podlodka

React Crew

10 – 14 февраля

Roadmap для React-разработчика

Алекс Рассудихин

Middle Frontend Developer, iDoctor

Доклад: «Долгожданный» React 19

«Долгожданный» React 19

ОТ АЛЕКСА РАССУДИХИНА

Podlodca React Crew #2

Немного обо мне

Рассудихин Алекс

Frontend Developer at iDoctor

Я Алекс, Milord Middle Frontend Developer. Люблю интересные задачи, печеньки и JavaScript 👋.

Временами пишу в Доку и веду свой блог.

Важный дисклеймер обо мне

- Имею большой багаж опыта на Vue.js, Nuxt;
- У меня нет ностальгии по старому React, пришёл недавно и сравниваю React больше с Vue, чем с другими версиями React.

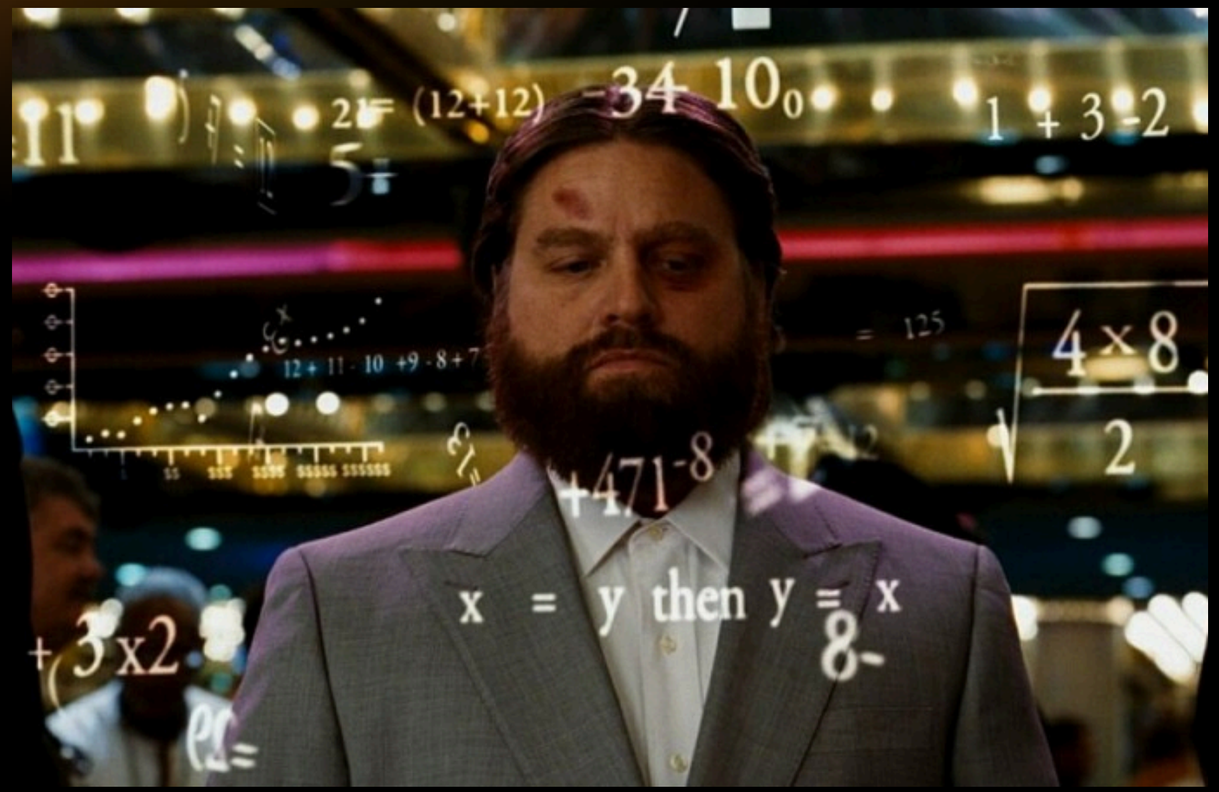
Что вас ждёт в докладе?

