wanasit / **chrono** Public

A natural language date parser in Javascript

MIT license

3.2k stars **335** forks

Star | Notifications

Code | Issues 109 | Pull requests 5 | Actions | Projects | Wiki | Security | Ins

master

wanasit Merge pull request #509 from jdicarlo-dh/expose-individual-locales ... on Apr 28 **911**

View code

README.md

Chrono (v2)

A natural language date parser in Javascript.

It is designed to handle most date/time format and extract information from any given text:

- *Today*, *Tomorrow*, *Yesterday*, *Last Friday*, etc
- *17 August 2013 - 19 August 2013*
- *This Friday from 13:00 - 16.00*
- *5 days ago*
- *2 weeks from now*
- *Sat Aug 17 2013 18:40:39 GMT+0900 (JST)*
- *2014-11-30T08:15:30-05:30*

Installation

With npm:

```
$ npm install --save chrono-node
```

```
import * as chrono from 'chrono-node';

chrono.parseDate('An appointment on Sep 12-13');
```

For Node.js:

```
const chrono = require('chrono-node');

// or `import chrono from 'chrono-node'` for ECMAScript
```

What's changed in the v2

For Users

- Chrono's default now handles only international English. While in the previous version, it tried to parse with all known languages.
- The current fully supported languages are `en`, `ja`, `fr`, `nl` and `ru` (`de`, `pt`, and `zh.hant` are partially supported).

For contributors and advanced users

- The project is rewritten in TypeScript
- New Parser and Refiner interface
- New source code structure. All parsers, refiners, and configuration should be under a locale directory (See. `locales/en`)

Note: v1.x.x will still be supported for the time being.

Usage

Simply pass a `string` to functions `chrono.parseDate` or `chrono.parse`.

```
import * as chrono from 'chrono-node';

chrono.parseDate('An appointment on Sep 12-13');
// Fri Sep 12 2014 12:00:00 GMT-0500 (CDT)

chrono.parse('An appointment on Sep 12-13');
/* [{
    index: 18,
    text: 'Sep 12-13',
    start: ...
}] */
```

For more advanced usage, here is the typescript definition of the `parse` function:

```
parse(text: string, ref?: ParsingReference, option?: ParsingOption): ParsedResult[] {...
```

Parsing Reference (Date / Timezone)

Today's "Friday" is different from last month's "Friday". The meaning of the referenced dates depends on when and where they are mentioned. Chrono lets you define the reference as `Date` or `ParsingReference` object:

```
// (Note: the exmaples run on JST timezone)

chrono.parseDate('Friday', new Date(2012, 8 - 1, 23));
// Fri Aug 24 2012 12:00:00 GMT+0900 (JST)

chrono.parseDate('Friday', new Date(2012, 8 - 1, 1));
// Fri Aug 03 2012 12:00:00 GMT+0900 (JST)

chrono.parseDate("Friday at 4pm", {
    // Wed Jun 09 2021 21:00:00 GMT+0900 (JST)
    // = Wed Jun 09 2021 07:00:00 GMT-0500 (CDT)
    instant: new Date(1623240000000),
    timezone: "CDT",
})
// Sat Jun 12 2021 06:00:00 GMT+0900 (JST)
// = Fri Jun 11 2021 16:00:00 GMT-0500 (CDT)
```

ParsingReference

- `instant?: Date` The instant when the input is written or mentioned
- `timezone?: string | number` The timezone where the input is written or mentioned. Support minute-offset (number) and timezone name (e.g. "GMT", "CDT")

Parsing Options

`forwardDate` (boolean) to assume the results should happen after the reference date (forward into the future)

```
const referenceDate = new Date(2012, 7, 25);
// Sat Aug 25 2012 00:00:00 GMT+0900 -- The reference date was Saturday

chrono.parseDate('Friday', referenceDate);
// Fri Aug 24 2012 12:00:00 GMT+0900 (JST) -- The day before was Friday

chrono.parseDate('Friday', referenceDate, { forwardDate: true });
// Fri Aug 31 2012 12:00:00 GMT+0900 (JST) -- The following Friday
```

