ها

هم تایپ‌ها و هم اینترفیس‌ها می‌تونن توسط کلاس‌ها Implement بشن. تفاوتی که اینجا وجود داره اینه که اگه یک تایپ با یک تایپ دیگه Union شده باشه (با استفاده از علامت `|`) این تایپ نمی‌تونه توسط کلاس‌ها `implement` بشه و خطا می‌گیریم.

برای آشنایی کامل با جزییات تفاوت تایپ‌ها و اینترفیس‌ها و مشاهدهٔ مثال‌ها این پست رو ببینین:

بیشتر بدانید

تفاوت Type Alias و Interface توی تایپ‌اسکریپت

تایپ Alias و اینترفیس شباهت‌های خیلی زیادی دارن و تقریباً هر جایی می‌تونن بجای همدیگه استفاده بشن. اما نحوهٔ استفاده و ساختن اونها متفاوت هست که توی این پست بررسی می‌کنیم

۴۲. درباره Core Web Vitals چی می‌دونید؟

Core Web Vitals به ۳ معیار مهم گفته میشه که توسط گوگل برای بررسی عملکرد یک وبسایت معرفی شده. این ۳ معیار شامل:

۱. Largest Contentful Paint (LCP)

گوگل توی این معیار، بزرگترین محتوای توی صفحه (مثل عکس، ویدئو، بلاک متنی) رو نگاه می‌کنه و بررسی می‌کنه که این محتوا توی چه مدت زمانی لود میشه. هر چقدر مدت زمان لود شدن این محتوا کمتر باشه، امتیاز صفحه برای LCP بهتر خواهد بود. مثلاً اگه بزرگترین المنت صفحه یک تصویر باشه و این تصویر توی مدت زمان زیادی لود بشه، این صفحه امتیاز پایینی برای LCP می‌گیره و نیاز به بهینه‌سازی داره. عدد خوب برای LCP یک صفحه از ۰ تا ۲.۵ ثانیه هست. عددی بین ۲.۵ تا ۴ ثانیه یک معیار متوسط هست و نیاز به بهینه‌سازی داره و عددی بزرگتر از ۴ ثانیه به عنوان یک صفحه ضعیف شناخته میشه.

۲. First Input Delay (FID)

این معیار برای بررسی کردن سرعت واکنش‌پذیری یک صفحه وقتی که لود میشه معرفی شده و توی اون، فاصله زمانی بین اولین تعامل کاربر با صفحه (مثل کلیک روی یک دکمه) و پاسخ مرورگر به اون تعامل اندازه‌گیری میشه. یک عدد خوب برای FID کمتر از ۱۰۰ میلی‌ثانیه هست. معمولاً صفحاتی که هنگام لود شدن عملیات سنگینی رو با جاوااسکریپت انجام میدن دارای یک FID ضعیف هستن.

۳. Cumulative Layout Shift (CLS)

این معیار برای بررسی کردن پایداری المنت‌ها توی یک صفحه هنگام لود شدن استفاده میشه. به قول معروف Visual Stability. توی یک صفحهٔ ضعیف، بخاطر لود شدن مرحله به مرحلهٔ المنت‌ها (مثل لود شدن یک عکس سنگین) شیفت‌ها و جابجایی‌های ناخواسته‌ای رخ میده که برای کاربر گیج‌کننده هست که در نتیجه UX و امتیاز اون صفحه پایین میاد.

۴۳. کاربرد Event Delegation توی جاوااسکریپت چیه؟

Event Delegation یک پترن هست و از اون هنگام کار با DOM و مدیریت کردن رویدادها برای داشتن کدهایی تمیزتر و با قابلیت توسعهٔ بیشتر استفاده می‌کنیم. برای مثال اگه چندین المنت مشابه داریم و می‌خوایم یک رویداد خاص (مثلاً `keyup`) همهٔ این المنت‌ها رو مدیریت کنیم، این الگو با استفاده از قابلیت

Event Propagation، بهمون اجازه میده تا با اضافه کردن هندلر (یا Listener) به المنت والد، بتونیم رویدادهای المنت‌های داخلی مدیریت کنیم. یعنی به جای اینکه برای تک تک این المنت‌ها، هندلر بنویسیم، هندلر رو به المنت والد اضافه می‌کنیم.

برای پیاده‌سازی این الگو می‌تونید این پست رو ببینید:

بیشتر بدانید

الگوی Event Delegation چیه؟

از این الگو هنگام کار با DOM و رویدادها استفاده می‌کنیم و با اون می‌تونیم کدهایی تمیزتر و با قابلیت توسعهٔ بیشتری داشته باشیم

۴۴. CORS چیه؟

CORS مخفف Cross-Origin Resource Sharing و به یعنی به اشتراک‌گذاری منابع برای درخواست‌هایی از منابعی غیر از سرور خودی هست. این یک قابلیت توی مرورگرها هست که به سرورها این امکان رو میده تا تعیین کنن که اطلاعات سرور (عکس، متن و ...) برای کدوم دامنه‌ها قابل دسترس باشه. وقتی درخواستی به یک سرور بزنیم و خطای CORS رو بگیریم، یعنی سرور اجازهٔ تعامل از سمت دامنه‌ای که این درخواست زدیم رو نداده.

برای اطلاعات بیشتر می‌تونین این پست رو ببینین:

بیشتر بدانید

CORS به زبان ساده

با قابلیتی از مرورگرها آشنا می‌شیم که به سرورها اجازه میده تا با اطمینان بیشتری اطلاعاتشون رو به اشتراک بذارن

۴۵. توی تایپ‌اسکریپت عبارت keyof typeof [value] رو توضیح بدید

این عبارت شامل ۲ عملگر متفاوت هست: `keyof` و `typeof` . این دو عملگر هنگام کار کردن با تایپ‌ها استفاده میشن و کمک می‌کنن از مقادیر فعلی تایپ‌هایی رو استخراج کنیم.

```ts
type T = { x: number; y: number };

const c: T = { x: 1, y: 2 };

type MyType = keyof typeof c; // "x" | "y"
```

ابتدا به صورت جدا اونها رو بررسی کنیم:

عملگر `typeof`

از این عملگر برای استخراج تایپ یک مقدار استفاده میشه و مشابه عملگر typeof جاوااسکریپت هست:

```ts
type T1 = { x: number; y: number };
const c1: T1 = { x: 1, y: 2 };

type T2 = typeof c1; // { x: number; y: number }
const c2: T2 = { x: 5, y: 10 };
```

عملگر `keyof`

اگر یک تایپ داریم و می‌خوایم یک تایپ Union مجزا از پراپرتی‌های اون تایپ بدست بیاریم، از `keyof` استفاده می‌کنیم:

```ts
type T1 = { x: number; y: number };
const c1: T1 = { x: 1, y: 2 };

type T2 = keyof T1; // "x" | "y"
const c2: T2 = "x";
```

توی این کد توی خط ۴ تونستیم یک تایپ از همهٔ از پراپرتی‌های تایپ خط ۱ داشته باشیم.

حالا وقتی یک مقدار داریم و می‌خوایم یک تایپ Union از پراپرتی‌های تایپ اون مقدار داشته باشیم، این دو عملگر رو با هم استفاده می‌کنیم:

```ts
const operations = {
  sum: () => {},
  multiply: () => {},
  divide: () => {}
};

type Operation = keyof typeof operations; // "sum" | "multiply" | "divide"

function math(operation: Operation, args: number[]) {
  // ...
}

math("sum", [1, 3, 5]);
```

۴۶. توی CSS واحدهای rem و em چه فرقی با هم دارن؟

این دو واحد شباهت زیادی به هم دارن و هر دو با نسبت مستقیم به مقدار پراپرتی `font-size` تغییر می‌کنن.

