Наборы и диапазоны [...]

Несколько символов или символьных классов в квадратных скобках `[…]` означают «искать любой символ из заданных».

Наборы

Для примера, `[eao]` означает любой из 3-х символов: `'a'`, `'e'` или `'o'`.

Это называется *набором*.

Наборы могут использоваться в регулярных выражениях вместе с обычными символами, например:

```
// найти [т или х], после которых идёт "оп"
alert( "Топ хоп".match(/[тх]оп/gi) ); // "топ", "хоп"
```

Обратите внимание, что в наборе несколько символов, но в результате он соответствует ровно одному символу.

Так что этот пример не даёт совпадений:

```
alert( "Вуаля".match(/В[уа]ля/) ); // null, нет совпаде
// ищет "В", затем [у или а], потом "ля"
// а в строке В, потом у, потом а
```

Шаблон ищет:

- `В`,
- затем *один* из символов `[уа]`,
- потом `ля`.

В этом случае совпадениями могут быть `Вуля` или `Валя`.

Диапазоны

Ещё квадратные скобки могут содержать *диапазоны символов*.

К примеру, `[a-z]` соответствует символу в диапазоне от `a` до `z`, или `[0-5]` – цифра от `0` до `5`.

В приведённом ниже примере мы ищем `"x"`, за которым следуют две цифры или буквы от `A` до `F`:

```
alert( "Exception 0xAF".match(/x[0-9A-F][0-9A-F]/g) );
```

Здесь в `[0-9A-F]` сразу два диапазона: ищется символ, который либо цифра от `0` до `9`, либо буква от `A` до `F`.

Если мы хотим найти буквы и в верхнем и в нижнем регистре, то мы можем добавить ещё диапазон `a-f`: `[0-9A-Fa-f]`. Или поставить у регулярного выражения флаг `i`.

Также мы можем использовать символьные классы внутри `[…]`.

Например, если мы хотим найти «символ слова» `\w` или дефис `-`, то набор будет: `[\w-]`.

Можем использовать и несколько классов вместе, например `[\s\d]` означает «пробельный символ или цифра».

Символьные классы – сокращения для наборов символов

Символьные классы – не более чем сокращение для наборов символов.

Например:

- **\d** – то же самое, что и `[0-9]`,
- **\w** – то же самое, что и `[a-zA-Z0-9_]`,
- **\s** – то же самое, что и `[\t\n\v\f\r ]`, плюс несколько редких пробельных символов Юникода.

Пример: многоязычный аналог \w

Так как символьный класс `\w` является всего лишь сокращением для `[a-zA-Z0-9_]`, он не найдёт китайские иероглифы, кириллические буквы и т.п.

Давайте сделаем более универсальный шаблон, который ищет символы, используемые в словах, для любого языка. Это очень легко с Юникод-свойствами: `[\p{Alpha}\p{M}\p{Nd}\p{Pc}\p{Join_C}]`.

Расшифруем его. По аналогии с классом `\w`, мы делаем свой набор, который включает в себя символы со следующими юникодными свойствами:

- `Alphabetic (Alpha)` – для букв,
- `Mark (M)` – для акцентов,
- `Decimal_Number (Nd)` – для цифр,
- `Connector_Punctuation (Pc)` – для символа подчёркивания `'_'` и подобных ему,
- `Join_Control (Join_C)` – два специальных кода `200c` и `200d`, используемые в лигатурах, например, арабских.

Пример использования:

```
let regexp = /[\p{Alpha}\p{M}\p{Nd}\p{Pc}\p{Join_C}]/gu

let str = `Hi 你好 12`;

// найдены все буквы и цифры
alert( str.match(regexp) ); // H,i,你,好,1,2
```

Конечно, этот шаблон можно адаптировать: добавить юникодные свойства или убрать. Более подробно о них было рассказано в главе Юникод: флаг "u" и класс \p{...}.

Юникодные свойства не работают в некоторых старых браузерах

Поддержка юникодных свойств `p{…}` была добавлена в Edge и Firefox относительно недавно. Если нужно реализовать поддержку `p{…}` для устаревших версий этих браузеров, можно использовать библиотеку XRegExp.

Или же использовать диапазоны символов в интересующем нас языке, например `[а-я]` для кириллицы.

Исключающие диапазоны

Помимо обычных диапазонов, есть «исключающие» диапазоны, которые выглядят как `[^…]`.

Они обозначаются символом каретки `^` в начале диапазона и соответствуют любому символу *за исключением заданных*.

Например:

- `[^aeyo]` – любой символ, за исключением `'a'`, `'e'`, `'y'` или `'o'`.