- Поговорим о нововведения в React 19:
  - Actions
  - Новые хуки;
  - use;
  - Предзагрузка ресурсов;
  - Мета-теги в компонентах;
  - Приятные мелочи;
- Подведём итоги о том, что мы видим в React 19;
- Поговорим о том, что нам ждать в React 20;

Поговорим о ACTIONS

Проблема

- Обработка асинхронной работы при отправке форм;
- Самостоятельная слежка за состояниями ошибок, загрузок и состояниями данных формы.



Когда пытаешься понять все эти 743 useState

Решение

actions - общее название для асинхронных функций, согласно конвенции создателей React. Ссылка;
Они:

- следят за состоянием загрузки;
- следят за оптимистичными обновлениями состояния;
- помогают обрабатывать ошибки форм;
- работают напрямую с html тегом <form>

Новые

ХУКИ

Что по новым хукам?

- useOptimistic (experimental 18.x);
- useFormStatus (experimental 18.x);
- useActionState (experimental 18.x).

Э-Э-ЭКСПЕРИМЕНТЫ!!

useActionState

- Позволяет обновить состояния ошибки и данных на основе работы с формой;
- Следит за состоянием загрузки.

useActionState

Принимает: асинхронную функцию и начальное состояние;
Возвращает: актуальное состояние, функцию для формы и состояние загрузки.

```
const [state, action, pending] = useActionState(fn, initialState);
```

useFormStatus

Вспомогательный хук, который можно использовать во вложенных компонентах, для получения статуса работы формы.

ДОЛОЙ ПРОПСЫ!

useFormStatus

Возвращает:

- pending - состояние загрузки формы;
- data - данные формы в формате FormData;
- method - метод form, указанный в html;
- action - функция, переданная в <form action={} />

```
const { pending, data, method, action } = useFormStatus();
```

А что с TS и валидацией?

- Совместим с Zod.
- С типизацией порядок, за исключением использования `as`

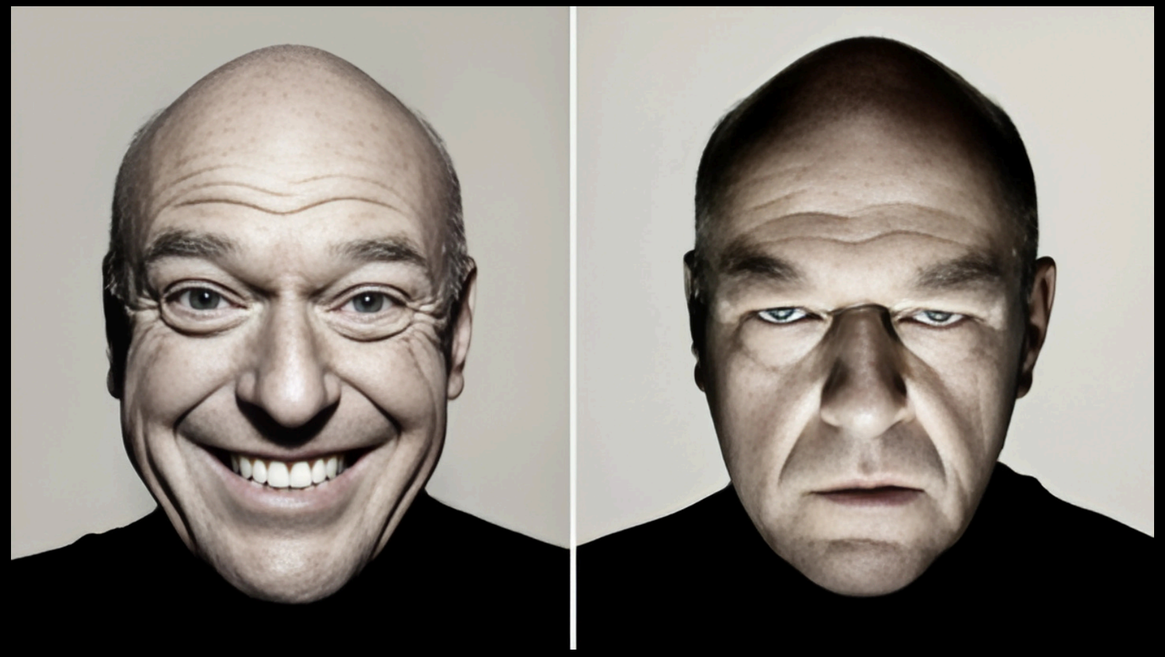
useOptimistic

Что он делает?