واحد em

توی محاسبات تایپوگرافی (مثل `font-size`) این واحد نسبت مستقیم با `font-size` **المنت والد** داره. و توی بقیه محاسبات این واحد نسبت مستقیم با اندازهٔ `font-size` **المنت فعلی** داره. برای مثال اگه `font-size` المنت والد برابر با `16px` باشه، با قرار دادن `font-size: 2em` برای المنت فرزند، مقدار نهایی برای `font-size` برابر با `32px` خواهد بود:

```css
.parent {
  font-size: 16px;
}

.parent > div {
  font-size: 2em; /*  16*2 = 32px  */
}
```

گفتیم که توی محاسبات غیرِ تایپوگرافی، این واحد نسبت مستقیم با اندازهٔ `font-size` **المنت فعلی** داره:

```
.element {
  font-size: 30px;
  width: 10em; /*  10*30 = 300px  */
}
```

واحد rem

این واحد نسبت مستقیم با اندازهٔ `font-size` المنت `root` (بیرونی‌ترین المنت، معمولاً `<html>`) داره:

```
:root {
  font-size: 25px;
}

article > div > p {
  font-size: 2rem;  /* 50px  */
  width: 20rem;     /* 500px  */
}
```

۴۷. دستور git stash چه کار می‌کنه؟

وقتی روی یک برنچ مشغول کار کردن هستیم و می‌خوایم به طور موقت بریم روی یک برنچ دیگه، در حالت عادی اگه تغییراتمون رو کامیت نکرده باشیم نمی‌تونیم این کار رو انجام بدیم. اما شرایطی هست که کامیت کردن این تغییرات مناسب نیست (مثلاً وقتی که هنوز توسعه کامل نشده) و می‌خوایم به هر شکل بریم روی یک برنچ دیگه. اینجا دستور `git stash` به کمکمون میاد. این دستور به طور موقت، تغییرات فعلی که هنوز کامیت نشدن رو به صورت لوکال ذخیره می‌کنه و اجازه میده بدون مشکل برنچمون رو عوض کنیم. نحوهٔ استفاده از این دستور به این صورت هست:

```
git stash
```

حالا می‌تونیم برنچ رو عوض کنیم:

```
git checkout bug-driven-development
```

بعد از اتمام کارمون وقتی که به برنچ ابتدایی برگشتیم:

```
git checkout initial-branch
```

باید تغییراتی که `git stash` برامون ذخیره کرده رو با دستور زیر به صورت دستی برگردونیم:

```
git stash pop
```

۴۸. کلمه‌کلیدی infer توی تایپ‌اسکریپت چه کار می‌کنه؟

کلمه‌کلیدی `infer` رو هنگام ساختن تایپ‌های شرطی (Conditional Type) می‌بینیم و از اون برای داشتن یک تایپ موقت توی Conditional Type ها استفاده می‌کنیم. `infer` باعث میشه تایپ‌هایی منعطف‌تر و Reusable داشته باشیم.

کد زیر رو در نظر بگیرید:

```
type MyReturnType<T> =
  T extends (...args: any) => infer U ? U : never;
```

کد بالا برای ساختن یک تایپ جدید از نوع خروجی یک تابع استفاده میشه. اینجا وقتی از `infer U` استفاده کردیم، به تایپ‌اسکریپت گفتیم که لطفاً وقتی که داره از این تایپ (`MyReturnType`) استفاده میشه، `infer U` رو جایگزین `U` کن. یعنی برای مثال:

```
type F = () => string;

type MyType = MyReturnType<F>; // string
```

تایپ‌اسکریپت وقتی می‌خواد کد بالا رو بررسی کنه، توی تایپ `MyReturnType` میاد `string` رو جایگزین `infer U` می‌کنه. یعنی:

```
type MyReturnType<T> =
  T extends (...args: any) => string ? string : never;
```

کد بالا به این معنی هست که اگه تایپ `T` از نوع تابعی بود که خروجی اون `string` هست، `string` ریترن کن، در غیر این صورت `never` . همونطور که دیدیم با `infer` تونستیم یک تایپ انعطاف‌پذیر داشته باشیم. یک مثال دیگه می‌تونه این باشه:

```
type PromiseReturnType<T> =
  T extends Promise<infer U> ? U : never;

type MyString = PromiseReturnType< Promise<string> >; //
string
```

تایپ بالا کمک می‌کنه نوع خروجی یک پرامیس رو بدست بیاریم.

۴۹. چرا می‌گیم ماژول‌های جاوااسکریپت Singleton هستن؟

Singleton بودن ماژول‌های جاوااسکریپت (منظور ES Modules) به این معنیه که وقتی یک ماژول رو چند بار توی جاهای مختلف برنامه Import می‌کنیم، جاوااسکریپت فقط بار اول عملیات پیاده‌سازی این ماژول رو انجام میده و برای استفاده‌های بعدی دیگه این عملیات پیاده‌سازی اتفاق نمیوفته و از همون Reference ماژول ابتدایی توی Import های بعدی استفاده میشه. با این کار مطمئن می‌شیم که کدهای توی یک ماژول فقط و فقط یک بار اجرا میشن و اطلاعاتی که توی یک ماژول وجود داره توی همهٔ Import ها مشترک هست.

۵۰. منظور از Progressive Enhancement توی توسعهٔ برنامه‌های فرانت‌اند چیه؟

Progressive Enhancement یک استراتژی ساختن صفحات وب هست که توی اون مهمترین اولویت، ساختن صفحات به نحوی هست محتوای اصلی صفحه برای همه نوع کاربران (موبایلی، دسکتاپ، مرورگر قدیمی) قابل دسترس و استفاده باشه. و ویژگی‌های اضافی در صورتی در دسترس باشه که مرورگر و دستگاه کاربر از اونها پشتیبانی کنه. به بیان ساده‌تر، هدف این استراتژی اینه که محتوای اصلی وبسایت برای اکثر کاربران قابل دسترس باشه و ویژگی‌های با اولویت کمتر فقط در شرایط مناسب در دسترس باشن. این استراتژی کمک می‌کنه تا همهٔ کاربرا بتونن بدون مشکل به موضوع اصلی برنامهٔ ما دسترسی داشته باشن و ویژگی‌های اضافی برای کاربری در دسترس باشه که مرورگر و دستگاه اون از اون ویژگی‌ها پشتیبانی می‌کنه.

Resources:

- https://stackoverflow.com/questions/60067100/why-is-the-infer-keyword-needed-in-typescript

۵۱. Call Stack توی جاوااسکریپت چیه؟

Call Stack یک ابزار هست که جاوااسکریپت با اون اجرای توابع توی برنامه رو مدیریت می‌کنه و مشخص می‌کنه برنامه توی چه مرحله‌ای هست و چه توابعی در صف اجرا هستن. منظور از Stack همون Data Structure معروف (LIFO) هست.

قبل از اجرای یک برنامهٔ جاوااسکریپتی، Call Stack یک محفظهٔ خالی هست. هر تابع به محض اینکه اجرا میشه توی Stack قرار می‌گیره و زمانی که اجرای اون تموم شد از Stack خارج میشه.

۵۲. منظور از Narrowing توی تایپ‌اسکریپت چیه؟

معنی واژه Narrowing یعنی محدود کردن یا باریک کردن. تایپ‌اسکریپت با عملیات Narrowing کاری می‌کنه که یک تایپ به شکل یک تایپ خاص‌تر و مشخص‌تر برای ما در دسترس باشه.

