Intl: интернационализация в JavaScript

Общая проблема строк, дат, чисел в JavaScript – они «не в курсе» языка и особенностей стран, где находится посетитель.

В частности:

Строки

При сравнении сравниваются коды символов, а это неправильно, к примеру, в русском языке оказывается, что `"ё" > "я"` и `"а" > "Я"`, хотя всем известно, что `я` – последняя буква алфавита и это она должна быть больше любой другой.

Даты

В разных странах принята разная запись дат. Где-то пишут 31.12.2014 (Россия), а где-то 12/31/2014 (США), где-то иначе.

Числа

В одних странах выводятся цифрами, в других – иероглифами, длинные числа разделяются где-то пробелом, где-то запятой.

Все современные браузеры, кроме IE10 (но есть библиотеки и для него) поддерживают стандарт ECMA 402, предназначенный решить эти проблемы навсегда.

Основные объекты

`Intl.Collator`

Умеет правильно сравнивать и сортировать строки.

`Intl.DateTimeFormat`

Умеет форматировать дату и время в соответствии с нужным языком.

`Intl.NumberFormat`

Умеет форматировать числа в соответствии с нужным языком.

Локаль

Локаль – первый и самый важный аргумент всех методов, связанных с интернационализацией.

Локаль описывается строкой из трёх компонентов, которые разделяются дефисом:

1. Код языка.
2. Код способа записи.
3. Код страны.

На практике не всегда указаны три, обычно меньше:

1. `ru` – русский язык, без уточнений.
2. `en-GB` – английский язык, используемый в Англии (`GB`).
3. `en-US` – английский язык, используемый в США (`US`).
4. `zh-Hans-CN` – китайский язык (`zh`), записываемый упрощённой иероглифической письменностью (`Hans`), используемый в Китае.

Также через суффикс `-u-*` можно указать расширения локалей, например `"th-TH-u-nu-thai"` – тайский язык (`th`), используемый в Таиланде (`TH`), с записью чисел тайскими буквами (๐, ๑, ๒, ๓, ๔, ๕, ๖, ๗, ๘, ๙) .

Стандарт, который описывает локали – RFC 5464, языки описаны в IANA language registry.

Все методы принимают локаль в виде строки или массива, содержащего несколько локалей в порядке предпочтения.

Если локаль не указана или `undefined` – берётся локаль по умолчанию, установленная в окружении (браузере).

Подбор локали localeMatcher

`localeMatcher` – вспомогательная настройка, которую тоже можно везде указать, она определяет способ подбора локали, если желаемая недоступна.

У него два значения:

- `"lookup"` – означает простейший порядок поиска путём обрезания суффикса, например `zh-Hans-CN` → `zh-Hans` → `zh` → локаль по умолчанию.
- `"best fit"` – использует встроенные алгоритмы и предпочтения браузера (или другого окружения) для выбора подходящей локали.

По умолчанию стоит `"best fit"`.

Если локалей несколько, например `["zh-Hans-CN", "ru-RU"]` то `localeMatcher` пытается подобрать наиболее подходящую локаль для первой из списка (китайская), если не получается – переходит ко второй (русской) и так далее. Если ни одной не нашёл, например на компьютере не совсем поддерживается ни китайский ни русский, то используется локаль по умолчанию.

Как правило, `"best fit"` является здесь наилучшим выбором.

Строки, Intl.Collator

Синтаксис:

```
1  // создание
2  let collator = new Intl.Collator([locales, [options]])
```

Параметры:

`locales`

Локаль, одна или массив в порядке предпочтения.

`options`

Объект с дополнительными настройками:

- `localeMatcher` – алгоритм выбора подходящей локали.
- `usage` – цель сравнения: сортировка `"sort"` или поиск `"search"`, по умолчанию `"sort"`.
- `sensitivity` – чувствительность: какие различия в символах учитывать, а какие – нет, варианты:
  - `base` – учитывать только разные символы, без диакритических знаков и регистра, например: `а ≠ б`, `е = ё`, `а = А`.
  - `accent` – учитывать символы и диакритические знаки, например: `а ≠ б`, `е ≠ ё`, `а = А`.
  - `case` – учитывать символы и регистр, например: `а ≠ б`, `е = ё`, `а ≠ А`.
  - `variant` – учитывать всё: символ, диакритические знаки, регистр, например: `а ≠ б`, `е ≠ ё`, `а ≠ А`, используется по умолчанию.
- `ignorePunctuation` – игнорировать знаки пунктуации: `true/false`, по умолчанию `false`.
- `numeric` – использовать ли численное сравнение: `true/false`, если `true`, то будет `12 > 2`, иначе `12 < 2`.
- `caseFirst` – в сортировке должны идти первыми прописные или строчные буквы, варианты: `"upper"` (прописные), `"lower"` (строчные)

или `"false"` (стандартное для локали, также является значением по умолчанию). Не поддерживается IE11.