- Позволяет «оптимистично» обновить состояние;
- Нужен преимущественно для ситуаций, когда мы 100% уверены в хорошем ответе от сервера.

Стоит помнить:

- Обработка ошибок.



useOptimistic

Принимает:

- Ссылку на реальный state;
- Функция обновления оптимистичного state;

Возвращает:

- Оптимистичный state;
- Обёртка над функций для обновления оптимистичного state.

```
const [optimisticState, addOptimistic] = useOptimistic(state, updateFn)
```

А что с TS?

Задействовано всего два типа:

- State - объект, с которым работаем.
- Action - принимаемое значение в reducer.

```
export function useOptimistic<State, Action>(
  passthrough: State,
  reducer: (state: State, action: Action) => State,
): [State, (action: Action) => void];
```

А с Zod?

Два способа:

- Можно провалидировать поле по нему до вызова addOptimistic;
- Можно прямо внутри функции addOptimistic и вместе с установкой значения, ещё и установить поле `error`.

Итоги по новым хукам

- Упрощается работа с формами;
- Жизнь разработчиков библиотек также упрощается;
- Типизировать можно, но есть одно но (`as`);
- Не конфликтует с валидациями, будь то кастомные или Zod.

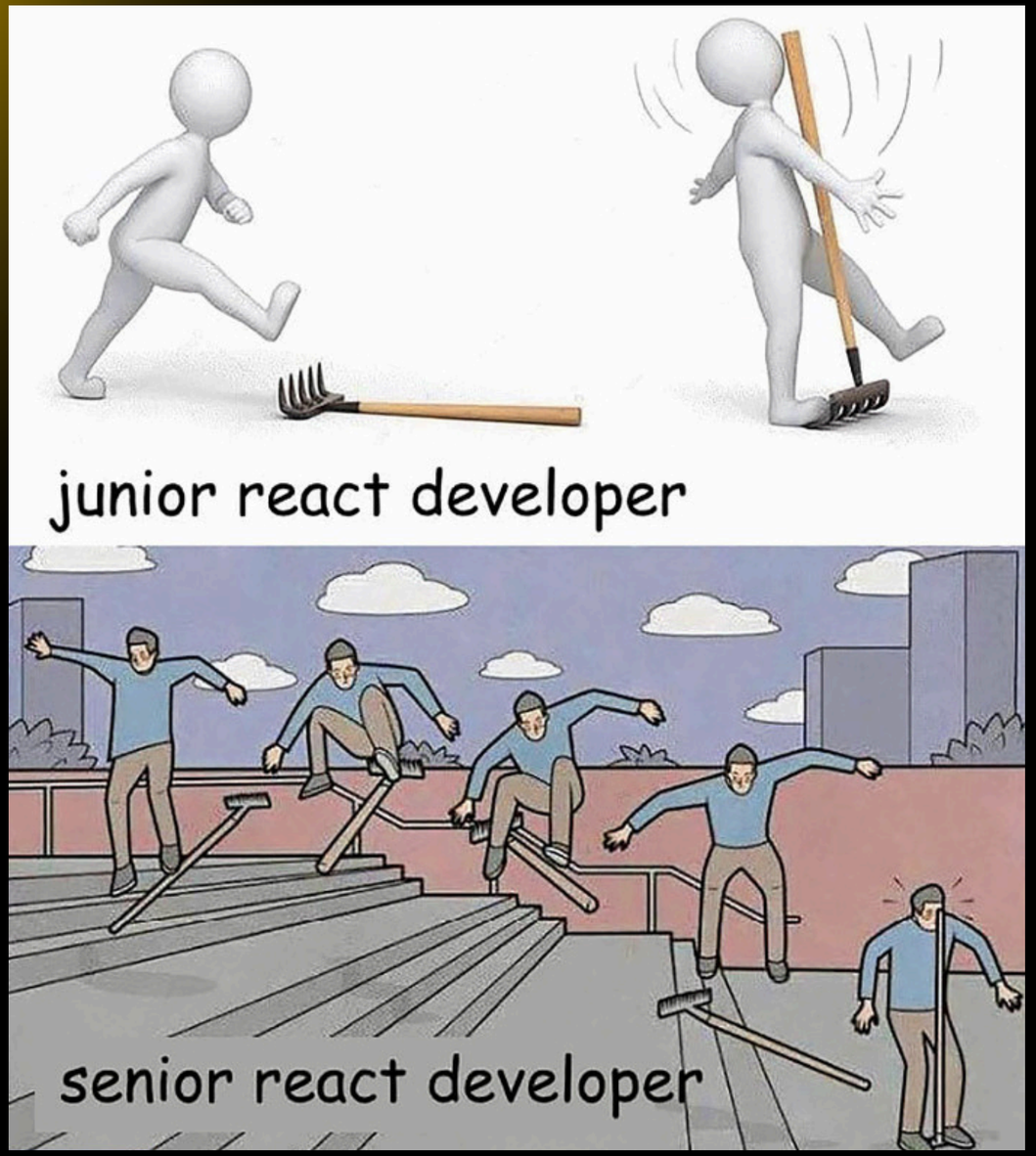
А что там с библиотеками?

