Native Internationalizaton

Making web content work across locales, each with it's own way to display dates, time is a challenge. Most of the time, when I hear about internationalization or locale aware data manipulation I hear about Moment.js or date-fns.

Both libraries are awesome, they allow you programmatically control how certain portions of text are presented to the user based on their locale (either a default or one they've provided).

However, there is also a built-in way to do these presentations. The Intl object is the namespace for the ECMAScript Internationalization API, which provides language sensitive string comparison, number formatting, and date, time formatting, and other language sensitive functions.

In order to work with this API, we have to learn more about locales . First of all, let's give a definition.

A locale is an identifier that refers to a set of user preferences such as:

- dates and times
- numbers and currencies
- translated names for time zones, languages, and countries
- measurement units
- sort-order (collation)

A locale is not case sensitive; using Uppercase names is only a convention.

The locale must be a string holding a BCP 47 language tag, and all part are separated by hyphens

Relative Time Format

I wrote about Intl.RelativeTimeFormat in an earlier blog post so I won't cover it in detail here, just enough to give you an idea of what it does.

The first step is to setup what one or more Relative Time Format Locales. In this case we set up a locale for English.

```
const rtf = new Intl.RelativeTimeFormat('en', {
  localeMatcher: 'best fit',
  style: 'long',
  numeric: 'auto',
});
```

Once we have the locale object for relative times we use the `format` methods with two parameters: the value and the unit that we want to use. Positive values indicate values in the future, negative values represent the past.

```
rtf.format(3.14, 'second');
// 'in 3.14 seconds'

rtf.format(-15, 'minute');
'minute'// '15 minutes ago'rmat(8, 'hour');
// 'in 8'hour'// 'in 8 hours'2, 'day');
// '2 days ago'day'// '2 days ago'
```

List format

Intl.ListFormat enable language-sensitive list formatting.

Different locales use different words to indicate separate the last character in a list and they use different words to indicate a conjunction (all objects together) or a disjunction (one object from the list).

The example below defines a default locale and a list of objects to work with.

```
const defaultLocale = 'en-US';
const list = ['Motorcycle', 'Bus', 'Car'];
```

Then we create new list format objects with different locales and types to show the difference in use and how the different locales (American English, Canadian Frnech and Chilean Spanish) handle the different use cases.

```
console.log(new Intl.ListFormat(defaultLocale, {
  style: 'long',
  type: 'conjunction'
}).format(list));
'conjunction'// > Motorcycle, Bus and Car(new Intl.ListFormat(defaultLoca
  style: 'short',
  type: 'disjunction'
}).format(list));
// > Motorcyc'short'// > Motorcycle, Bus or Cartl.ListFormat('fr-CA', {
  style: 'long',
  type: 'conjunction'
  }).format(list));
// > Motorcycle, Bus'fr-CA'// > Motorcycle, Bus et CarFormat('fr-CA', {
   style: 'short',
   type: 'disjunction'
}).format(list));
// > Motorcycle, Bus ou Car'fr-CA'// > Motorcycle, Bus ou Car'es-CL', {
  style: 'long',
  type: 'conjunction'
}).format(list));
// > Motorcycle, Bus y Car

 conso'es-CL'// > Motorcycle, Bus y Car', {
   style: 'short',
   type: 'disjunction'
}).format(list));
// > Motorcycle, Bus o Car

'short'// > Motorcycle, Bus o Car
```

DateTime Format

Intl.DateTimeFormat enables language-sensitive date and time formatting.

The example below creates a new date object.

```
const defaultLocale = 'en-US';
const date = new Date('December 17, 1995');
```

Then we format the date in our default locale (American English), Canadian French and Chilean Spanish. The expected result is shown in a comment under the command.

```
console.log(new Intl.DateTimeFormat(defaultLocale)
  .format(date));
// expected output: "12/17/1995"

console.log(new Intl.DateTimeFormat('fr-CA')
  .format(date));
'fr-CA'// expected output: "1995-12-17"le.log(new Intl.DateTimeFormat('es
  .format(date));
// expe'es-CL'// expected output: "17-12-1995"
```

Format Range

`formatRange` is only available on Chrome 76+. Chrome 76 hasn't hit the stable channel at the time of this post

Intl.formatRange is an extension to `DateTimeFormat` that allows you to do range of dates, for example, from January 10th to 20th.

We first set the dates that we want to work with. I set three dates to provide for different examples

```
let date1 = new Date(2007, 0, 10);
let date2 = new Date(2007, 0, 15);
let date3 = new Date(2007, 0, 20);
```

We then create a `DateTimeFormat` object. Because we're using all three values (year, month, date) the results will also incorporate all three.

```
let fmt2 = new Intl.DateTimeFormat("en", {
    year: 'numeric',
    month: 'short',
    day: 'numeric'
});
```

I test by logging to console. The results are in comments after each partiicular test.

```
console.log(fmt2.format(date1));
// Jan 10, 2007
console.log(fmt2.formatRange(date1, date2));
// Jan 10 – 15, 2007
console.log(fmt2.formatRange(date1, date3));
// Jan 10 – 20, 2007
```

If we know that we're working on a single year we can eliminate the year field. The new `DateTimeFormat` object looks like this.

```
let fmt3 = new Intl.DateTimeFormat("en", {
    month: 'short',
    day: 'numeric'
});
```

And the results omit the year since we didn't include it in the object we are using to format the ranges.

```
console.log(fmt3.format(date1));
// Jan 10
console.log(fmt3.formatRange(date1, date2));
// Jan 10 – 15
console.log(fmt3.formatRange(date1, date3));
// Jan 10 – 20
```

Once we get better browser coverage, using the native internationalization libraries will reduce app payload by removing libraries like Moment of date-fns. Until then you'll have to feature detect support foor the specific APIs and provide a

fallback where native APIs are not supported.