Having Fun with ES Proxies

Clarke's Third Law in Action

A talk by Christophe Porteneuve at dotJS 2025

whoami

```
const christophe = {
  family: { wife: '👩 Élodie', sons: ['👦🏻 Maxence', '👦🏻 Elliott'] },
  city: 'Paris, FR',
  company: {
    name: 'Doctolib', // careers.doctolib.fr 😉
    hiring: true,
    superCool: true
  },
  webDevSince: 1995, // 😱
  mightBeKnownFor: [
    'Prototype.js',
    'Prototype and Script.aculo.us (“The Bungie Book”)',
    'dotJS',
    'Paris Web',
    'NodeSchool Paris',
  ],
}
```

About these slides…

These are interactive. Hover on the bottom-left corner for the toolbar.

You also get a lot of keyboard shortcuts:

- `f` to toggle **fullscreen**
- `d` to toggle **dark mode**
- `g` to go to any slide directly, by number or fuzzy title match
- `o` to toggle the **overview** layout (see all slides as thumbs)
- `←` / `→` / `Space` to navigate step-by-step
- `↑` / `↓` to navigate slide-by-slide
- `Home` / `End` (or `Cmd` `↑` / `Cmd` `↓`) to go to first / last slide
- Hover on the bottom-left corner for a toolbar (including a PDF download link!)
- Many code examples have links to their source files so you can play with the whole implementation


Lay of the land
You likely haven’t used this stuff yet.

What are proxies?

They're **not** network proxies 😉

- A proxy is a **wrapper** around an object (which could be a function)
- It can **intercept any interaction** your code has with that object… through the wrapper.
- It's the latest step in **JS metaprogramming**
- **Can't be polyfilled.**
- Showed up in **ES2015**, available across browsers since Sep 2016, and Node 6+.
- Deemed **baseline** for a long time.

You build one like this

```
new Proxy(obj, {
  // handler object, usually provides "traps":  get, set, has, apply, etc.
})
```

- **Target** = the proxied object
- **Prop / Key** = property being used (looked up, read, written to or whose descriptor is defined)
- **Value** = property value being set
- For `apply` and `construct` you get arguments, `apply` also has the binding (`this`)

You build one like this

```
new Proxy(obj, {
  // handler object, usually provides "traps":  get, set, has, apply, etc.
})
```

