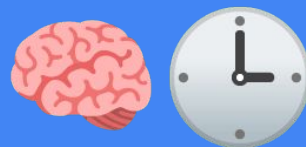


How to outsmart time




Building futuristic JavaScript applications using Temporal

Ujjwal Sharma (@ryzokuken)

 HolyJS Piter, April 2021

About me

- Ujjwal Sharma (@ryzokuken)
- Compilers Hacker at Igalia 
- TC39 Delegate
- Temporal Champion
- Node.js Core Collaborator
- V8 Contributor
- International Speaker

Recap



- Date severely outdated, has serious issues.
- Popular third party libraries for date/time handling.
- Quite a few problems exist, need to do something.
- Temporal: state-of-the-art date/time handling in JS.
- Ergonomic API with special focus on common use-cases.
- Powerful feature set accommodating complex use-cases.
 - Calendar Support
 - Custom Time Zones and Calendars



Ujjwal Sharma – Tempus Fugit: A story of time



Watch later



Share



Tempus Fugit: A story of time



Ujjwal Sharma
Igalia

Temporal is now Stage 3! 🎉

What does that mean? 🤔

- All the tiny details have been discussed.
- The specification text has been approved.
- The committee is satisfied with the design.
- Time to start implementing and using Temporal.
- Polyfill implementations.
- Browser implementations.

Stage Process

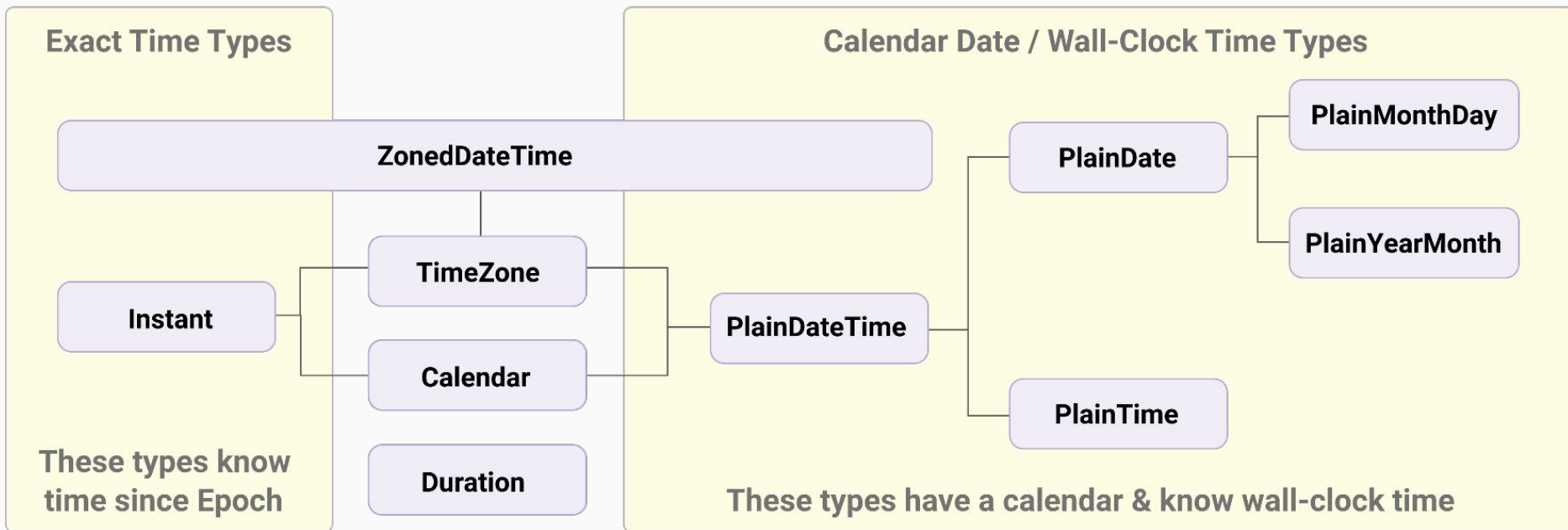
- Stage 0: Strawperson
 - Just an idea
- Stage 1: Proposal
 - Describe shape of solution
 - Identify potential blockers

Stage Process

- Stage 2: Draft
 - Describe precise syntactic and semantic details
- Stage 3: Candidate
 - Further feedback from implementations and users
- Stage 4: Finished
 - Tested and ready for addition to the standard

What changed? 🤔

- Absolute renamed to Instant.
- DateTime and friends prefixed with “Plain”.
- ZonedDateTime! ✨
- New functionality like rounding.
- Improved ergonomics.
- Various cleanups and bugfixes.
- No more subclassing.