`timezones` Override or add custom mappings between timezone abbreviations and offsets. Use this when you want Chrono to parse certain text into a given timezone offset. Chrono supports both unambiguous (normal) timezone mappings and ambigous mappings where the offset is different during and outside of daylight savings.

```
// Chrono doesn't understand XYZ, so no timezone is parsed
chrono.parse('at 10:00 XYZ', new Date(2023, 3, 20))
// "knownValues": {"hour": 10, "minute": 0}

// Make Chrono parse XYZ as offset GMT-0300 (180 minutes)
chrono.parse('at 10:00 XYZ', new Date(2023, 3, 20), { timezones: { XYZ: -180 } })
// "knownValues": {"hour": 10, "minute": 0, "timezoneOffset": -180}

// Make Chrono parse XYZ as offset GMT-0300 outside of DST, and GMT-0200 during DST. Ass
import { getLastDayOfMonthTransition } from "timezone";
import { Weekday, Month } from "parsing";

const parseXYZAsAmbiguousTz = {
  timezoneOffsetDuringDst: -120,
  timezoneOffsetNonDst: -180,
  dstStart: (year: number) => getLastWeekdayOfMonth(year, Month.FEBRUARY, Weekday.SUNDA
  dstEnd: (year: number) => getLastWeekdayOfMonth(year, Month.SEPTEMBER, Weekday.SUNDAY,
};
// Parsing a date which falls within DST
chrono.parse('Jan 1st 2023 at 10:00 XYZ', new Date(2023, 3, 20), { timezones: { XYZ: par
// "knownValues": {"month": 1, ..., "timezoneOffset": -180}

// Parsing a non-DST date
chrono.parse('Jun 1st 2023 at 10:00 XYZ', new Date(2023, 3, 20), { timezones: { XYZ: par
// "knownValues": {"month": 6, ..., "timezoneOffset": -120}
```

Parsed Results and Components

ParsedResult

- `refDate: Date` The reference date of this result
- `index: number` The location within the input text of this result
- `text: string` The text this result that appears in the input
- `start: ParsedComponents` The parsed date components as a ParsedComponents object
- `end?: ParsedComponents` Similar to `start`
- `date: () => Date` Create a javascript Date

ParsedComponents

- `get: (c: Component) => number | null` Get known or implied value for the component
- `isCertain: (c: Component) => boolean` Check if the component has a known value
- `date: () => Date` Create a javascript Date

For example:

```
const results = chrono.parse('I have an appointment tomorrow from 10 to 11 AM');

results[0].index;     // 22
results[0].text;      // 'tomorrow from 10 to 11 AM'
results[0].refDate;   // Sat Dec 13 2014 21:50:14 GMT-0600 (CST)

// `start` is Sat Dec 14 2014 10:00:00
results[0].start.get('day');    // 14 (the 14th, the day after refDate)
results[0].start.get('month');  // 12 (or December)
results[0].start.get('hour');   // 10
results[0].start.date();        // Sun Dec 14 2014 10:00:00 GMT-0600 (CST)

...
results[0].end.date();  // Sun Dec 14 2014 11:00:00 GMT-0600 (CST)
```

Strict vs Casual configuration

Chrono comes with `strict` mode that parse only formal date patterns.

```
// 'strict' mode
chrono.strict.parseDate('Today');       // null
chrono.strict.parseDate('Friday');      // null
chrono.strict.parseDate('2016-07-01');  // Fri Jul 01 2016 12:00:00 ...
chrono.strict.parseDate('Jul 01 2016'); // Fri Jul 01 2016 12:00:00 ...

// 'casual' mode (default)
chrono.parseDate('Today');              // Thu Jun 30 2016 12:00:00 ...
chrono.casual.parseDate('Friday');      // Fri Jul 01 2016 12:00:00 ...
chrono.casual.parseDate('2016-07-01');  // Fri Jul 01 2016 12:00:00 ...
chrono.casual.parseDate('Jul 01 2016'); // Fri Jul 01 2016 12:00:00 ...
```

Locales

By default, Chrono is configured to handle **only international English**. This differs from the previous version of Chrono that would try all locales by default.

There are several locales supported contributed by multiple developers under `./locales` directory.

```
// default English (US)
chrono.parseDate('6/10/2018');

chrono.en.parseDate('6/10/2018');       // June 10th, 2018
chrono.en.GB.parseDate('6/10/2018');    // October 6th, 2018

chrono.ja.parseDate('□□□□□□□□□□');
```

Current supported locale options are: `en`, `ja`, `fr`, `nl` and `ru` (`de`, `pt`, and `zh.hant` are partially supported).