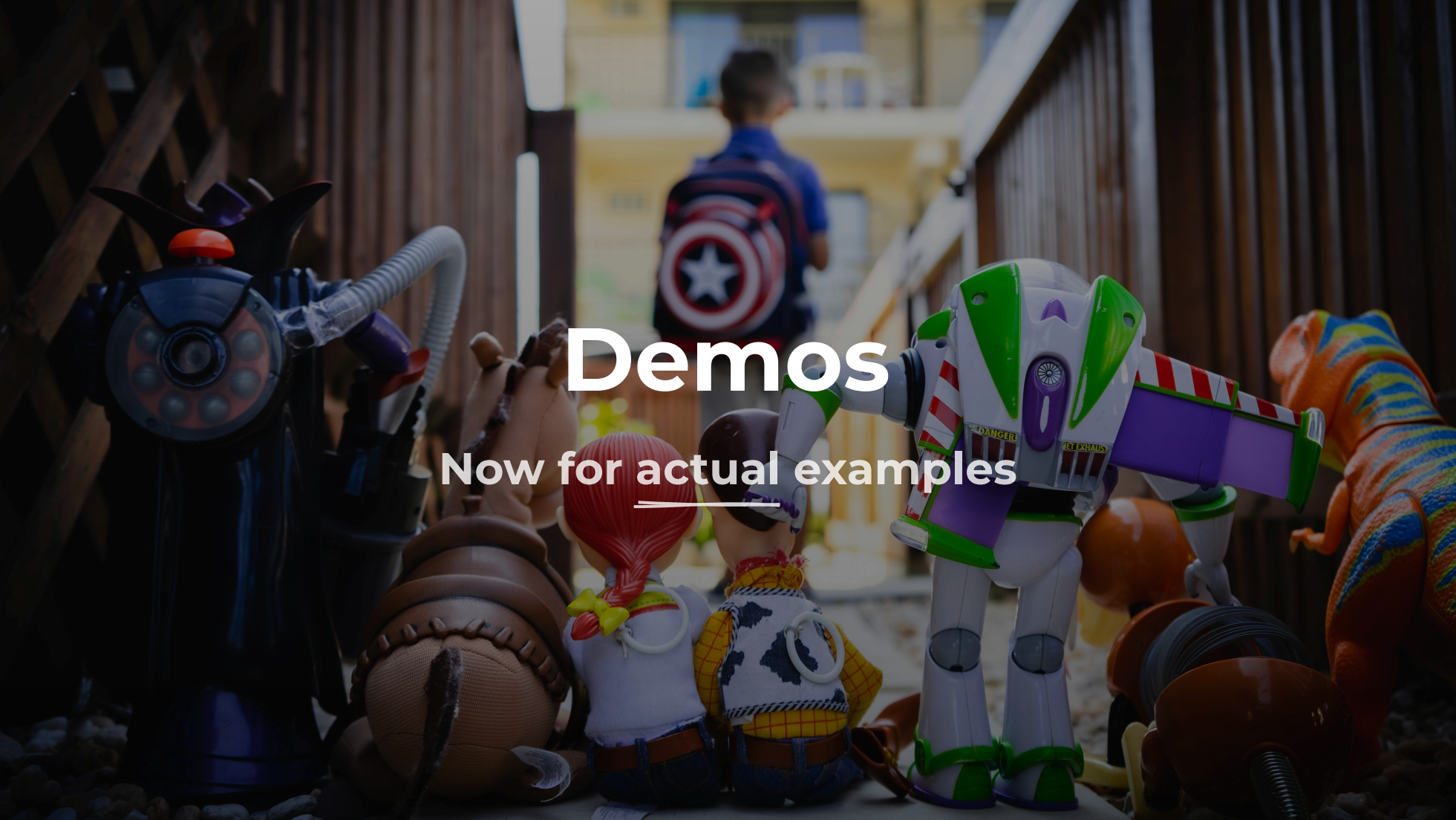
- **Target** = the proxied object
- **Prop / Key** = property being used (looked up, read, written to or whose descriptor is defined)
- **Value** = property value being set
- For `apply` and `construct` you get arguments, `apply` also has the binding (`this`)

`Reflect` provides all the **built-in behaviors** for every **trap**, so we can delegate to them.

Here’s a dumb example

```
const conference = { name: 'dotJS', year: 2025 }
const proxiedConf = new Proxy(conference, {
  get(target, key, receiver) {
    console.log(`Reading ${key as string} off`, target)
    return Reflect.get(target, key, receiver)
  },
})

// console.log(proxiedConf.name)
```


Demos
Now for actual examples

“Type-safe” object properties

```ts
function makeTypeSafish(obj: object) {
  return new Proxy(obj, {
    set(target, key, value) {
      if (key in target) {
        const knownType = typeof target[key];

        if (knownType !== typeof value) {
          throw new Error(`Property ${key as string} expects a ${knownType}`);
        }
      }

      return Reflect.set(target, key, value);
    },
  });
}
```

🌎 » type-safe-properties.ts

“Type-safe” object properties

```ts
function makeTypeSafish(obj: object) {
  return new Proxy(obj, {
    set(target, key, value) {
      if (key in target) {
        const knownType = typeof target[key];

        if (knownType !== typeof value) {
          throw new Error(`Property ${key as string} expects a ${knownType}`);
        }
      }

      return Reflect.set(target, key, value);
    },
  });
}
```

