Record & Tuple

Stage 1 Update @ TC39 March 2020

Robin Ricard & Rick Button
Bloomberg

Advisor: Daniel Ehrenberg
Igalia

First, a refresher!

```
const titleRecord = #{
    text: "Record and Tuple",
    font: "Comic Sans",
};
const wordsTuple = #["Record", "Tuple"];
```

Settling down on the syntax

~~# is the new Object.freeze~~

#{ / #[

We intend to keep #

- We kept the option open in the issue tracker until now
- Most feedback by people that tried the syntax goes towards # instead of {| |} and [| |]
- Adding a closing | is cumbersome to type

Are we ok with # ?

```
const contents = #[
    #{
        text: "Record and Tuple",
        font: "Comic Sans",
    },
    #{
        text: "An ECMA TC39 Stage 1 Proposal",
    },
];
```

```
const title = #{
    text: "Record and Tuple",
    font: "Comic Sans",
};
const description = #{
    text: "An ECMA TC39 Stage 1 Proposal",
};
const contents = #[title, description];
```

at each level

Declaring Record and Tuple structures is not recursive

- Original implicit behavior was hard to reliably describe
- Grammar is simpler

Spec text started

Mostly grammar and introductions

- Grammar without lookaheads
- Methods in `Record` are excluded in grammar
- Holes in `Tuple` are excluded as well
- Feedback welcome

Referring to objects from Record & Tuple

R&T are deeply immutable

Design Goal: Prevent issues with equality, cloning and accidental mutations.

```
#{ a: {} } !== #{ a: {} }
```

R&T are deeply immutable

They can only contain primitive types… How do we refer to objects in them?

```
const vdomButton = #{
    type: "button",
    children: #[
        "click me!"
    ],
    props: #{
        onClick:
            () => alert("clicked!"),
    },
    ref: document.getElementById("my-button"),
};
```

TypeError!

RefCollection

We have an upcoming Stage 0 proposal to reference objects automatically using symbols.

```
const rc = new RefCollection();

const vdomButton = #{
    type: "button",
    children: #[
        "click me!"
    ],
    props: #{
        onClick: rc.ref(
            () => alert("clicked!")
        ),
    },
    ref: rc.ref(
        document.getElementById("my-button")
    ),
};

rc.deref(vdomButton.props.onClick)();
// => clicked!
```

Early stage 0 proposal

https://github.com/rricard/proposal-refcollection

“Deep Path Properties”

With objects

If using objects, normal assignment is enough.

```
const state = {
    counters: [
        { name: "Counter 1", value: 1 },
        { name: "Counter 2", value: 0 },
        { name: "Counter 3", value: 123 },
    ],
    metadata: {
        lastUpdate: 1584382969000,
    }
};

state.counters[0].value = 2;
state.counters[1].value = 1;
state.metadata.lastUpdate = 1584383011300;
```

With records

Records are immutable, so we can create a new record with spread syntax

```
const state2 = #{
    ...state,
    counters: #[
        #{
            ...state.counters[0],
            value: 2,
        },
        #{
            ...state.counters[1],
            value: 1,
        },
        ...state.counters,
    ],
    metadata: #{
        ...state.metadata,
        lastUpdate: 1584383011300,
    },
}
```

Can we make updates in deeply nested immutable structures more practical?

“Deep Path Properties”

Upcoming Stage 0 proposal for additional property syntax for records.

```
const state2 = #{
    ...state,
    counters[0].value: 2,
    counters[1].value: 1,
    metadata.lastUpdate: 1584383011300,
};
```

Early stage 0 proposal

https://github.com/rickbutton/proposal-deep-path-properties-for-record

Equality Semantics

- Discussed in issues #20 / #65
- Chosen in order to not introduce more values for which `===` and `Object.is()` differ
- `==` equivalent to `===`

```
#[-0]  !== #[+0]
#[NaN] === #[NaN]

#[-0]  != #[+0]
#[NaN] == #[NaN]
```

Synergy with Read-only Collections Proposal

- Presented for Stage 1 in October 2019
- Investigated possible synergy in API surface
- Currently, Read-only Collections proposal doesn’t an API for read-only arrays; synergy with `Tuple` isn’t possible.