Стало проще без библиотек в:

- Небольшие проекты;
- Не особо сложные формы;

Бонус:

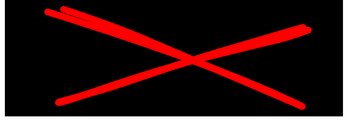
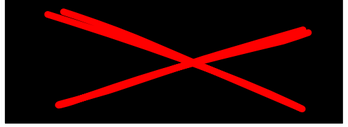
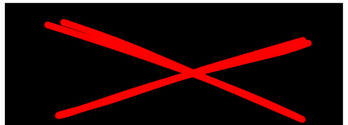
- Новые хуки помощь в разработке библиотек, т.к. на их основе можно реализовать что-то новое.


junior react developer
senior react developer

ХУК ИЛИ

НЕ ХУК

Почему не хук?

	Хуки	Use
Вызов в циклах, условных конструкциях		
Нужно соблюдать порядок вызова		
Применение вне хуков и FC		

use

use - это функция для чтения промисов или контекста:

- Его можно вызывать после раннего `return null`;
- Можно получать значения промисов и контекста внутри условий.

А что с типами?

```
export type Usable<T> = PromiseLike<T> | Context<T>;
export function use<T>(usable: Usable<T>): T;
```

PRELOAD РЕСУРСОВ

Функции для preload-а ресурсов

Используются для ускорения работы приложения путём предварительной загрузки ресурсов, будь то скрипты, шрифты или стили. Функции:

- preinit;
- preload;
- prefetchDNS;
- preconnect.

Я - СКОРОСТЬ!

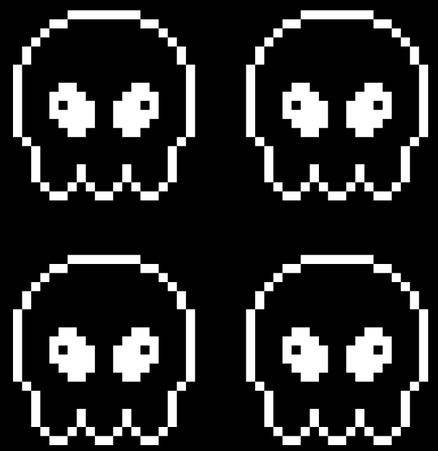
Как использовать?

```
function MyComponent() {
  preinit('https://.../path/script.js', { as: 'script' })
  preload('https://.../path/font.woff', { as: 'font' })
  preload('https://.../path/style.css', { as: 'style' })
  prefetchDNS('https://...')
  preconnect('https://...')
}
```

Meta-данные

Метаданные в компонентах

```
function BlogPost({post}) {
  return (
    <article>
      <h1>{post.title}</h1>
      <title>{post.title}</title>
      <meta name="author" content="Josh" />
      <link rel="author" href="https://twitter.com/joshcstory/" />
      <meta name="keywords" content={post.keywords} />
      <p>
        Eee equals em-see-squared...
      </p>
    </article>
  )
}
```



Не доработали, получается?