🌎 » type-safe-properties.ts

“Type-safe” object properties

```ts
function makeTypeSafish(obj: object) {
  return new Proxy(obj, {
    set(target, key, value) {
      if (key in target) {
        const knownType = typeof target[key];

        if (knownType !== typeof value) {
          throw new Error(`Property ${key as string} expects a ${knownType}`);
        }
      }

      return Reflect.set(target, key, value);
    },
  });
}
```

🌎 » type-safe-properties.ts

“Type-safe” object properties

```ts
function makeTypeSafish(obj: object) {
  return new Proxy(obj, {
    set(target, key, value) {
      if (key in target) {
        const knownType = typeof target[key];

        if (knownType !== typeof value) {
          throw new Error(`Property ${key as string} expects a ${knownType}`);
        }
      }

      return Reflect.set(target, key, value);
    },
  });
}
```

🌎 » type-safe-properties.ts

“Type-safe” object properties

```
const bestConf = makeTypeSafish({ name: 'dotJS' })
```

“Type-safe” object properties

```
const bestConf = makeTypeSafish({ name: 'dotJS' })

bestConf.name = 42
// 💥 Throws with "Property name expects a string"
```

“Type-safe” object properties

```
const bestConf = makeTypeSafish({ name: 'dotJS' })

bestConf.edition = 10
// ✅ OK

bestConf.edition = '10th'
// 💥 Throws with "Property edition expects a number"
```

Shout-out: `tpyo`

The lib that defeats typos 🤣

```
1  import tpyo from 'tpyo'
2
3  const woah = tpyo(['dotAI', 'dotJS'])
4  woah.pousse('dotCSS', 'dotSwift')
5  console.log(woah.longueur, woah.avril(3), woah.plop(), woah.shot())
```

```
4, dotSwift, dotSwift, dotAI
```

Finds the closest existing property using Levenshtein distances 🤯

Negative array indices

```ts
function allowNegativeIndices(arr: Array<unknown>) {
  return new Proxy(arr, {
    get(target, prop, receiver) {
      const index = Number(prop);
      if (index < 0) prop = String(index + target.length);
      return Reflect.get(target, prop, receiver);
    },
    set(target, prop, value, receiver) {
      const index = Number(prop);
      if (index < 0) prop = String(index + target.length);
      return Reflect.set(target, prop, value, receiver);
    },
  });
}
```

» negative-array-playground.ts

Negative array indices

```ts
function allowNegativeIndices(arr: Array<unknown>) {
  return new Proxy(arr, {
    get(target, prop, receiver) {
      const index = Number(prop);
      if (index < 0) prop = String(index + target.length);
      return Reflect.get(target, prop, receiver);
    },
    set(target, prop, value, receiver) {
      const index = Number(prop);
      if (index < 0) prop = String(index + target.length);
      return Reflect.set(target, prop, value, receiver);
    },
  });
}
```

» negative-array-playground.ts

Negative array indices

```
const allTheDots = allowNegativeIndices([
  'dotJS', 'dotAI',
  'dotRB', 'dotCSS', 'dotScale', 'dotSecurity', 'dotSwift',
])

allTheDots[0]  // ⇒ 'dotJS'
allTheDots[-1] // ⇒ 'dotSwift'
allTheDots[-2] // ⇒ 'dotSecurity'
```

Negative array indices

```
const allTheDots = allowNegativeIndices([
  'dotJS', 'dotAI',
  'dotRB', 'dotCSS', 'dotScale', 'dotSecurity', 'dotSwift',
])

allTheDots[0]  // ⇒ 'dotJS'
allTheDots[-1] // ⇒ 'dotSwift'
allTheDots[-2] // ⇒ 'dotSecurity'
allTheDots[-3] = 'dotMontéeÀLÉchelle' // Oh là là the frenchness 🇫🇷
```

Negative array indices

```ts
const allTheDots = allowNegativeIndices([
  'dotJS', 'dotAI',
  'dotRB', 'dotCSS', 'dotScale', 'dotSecurity', 'dotSwift',
])

allTheDots[0]  // => 'dotJS'
allTheDots[-1] // => 'dotSwift'
allTheDots[-2] // => 'dotSecurity'
allTheDots[-3] = 'dotMontéeÀLÉchelle'
console.log(allTheDots[4])
console.log(Object.keys(allTheDots))