توی عکس همونطور که می‌بینیم، پارامتر `x` ممکنه رشته باشه یا عدد:

```ts
function example(x: string | number) {
  if (typeof x === "string") {
    x.toUpperCase();
  } else {
    x.toFixed(2);
  }
}
```

دیتی

توی این کد دو عملیات Narrowing انجام گرفته: یکی برای بلاک `if` و یکی هم برای `else`. با این ویژگی، با پارامتر `x` توی بلاک `if` فقط به صورت رشته رفتار میشه. برای مثال اگه متد `toFixed` رو صدا بزنیم خطا می‌گیریم. چون توی این بلاک، `x` فقط به صورت رشته قابل دسترس هست. همچنین توی بلاک `else` تایپ‌اسکریپت می‌دونه که نوع `x` دیگه رشته نیست و قطعاً عددی هست. پس با اون مثل یک عدد رفتار می‌کنه.

۵۳. چه زمانی استفاده از WebSocket مناسب نیست؟

WebSocket یک پروتکل برای برقراری ارتباط دو طرفه بین کلاینت و سرور هست. برخلاف پروتکل HTTP که توی اون برای دریافت اطلاعات سرور باید همیشه درخواست‌ها رو پیاده‌سازی و ارسال کنیم، توی WebSocket ارتباط دائمی هست و اطلاعات جدید سرور به محض اینکه قابل دسترس باشن به کلاینت ارسال میشن و بلعکس. یعنی یک ارتباط دو طرفه بین کلاینت و سرور.

WebSocket در مقایسه با HTTP مزایای زیر رو داره:

۱. ارتباط دائمی به صورت Stateful بین سرور و کلاینت
۲. ارتباط شبیه Real time. به دلیل حذف شدن عملیات زمان‌برِ برقراری ارتباط بین سرور و کلاینت
۳. ارتباط دو طرفه که توی اون سرور و کلاینت به‌صورت همزمان می‌تونن اطلاعات رد و بدل کنن

حالا با توجه به سوال، چه زمانی استفاده از WebSocket پیشنهاد نمیشه؟ هر ابزاری برای کاربرد خاص معرفی شده و استفاده از اون توی شرایط نامناسب نتیجهٔ منفی به همراه داره. برقراری یک ارتباط WebSocket پیچیدگی و هزینه‌های بالاتری نسبت به HTTP داره و استفاده از اون برای تعامل معمولی سرور و کلاینت برای دریافت و ارسال اطلاعات پیشنهاد نمیشه. کاربرد WebSocket بیشتر برای ارتباطات Real-time مثل چت هست و اگه چنین استفاده‌هایی نداریم بهتره از HTTP استفاده کنیم. همچنین برای رد و بدل کردن اطلاعات بزرگ مثل آپلود فایل، پروتکل‌هایی مثل FTP گزینه‌های مناسب‌تری در مقایسه با WebSocket هستن.

۵۴. درباره پراپرتی position توی CSS چه چیزهایی می‌دونین؟

پراپرتی `position` توی CSS برای تنظیم کردن موقعیت یک المنت توی صفحه به کار میره. این پراپرتی شامل ۵ مقدار زیر هست:

static: مقدار پیشفرض این پراپرتی هست و المنت‌های دارای این مقدار، طبق موقعیت قرارگیری اونها توی صفحه و طبق جریان طبیعی صفحه موقعیت‌بندی

میشن و قابل جابجایی توسط پراپرتی‌هایی مثل `top` و `left` و `z-index` نیستن.

relative: یک المنت دارای این مقدار، متناسب با موقعیت فعلی و طبیعی خودش توی صفحه می‌تونه با پراپرتی‌هایی مثل `top` و `left` و `z-index` جابجا بشه. بقیه المنت‌های توی صفحه تحت تأثیر موقعیت و جابجایی این المنت قرار نمی‌گیرن.

absolute: یک المنت دارای این مقدار، متناسب با اولین المنت بیرونی (والد) دارای `position` (غیر از `static`) می‌تونه با پراپرتی‌هایی مثل `top` و `left` و `z-index` موقعیت‌بندی بشه. برای مثال وقتی از کد زیر برای یک المنت استفاده می‌کنیم:

```
.element {
  position: absolute;
  top: 0;
}
```

این المنت توی موقیعت `top: 0` نزدیک‌ترین والد دارای `position` قرار می‌گیره. چنین المنتی از جریان طبیعی ساختار صفحه بیرون میره و فضای مشخصی نمی‌تونه داشته باشه.

fixed: یک المنت دارای این مقدار، متناسب با بیرونی‌ترین قسمت صفحه (Window) می‌تونه با پراپرتی‌هایی مثل `top` و `left` و `z-index` موقعیت‌بندی بشه. مثل `absolute`، چنین المنتی از جریان طبیعی ساختار صفحه خارج میشه و فضای مشخصی نمی‌تونه داشته باشه. المنت‌های `position: fixed` همیشه یک نقطه ثابتی توی صفحه دارن و موقعیت اونها حتی با اسکرول هم تغییر نمی‌کنه.

sticky: موقعیت یک المنت با این مقدار، بر اساس موقعیت نزدیک‌ترین والدِ قابلِ اسکرول تعریف میشه. وقتی اسکرول کاربر از نقطهٔ خاصی عبور نکرده باشه، این المنت مثل `position: relative` عمل می‌کنه و با عبور کردن اسکرول کاربر از اون نقطه، این المنت مثل `position: fixed` موقعیت‌بندی میشه.

۵۵. چطوری المنتی داشته باشیم که با تغییر border و padding اون طول و عرض اون ثابت بمونه؟

این کار با دادن مقدار `border-box` به پراپرتی `box-sizing` توی CSS قابل انجام هست:

```
.element {
  box-sizing: border-box;
  width: 200px;
  border: 10px solid;
}
```

توی کد بالا با تغییر کردن مقدار `border` ، عرض المنت روی مقدار 200 ثابت باقی می‌مونه. یعنی برای مثال با اضافه شدن `10px` برای `border` ، محتوای داخلی المنت به 180 کاهش پیدا می‌کنه تا عرض نهایی المنت برابر با 200 بشه.

مقدار پیشفرض پراپرتی `box-sizing` برابر با `content-box` هست. اگه کد بالا رو به شکل زیر تغییر بدیم:

```
.element {
  box-sizing: content-box;
  width: 200px;
  border: 10px solid;
}
```

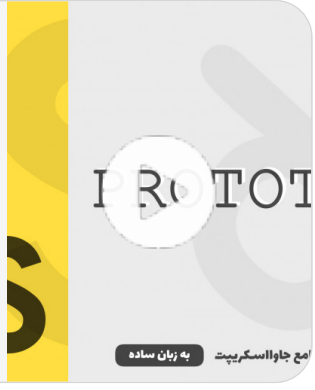
عرض نهایی المنت برابر با 220 (200 + 10 + 10) خواهد بود. یعنی اندازهٔ Border روی عرض نهایی تأثیر داره. مشاهدهٔ دمو

۵۶. از پروتوتایپ توی جاوااسکریپت چی می‌دونید؟

معنی لغوی پروتوتایپ یعنی **نمونه‌ی اولیه**. هر مقداری که توی جاوااسکریپت تعریف می‌کنیم، یک سری از ویژگی‌هاش رو از یک نمونهٔ اولیه به ارث می‌بره. این ویژگی‌ها شامل متدها و پراپرتی‌های مفید هستن که ما فکر می‌کنیم توی مقداری که ساختیم وجود دارن. در صورتی که موقع ساخته شدن، از یک والد به ارث برده میشن. مثل پراپرتی `length` توی رشته‌ها و آرایه‌ها:

```
const str = "Hello, I'm not lazy; I'm just waiting for AI
to write my code";
alert(str.length); // 61

const arr = [1, 2, 3, 4, 5];
alert(arr.length); // 5
```

توی این کد ما از یک پراپرتی به اسم `length` طوری استفاده کردیم که انگار توی مقادیر متغیرهای ما از قبل تعریف شده. اما ظاهراً چنین پراپرتی‌ای توی این مقادیر دیده نمیشه. به این دلیل که این مقادیر هنگام ساخته شدن این پراپرتی رو از یک والد به ارث بردن. برای آشنایی کامل با پروتوتایپ‌های جاوااسکریپت این پست رو ببینین:



بیشتر بدانید

[ویدئو] پروتوتایپ | آموزش جاوااسکریپت به زبان ساده

اساس جاوااسکریپت رو آبجکت‌ها و پروتوتایپ‌ها تشکیل میدن و درک پروتوتایپ‌ها کمک بزرگی به درک نحوه کارایی جاوااسکریپت می‌کنه

۵۷. Authorization و Authentication چه فرقی با هم دارن؟

این دو کلمه که ظاهری شبیه به هم دارن و توی مباحث امنیتی با اینکه گاهی اوقات به جای همدیگه مورد استفاده قرار می‌گیرن، معنای متفاوتی دارن.