React Helmet не дублирует метатеги.

```
export const MetaTags = () => {
  return (
    <div>
      <title>MetaTags title 1</title>
      <title>MetaTags title 2</title>
      <title>MetaTags title 3</title>
    </div>
  );
};
```

```
<title>MetaTags title 1</title>
<title>MetaTags title 2</title>
<title>MetaTags title 3</title>
```

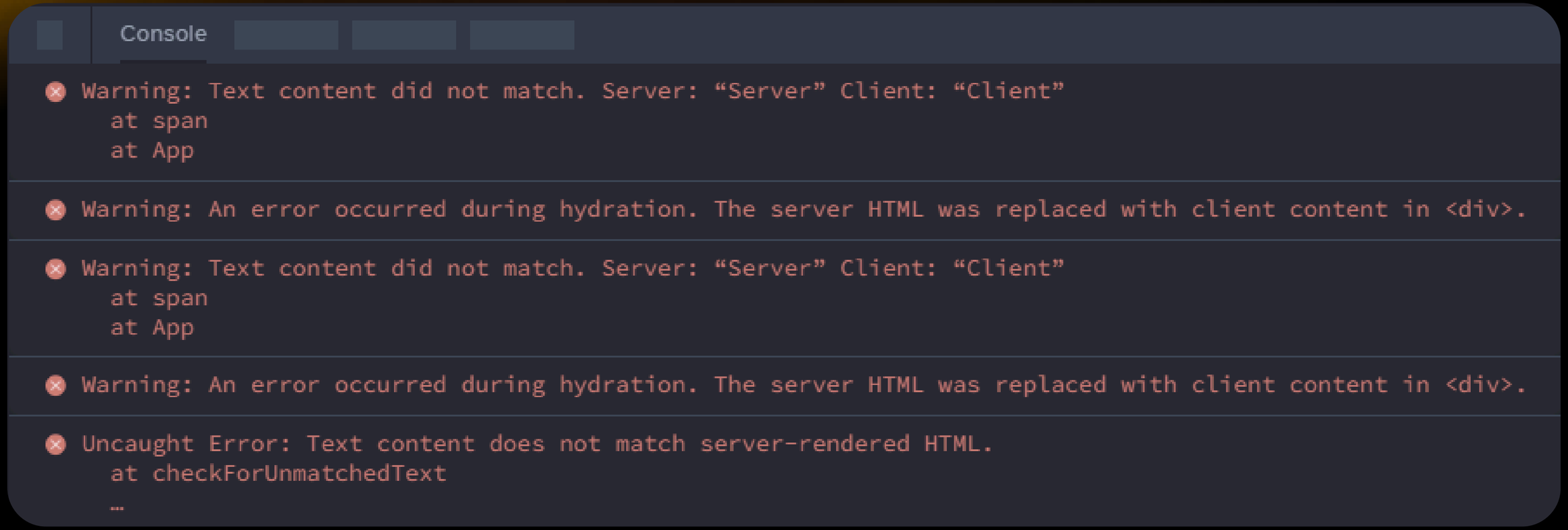
А что с читаемостью?

React Helmet куда лучше читается.

```
export const MetaTags = () => {
  return (
    <Helmet>
      <title>MetaTags title 1</title>
      <title>MetaTags title 2</title>
      <title>MetaTags title 3</title>
    </Helmet>
  );
};
```

Ошибки

Гидрация. Было.



Гидрация. Стало.

