

Раздел

Объекты: основы

Навигация по уроку

Проблема
«несуществующего
свойства»

Опциональная цепочка

Сокращённое вычисление

Другие варианты
применения: ?.(), ?[]

Итого

Комментарии


Поделиться



Редактировать на GitHub

 → Язык программирования JavaScript

→ Объекты: основы

 30-го ноября 2020

Опциональная цепочка '?!'

Новая возможность

Эта возможность была добавлена в язык недавно. В старых браузерах может понадобиться полифил.

Опциональная цепочка `?.` — это безопасный способ доступа к свойствам вложенных объектов, даже если какое-либо из промежуточных свойств не существует.

Проблема «несуществующего свойства»

Если вы только начали читать учебник и изучать JavaScript, то, возможно, эта проблема вам пока незнакома, но она достаточно распространена.

Например, рассмотрим объекты для пользователей `user`. У большинства пользователей есть адрес `user.address` с улицей `user.address.street`, но некоторые адрес не указали.

В этом случае при попытке получить свойство `user.address.street` будет ошибка:

```
1 let user = {}; // пользователь без свойства address
2
3 alert(user.address.street); // ошибка!
```

Это нормальный результат, так работает JavaScript, но во многих реальных ситуациях удобнее было бы получать не ошибку, а просто `undefined` («нет улицы»).

Или ещё пример. В веб-разработке нам бывает нужно получить данные об HTML-элементе, который иногда может отсутствовать на странице:

```
1 // Произойдёт ошибка, если querySelector(...) равен null
2 let html = document.querySelector('.my-element').innerHTML
```

До появления `?.` в языке для решения подобных проблем использовался оператор `&&`.

Например:

```
1 let user = {}; // пользователь без адреса
2
3 alert( user && user.address && user.address.street ); /
```

Использование логического И со всей цепочкой свойств гарантирует, что все они существуют (а если нет — вычисление прекращается), но это довольно длинная и громоздкая конструкция.

Опциональная цепочка

Опциональная цепочка `?.` останавливает вычисление и возвращает `undefined`, если часть перед `?.` имеет значение `undefined` или `null`.

Для краткости в этой статье мы будем говорить о значении, что оно «существует», если оно отличается от `null` или `undefined`.

Раздел

Объекты: основы

Навигация по уроку

Проблема
«несуществующего
свойства»

Опциональная цепочка

Сокращённое вычисление

Другие варианты
применения: ?.(), ?[]

Итого

Комментарии

Поделиться



Редактировать на GitHub

Вот безопасный способ обратиться к свойству `user.address.street`:

```
1 let user = {}; // пользователь без адреса
2
3 alert( user?.address?.street ); // undefined (без ошибок)
```

Чтение адреса с помощью конструкции `user?.address` выполняется без ошибок, даже если объекта `user` не существует:

```
1 let user = null;
2
3 alert( user?.address ); // undefined
4 alert( user?.address.street ); // undefined
```

Обратите внимание, что синтаксис `?.` делает необязательным только свойство перед ним, а не какое-либо последующее.

В приведённом выше примере конструкция `user?.` допускает, что переменная `user` может содержать `null/undefined`.

С другой стороны, если объект `user` существует, то в нём должно быть свойство `user.address`, иначе выполнение `user?.address.street` вызовет ошибку из-за второй точки.

⚠ Не злоупотребляйте опциональной цепочкой

Используйте `?.` только тогда, когда допускаете ситуацию, что значение перед ним не существует.

Например, если по нашей логике объект `user` точно существует, но его свойство `address` является необязательным, то предпочтительнее использовать следующую конструкцию: `user.address?.street`.

Тогда если переменная `user` по ошибке окажется пустой, мы увидим программную ошибку и исправим это.

⚠ Переменная перед `?.` должна быть объявлена

Если переменной `user` вообще не существует, то выражение `user?.anything` выдаст ошибку:

```
1 // ReferenceError: user is not defined
2 user?.address;
```

Объявление переменной (например `let/const/var user`) обязательно должно быть. Опциональная цепочка работает только с существующими переменными.