Authentication

Authentication یعنی احراز هویت. یعنی بررسی کردن اینکه کاربر مورد نظر کی هست. برای مثال وقتی توی برنامه‌ای عملیات لاگین انجام می‌دیم، در واقع داریم Authentication انجام می‌دیم و می‌خوایم به برنامه بگیم که ما کی هستیم و هویت ما چیه.

Authorization

Authorization یعنی اجازه یا مجوز. یعنی بررسی کردن اینکه کاربر مورد نظر اجازه انجام دادن یک کار خاص رو داره یا نه. برای مثال می‌خوایم بررسی کنیم که آیا یک کاربر اجازهٔ دسترسی به یک فایل رو داره یا نه. اینجا باید Authorization انجام بدیم و مجوزهای کاربر رو بررسی کنیم. توی این شرایط ممکنه کاربر احراز هویت شده باشه اما مجوز انجام یک کار مشخص رو نداشته باشه.

۵۸. منظور از عملیات Non-Blocking I/O توی جاوااسکریپت چیه؟

عملیات Non-Blocking I/O که شاید اون رو بیشتر برای Node.js شنیده باشیم، به

عملیاتی گفته میشه که توی اون یک برنامه می‌تونه پردازش‌هایی Asynchronous روی ورودی‌ها و خروجی‌ها داشته باشه، بطوری که هنگام کار، Thread اصلی بلاک نشه و برای بقیه محاسبات آزاد باقی بمونه. عملیاتی مثل خوندن فایل از دیسک، اتصال و کار با دیتابیس و برقراری و مدیریت درخواست‌های HTTP.

توی روش‌های سنتی، یک زبان نمی‌تونست چنین عملیاتی رو جداگونه مدیریت کنه و برنامه می‌بایست درخواست بعدی رو بعد از به پایان رسیدن درخواست فعلی پردازش می‌کرد که نتیجهٔ اون، سرعت پایین پردازش درخواست‌ها بود.

اما جاوااسکریپت با پیاده‌سازی مکانیزم‌هایی مثل Event Loop و ارائه دادن ویژگی‌هایی مثل Promise ها و Async/Await می‌تونه چنین عملیاتی رو به صورت Asynchronous پردازش کنه که در نتیجهٔ اون سرعت پردازش و بطور کلی برنامهٔ ما رو بالاتر می‌بره.

۵۹. Virtual DOM چیه؟

Virtual DOM یک کپی از DOM واقعی هست که توی حافظه نگهداری میشه و هدف اون بهبود سرعت و عملکرد برنامه برای بروزرسانی UI هست. Virtual DOM که بیشتر توی فریم‌ورک‌های فرانت‌اند مثل ویو و ری‌اکت دیده میشه، یک واسط بین State برنامه و DOM اصلی هست و کمک می‌کنه عملیات بروزرسانی UI به طور بهینه‌تر و سریع‌تر انجام بشه.

روش سنتی بروزرسانی UI که توی اون DOM بطور دستی و مستقیم آپدیت می‌شد، یک روش سنگین و پرهزینه (مخصوصاً برای ساختارهای پیچیدهٔ DOM) به حساب میومد که واکنش‌گرایی و سرعت پایین صفحه نتیجهٔ اون بود.

Virtual DOM در ابتدا که صفحه رندر میشه از DOM واقعی ساخته و توی حافظه نگهداری میشه. وقتی که یک State توی برنامه تغییر می‌کنه، Virtual DOM ساختار خودش رو بروزرسانی می‌کنه و در صورت نیاز، با روش‌هایی بهینه‌شده و کم‌هزینه تصمیم به بروزرسانی UI و DOM اصلی می‌گیره. که نتیجهٔ این روش داشتن یک صفحهٔ سریع با واکنش‌پذیری بالا هست.

۶۰. منظور از Pseudo-elements و Pseudo-classes توی CSS چیه؟

Pseudo-elements (شبه المنت) به المنت‌هایی گفته میشه که در حالت عادی توی DOM نیستن. مثل المنت‌هایی که با `after::` ساخته و به صفحه اضافه میشن:

```
a::after {
  content: "*";
}
```

کد بالا به انتهای همهٔ المنت‌های `<a>` توی صفحه یک `*` اضافه می‌کنه. اینجا `*` یک Pseudo-element هست.

Pseudo-classes به سلکتورهایی مثل `hover` و `focus` گفته میشه که در شرایط خاصی المنت‌ها رو انتخاب می‌کنن:

```
a:hover {
  color: darkblue;
}
```

توی کد بالا `:hover` یک Pseudo-class هست.

...

Resources:

- https://developer.mozilla.org/en-US/docs/Web/CSS/position

۶۱. المنت Template توی HTML برای چه کاری هست؟

`<template>` یک المنت خاص توی HTML برای نگهداری محتوایی هست که در حالت عادی و خودکار توسط مرورگر در زمان لود شدن صفحه رندر نمیشه و می‌تونه بعداً توسط جاوااسکریپت استفاده و رندر بشه. به بیان ساده‌تر، `<template>` مثل یک محفظه برای نگهداری یک محتوای HTML ی هست که بعداً می‌تونه به صورت داینامیک توسط جاوااسکریپت به صفحه اضافه بشه:

```
<body>

  <template id="list-item-template">
    <li>
      <h2 class="title"></h2>
      <p class="description"></p>
    </li>
  </template>

  <ul></ul>

  <script>
    const template = document.querySelector('#list-item-
template');
    const list = document.querySelector('ul');

    const item = template.content.cloneNode(true);
    item.querySelector('.title').innerText = "Hello
world";

    list.appendChild(item);
  </script>
</body>
```

۶۲. منظور از HOF توی جاوااسکریپت چیه و چه مزایایی داره؟

HOF مخفف Higher-Order Function هست و به تابعی گفته میشه که می‌تونه یک تابع دیگه رو به عنوان ورودی بپذیره یا یک تابع رو به عنوان خروجی `return` کنه. در واقع HOF یک قابلیت مهم توی برنامه‌نویسی فانکشنال به حساب میاد. این قابلیت اجازه میده که با توابع مثل بقیه نوع‌های داده‌ای مثل رشته و آبجکت به عنوان First-class citizens رفتار کنیم و بتونیم اونها رو به متغیرها نسبت بدیم و

یا به عنوان ورودی/خروجی توابع از اونها استفاده کنیم:

```
function higherOrderFunction(callback) {
  callback();
}

higherOrderFunction(
  () => alert("Hello World")
);

higherOrderFunction(
  () => console.log("Hello World")
);
```

از مزایای این قابلیت اینه که باعث میشه اجزای برنامهٔ ما Encapsulation و انعطاف‌پذیری بیشتری داشته باشن.

۶۳. URL و URI چه فرقی با هم دارن؟

URI یک مفهوم کلی‌تر نسبت به URL هست. به عبارت دیگه URL نوعی URI به حساب میاد. ولی نه بلعکس.