```

```
dotMontéeÀLÉchelle
["0", "1", "2", "3", "4", "5", "6"]
```

» negative-array-playground.ts

Mini observables

```ts
type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }

function makeObservable(obj: object) {
  const subscribers = new Set<Subscriber>()
  const proxy = new Proxy(obj, {
    …
    set(target, key, value, receiver) {
      const previous = target[key as keyof typeof target]
      const result = Reflect.set(target, key, value, receiver)
      subscribers.forEach((sub) => sub({ key, previous, value }))
      return result
    }
  }) as Subscribable

  proxy.$subscribe = …

  return proxy
}
```

🌎 » mini-observables.ts

Mini observables

```ts
type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }

function makeObservable(obj: object) {
  const subscribers = new Set<Subscriber>()
  const proxy = new Proxy(obj, {
    …
    set(target, key, value, receiver) {
      const previous = target[key as keyof typeof target]
      const result = Reflect.set(target, key, value, receiver)
      subscribers.forEach((sub) => sub({ key, previous, value }))
      return result
    }
  }) as Subscribable

  proxy.$subscribe = …

  return proxy
}
```

🌎 » mini-observables.ts

Mini observables

```ts
type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }

function makeObservable(obj: object) {
  const subscribers = new Set<Subscriber>()
  const proxy = new Proxy(obj, {
    …
    set(target, key, value, receiver) {
      const previous = target[key as keyof typeof target]
      const result = Reflect.set(target, key, value, receiver)
      subscribers.forEach((sub) => sub({ key, previous, value }))
      return result
    }
  }) as Subscribable

  proxy.$subscribe = …

  return proxy
}
```

🌎 » mini-observables.ts

Mini observables

```ts
type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }

function makeObservable(obj: object) {
  const subscribers = new Set<Subscriber>()
  const proxy = new Proxy(obj, {
    …
    set(target, key, value, receiver) {
      const previous = target[key as keyof typeof target]
      const result = Reflect.set(target, key, value, receiver)
      subscribers.forEach((sub) => sub({ key, previous, value }))
      return result
    }
  }) as Subscribable

  proxy.$subscribe = …

  return proxy
}
```

🌎 » mini-observables.ts

Mini observables

```ts
type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }

function makeObservable(obj: object) {
  const subscribers = new Set<Subscriber>()
  const proxy = new Proxy(obj, {
    …
    set(target, key, value, receiver) {
      const previous = target[key as keyof typeof target]
      const result = Reflect.set(target, key, value, receiver)
      subscribers.forEach((sub) => sub({ key, previous, value }))
      return result
    }
  }) as Subscribable

  proxy.$subscribe = …

  return proxy
}
```

🌎 » mini-observables.ts

Mini observables

```ts
type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }

function makeObservable(obj: object) {
  const subscribers = new Set<Subscriber>()
  const proxy = new Proxy(obj, {
    deleteProperty(target, key) {
      const previous = target[key as keyof typeof target]
      const result = Reflect.deleteProperty(target, key)
      subscribers.forEach((sub) => sub({ key, previous, deleted: true }))
      return result
    },
    …
  }) as Subscribable

  proxy.$subscribe = …

  return proxy
}
```

🌎 » mini-observables.ts

Mini observables

```ts
type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }

function makeObservable(obj: object) {
  const subscribers = new Set<Subscriber>()
  const proxy = new Proxy(obj, {
    deleteProperty(target, key) {
      const previous = target[key as keyof typeof target]
      const result = Reflect.deleteProperty(target, key)
      subscribers.forEach((sub) => sub({ key, previous, deleted: true }))
      return result
    },
    …
  }) as Subscribable

  proxy.$subscribe = …

  return proxy
}
```