В подавляющем большинстве случаев подходят стандартные параметры, то есть `options` указывать не нужно.

Использование:

```
let result = collator.compare(str1, str2);
```

Результат `compare` имеет значение `1` (больше), `0` (равно) или `-1` (меньше).

Например:

```
let collator = new Intl.Collator();

alert( "ёжик" > "яблоко" ); // true (ёжик больше, что н
alert( collator.compare("ёжик", "яблоко") ); // -1 (ёжи
```

Выше были использованы полностью стандартные настройки. Они различают регистр символа, но это различие можно убрать, если настроить чувствительность `sensitivity`:

```
let collator = new Intl.Collator();
alert( collator.compare("ЁжиК", "ёжик") ); // 1, разные

let collator = new Intl.Collator(undefined, {
  sensitivity: "accent"
});
alert( collator.compare("ЁжиК", "ёжик") ); // 0, одинак
```

Даты, Intl.DateTimeFormat

Синтаксис:

```
// создание
let formatter = new Intl.DateTimeFormat([locales, [opti
```

Первый аргумент – такой же, как и в `Collator`, а в объекте `options` мы можем определить, какие именно части даты показывать (часы, месяц, год...) и в каком формате.

Полный список свойств `options`:

Свойство	Описание	Возможные значения	По умолчанию
`localeMatcher`	Алгоритм подбора локали	`lookup`, `best fit`	`best fit`
`formatMatcher`	Алгоритм подбора формата	`basic`, `best fit`	`best fit`
`hour12`	Включать ли время в 12-часовом формате	`true` – 12-часовой формат, `false` – 24-часовой	
`timeZone`	Временная зона	Временная зона, например `Europe/Moscow`	`UTC`
`weekday`	День недели	`narrow`, `short`, `long`	
`era`	Эра	`narrow`, `short`, `long`	
`year`	Год	`2-digit`, `numeric`	`undefined` или `numeric`

Свойство	Описание	Возможные значения	По умолчанию
`month`	Месяц	`2-digit`, `numeric`, `narrow`, `short`, `long`	`undefined` или `numeric`
`day`	День	`2-digit`, `numeric`	`undefined` или `numeric`
`hour`	Час	`2-digit`, `numeric`	
`minute`	Минуты	`2-digit`, `numeric`	
second	Секунды	`2-digit`, `numeric`	
`timeZoneName`	Название таймзоны (нет в IE11)	`short`, `long`	

Все локали обязаны поддерживать следующие наборы настроек:

- weekday, year, month, day, hour, minute, second
- weekday, year, month, day
- year, month, day
- year, month
- month, day
- hour, minute, second

Если указанный формат не поддерживается, то настройка `formatMatcher` задаёт алгоритм подбора наиболее близкого формата: `basic` – по стандартным правилам и `best fit` – по умолчанию, на усмотрение окружения (браузера).

Использование:

```
1  let dateString = formatter.format(date);
```

Например:

```
1  let date = new Date(2014, 11, 31, 12, 30, 0);
2
3  let formatter = new Intl.DateTimeFormat("ru");
4  alert( formatter.format(date) ); // 31.12.2014
5
6  let formatter = new Intl.DateTimeFormat("en-US");
7  alert( formatter.format(date) ); // 12/31/2014
```

Длинная дата, с настройками:

```
1  let date = new Date(2014, 11, 31, 12, 30, 0);
2
3  let formatter = new Intl.DateTimeFormat("ru", {
4    weekday: "long",
5    year: "numeric",
6    month: "long",
7    day: "numeric"
8  });
9
10 alert( formatter.format(date) ); // среда, 31 декабря 2
```

Только время:

```
1  let date = new Date(2014, 11, 31, 12, 30, 0);
2
3  let formatter = new Intl.DateTimeFormat("ru", {
4    hour: "numeric",
5    minute: "numeric",
```

```
  second: "numeric"
});

alert( formatter.format(date) ); // 12:30:00
```

Числа, Intl.NumberFormat

Форматтер `Intl.NumberFormat` умеет красиво форматировать не только числа, но и валюту, а также проценты.