URI

این واژه مخفف Uniform Resource Identifier هست و هدف اون ارائه دادن یک راه جامع و پایدار برای دسترسی به ریسورس‌هاست. یک ریسورس می‌تونه هر چیزی مثل یک کتاب و یک وسیلهٔ فیزیکی، آدرس یک صفحهٔ وب و فایل و عکس و ویدئو باشه. برای مثال، آدرس زیر یک URI برای دسترسی به یک عکس روی یک کامپیوتر شخصی هست:

```
file:///C:/Users/YourUsername/Pictures/example.jpg
```

URI زیرمجموعه‌هایی مثل URL و URN داره که URL معروف‌ترین نوع URI هست.

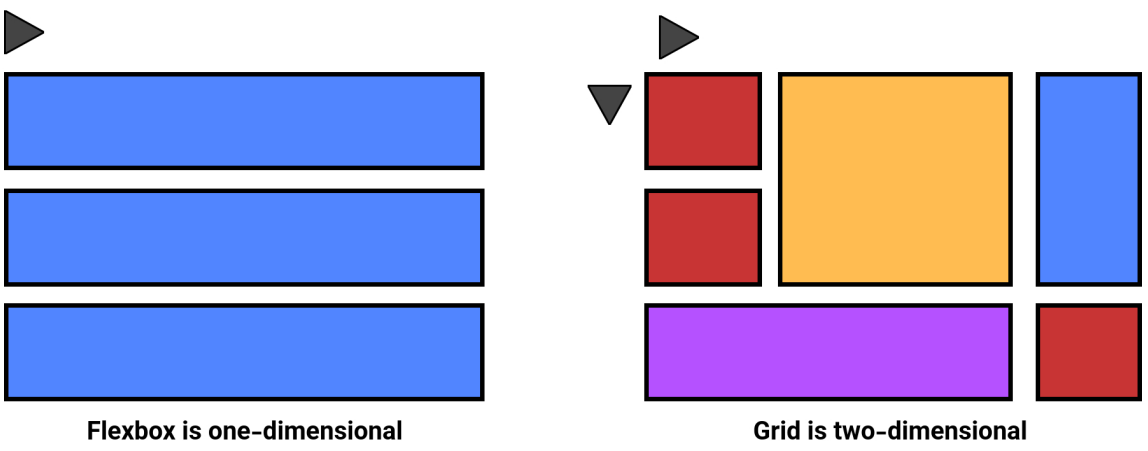
URL

این واژه مخفف Uniform Resource Locator و نوعی URI هست که به طور اختصاصی برای آدرس‌دهی ریسورس‌ها (صفحات، فایل‌ها، تصاویر و ...) روی بستر اینترنت به کار میره. برای مثال آدرس‌های زیر همگی URL هستن:

```
https://www.example.com
https://www.example.com/index.html
https://www.example.com/search?q=apple&type=fruits

ftp://username:password@ftp.example.com/path/to/file
```

۶۴. توی CSS چه زمانی Grid رو به Flexbox ترجیح بدیم؟

هر دو ویژگی برای ساختن Layout ها توی صفحه استفاده میشن. اما Flexbox بیشتر برای Layout های ساده و یک‌بعدی استفاده میشه. در حالی که کاربرد Grid بیشتر برای ساختن Layout های پیچیده‌تر و دو بعدی هست. عکس زیر به خوبی کاربرد این دو ویژگی رو نشون میده:



۶۵. از TDZ توی جاوااسکریپت چی می‌دونید؟

TDZ مخفف Temporal Dead Zone هست و زمانی به دنیای جاوااسکریپت پا گذاشت که `let` و `const` معرفی شدن. TDZ به قسمتی از کد گفته میشه که یک متغیر به دلیل اینکه هنوز پیاده‌سازی نشده قابل استفاده نیست.

همونطور که می‌دونیم متغیرهایی با `let` و `const` ساخته میشن، قبل از پیاده‌سازی قابل استفاده نیستن:

```
// ReferenceError: can't access lexical declaration 'x'
before initialization
alert(x);

let x = 10;
```

با اجرا شدن کد بالا خطا می‌گیریم که متغیر `x` با اینکه تعریف شده، نمی‌تونه استفاده بشه. بنابراین توی خط ۲ متغیر `x` توی TDZ قرار داره.

۶۶. چرا پیشنهاد میشه از حلقهٔ for...in جاوااسکریپت استفاده نکنیم؟

با استفاده از حلقه `for...in` می‌تونیم روی پراپرتی‌های یک آبجکت پیمایش انجام بدیم و به اونها دسترسی داشته باشیم. اگه با پروتوتایپ‌ها آشنایی دارید، باید بدونیم که این حلقه، پیمایش رو روی پراپرتی‌های پروتوتایپ‌ها هم انجام میده. برای مثال وقتی چنین کدی داشته باشیم:

```
const obj1 = { a: 1 };
const obj2 = { b: 1 };

Object.setPrototypeOf(obj1, obj2);

for (const key in obj1) {
  alert(key); // a, b
}
```

توی این کد `obj2` رو به عنوان پروتوتایپ `obj1` قرار دادیم و دیدیم که وقتی برای `obj1` از حلقه `for...in` استفاده کردیم، تونستیم به پراپرتی‌های آبجکت `obj2` هم دسترسی داشته باشیم. این یعنی پیمایش‌های ناخواسته و اضافی. پس باید بدونیم که ممکنه بهینگی و سرعت برنامه به این دلیل پایین بیاد. در کل پیشنهاد میشه جز برای تست و دیباگ برنامه از حلقه `for...in` استفاده نکینم. بجای اون بهتره از حلقه for...of استفاده کنیم.

۶۷. چه زمانی از تگ section توی HTML استفاده نکنیم؟

اگه توی صفحه المنت‌هایی مرتبط به هم داریم و قصد داریم اونها رو گروه‌بندی کنیم، از تگ `<section>` استفاده می‌کنیم. شاید این کار رو قبلاً با تگ `<div>` انجام می‌دادیم. اما `<section>` بر خلاف `<div>` ، یک تگ Semantic هست و هدف اون ارائه دادن روشی معنادار برای گروه‌بندی کردن المنت‌هایی هست که از لحاظ محتوا و معنا به هم مرتبط هستن. پس اگه المنت‌هایی داریم که هیچ ارتباطی به هم ندارن و از لحاظ معنایی مفهوم خاصی رو منتقل نمی‌کنن، باید `<div>` رو به `<section>` ترجیح بدیم. به طور کلی نباید از `<section>` استفاده کنیم اگر:

- اگه قصد گروه‌بندی المنت‌هایی داریم که هیچ ارتباطی به هم ندارن. مثلاً دو تصویر غیر مرتبط
- اگه نمی‌تونیم برای `<section>` از المنت‌های Heading مثل `h2` استفاده کنیم. چون طبق استانداردها، هر `<section>` باید توسط یک

المنت Heading توصیف بشه

- اگه از المنت مرتبط‌تری بجای `<section>` می‌تونیم استفاده کنیم. مثلاً از `<footer>` برای گروه‌بندی عناصر فوتر و `<article>` برای محتوای یک مقاله