Сокращённое вычисление

Как уже говорилось, `?.` немедленно останавливает вычисление, если левой части не существует.

Таким образом, последующие вызовы функций или операции не будут выполнены.

Например:

```
1 let user = null;
2 let x = 0;
3
4 user?.sayHi(x++); // нет user, поэтому до x++ вычисления
5
```

Раздел

[Объекты: основы](#)

Навигация по уроку

Проблема
«несуществующего
свойства»

Опциональная цепочка

Сокращённое вычисление

Другие варианты
применения: ?.(), ?.[]

Итого

Комментарии

Поделиться

[Редактировать на GitHub](#)

Другие варианты применения: ?.(), ?.[]

Опциональная цепочка `?.` — это не оператор, а специальная синтаксическая конструкция, которая также работает с функциями и квадратными скобками.

Например, `?.()` используется для вызова потенциально несуществующей функции.

В следующем примере не у всех пользователей есть метод `admin`:

```

1 let user1 = {
2   admin() {
3     alert("Я администратор");
4   }
5 }
6
7 let user2 = {};
8
9 user1.admin?.(); // Я администратор
10 user2.admin?.();

```

В обоих вызовах сначала используем точку (`user1.admin`), чтобы получить свойство `admin`, потому что объект пользователя точно существует, к нему можно обратиться без какой-либо ошибки.

Затем уже `?.()` проверяет левую часть: если функция `admin` существует, то она выполнится (это так для `user1`). Иначе (для `user2`) вычисление остановится без ошибок.

Также существует синтаксис `?.[]`, если значение свойства требуется получить с помощью квадратных скобок `[]`, а не через точку `.`. Как и в остальных случаях, такой способ позволяет защититься от ошибок при доступе к свойству объекта, которого может не быть.

```

1 let user1 = {
2   firstName: "Иван"
3 };
4
5 let user2 = null; // Представим, что пользователь не ав
6
7 let key = "firstName";
8
9 alert( user1?.[key] ); // Иван
10 alert( user2?.[key] ); // undefined
11
12 alert( user1?.[key]?.something?.not?.existing ); // unde

```

Кроме этого, `?.` можно совместно использовать с `delete`:

```

1 delete user?.name; // Удалить user.name, если пользоват

```

Раздел

Объекты: основы

Навигация по уроку

Проблема
«несуществующего
свойства»

Опциональная цепочка

Сокращённое вычисление

Другие варианты
применения: `?()`, `?[]`

Итого

Комментарии

Поделиться



Редактировать на GitHub



⚠ Можно использовать `?.` для безопасного чтения и удаления, но не для записи

Опциональная цепочка `?.` не имеет смысла в левой части присваивания.

Например:

```
1 let user;
2
3 user?.name = "John"; // Ошибка, это не сработает
4 // это по сути то же самое что undefined = "John"
```

Она недостаточно «умна» для этого.

Итого

Синтаксис опциональной цепочки `?.` имеет три формы:

1. `obj?.prop` – возвращает `obj.prop`, если существует `obj`, и `undefined` в противном случае.
2. `obj?.[prop]` – возвращает `obj[prop]`, если существует `obj`, и `undefined` в противном случае.
3. `obj.method?.()` – вызывает `obj.method()`, если существует `obj.method`, в противном случае возвращает `undefined`.

Как мы видим, все они просты и понятны в использовании. `?.` проверяет левую часть выражения на равенство `null/undefined`, и продолжает дальнейшее вычисление, только если это не так.

Цепочка `?.` позволяет без возникновения ошибок обратиться к вложенным свойствам.

Тем не менее, нужно разумно использовать `?.` — только там, где это уместно, если допустимо, что левая часть не существует. Чтобы таким образом не скрывать возможные ошибки программирования.

Проводим [курсы по JavaScript и фреймворкам](#). ✕

💬 Комментарии

перед тем как писать...