Console

Uncaught Error: Hydration failed because the server rendered HTML didn't match the client. As a result this tree will be regenerated on the client. This can happen if an SSR-ed Client Component used:

- A server/client branch `if (typeof window !== 'undefined')`.
- Variable input such as `Date.now()` or `Math.random()` which changes each time it's called.
- Date formatting in a user's locale which doesn't match the server.
- External changing data without sending a snapshot of it along with the HTML.
- Invalid HTML tag nesting.

It can also happen if the client has a browser extension installed which messes with the HTML before React loaded.

https://react.dev/link/hydration-mismatch

```
  <App>
    <span>
+     Client
-     Server
```

Остальные ошибки. Было.

Console

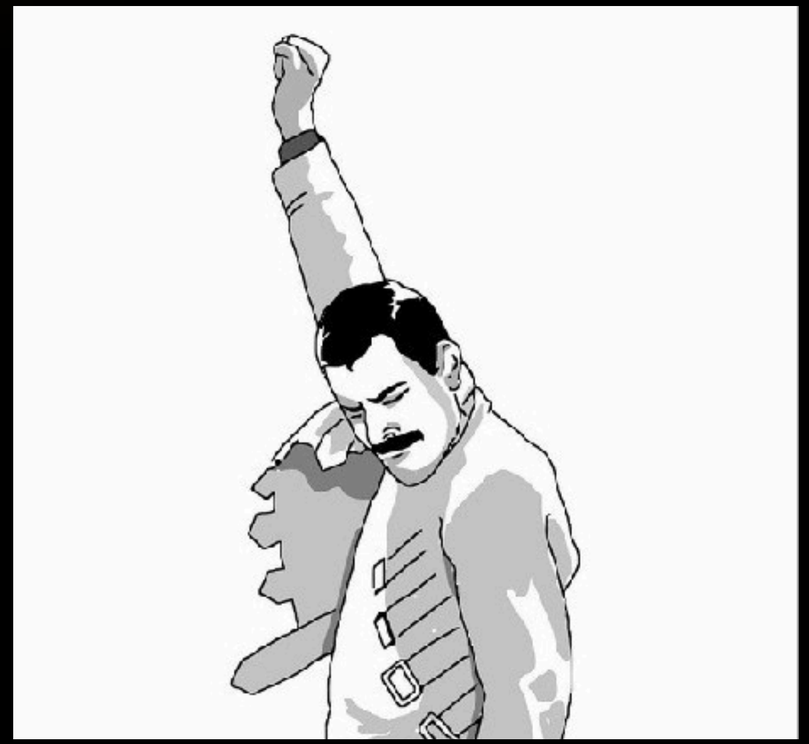
```
Uncaught Error: hit
    at Throws
    at renderWithHooks
    …

Uncaught Error: hit      <-- Duplicate
    at Throws
    at renderWithHooks
    …

The above error occurred in the Throws component:
    at Throws
    at ErrorBoundary
    at App

  React will try to recreate this component tree from scratch using the error boundary you provided,
  ErrorBoundary.
```

Остальные ошибки. Стало.

Небольшие радости

Мелкие улучшения

- ref стал полноценным пропсом без forwardRef;
- <Context.Provider> стал просто <Context>



Server components

Серверные компоненты

Они наконец-то stable, хотя ими все пользовались и без этого в Next.js 14.

Подробнее у Дэниза Демирсой в докладе «Next (зачеркнуто) React Server Components»

Переходим к

заключению

О хорошем

- React помогает нам работать с формами;
- Улучшили ошибки;
- Стабилизировали промежуточный React 18, потому и «долгожданный».

О плохом

- А зачем вообще были мета-данные?
- `as` в useActionState.

Вангую
за React 20

Экспериментальные фичи

- taintObjectReference - не передаём с сервера на клиент;
- taintUniqueValue - не передаём уникальные значения с сервера на клиент (токены, ключи и т.д.).

Мои мысли

- Надеюсь, убьют уже React Helmet;
- Продолжат замещать другие библиотеки;
- Внедрение View Transition API.

Спасибо за внимание!

webdev.kz