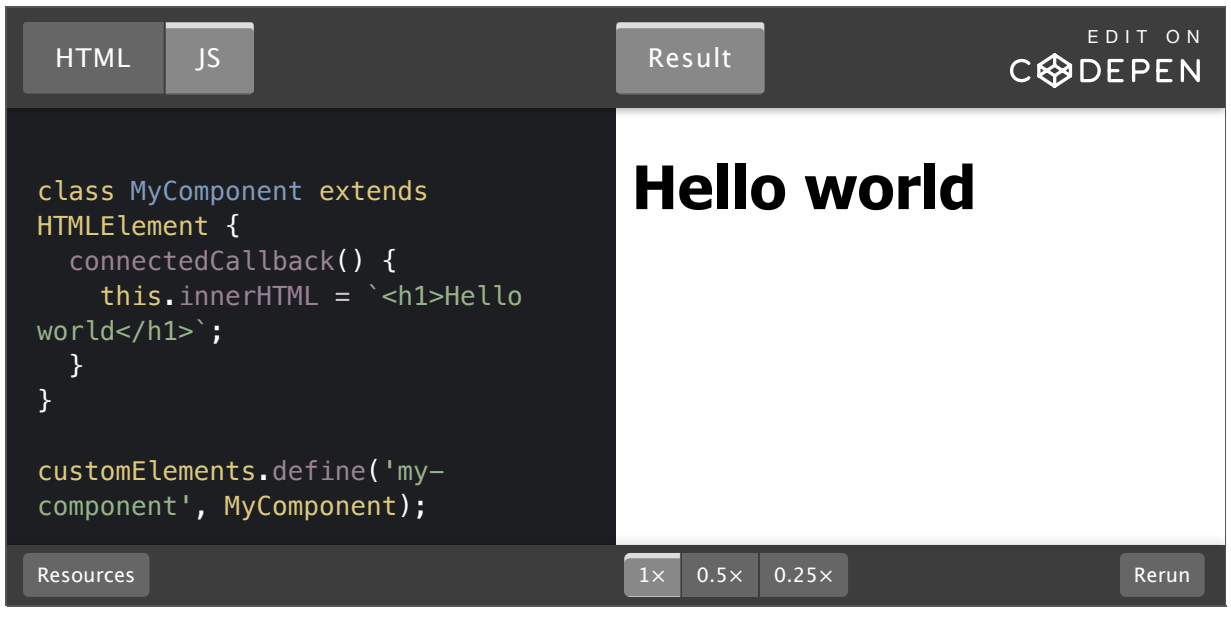
۶۸. از Web Components چی می‌دونید؟

اگه از فریم‌ورک‌هایی مثل ری‌اکت و ویو استفاده کرده باشین، با مفهوم و مزایای کامپوننت‌ها آشنایی دارید. وب کامپوننت‌ها مجموعه‌ای از API های وب هستن که برای ساختن کامپوننت‌هایی اختصاصی، با قابلیت استفاده مجدد (Reusable) و کپسوله‌شده (Encapsulated) بدون نیاز به فریم‌ورک خاصی و با استفاده از HTML و جاوااسکریپت و CSS استفاده میشن.

منظور از Encapuslated بودن اینه که این کامپوننت‌ها خاصیتی دارن که اجازه میدن اجزایی مثل Modal و Dialog بسازیم به‌طوری که بقیه المنت‌های توی صفحه تحت تاثیر استایل‌ها و جاوااسکریپتی که توی اون استفاده شده قرار نگیرین که این قابلیت با استفاده از یک ویژگی به اسم Shadow DOM قابل انجام هست.

Shadow DOM در واقع یک نوع DOM با ویژگی‌های اختصاصی هست که محتوای اون مثل HTML و جاوااسکریپت‌ها و استایل‌ها ایزوله هستن و نمی‌تونن روی DOM اصلی تاثیر بذارن.

همونطور که گفتیم، یک وب کامپوننت رو می‌تونیم بدون نیاز به ابزار یا فریم‌ورک خاصی بسازیم. در واقع از همین کامپوننت‌ها می‌تونیم خیلی راحت توی فریم‌ورک‌ها به عنوان یک المنت مستقل استفاده کنیم. کد زیر یک نمونهٔ ساده از یک وب کامپوننت هست:



۶۹. توی CSS منظور از Specificity چیه و چه قوانینی داره؟

حتماً دیدین که Rule هایی که توی بعضی از سلکتورها می‌نویسیم، نسبت به بقیه Rule ها قدرت بیشتری دارن و در نتیجه روی المنت اعمال میشن. به این قضیه میگن CSS Specificity. هر چی Specificity یک سلکتور بالاتر باشه، قدرت بیشتری داره تا روی المنت اعمال بشه.

Specificity به مجموعه‌ای از قوانینی گفته میشه که مرورگر با استفاده از اونها می‌تونه تصمیم بگیره برای یک المنت کدوم استایل اولویت بالاتری داره و باید روی اون المنت اعمال بشه. برای مثال اگه چنین استایل‌هایی داریم:

```
div {
  background: red;
}

div.blue {
  background: blue;
}
```

مرورگر با استفاده از قوانین Specificity می‌تونه تصمیم بگیره کدوم `background` برای المنت `div` توی صفحه باید اعمال بشه. توی CSS هر Selector یک اولویت و به بیان فنی‌تر Specificity مشخصی داره:

۱. استایل‌های Inline (بالاترین اولویت):

```
<h1 style="color: #fff;">
```

۲. سلکتور ID

```
#navbar {
  color: green
}
```

۳. سلکتورهای `class` و اتریبیوت و pseudo-class ها

```
.green {
  color: green;
}

[href] {
  color: yellow;
}

div:hover {
  color: red;
}
```

۴. المنت‌ها و شبه‌المنت‌ها (پایین‌ترین اولویت)

```css
button {
  color: blue;
}

a::before {
  color: green;
}
```

حتماً با دستور `!important` آشنایی دارید. اگه از این دستور برای پراپرتی CSS استفاده کنیم، اون پراپرتی با بالاترین Specificity روی المنت اعمال میشه و Selector دیگه‌ای با هر ارزش Specificity نادیده گرفته میشه.

۷۰. چه زمانی از اتریبیوت‌های defer و async روی تگ script استفاده کنیم؟

در حالت عادی اسکریپت‌هایی که به صفحه اضافه می‌کنیم، به قول معروف Render-blocking هستن و مرورگر تا زمانی که اونها رو دانلود نکنه، نمی‌تونه ادامهٔ پردازش صفحه و DOM رو انجام بده. اما راه‌هایی وجود داره که بتونیم از یک اسکریپت توی هر جایی از صفحه استفاده کنیم بدون اینکه نگران بلاک شدن پردازش‌های مرورگر باشیم. اون راه‌ها استفاده از اتریبیوت‌های `defer` و `async` هست.

`defer` به مرورگر میگه منتظر نباش تا من لود بشم. بنابراین پردازش صفحه متوقف نمیشه و مرورگر میره به ادامه پردازش صفحه می‌پردازه. اون اسکریپت توی بک‌گراند لود میشه و به محض اینکه DOM به‌طور کامل ساخته شد اجرا میشه. بنابراین `defer` باعث وقفه توی کار مرورگر نمیشه.

اسکریپت `async` هم توی بک‌گراند دانلود و بعد اجرا میشه. پردازش DOM و همچنین بقیه اسکریپت‌ها منتظر این اسکریپت نمی‌مونن و کاملاً مستقل از همدیگه به کارشون می‌پردازن. به بیان ساده‌تر، اسکریپت `async` به محض اینکه لود شد اجرا میشه؛ بدون توجه به اینکه آیا صفحه به طور کامل لود شده یا نه.

به طور کلی `defer` و `async` رو به تگ‌های script ـی اضافه می‌کنیم که قراره یک اسکریپت خارجی رو دانلود و اجرا کنن. این دو اتریبیوت باعث میشن که اسکریپت‌ها به صورت مستقل دانلود بشن تا وقفه‌ای توی پردازش صفحه ایجاد نشه. توی `defer` یک اسکریپت زمانی اجرا میشه که DOM کاملاً توسط مرورگر پردازش شده باشه. اما `async` به این توجه نمی‌کنه که آیا DOM لود شده یا نه و یا مرورگر هنوز مشغول پردازش صفحه هست یا نه.

برای آشنایی بیشتر می‌تونید این پست رو ببینید:

بیشتر بدانید

اسکریپت‌های defer و async

با قابلیتی آشنا می‌شیم که بهمون اجازه میده تا بتونیم یک صفحه HTML رو سریع‌تر لود کنیم

. . .