Синтаксис:

```
let formatter = new Intl.NumberFormat([locales[, option

formatter.format(number); // форматирование
```

Параметры, как и раньше – локаль и опции.

Список опций:

Свойство	Описание	Возможные значения	По умолч
`localeMatcher`	Алгоритм подбора локали	`lookup`, `best fit`	`best fi`
`style`	Стиль форматирования	`decimal`, `percent`, `currency`	`decimal`
`currency`	Алфавитный код валюты	См. Список кодов валюты, например `USD`	
`currencyDisplay`	Показывать валюту в виде кода, локализованного символа или локализованного названия	`code`, `symbol`, `name`	`symbol`
`useGrouping`	Разделять ли цифры на группы	`true`, `false`	`true`
`minimumIntegerDigits`	Минимальное количество цифр целой части	от `1` до `21`	`21`
`minimumFractionDigits`	Минимальное количество десятичных цифр	от `0` до `20`	для чисел валюты з
`maximumFractionDigits`	Максимальное количество десятичных цифр	от `minimumFractionDigits` до `20`.	для чисел `max(min 3)`, для п валюты з
`minimumSignificantDigits`	Минимальное количество значимых цифр	от `1` до `21`	`1`
`maximumSignificantDigits`	Максимальное количество значимых цифр	от `minimumSignificantDigits` до `21`	`minimum`

Пример без опций:

```
let formatter = new Intl.NumberFormat("ru");
alert( formatter.format(1234567890.123) ); // 1 234 567
```

С ограничением значимых цифр (важны только первые 3):

```
let formatter = new Intl.NumberFormat("ru", {
  maximumSignificantDigits: 3

```


```
4  });
   alert( formatter.format(1234567890.123) ); // 1 230 000
```

С опциями для валюты:

```
1  let formatter = new Intl.NumberFormat("ru", {
2    style: "currency",
3    currency: "GBP"
4  });
5
6  alert( formatter.format(1234.5) ); // 1 234,5 £
```

С двумя цифрами после запятой:

```
1  let formatter = new Intl.NumberFormat("ru", {
2    style: "currency",
3    currency: "GBP",
4    minimumFractionDigits: 2
5  });
6
7  alert( formatter.format(1234.5) ); // 1 234,50 £
```

Методы в Date, String, Number

Методы форматирования также поддерживаются в обычных строках, датах, числах:

`String.prototype.localeCompare(that [, locales [, options]])`

Сравнивает строку с другой, с учётом локали, например:

```
1  let str = "ёжик";
2
3  alert( str.localeCompare("яблоко", "ru") ); // -1
```

`Date.prototype.toLocaleString([locales [, options]])`

Форматирует дату в соответствии с локалью, например:

```
1  let date = new Date(2014, 11, 31, 12, 00);
2
3  alert( date.toLocaleString("ru", { year: 'numeric', mon
```

`Date.prototype.toLocaleDateString([locales [, options]])`

То же, что и выше, но опции по умолчанию включают в себя год, месяц, день

`Date.prototype.toLocaleTimeString([locales [, options]])`

То же, что и выше, но опции по умолчанию включают в себя часы, минуты, секунды

`Number.prototype.toLocaleString([locales [, options]])`

Форматирует число, используя опции `Intl.NumberFormat`.

Все эти методы при запуске создают соответствующий объект `Intl.*` и передают ему опции, можно рассматривать их как укороченные варианты вызова.

Старые IE

В IE10 рекомендуется использовать полифил, например библиотеку https://github.com/andyearnshaw/Intl.js.

Задачи

Поделиться

Редактировать на GitHub

Отсортируйте массив с буквой ё

важность: 5

Используя `Intl.Collator`, отсортируйте массив:

```
let animals = ["тигр", "ёж", "енот", "ехидна", "АИСТ",

// ... ваш код ...

alert( animals ); // АИСТ,ёж,енот,ехидна,тигр,ЯК
```

В этом примере порядок сортировки не должен зависеть от регистра.

Что касается буквы `"ё"`, то мы следуем обычным правилам сортировки буквы ё, по которым «е» и «ё» считаются одной и той же буквой, за исключением случая, когда два слова отличаются только в позиции буквы «е» / «ё» – тогда слово с «е» ставится первым.

решение

Комментарии

перед тем как писать...