Оптимизация переименования ключа в объекте JavaScript

Быстрый ответ

Для **переименования ключа** в объекте JavaScript, вам потребуется **присвоить значение** новому ключу и впоследствии **удалить старый**:

JS Скопировать код

```
let obj = { oldKey: 'value' };
// В программировании порой происходят настоящие чудеса, пра
obj.newKey = obj.oldKey;
delete obj.oldKey;
```

Если вы следуете принципу **неизменности данных**, вы можете перенести свойства в **новый объект**:

JS Скопировать код

```
let newObj = { ...obj, newKey: obj.oldKey };
// Вуаля, старый ключ будто испарился! 🕶
delete newObj.oldKey;
```

Методы и стратегии переименования ключей

Сохранение атрибутов: применение `Object.defineProperty`

Если вы хотите сохранить атрибуты старого ключа, метод `Object.defineProperty` поможет осуществить переименование незаметно:

JS Скопировать код

```
Object.defineProperty(obj, 'newKey', Object.getOwnPropertyDe
delete obj.oldKey;
```

Избегание бессмысленных действий: не переименовываем ключ в себя самого

Нет смысла переименовывать ключ в то же значение, что у него было. Это зря потраченные ресурсы, как если бы собака без конца гонялась за своим хвостом. 

JS Скопировать код

```
if (obj.hasOwnProperty('oldKey') && 'oldKey' !== 'newKey') {
  Object.defineProperty(obj, 'newKey', Object.getOwnProperty
  delete obj.oldKey;
}
```

Универсальность: шаг к модульному коду

Функция для переименования ключей

Чтобы облегчить процесс, вы можете создать **функцию**, которая будет переименовывать ключи "на лету":

JS Скопировать код

```
let renameKey = (obj, oldKey, newKey) => {
  obj[newKey] = obj[oldKey];
  delete obj[oldKey];
  return obj;
};
```

Цепочка вызовов: потоковый интерфейс

Потоковый интерфейс позволяет выполнять последовательное переименование, что является удобным и простым, словно вы раздаете заказы на автомойке:

JS Скопировать код

```
function FluentRenamer(obj) {
  this.obj = { ...obj };
}
FluentRenamer.prototype.rename = function(oldKey, newKey) {
  if (this.obj.hasOwnProperty(oldKey)) {
    Object.defineProperty(this.obj, newKey,
    Object.getOwnPropertyDescriptor(this.obj, oldKey));
    delete this.obj[oldKey];
  }
  return this;
};
```

Применение ES6 для переименования ключей

Однострочное переименование через деструктуризацию

Мы применяем **деструктуризацию** и оператор **остатка** для элегантного решения:

JS Скопировать код

```
// Словно фокус: ключ исчезает и появляется снова!
const { oldKey, ...rest } = obj;
obj = { newKey: oldKey, ...rest };
```

Работа с неизменяемыми структурами данных

Создание объекта с новыми ключами

Познакомьтесь с принципами функционального программирования при помощи `Object.entries` и `Array.prototype.map`:

JS Скопировать код

```
let renamedObj = Object.fromEntries(
  Object.entries(obj).map(([key, value]) =>
    key === 'oldKey' ? ['newKey', value] : [key, value])
);
```

Визуализация

Представьте, что наш **объект** — это **шкаф** с ярлыками на дверцах. Мы просто меняем ярлыки местами:
Было: [🏷️ "oldKey": 📦, 🏷️ "otherKey": 📂]
Меняем "oldKey" на "newKey":

JS Скопировать код

```
let obj = { oldKey: "value" };
// Простое переклеивание ярлыков!
obj["newKey"] = obj["oldKey"];
delete obj["oldKey"];
```

Стало: [🏷️ "newKey": 📦, 🏷️ "otherKey": 📂]
Мы не затрагивали содержимое, только заменили метку.