Importing specific locales

Chrono exports all locale options by default for simplicity. However, this can cause issues when using Chrono if you're using a Node.js runtime that was built with the Intl module disabled (with the `--without-intl` flag), such as:

```
Invalid regular expression: /* omitted */: Invalid property name in character class
```

This is because the Intl module is required to handle special characters, such as Cyrillic (`ru`).

To avoid this, you can specify only the locale(s) you want to import:

```
// CommonJS (Node.js)
const chrono = require('chrono-node/en')

// ECMAScript
import chrono from 'chrono-node/en'

// TypeScript
// Warning: `moduleResolution` must be set to `node16` or `nodeNext` in tsconfig.json`
import * as chrono from 'chrono-node/en'
```

Customize Chrono

Chrono's extraction pipeline configuration consists of `parsers: Parser[]` and `refiners: Refiner[]`.

- First, each parser independently extracts patterns from input text input and create parsing results (ParsingResult).
- Then, the parsing results are combined, sorted, and refined with the refiners. In the refining phase, the results can be filtered-out, merged, or attached with additional information.

Parser

```
interface Parser {
    pattern: (context: ParsingContext) => RegExp,
    extract: (context: ParsingContext, match: RegExpMatchArray) =>
        (ParsingComponents | ParsingResult | {[c in Component]?: number} | null)
}
```

Parser is a module for low-level pattern-based parsing. Ideally, each parser should be designed to handle a single specific date format.

User can create a new parser for supporting new date formats or languages by providing RegExp pattern `pattern()` and extracting result or components from the RegExp match `extract()`.

```
const custom = chrono.casual.clone();
custom.parsers.push({
    pattern: () => { return /\bChristmas\b/i },
    extract: (context, match) => {
        return {
            day: 25, month: 12
        }
    }
});

custom.parseDate("I'll arrive at 2.30AM on Christmas night");
// Wed Dec 25 2013 02:30:00 GMT+0900 (JST)
// 'at 2.30AM on Christmas'
```

Refiner

```
interface Refiner {
    refine: (context: ParsingContext, results: ParsingResult[]) => ParsingResult[]
}
```

Refiner is a higher level module for improving or manipulating the results. User can add a new type of refiner to customize Chrono's results or to add some custom logic to Chrono.

```
const custom = chrono.casual.clone();
custom.refiners.push({
    refine: (context, results) => {
        // If there is no AM/PM (meridiem) specified,
        //  let all time between 1:00 - 4:00 be PM (13.00 - 16.00)
        results.forEach((result) => {
            if (!result.start.isCertain('meridiem') &&
                result.start.get('hour') >= 1 && result.start.get('hour') < 4) {

                result.start.assign('meridiem', 1);
                result.start.assign('hour', result.start.get('hour') + 12);
            }
        });
        return results;
    }
});

// This will be parsed as PM.
// > Tue Dec 16 2014 14:30:00 GMT-0600 (CST)
custom.parseDate("This is at 2.30");

// Unless the 'AM' part is specified
```

```
// > Tue Dec 16 2014 02:30:00 GMT-0600 (CST)
custom.parseDate("This is at 2.30 AM");
```

In the example, the custom refiner assigns PM to parsing results with ambiguous meridiem. The `refine` method of the refiner class will be called with parsing results (from parsers or other previous refiners). The method must return an array of the new results (which, in this case, we modified those results in place).

More documentation

Checkout the Typescript Documentation in the project's Github page.

Development Guides

This guide explains how to set up chrono project for prospective contributors.

```
# Clone and install library
$ git clone https://github.com/wanasit/chrono.git chrono
$ cd chrono
$ npm install
```

Parsing date from text is complicated. A small change can have effects on unexpected places. So, Chrono is a heavily tested library. Commits that break a test shouldn't be allowed in any condition.

Chrono's unit testing is based-on Jest.

```
# Run the test
$ npm run test

# Run the test in watch mode
$ npm run watch
```

Chrono's source files is in `src` directory. The built bundle (`dist/*`) is created by running Webpack via the following command

```
$ npm run build
```

Releases 44

v2.6.3 Latest
on Apr 9

+ 43 releases

Packages

No packages published

Used by 15.7k

+ 15,724

Contributors 67

+ 56 contributors

Languages

- **TypeScript** 99.9%
- **JavaScript** 0.1%