Babel

Syntax &
Polyfill &
Playground

babel / babel

Added support for record and tuple syntax. #10865

Merged **nicolo-ribaudo** merged 18 commits into `babel:master` from `rickbutton:rb/record-and-tuple` 4 days ago

7.9.0 Released: Smaller preset-env output, Typescript 3.8 support and a new JSX transform

March 16, 2020 by Nicolò Ribaudo

• • •

Lastly, `@babel/parser` now supports an additional ECMAScript proposal: Record & Tuple. Please note that this is only parser support, and the transforms are still being worked on.

Support for Record & Tuple syntax landed in Babel 7.9.0!

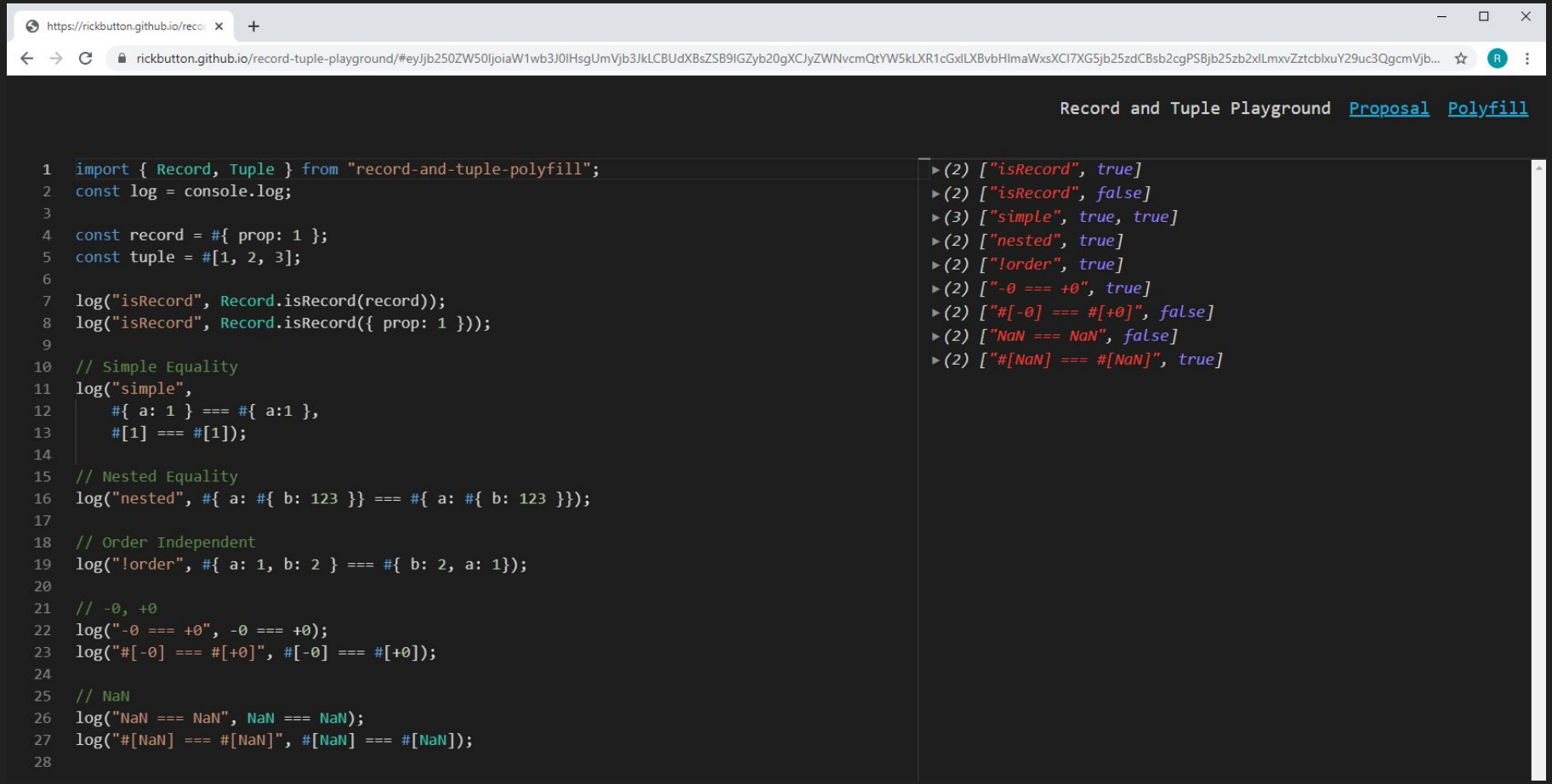
```
const record = Record({ a: 1, b: 2, c: 3 });
const tuple = Tuple("a", "b");
assert(Record.keys(record) === tuple.pushed("c"));

// with babel-plugin-proposal-record-and-tuple
assert(#[1,2,3] === #[1,2,3]);
```