Summary

- Instant and Plain* types work as previously talked about.
- ZonedDateTime is the combination of an Instant and a TimeZone.
- All arithmetic operations are done using Durations.
- TimeZones are used in ZDT primarily, direct conversion removed.
- Calendars are used for Date and all supersets.
- All other features could be added in a v2 proposal.

draft-ryzokuken-datetime-extended



- ISO8601/RFC3339 old and limited.
- Ad-hoc formats with additional time zone.
- Need to also add calendar into the mix.
- The need for a generalized extension format.
- The need to standardize.
- Working through the standards process.

ISO? IETF? RFC?

- ISO = International Organization for Standardization
 - ISO 8601
 - ISO/TC 154/WG 5
 - CalConnect
- IETF = Internet Engineering Task Force
- RFC = Request for Comments
- I-D = Individual Draft

ISO 8601 / RFC 3339

Time Zone
Extension

Calendar
Extension

2020-08-05T20:06:13+09:00[Asia/Tokyo][u-ca=japanese]

PlainMonthDay

TimeZone

Calendar
(Used in
all types
except
Instant)

PlainYearMonth

PlainDate

PlainTime

PlainDateTime

Instant (PlainDateTime with offset or "Z" suffix)

ZonedDateTime (PlainDateTime+offset+TimeZone)

Let's make an invoice calculator! □

Step 1: Pick a date-time picker

- Pick a date-time picker component that fits rendering strategy.
- Should return an ISO-8601 string.
 - Should return a Temporal type?
- There are already many you can pick from!
 - react-datetime-picker (React)
 - datetimepicker (jQuery)


```
Temporal.PlainDateTime.from(myString)
```

Step 3: Two date-times? Find the difference!

- When you have a start point and an end point, you can find the difference.
- Durations can be both positive and negative, direction is important!
 - Note when adding durations especially.
 - Also especially when dealing with money!
- You can check the sign with `duration.sign`
- You can find the absolute value by `duration.abs()`

Step 3: Two date-times? Find the difference!

```
const earlier = Temporal.Instant.from('2020-01-09T00:00Z');
const later = Temporal.Instant.from('2020-01-09T04:00Z');

const result = later.since(earlier, {
  largestUnit: 'hours'
}); // 'PT4H'

const result2 = earlier.until(later, {
  largestUnit: 'minutes'
}); // 'PT240M'
```

Step 4: Find out how much you worked!

- Once you have an array of durations, you can add all of them together.
- `durations.reduce(`
 `(total, current) => total.add(current),`
 `new Temporal.Duration()`
 `);`
- `const total = Temporal.Duration.from('PT0S');`
 `durations.forEach(duration => total.add(duration));`
- Remember to call `abs()` if you need to!

Duration Interchange Format

- `Temporal.Duration.from({
 years: 1, months: 2, weeks: 3, days: 4,
 hours: 5, minutes: 6, seconds: 7})
 .toString(); // 'P1Y2M3W4DT5H6M7S'`
- Can use fractions! (careful)

Step 5a: by the hour

- Depending on the contract, you might want to charge per day or per hour.
- The math is easy! In fact, it's built into Temporal.
- For bringing things to a single unit, just use `total(...)`.

```
// How many 24-hour days is 1,000,000 seconds?
```

```
d = Temporal.Duration.from('PT1000000S');
```

```
d.total({ unit: 'hours' }); // 277.77777777777777
```

Step 5b: Relativity is important!

```
// Find totals in months, with and without taking DST into account
d = Temporal.Duration.from({ hours: 2756 });
d.total({
  relativeTo: '2020-01-01T00:00+01:00[Europe/Rome]',
  unit: 'months'
}); // => 3.7958333333333334
d.total({
  unit: 'months',
  relativeTo: '2020-01-01'
}); // => 3.79444444444444443
```

Step 5c: Rounding for the win!

- The final value can be rounded up or down, depending on the contract.
- Sometimes you don't charge by a "X", but rather "n Xs".
- `round(...)` to the rescue!

```
d = Temporal.Duration.from({ minutes: 6 });  
d.round({  
  smallestUnit: 'minutes',  
  roundingIncrement: 5,  
  roundingMode: 'ceil' }); // => PT10M
```


Step 6: Profit 

Assignment Time 

Links to the future (and present)

- [Polyfill](#)
- [V8 tracking issue](#)
- [SpiderMonkey tracking issue](#)
- [JavaScriptCore tracking issue](#)
- [core-js tracking issue](#)
- [Temporal v2](#)

Special Thanks 🙏

- Temporal Champions
- Moment.js Maintainers
- Temporal Stage 3 Reviewers
- ECMA 262 Editors
- HolyJS Organizers and PC

спасибо!

