Конкурентный рендеринг

подход к рендерингу компонентов, позволяющий выполнять несколько задач одновременно, переключаясь между ними в зависимости от приоритетов и доступности ресурсов

Предполагает **Прерывание рендеринга** одного компонента, чтобы выполнить другой, более важный; **Приоритет задач**: обработка событий ввода данных пользователем, более приоритетная и выполняются немедленно, тогда как рендеринг результата поиска, могут быть отложены; **Отмена рендеров**: если данные или состояние меняются до завершения предыдущего рендера, React может отменить текущий рендер и начать новый с учетом новых данных

Преимущества: Отзывчивость, Плавность, Управление ресурсами

---

useTransition

Хук устанавливает приоритеты рендеринга, разделяя обновления на критические и фоновые. Оборачивает дорогостоящую функцию action (у функции должен быть суффикс action) в **startTransition,** тем самым задавая этой функции низкий приоритет

const [isPending, startTransiton] = useTransition()

Особенность startTransition:

Требуется обернуть все set функции после await в дополнительный startTransition

startTransition синхронный, setTimeout внутри него не работает, нужно наоборот startTransition внутри setTimeout

если внутри action нет set, нужно использовать useDeferrredValue

startTransition всегда остаётся одной и той же между рендерами

startTransition прерывается другими set

startTransition нельзя использовать для управления text input

startTransition объединяются (batch) если их несколько (при обработке нескольких событий, н.п. клик)

startTransition не блокирует интерфейс, поэтому не использует suspense fallback, также можно использовать useOptimistic

startTransition в навигации:

Можно прервать переход если пользователь передумал

Не показывает fallback индикатор загрузки, у пользователя ощущение быстрой реакции на его действия

StartTransition завершить все side effects до перехода на новый роут

Сценарий: сетевая загрузка данных, фильтрация, перерисовка карт, сложные и длительные вычисления

Плюсы: Управление приоритетом, фоновое выполнение, не отменяет задачу, подходит для задач которым необходимо отображать состояние перехода isPending

---

useDeferredValue

Хук, который позволяет создавать отложенную копию состояния. Полезно для улучшения плавности интерфейса при частых изменениях состояния

const deferredValue = useDeferredValue(value);

Особенность:

Внутри startTransition useDeferredValue бесполезен, так как action уже оптимизирован (отложен)

useDeferredValue принимает примитивы или объекты созданные вне цикла отрисовки компонента иначе это вызовет ненужный рендер

useDeferredValue сравнивает value с помощью Object.is, не сразу перерисовывает компонент, а планирует фоновый процесс, позволяя завершиться другим пользовательским событиями, если value изменилось запланированный фоновый процесс перерисовки отменяется и планируется новый. Т.о. рендеринг начнется, когда пользователь перестанет генерировать события

useDeferredValue не будет использовать Suspense fallback, а покажет старое значение

useDeferredValue не предоставляет механизма для контроля или предотвращения дублирующих запросов, (как получиться)

Индикатор deferredValue === value нужно делать самому

Для оптимизации используй memo

Сценарий: фильтрация, перерисовка карт, другие операции требующие плавности и способные терпеть небольшие задержки.

В отличие от Debounce не блокирует операции а устанавливает приоритеты, также Debounce используется для сетевых запросов, а useDeferredValue для операций с низким приоритетом относительно интерфейса

Плюсы:

Помогает уменьшить мерцания и другие визуальные артефакты, возникающие при частых изменениях состояния.

Позволяет избежать частых перерисовок. Ленивый рендеринг