- `[^0-9]` – любой символ, за исключением цифры, то же, что и `\D`.
- `[^\s]` – любой непробельный символ, то же, что и `\S`.

Пример ниже ищет любые символы, кроме латинских букв, цифр и пробелов:

```
alert( "alice15@gmail.com".match(/[^\d\sA-Z]/gi) ); // @ .
```

Экранирование внутри [...]

Обычно, когда мы хотим найти специальный символ, нам нужно экранировать его, например `\.`. А если нам нужна обратная косая черта, тогда используем `\\`, т.п.

В квадратных скобках большинство специальных символов можно использовать без экранирования:

- Символы `. + ( )` не нужно экранировать никогда.
- Тире `-` не надо экранировать в начале или в конце (где оно не задаёт диапазон).
- Символ каретки `^` нужно экранировать только в начале (где он означает исключение).
- Закрывающую квадратную скобку `]`, если нужен именно такой символ, экранировать нужно.

Другими словами, разрешены без экранирования все специальные символы, кроме случаев, когда они означают что-то особое в наборах.

Точка `.` внутри квадратных скобок – просто точка. Шаблон `[.,]` будет искать один из символов: точку или запятую.

В приведённом ниже примере регулярное выражение `[-().^+]` ищет один из символов `-().^+`:

```
// Нет необходимости в экранировании
let regexp = /[-().^+]/g;

alert( "1 + 2 - 3".match(regexp) ); // Совпадения +, -
```

...Впрочем, если вы решите экранировать «на всякий случай», то не будет никакого вреда:

```
// Экранирование всех возможных символов
let regexp = /[\-\(\)\.\^\+]/g;

alert( "1 + 2 - 3".match(regexp) ); // также работает:
```

Наборы и флаг «u»

Если в наборе есть суррогатные пары, для корректной работы обязательно нужен флаг `u`.

Например, давайте попробуем найти шаблон `[𝒳𝒴]` в строке `𝒳`:

```
alert( '𝒳'.match(/[𝒳𝒴]/) ); // покажет странный символ,
// (поиск был произведён неправильно, вернулась только
```

Результат неверный, потому что по умолчанию регулярные выражения «не знают» о существовании суррогатных пар.

Движок регулярных выражений думает, что `[𝒳𝒴]` – это не два, а четыре символа:

1. левая половина от `𝒳` (1),
2. правая половина от `𝒳` (2),
3. левая половина от `𝒴` (3),

4. правая половина от 𝒳 (4).

Мы даже можем вывести их коды:

```
1  for(let i=0; i<'𝒳𝒴'.length; i++) {
2    alert('𝒳𝒴'.charCodeAt(i)); // 55349, 56499, 55349, 56
3  };
```

То есть в нашем примере выше ищется и выводится только левая половина от 𝒳.

Если добавить флаг `u`, то всё будет в порядке:

```
1  alert( '𝒳'.match(/[𝒳𝒴]/u) ); // 𝒳
```

Аналогичная ситуация произойдёт при попытке искать диапазон: `[𝒳-𝒵]`.

Если мы забудем флаг `u`, то можем нечаянно получить ошибку:

```
1  '𝒳'.match(/[𝒳-𝒵]/); // Error: Invalid regular expressio
```

Причина в том, что без флага `u` суррогатные пары воспринимаются как два символа, так что `[𝒳-𝒵]` воспринимается как `[<55349><56499>-<55349><56500>]` (каждая суррогатная пара заменена на коды). Теперь уже отлично видно, что диапазон `56499-55349` некорректен: его левая граница больше правой, это и есть формальная причина ошибки.

При использовании флага `u` шаблон будет работать правильно:

```
1  // поищем символы от 𝒳 до 𝒵
2  alert( '𝒳'.match(/[𝒳-𝒵]/u) ); // 𝒳
```

Задачи

Java[^script]

У нас есть регулярное выражение `/Java[^script]/`.

Найдёт ли оно что-нибудь в строке `Java`? А в строке `JavaScript`?

решение

Найдите время как hh:mm или hh-mm

Время может быть в формате `часы:минуты` или `часы-минуты`. И часы, и минуты имеют две цифры: `09:00` или `21-30`.

Напишите регулярное выражение, чтобы найти время:

```
1  let regexp = /your regexp/g;
2  alert( "Завтрак в 09:00. Ужин в 21-30".match(regexp) );
```

P.S. В этой задаче мы предполагаем, что время всегда правильное, нет необходимости отфильтровывать плохие строки, такие как «45:67». Позже мы разберёмся с этим.

решение

Комментарии

перед тем как писать...

Поделиться

Редактировать на GitHub