🌎 » mini-observables.ts

Mini observables

```ts
type Subscribable = { $subscribe: (sub: Subscriber) ⇒ Subscribable }

function makeObservable(obj: object) {
  const subscribers = new Set<Subscriber>()
  const proxy = new Proxy(obj, {
    deleteProperty(target, key) {
      const previous = target[key as keyof typeof target]
      const result = Reflect.deleteProperty(target, key)
      subscribers.forEach((sub) ⇒ sub({ key, previous, deleted: true }))
      return result
    },
    …
  }) as Subscribable

  proxy.$subscribe = …

  return proxy
}
```

» mini-observables.ts

Mini observables

```ts
type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }

function makeObservable(obj: object) {
  const subscribers = new Set<Subscriber>()
  const proxy = new Proxy(obj, {
    deleteProperty(target, key) {
      const previous = target[key as keyof typeof target]
      const result = Reflect.deleteProperty(target, key)
      subscribers.forEach((sub) => sub({ key, previous, deleted: true }))
      return result
    },
    …
  }) as Subscribable

  proxy.$subscribe = …

  return proxy
}
```

🌎 » mini-observables.ts

Mini observables

```ts
type Notification =
  | { key: PropertyKey, previous: unknown, value: unknown }
  | { key: PropertyKey, previous: unknown, deleted: true }
type Subscriber = (notif: Notification) => void
type Subscribable = { $subscribe: (sub: Subscriber) => Subscribable }

function makeObservable(obj: object) {
  const subscribers = new Set<Subscriber>()
  const proxy = …

  proxy.$subscribe = (subscriber) => {
    subscribers.add(subscriber)
    return proxy
  }

  return proxy
}
```

🌎 » mini-observables.ts

Mini observables

```ts
const propLog: PropertyKey[] = []
const foo = makeObservable({ name: 'dotJS', ranking: '👍' })
  .$subscribe(console.debug)
  .$subscribe(({ key }) => propLog.push(key))

// foo.name += ' 2025'
// foo.ranking = '💫'
// foo.hasAfterParty = true
// delete foo.hasAfterParty
// console.log(propLog)

```

» mini-observables.ts

Shout-out: Vue.js 3 relies on ES proxies

```
<script setup>
import { reactive } from 'vue'

const state = reactive({ count: 0 })
</script>

<template>
  <button @click="state.count += 1">
   {{ state.count }}
  </button>
</template>
```

It's more powerful and performant than Vue 2's reactivity. For instance, no custom syntax for reacting to property deletion (`deleteProperty` trap), no need to have all properties defined from the start either (the `set` trap triggers on both existing and new properties).

TTL Refs

```ts
function makeTTL(fx: Function, ttlMS: number) {
  const expiry = Date.now() + ttlMS
  const fxName = fx.name || fx['displayName'] || 'Function'

  return new Proxy(fx, {
    apply(target, thisArg, args) {
      if (Date.now() ⩾ expiry) {
        throw new Error(`${fxName} cannot be called anymore (${ttlMS}ms TTL expired)`)
      }

      return Reflect.apply(target, thisArg, args)
    }
  })
}
```

» ttl-refs.ts

TTL Refs

```ts
function makeTTL(fx: Function, ttlMS: number) {
  const expiry = Date.now() + ttlMS
  const fxName = fx.name || fx['displayName'] || 'Function'

  return new Proxy(fx, {
    apply(target, thisArg, args) {
      if (Date.now() ⩾ expiry) {
        throw new Error(`${fxName} cannot be called anymore (${ttlMS}ms TTL expired)`)
      }

      return Reflect.apply(target, thisArg, args)
    }
  })
}
```

» ttl-refs.ts

TTL Refs

```ts
function sayHi(whom: string) { console.log(`Hi ${whom}!`) }
const sayHiFast = makeTTL(sayHi, 500)

sayHiFast('John')
// ✅ Hi John

setTimeout(() => sayHiFast('Nessrine'), 400)
// ✅ Hi Nessrine!

setTimeout(() => sayHiFast('Sylvain'), 500)
// 💥 sayHi cannot be called anymore (500ms TTL expired)