Resources:

- https://blog.webdevsimplified.com/2020-06/template-tag/

۷۱. از gzip چی می‌دونید؟

یکی از مهمترین تکنیک‌های افزایش سرعت برنامه‌های وب استفاده از gzip هست. gzip یک الگوریتم و قابلیت هست که باید توی سرور فعالسازی و کانفیگ بشه و هدف اون فشرده‌سازی و کاهش دادن اندازهٔ فایل‌هایی هست که از سرور به سمت کلاینت فرستاده میشن. به‌طوری که حجم یک فایل می‌تونه تا ۹۰٪ کاهش پیدا کنه که در نتیجه معیارهایی مثل سرعت، UX و SEO برنامهٔ ما بهبود پیدا می‌کنه. gzip معمولاً به صورت پیشفرض روی همهٔ سرورها فعال هست و همچنین تقریباً همهٔ مرورگرها از اون به‌طور خودکار پشتیبانی می‌کنن.

۷۲. منظور از Transitive Dependency توی فایل package.json چیه؟

فرض کنیم قصد داریم پکیج A رو توسط دستور `npm install` به برنامه اضافه کنیم. فرض کنیم پکیج A توی خودش وابسته به پکیج B هست. این یعنی توسعه‌دهندهٔ پکیج A، پکیج B رو توی قسمت `dependencies` فایل `package.json` پکیج A قرار داده. حالا فرض کنیم پکیج B هم وابسته به یک پکیج دیگه به اسم C هست. پس پکیج A بطور غیر مستقیم به پکیج C هم وابستگی داره. به نحوه وابستگی پکیج A به C، میگن وابستگی Transitive. یعنی پکیج A برای فعالیتش نیاز به حضور پکیج C داره:

```
A -> B -> C -> D ...
```

پس به وابستگی‌های غیر مستقیم می‌گیم Transitive Dependency.

وقتی توی مسیری که فایل `package.json` وجود داره، دستور `npm install` رو بزنیم، همه وابستگی‌های Transitive هم نصب و به پوشهٔ `node_modules` اضافه میشن. دقت کنین که وابستگی‌های Transitive باید توی قسمت `dependencies` نوشته شده باشن. وابستگی‌های Transitive که توی `devDependencies` لیست شدن نصب نخواهند شد.

۷۳. چه چالش‌هایی برای یک توسعه‌دهندهٔ سینیور وجود داره؟

یک توسعه‌دهندهٔ سینیور بودن، چیزی فراتر از مهارت‌های فنی و هارد اسکیل‌ها هست و معمولاً یک سینیور واقعی/غیر واقعی رو میشه بدون نگاه کردن به کدها تشخیص داد. به بیان ساده‌تر، اگر بهترین کدنویس دنیا باشیم ولی بعضی از مهارت‌ها رو نداشته باشیم، نمی‌تونیم بگیم سینیور هستیم. این مهارت‌ها شامل:

۱. نقش رهبری و مدیریت تیم: به عنوان یک توسعه‌دهندهٔ سینیور همیشه باید مایل باشیم به درستی و به شکل ساده و قابل فهم به سوالات افراد تازه‌کار جواب بدیم و در برابر اونها صبور و مشخص‌کنندهٔ مسیر باشیم.

۲. مدیریت پروژه: باید مایل باشیم که توی تصمیم‌گیری‌ها شرکت کنیم و نظرمون رو ارائه بدیم. این تصمیم‌گیری‌ها می‌تونه شامل مباحثی مثل انتخاب تکنولوژی و ابزارها باشه و یا مباحث عمومی‌تر مثل برنامه‌ریزی و تقسیم‌بندی کردن کارها.

۳. شناخت نیازهای مشتری: در اکثر موارد مشتری کوچیک‌ترین اطلاعاتی از کاری که انجام می‌دیم و خروجی‌ای که قراره تحویل بدیم نداره و یک توسعه‌دهندهٔ سینیور خوب باید بتونه نیازهای گفته‌شده و ناگفتهٔ مشتری رو شناسایی کنه.

۴. مهارت‌های فنی همیشه به‌روز: برای اینکه بتونیم توی تصمیم‌گیری‌ها شرکت کنیم، به افراد تازه‌کار راهنمایی بدیم و نیازهای مشتری رو به درستی شناسایی کنیم، داشتن اطلاعات به‌روز و آشنایی با جدیدترین تکنولوژی‌ها، ابزارها و تکنیک‌های برنامه‌نویسی، یکی از مهمترین ملاک‌های یک توسعه‌دهندهٔ سینیور به حساب میاد.

۵. تعادل کار و زندگی: همهٔ ما توی برهه‌ای از زندگی ممکنه دچار افراط و تفریط بشیم و هیچ خط وسطی وجود نداره. ولی تلاش برای حفظ و بازگشتن به نقطهٔ تعادل یکی از مهمترین کارهایی هست که باید آگاهانه انجامش بدیم. در غیر این صورت دچار فروپاشی خواهیم شد.

بیشتر بدانید

تفاوت توسعه‌دهنده Junior و Senior

تفاوت بین توسعه‌دهنده های Junior، Mid-Level و Senior رو امروز توی این مقاله با هم بررسی میکنیم

۷۴. منظور و هدف از Semantic HTML چیه؟

Semantic HTML یک روش نوشتن کدهای HTML با استفاده از تگ‌هایی مثل `main` و `footer` و `header` هست و هدف اون داشتن کدهایی هست که

هم برای توسعه‌دهنده‌ها و هم برای ابزارهای تفسیرکنندهٔ کدها خواناتر و معنادارتر باشه.

توی روش‌های قدیمی رایج بود که از یک المنت نامربوط برای انجام یک کار استفاده بشه. مثلاً استفاده از المنت `table` برای قسمت‌بندی صفحه. اما این کار باعث سردرگمی توسعه‌دهنده‌ها و ابزارهایی مثل مرورگرها می‌شد. به همین دلیل المنت‌هایی با معنا معرفی شدن که به این المنت‌ها گفته میشه Semantic Elements.

توی Semantic HTML هر المنتی یک معنی اختصاصی داره. مثلاً تگ‌های `h1` تا `h6` که هر کدوم وظیفهٔ مشخصی دارن و باید توی جای درست ازشون استفاده بشه. اگه قصد داریم صفحه‌ای تمیزتر، خواناتر، Accessible و با سئوی بهتر داشته باشیم، یکی از مهمترین راه‌ها استفاده از Semantic HTML هست.

۷۵. توی تایپ‌اسکریپت تایپ‌های void و never چه فرقی باهم دارن؟

وقتی تابعی رو می‌بینیم که خروجی اون `void` هست، به این معنیه که اون تابع چیزی رو ریترن نمی‌کنه. بنابراین اگه تابعی داریم که هیچ چیزی ریترن نمی‌کنه، برای نوع خروجی اون از تایپ `void` استفاده می‌کنیم. اما بعضی توابع ممکنه اصلاً به مرحلهٔ ریترن کردن نرسن و کدهای توی این تابع هیچوقت به طول کامل - از ابتدا تا انتها - اجرا نشن. بنابراین از تایپ `never` برای تابعی استفاده می‌کنیم که حدس می‌زنیم به مرحلهٔ ریترن کردن نرسه.

جزییات بیشتر و مثال‌ها:

۷۶. چه زمانی از State management توی برنامه‌ها استفاده کنیم؟

با استفاده از ابزارهای State management مثل Redux و Pinia می‌تونیم راحت‌تر و منسجم‌تر اطلاعات و به قول معروف State های توی برنامه رو مدیریت کنیم.