Implements a graph of interned frozen objects, native `===` works as expected.

Uses `WeakRef` and `FinalizationRegistry` to prevent memory leaks.

https://github.com/bloomberg/record-tuple-polyfill

Please try the Record and Tuple Playground!



Next Steps for Record and Tuple

- Address questions presented in this meeting:
  - Syntax - #{} / #[]
  - Grammar (No methods, no holes)
  - Equality Semantics
- Incorporate this feedback from this meeting into the experimental polyfill.
- Publish experimental polyfill in a few weeks and encourage non-production experimentation, and gather feedback through issues and surveys.

- If feedback is positive, we plan to propose Record and Tuple for Stage 2 in 4-6 months. Is this reasonable?

Discussion!

Bonus!

Name clash between spec-internal Record and Record

1 Overview

R&T Draft

1.1 ECMAScript Overview

ECMAScript is object-based: basic language and host facilities are provided by objects, and an ECMAScript program is a cluster of communicating objects. In ECMAScript, an *object* is a collection of zero or more *properties* each with *attributes* that determine how each property can be used—for example, when the Writable attribute for a property is set to **false**, any attempt by executed ECMAScript code to assign a different value to the property fails. Properties are containers that hold other objects, *primitive values*, or *functions*. A primitive value is a member of one of the following built-in types: **Undefined**, **Null**, **Boolean**, **Number**, **BigInt**, **String**, ~~and~~ **Symbol**; **Record and Tuple;** an object is a member of the built-in type **Object**; and a function is a callable object. A function that is associated with an object via a property is called a *method*.

Draft ECMA-262

6.2.1 The List and Record Specification Types

The *List* type is used to explain the evaluation of argument lists (see 12.3.8) in **new** expressions, in function calls, and in other algorithms where a simple ordered list of values is needed. Values of the List type are simply ordered sequences of list elements containing the individual values. These sequences may be of any length. The elements of a list may be randomly accessed using 0-origin indices. For notational convenience an array-like syntax can be used to access List elements. For example, *arguments*[2] is shorthand for saying the 3rd element of the List *arguments*.

For notational convenience within this specification, a literal syntax can be used to express a new List value. For example, « 1, 2 » defines a List value that has two elements each of which is initialized to a specific value. A new empty List can be expressed as « ».

The *Record* type is used to describe data aggregations within the algorithms of this specification. A Record type value consists of one or more named fields. The value of each field is either an ECMAScript value or an abstract value represented by a name associated with the Record type. Field names are always enclosed in double brackets, for example [[Value]].

"Record" name clashes with spec-internal "Record Specification Type" #96

Edit New issue

Open **rricard** opened this issue 21 days ago · 4 comments

rricard commented 21 days ago • edited Member

Whenever we refer to "Record" inside of a spec document, we start referring to 6.2.1 The List and Record Specification Types which is a spec-internal notion.

This is going to be a big issue that will hamper our ability to write accurate spec text.

This yields multiple questions:

- (asking spec editors) Could we be able to not match "Record" in spec text with the spec-internal Record Specification Type?
- (asking spec editors) Is it possible to change the spec-internal Record term to something similar?
- Do we want to consider other terms instead of Record for this proposal?