```

```
Hi John!
+401ms:, Hi Nessrine!
+502ms:, Error: "sayHi cannot be called anymore (500ms TTL expired)"
```

» ttl-refs.ts

“URL builder”

```ts
// We aim for this:
// const builder = makeURLBuilder('https://api.acme.biz')
// builder.buy.bacon.and.eggs() ⇒ https://api.acme.biz/buy/bacon/and/eggs

function makeURLBuilder(domain: string): URLBuilder {
  const parts: string[] = []

  return new Proxy(buildURL, {
    has: () ⇒  true,
    get(_target, prop, receiver) {
      parts.push(prop as string)
      return receiver
    },
  }) as URLBuilder

  …
}
```

🌎 » url-builder.ts

“URL builder”

```ts
// We aim for this:
// const builder = makeURLBuilder('https://api.acme.biz')
// builder.buy.bacon.and.eggs() ⇒ https://api.acme.biz/buy/bacon/and/eggs

function makeURLBuilder(domain: string): URLBuilder {
  const parts: string[] = []

  return new Proxy(buildURL, {
    has: () ⇒  true,
    get(_target, prop, receiver) {
      parts.push(prop as string)
      return receiver
    },
  }) as URLBuilder

  …
}
```

🌎 » url-builder.ts

“URL builder”

```ts
// We aim for this:
// const builder = makeURLBuilder('https://api.acme.biz')
// builder.buy.bacon.and.eggs() ⇒ https://api.acme.biz/buy/bacon/and/eggs

function makeURLBuilder(domain: string): URLBuilder {
  const parts: string[] = []

  return new Proxy(buildURL, {
    has: () ⇒  true,
    get(_target, prop, receiver) {
      parts.push(prop as string)
      return receiver
    },
  }) as URLBuilder

  …
}
```

🌎 » url-builder.ts

“URL builder”

```ts
// We aim for this:
// const builder = makeURLBuilder('https://api.acme.biz')
// builder.buy.bacon.and.eggs() ⇒ https://api.acme.biz/buy/bacon/and/eggs

function makeURLBuilder(domain: string): URLBuilder {
  const parts: string[] = []

  …

  function buildURL(qs: object = {}) {
    const url = new URL([domain, ...parts].join('/'))
    for (const [key, value] of Object.entries(qs)) {
      url.searchParams.append(key, value)
    }
    parts.length = 0
    return url.toString()
  }
}
```

🌎 » url-builder.ts

“URL builder”

```ts
const builder = makeURLBuilder('https://api.acme.biz')
console.log(builder.buy.bacon.and.eggs())

console.log(builder.categories[124].discounts({ mode: 'sale', pageSize: 10 }))

```

```
https://api.acme.biz/buy/bacon/and/eggs
https://api.acme.biz/categories/124/discounts?mode=sale&pageSize=10
```

» url-builder.ts

VCR (record & replay)

Here's what we'd like to be able to do.

```ts
const { vcr, replay } = makeVCR()
vcr.foo.bar.baz({ name: 'hello' })
vcr.foo.quux = 42
vcr.foo.doo('amazing').name = 'YOLO'

const subject: DemoSubject = {
  foo: {
    bar: { baz({ name }: { name: string }) { this.name = name } },
    doo(text: string) { this.kind = text.toUpperCase(); return this },
  },
}

replay(subject)
```

🌎 » vcr.ts

VCR (record & replay)

Here's what we'd like to be able to do.

```ts
const { vcr, replay } = makeVCR()
vcr.foo.bar.baz({ name: 'hello' })
vcr.foo.quux = 42
vcr.foo.doo('amazing').name = 'YOLO'

const subject: DemoSubject = {
  foo: {
    bar: { baz({ name }: { name: string }) { this.name = name } },
    doo(text: string) { this.kind = text.toUpperCase(); return this },
  },
}

replay(subject)
```