State management کمک می‌کنه تا اطلاعات برنامه توی یک جای مشخص تعریف و متمرکز بشه و همچنین از راه‌های مشخص بشه به اونها دسترسی داشت و یا اونها رو تغییر داد. این کار باعث میشه اطلاعات امنیت بیشتری داشته باشن. همچنین باعث میشه برنامه‌ای خواناتر، ساده‌تر، Maintainable و Reusable داشته باشیم. اما State management ها مثل هر ابزار دیگه‌ای برای هر شرایطی مناسب نیستن و استفاده از اونها ممکن باعث پیچیدگی بیش‌از حد برنامه بشه. به طور کلی توی شرایط زیر بهتره از State management ها استفاده نکنیم:

۱. یک برنامهٔ کوچیک. شاید این مهمترین دلیلی باشه که از State management استفاده نکنیم. اگه برنامه و اطلاعات ما بزرگ نیست بهتره دنبال راه‌های دیگه برای مدیریت اطلاعات باشیم. مثل استفاده از Context و React Query توی ری‌اکت.

۲. اطلاعات Local یک کامپوننت: اگه اطلاعات یک کامپوننت فقط مربوط به همون کامپوننت هست، باید توی همون کامپوننت نگهداری بشه.

۷۷. چه تکنیک‌هایی برای نمایش بهینهٔ تصاویر توی صفحهٔ وب می‌شناسید؟

یک تصویر ممکنه حجمی به اندازهٔ کل جاوااسکریپت و CSS برنامه داشته باشه. پس بهینه کردن تصاویر تاثیر زیادی توی سرعت و عملکرد یک برنامه داره. با استفاده از تکنیک‌های زیر می‌تونیم تصاویر رو توی صفحه به شکل بهینه‌تر نمایش بدیم:

۱. **استفاده از فرمت مناسب** مثل JPG و WebP که می‌تونه حجم عکس رو تا حد زیادی کاهش بده.

۲. **فشرده‌سازی تصاویر**: ابزارهایی مثل Squoosh وجود دارن که با فشرده‌سازی عکس‌ها کمک می‌کنن حجم عکس‌ها رو تا حد زیادی بدون افت کیفیت کاهش بدیم.

۳. **تصاویر Responsive**: اگه برای اندازه‌های مختلف صفحه تصاویر مختلفی داریم، می‌تونیم از اتریبیوت `srcset` تگ `img` استفاده کنیم تا تصویر مناسبی رو برای اندازه‌های مختلف صفحه نمایش بدیم.

۴. **استفاده از Lazy Loading**: با استفاده از ابزارهای اختصاصی و همچنین اتریبیوت `lazy` روی تگ `img` می‌تونیم کاری کنیم که تصاویر فقط زمانی دانلود و نمایش داده بشن که توی محدودهٔ Viewport هستن.

۵. **استفاده از CDN ها**: CDN ها سرورهای اختصاصی، سریع و بهینه برای نگهداری و دانلود فایل‌های استاتیک مثل تصاویر و ویدئوها هستن.

۶. استفاده از Preload و Preconnect برای تصاویر

۷۸. Debounce و Throttle چه فرقی با هم دارن؟

وقتی بحث بهینگی و کارایی برنامه‌های فرانت‌اند میشه، دو تکنیک مهم ظاهر میشن: **Throttle** و **Debounce**. هر دو تکنیک شباهت‌های زیادی دارن و برای به تاخیر انداختن اجرای یک قطعه کد به کار میرن. به قول معروف برای *Rate Limiting*.

تابع Throttle مشخص میکنه که کدهای ما توی یک بازه زمانی مشخص فقط یک بار اجرا بشه. مثلاً می‌خوایم یک قطعه کد توی بازه زمانی ۱۰ ثانیه، حداکثر یک‌بار اجرا بشه. اینجا از تابع Throttle استفاده می‌کنیم.

با تابع Debounce می‌تونیم مطمئن بشیم که یک قطعه کد دوباره اجرا نمیشه مگر اینکه مقدار مشخصی از زمان گذشته باشه. معروف‌ترین مثال Debounce برای پیاده‌سازی جستجوی لحظه‌ای هست. ابتدا صبر می‌کنیم تا کاربر دست از تایپ کردن بکشه و بعد جستجو رو شروع می‌کنیم.

برای آشنایی کامل و نحوهٔ پیاده‌سازی این توابع توی جاوااسکریپت این پست رو ببینید:

بیشتر بدانید

پیاده‌سازی توابع Throttle و Debounce در جاوااسکریپت

با دو تابع آشنا می‌شیم که کمک می‌کنن برنامهٔ بهینه‌تر و سریع‌تر و کاربرد پذیرتر داشته باشیم

۷۹. چرا پیشنهاد میشه از await توی حلقه‌ها استفاده نکنیم؟

شاید با چنین کدی مواجه شده باشیم:

```
for (const url of urls) {
  const response = await fetch(url);
}
```

اما این کد کاملاً غیر بهینه به حساب میاد. باید بدونیم که یکی از هدف‌های async/await پیاده‌سازی قابلیت پردازش موازی و مدیریت کردن عملیات ناهمگام هست.

وقتی توی هر پیمایش حلقه از `await` استفاده می‌کنیم، یه جورایی مزیت‌های پردازش موازی رو نادیده گرفتیم. توی این شرایط، عملیات موازی نیست. بلکه متوالی هست. چونکه پیمایش بعدی باید صبر کنه تا عملیات ناهمگام فعلی تموم بشه. پس بهتره کاری کنیم که عملیات ناهمگام بصورت موازی اجرا بشن.

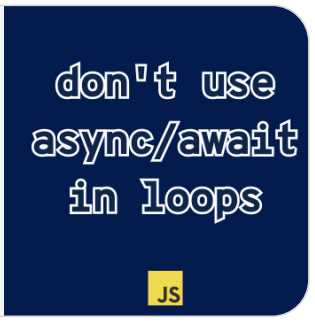
بهتره توی حقله منتظر نتیجهٔ عملیات ناهمگام نباشیم. تک تک این عملیات رو توی یک آرایه قرار بدیم و نهایتاً بیرون از حلقه از `Promise.all()` استفاده کنیم تا از مزیت پردازش‌های موازی استفاده کرده باشیم:

```
(async () => {
  const responses = [];

  for (const url of urls) {
    const response = fetch(url);
    responses.push(response);
  }

  await Promise.all(responses);
})();
```

بیشتر بدانید

چرا از await داخل حلقه‌ها استفاده نکنیم؟

یاد می‌گیرم که چرا استفاده از await توی حلقه‌ها کار مناسبی نیست و با یک روش جایگزین آشنا می‌شیم



۸۰. چرا اجرای کد زیر هیچوقت به پایان نمیرسه؟

```
let x = true;

setTimeout(() => {
  x = false;
}, 0);

while (x === true) {
  console.log(`(ツ)`);
}
```

توی این کد باید بدونیم که حلقهٔ `while` هیچوقت به پایان نمی‌رسه. به این دلیل که هیچوقت نوبت به اجرای `setTimeout` نمی‌رسه که `x` رو `false` کنه. به بیان تخصصی‌تر، `setTimeout` یک Web API هست و اولویت پایین‌تری نسبت به کدهای داخلی زبان جاوااسکریپت دارن و زمانی اجرا میشن که Call Stack خالی باشه. پس به دلیل اینکه حلقهٔ `while` همچنان با `x = true` در حال اجرا هست، هیچوقت کارش تموم نمیشه و در نتیجه Call stack هم خالی نمیشه تا تابعی که به `setTimeout` پاس داده شده رو اجرا کنه.

. . .

Resources:

- https://maximorlov.com/linting-rules-for-asynchronous-code-in-javascript/#2-no-await-in-loop
- https://eslint.org/docs/rules/no-await-in-loop

خب دوستان، ۸۰ سوال رو با هم بررسی کردیم. امیدوارم استفاده کنین و توی مسیر موفقیت و خوشبختی باشین. علاوه بر وبسایت دیتی، می‌تونین از طریق تلگرام و لینکدین با من در ارتباط باشین ✌️😉

۱۴۰۳/۰۱/۲۱