OPTIMISATION DE REACT

TECHNIQUES D'OPTIMISATION DES PERFORMANCES DE REACT

Maintenir la composante locale de l'État si nécessaire

- Dans les applications complexes et de grande taille, les mises à jour d'état sont fréquentes et peuvent être asynchrones ou déclenchées par l'utilisateur. Les composants sont souvent rendus plusieurs fois avant toute interaction utilisateur. Les développeurs doivent détecter et éviter les rendus inutiles, surtout lorsqu'ils se produisent dans les composants parents, ce qui entraîne des redessinages inutiles de toute la hiérarchie de composants, gaspillant des cycles CPU.
- Pour éviter les rendus inutiles, il est conseillé de localiser l'état uniquement dans les composants concernés. Cependant, lorsqu'un état global doit être transmis aux composants enfants, il est crucial d'apprendre comment empêcher les rendus des composants enfants non affectés.

Mémorisation des composants React pour éviter les rerendus inutiles

- La mémorisation en cache les résultats des appels de fonctions coûteux, accélérant ainsi le code. En React, pour un composant simple qui se rend à chaque changement de props, utiliser la mémorisation est bénéfique si ces props changent rarement. Cela évite les rendus inutiles et améliore les performances.
- 1. **En utilisant React.memo()**
- 2. **En utilisant le hook useCallback**
- 3. **En utilisant le hook useMemo()**

Fractionnement du code dans React à l'aide de la fonction import() dynamique

- La taille du bundle JavaScript ralentit le chargement des applications. Par défaut, webpack inclut tout dans un seul bundle, forçant le navigateur à télécharger l'ensemble du code, y compris les parties rarement utilisées et les bibliothèques lourdes, même si seule la page "about us" est nécessaire.
- Le fractionnement du code permet de charger paresseusement et de manière asynchrone certaines parties du code, utilisant React.lazy et Suspense pour optimiser le chargement des routes avec react-router. Pour les applications client, c'est simple, mais plus complexe côté serveur, car tout le code doit être prêt à être rendu. React.lazy et Suspense ne fonctionnent pas pour le rendu côté serveur. Le fractionnement du code divise un gros fichier bundle en plusieurs morceaux, chargés à la demande, améliorant ainsi les performances des pages dans une application React complexe.

Fractionnement du code dans React à l'aide de la fonction import() dynamique

```
const Home = React.lazy(() => import("./components/Home"));
const About = React.lazy(() => import("./components/About"));
```

Après l'importation, nous devons effectuer le rendu des composants paresseux au sein d'un composant Suspense comme suit :

```
<React.Suspense fallback={<p>Loading page...</p>}>
  <Route path="/" exact>
    <Home />
  </Route>
  <Route path="/about">
    <About />
  </Route>
</React.Suspense>
```

Virtualiser les longues listes

- La virtualisation des listes, ou fenêtrage, améliore les performances en ne rendant qu'un sous-ensemble de lignes d'une longue liste de données à la fois. Cela réduit le temps de rendu et le nombre de nœuds DOM créés. Les bibliothèques React populaires comme react-window et react-virtualized offrent des composants réutilisables pour afficher des listes, des grilles et des données tabulaires.

Lazyload des images dans React

- Le lazyload des images, comme le fenêtrage, empêche la création de nœuds DOM inutiles, améliorant ainsi les performances de notre application React.
- Les bibliothèques de chargement lazyload populaires pour les projets React comprennent react-lazyload et react-lazy-load-image-component.

Fichiers Chunk multiples

- Chaque application commence avec quelques composants, mais en ajoutant des fonctionnalités et des dépendances, le fichier de production peut devenir énorme. Pour résoudre ce problème, utilisez le plugin "SplitChunksPlugin" de webpack. Ce plugin divise les fichiers, permettant au navigateur de les mettre en cache moins souvent et de télécharger les ressources en parallèle, réduisant ainsi le temps de chargement. Il s'attaque aussi à la duplication de code, en plaçant les modules coûteux dans des morceaux séparés pour éviter de charger du contenu redondant.

Comment optimiser une application frontend de manière globale ?

Utiliser un CDN

- Utiliser un réseau de diffusion de contenu (CDN) est crucial. Il sert vos ressources depuis un emplacement proche de l'utilisateur, offrant une bande passante plus large et des latences plus faibles. Les CDN, conçus pour la vitesse et souvent peu coûteux, peuvent aussi agir comme proxy et mettre en cache vos appels API. Ils respectent les en-têtes Cache-Control, permettant de définir des délais d'expiration différents pour chaque ressource. Les CDN modernes peuvent ignorer certains paramètres de requête, mettre en cache les cookies et ajouter des en-têtes personnalisés, comme l'adresse IP d'origine ou la géolocalisation IP.