As far as alternate names goes for this proposal or for renaming the spec-internal Record Specification

Assignees

No one—assign yourself

Labels

bug

question

undecided point

Projects

None yet

Milestone

ljharb commented 21 days ago — Member

I think even if we resolve the ecmarkup issue, having two kinds of "record" in the spec will be very confusing.

I'd prefer that either the existing type, or this proposal, be renamed, to avoid that.

bakkot commented 20 days ago — Member

The name of this type in this proposal affects far, far more people than the name of the spec-internal type, so please don't let this conflict affect your choice of naming for this proposal.

I agree that this would require addressing, but if we collectively decide that "record" is the right thing to call the thing in this proposal when ignoring concerns about the spec-internal type, then we'll go with that and make the necessary editorial changes to the spec to accommodate that.

 8

Grammar

3 ECMAScript Language: Lexical Grammar

3.1 Punctuators

Punctuator ::
- *OptionalChainingPunctuator*
- *NumberSignPunctuator*
- *OtherPunctuator*

OptionalChainingPunctuator ::
- `?.` [lookahead ∉ *DecimalDigit*]

NumberSignPunctuator ::
- `#{`
- `#[`

OtherPunctuator :: **one of**

`{ ( ) [ ] . ... ; , < > <= >= == != === !== + - * % ** ++ -- << >> >>> & | ^ ! ~ && || ?? ? : = += -= *= %= **= <<= >>= >>>= &= |= ^= =>`

DivPunctuator ::
- `/`
- `/=`

RightBracePunctuator ::
- `}`

Trying to avoid a lookahead

4 ECMAScript Language: Expressions

4.1 Primary Expression

Syntax

PrimaryExpression `[Yield, Await]` :

- **this**
- *IdentifierReference* `[?Yield, ?Await]`
- *Literal*
- *ArrayLiteral* `[?Yield, ?Await]`
- *ObjectLiteral* `[?Yield, ?Await]`
- *RecordLiteral* `[?Yield, ?Await]`
- *TupleLiteral* `[?Yield, ?Await]`
- *FunctionExpression*
- *ClassExpression* `[?Yield, ?Await]`
- *GeneratorExpression*
- *AsyncFunctionExpression*
- *AsyncGeneratorExpression*
- *RegularExpressionLiteral*
- *TemplateLiteral* `[?Yield, ?Await, ~Tagged]`
- *CoverParenthesizedExpressionAndArrowParameterList* `[?Yield, ?Await]`

4.1.1 Record Initializer

Syntax

RecordLiteral`[Yield, Await]` :

`#{ }`

`#{` *RecordPropertyDefinitionList*`[?Yield, ?Await]` `}`

`#{` *RecordPropertyDefinitionList*`[?Yield, ?Await]` `, }`

RecordPropertyDefinitionList`[Yield, Await]` :

RecordPropertyDefinition`[?Yield, ?Await]`

RecordPropertyDefinitionList`[?Yield, ?Await]` `,` *RecordPropertyDefinition*`[?Yield, ?Await]`

RecordPropertyDefinition`[Yield, Await]` :

IdentifierReference`[?Yield, ?Await]`

PropertyName`[?Yield, ?Await]` `:` *AssignmentExpression*`[+In, ?Yield, ?Await]`

`...` *AssignmentExpression*`[+In, ?Yield, ?Await]`

Methods are forbidden

4.1.2 Tuple Initializer

Syntax

TupleLiteral`[Yield, Await]` :

`#[ ]`

`#[` *TupleElementList*`[?Yield, ?Await]` `]`

`#[` *TupleElementList*`[?Yield, ?Await]` `, ]`

TupleElementList`[Yield, Await]` :

AssignmentExpression`[+In, ?Yield, ?Await]`

SpreadElement`[?Yield, ?Await]`

TupleElementList`[?Yield, ?Await]` `,` *AssignmentExpression*`[+In, ?Yield, ?Await]`

TupleElementList`[?Yield, ?Await]` `,` *SpreadElement*`[?Yield, ?Await]`

Holes are forbidden