🌎 » vcr.ts

VCR (record & replay)

Here's what we'd like to be able to do.

```ts
const { vcr, replay } = makeVCR()
vcr.foo.bar.baz({ name: 'hello' })
vcr.foo.quux = 42
vcr.foo.doo('amazing').name = 'YOLO'

const subject: DemoSubject = {
  foo: {
    bar: { baz({ name }: { name: string }) { this.name = name } },
    doo(text: string) { this.kind = text.toUpperCase(); return this },
  },
}

replay(subject)
```

» vcr.ts

VCR (record & replay)

Here's what we'd like to be able to do.

```ts
const { vcr, replay } = makeVCR()
vcr.foo.bar.baz({ name: 'hello' })
vcr.foo.quux = 42
vcr.foo.doo('amazing').name = 'YOLO'

const subject: DemoSubject = {
  foo: {
    bar: { baz({ name }: { name: string }) { this.name = name } },
    doo(text: string) { this.kind = text.toUpperCase(); return this },
  },
}

replay(subject)
```

» vcr.ts

VCR (record & replay)

```ts
function makeVCR({
  pathPrefix = [],
  ops = [],
}: { pathPrefix?: PropPath; ops?: Op[] } = {}): VCRTuple {
  const vcr = new Proxy(() => {}, {
    get: (_, prop) => makeVCR({ pathPrefix: [...pathPrefix, prop], ops }).vcr,
    apply(_, __, args) {
      const propPath = [...pathPrefix, { args }]
      ops.push({ kind: 'read', prop: propPath })
      return makeVCR({ pathPrefix: propPath, ops }).vcr
    },
    set(_, prop, value) {
      ops.push({ kind: 'write', prop: [...pathPrefix, prop], value })
      return true
    },
  }) as VCR

  return { vcr, replay: $replay.bind(null, ops) }
}
```

VCR (record & replay)

```ts
function makeVCR({
  pathPrefix = [],
  ops = [],
}: { pathPrefix?: PropPath; ops?: Op[] } = {}): VCRTuple {
  const vcr = new Proxy(() => {}, {
    get: (_, prop) => makeVCR({ pathPrefix: [...pathPrefix, prop], ops }).vcr,
    apply(_, __, args) {
      const propPath = [...pathPrefix, { args }]
      ops.push({ kind: 'read', prop: propPath })
      return makeVCR({ pathPrefix: propPath, ops }).vcr
    },
    set(_, prop, value) {
      ops.push({ kind: 'write', prop: [...pathPrefix, prop], value })
      return true
    },
  }) as VCR

  return { vcr, replay: $replay.bind(null, ops) }
}
```

VCR (record & replay)

```ts
function makeVCR({
  pathPrefix = [],
  ops = [],
}: { pathPrefix?: PropPath; ops?: Op[] } = {}): VCRTuple {
  const vcr = new Proxy(() => {}, {
    get: (_, prop) => makeVCR({ pathPrefix: [...pathPrefix, prop], ops }).vcr,
    apply(_, __, args) {
      const propPath = [...pathPrefix, { args }]
      ops.push({ kind: 'read', prop: propPath })
      return makeVCR({ pathPrefix: propPath, ops }).vcr
    },
    set(_, prop, value) {
      ops.push({ kind: 'write', prop: [...pathPrefix, prop], value })
      return true
    },
  }) as VCR

  return { vcr, replay: $replay.bind(null, ops) }
}
```

VCR (record & replay)

```ts
function makeVCR({
  pathPrefix = [],
  ops = [],
}: { pathPrefix?: PropPath; ops?: Op[] } = {}): VCRTuple {
  const vcr = new Proxy(() => {}, {
    get: (_, prop) => makeVCR({ pathPrefix: [...pathPrefix, prop], ops }).vcr,
    apply(_, __, args) {
      const propPath = [...pathPrefix, { args }]
      ops.push({ kind: 'read', prop: propPath })
      return makeVCR({ pathPrefix: propPath, ops }).vcr
    },
    set(_, prop, value) {
      ops.push({ kind: 'write', prop: [...pathPrefix, prop], value })
      return true
    },
  }) as VCR

  return { vcr, replay: $replay.bind(null, ops) }
}
```