Utilisez des animations CSS plutôt que des animations JS

- Les animations offrent une expérience utilisateur fluide et agréable et peuvent être créées de trois façons : les transitions CSS, les animations CSS, et le JavaScript.
- Les transitions CSS permettent d'animer facilement entre deux états CSS, comme le repos et le survol d'un bouton. Une transition commence toujours à partir du style actuel, même si l'élément est déjà en transition.
- Les animations CSS permettent de définir des animations entre un ensemble de valeurs initiales et un ensemble de valeurs finales, en utilisant des styles et des images clés pour indiquer les états de début, de fin, et les points intermédiaires.

Ne plus utiliser CSS-in-JS

- La plupart des bibliothèques génèrent un CSS qui se retrouve dans une balise de style, ce qui peut entraîner une page HTML massive en rendu côté serveur, impactant les performances. Bien que vous receviez le CSS nécessaire, il ne peut pas être mis en cache, et le code de génération est inclus dans le bundle JavaScript.
- Des bibliothèques CSS-in-JS à temps d'exécution zéro, comme linaria, permettent d'écrire des styles dans des fichiers JS et de les extraire en fichiers CSS lors de la construction. Elles prennent en charge le style dynamique basé sur des accessoires via des variables CSS.

Utiliser un rendu côté serveur (SSR) plutot qu'un rendu côté client (CSR)

- Les applications côté client nécessitent que le navigateur récupère, analyse et exécute tout le JavaScript pour générer le document visible par les utilisateurs, ce qui peut être long, surtout sur des appareils lents.
- Le rendu côté serveur améliore l'expérience utilisateur en permettant au navigateur de récupérer et d'afficher le document complet avant de charger et d'exécuter le code JavaScript. Cela évite un rendu lent du DOM sur le client et hydrate plutôt l'état. Bien que l'application ne soit pas immédiatement interactive, elle se charge beaucoup plus rapidement. Après le chargement initial, elle fonctionne comme une application côté client classique, sans revenir à un rendu côté serveur inefficace.

Optimiser les dépendances NPM

- Pour réduire la taille de votre paquet d'applications, examinez le code utilisé à partir des dépendances. Par exemple, Moment.js inclut des fichiers localisés pour plusieurs langues. Si vous n'avez pas besoin de cette fonctionnalité, vous pouvez supprimer les locales inutilisées avec le moment-locales-webpack-plugin. De même, avec Lodash, si vous n'utilisez que quelques méthodes sur des centaines disponibles, il est inutile d'inclure toutes ces méthodes dans votre paquet final.
- Pour évaluer la taille des importations dans votre éditeur, utilisez l'extension Import Cost pour VS. Optez pour des bibliothèques et composants qui offrent la même fonctionnalité, mais avec une taille de bundle plus réduite, car certaines bibliothèques proposent des variations qui remplissent les mêmes besoins pour vos applications web.

Analyse et optimisation de votre bundle Webpack

- Avant de déployer votre application en production, inspectez et analysez votre bundle pour éliminer les plugins ou modules inutiles. Utilisez la bibliothèque **webpack-bundle-analyzer**, qui offre une visualisation interactive en treemap de la taille des fichiers de sortie de webpack. Cela vous permet de voir quels modules sont inclus et ceux qui occupent le plus d'espace.
- Bien que les bundlers modernes prennent en charge un secouage d'arbre (treeshaking) sophistiqué pour supprimer automatiquement le code inaccessible, il est essentiel de rester vigilant. Ce module vous aidera à :

1. Découvrir le contenu réel de votre paquet.
2. Identifier les modules qui consomment le plus d'espace.
3. Trouver les modules qui ont été ajoutés accidentellement.

Pour optimiser davantage votre application, voici quelques éléments supplémentaires à considérer :

1.Définissez les en-têtes Cache-Control : Ces directives dans les en-têtes HTTP contrôlent la mise en cache dans les navigateurs et les caches partagés, réduisant les demandes multiples au serveur. Bien que cela n'affecte pas le temps de chargement initial, cela réduit considérablement les temps de chargement ultérieurs.

2.Optimisez les SVG : Les SVG sont évolutifs et généralement plus légers que les images matricielles. Cependant, certains SVG peuvent contenir des éléments inutilisés qui augmentent le temps de chargement. Utilisez des balises <img>pour les afficher, mais cela limite les possibilités de stylisation et d'animation. Si l'animation est essentielle, utilisez-les directement.

3.Utilisez WebP : Ce format d'image proposé par Google réduit la taille des fichiers avec une perte de qualité minimale. Les images WebP sont généralement 25 à 35 % plus petites que les JPEG tout en maintenant la même qualité. WebP prend également en charge les animations, comme les GIF.

4.Hébergez vos polices vous-même : En hébergeant les polices localement, le navigateur peut trouver '@font-face' directement dans le CSS de l'application, ce qui permet de télécharger les fichiers de police via la même connexion à votre serveur ou CDN, rendant le processus plus rapide et éliminant une récupération supplémentaire.