VCR (record & replay)

```ts
function makeVCR({
  pathPrefix = [],
  ops = [],
}: { pathPrefix?: PropPath; ops?: Op[] } = {}): VCRTuple {
  const vcr = new Proxy(() => {}, {
    get: (_, prop) => makeVCR({ pathPrefix: [...pathPrefix, prop], ops }).vcr,
    apply(_, __, args) {
      const propPath = [...pathPrefix, { args }]
      ops.push({ kind: 'read', prop: propPath })
      return makeVCR({ pathPrefix: propPath, ops }).vcr
    },
    set(_, prop, value) {
      ops.push({ kind: 'write', prop: [...pathPrefix, prop], value })
      return true
    },
  }) as VCR

  return { vcr, replay: $replay.bind(null, ops) }
}
```

VCR (record & replay)

```ts
const { vcr, replay } = makeVCR()
vcr.foo.bar.baz({ name: 'hello' })
vcr.foo.quux = 42
vcr.foo.doo('amazing').name = 'YOLO'

const subject: DemoSubject = {
  foo: {
    bar: { baz({ name }: { name: string }) { this.name = name } },
    doo(text: string) { this.kind = text.toUpperCase(); return this },
  },
}

replay(subject)
console.log(subject)
```

```
{ "foo": { "bar": { "name": "hello" }, "quux": 42, "kind": "AMAZING", "name": "YOLO" } }
```

» vcr.ts

Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// Without Immer

const nextState = [...baseState]
nextState[1] = { ...nextState[1], done: true }
nextState.push({ title: 'Tweet about it' })
```

Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// Without Immer

const nextState = [...baseState]
nextState[1] = { ...nextState[1], done: true }
nextState.push({ title: 'Tweet about it' })
```

Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// Without Immer

const nextState = [...baseState]
nextState[1] = { ...nextState[1], done: true }
nextState.push({ title: 'Tweet about it' })
```

Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// Without Immer

const nextState = [...baseState]
nextState[1] = { ...nextState[1], done: true }
nextState.push({ title: 'Tweet about it' })
```

Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// With Immer
import { produce } from 'immer'

const nextState = produce(baseState, (draft) => {
    draft[1].done = true
    draft.push({ title: 'Tweet about it' })
})

// `baseState` is unchanged
// `nextState` is the deep, copy-on-write variant
```

Shout-out: Immer.js is amazing

Michel Weststrate (also of MobX fame) made this incredible library that provides transparent immutability. It's sort of VCR done right, and on steroids.

```
// With Immer
import { produce } from 'immer'

const nextState = produce(baseState, (draft) => {
    draft[1].done = true
    draft.push({ title: 'Tweet about it' })
})

// `baseState` is unchanged
// `nextState` is the deep, copy-on-write variant
```

Shout-out: Immer.js is amazing

It's also great in combination with React, with or without Redux. Basic example:

```
const [todos, setTodos] = useImmer([
  { id: 'React', title: 'Learn React', done: true },
  { id: 'Immer', title: 'Try Immer', done: false }
])

function handleToggle(id) {
  setTodos((draft) => {
    const todo = draft.find((todo) => todo.id === id)
    todo.done = !todo.done
  })
}
```

Also has built-in solutions for `useReducer`, Redux reducers, etc.

🙏 Thank you! 😊

These slides are on `bit.ly/dotjs-esproxies`

Christophe: `@porteneuve@piaille.fr` / LinkedIn

Photo credits: Cloudy sky par mosi knife, Confusion by Ayo Ogunsende and toys by Chris Hardy, all sources from Unsplash.