Undo/Redo大作戰：State Management

Jas Chen

2015/04/15 @ Magic6 meetup

Agenda

- About me
- The problem of optimistic update: Deep copy
- The problem of render performance: Deep compare
- The solution: Immutable data

The problem of optimistic update: Deep copy

Optimistic Update

Traditional way

```
// wait for server to update,
// this will take a while
$.post("/api/user", user)
    .done(function () {
        // server says ok
        userList.push(user);
        renderUserList();
    });
```

Optimistic update

```
// I believe update will success
userList.push(user);
renderUserList();
$.post("/api/user", user);
```

Optimistic update will not wait for server response, user will see the change immediately, this brings better user experience.

But, what if update fail?

Traditional way

```
// wait for server to update,
// this will take a while
$.post("/api/user", user)
    .done(function () {
        userList.push(user);
        renderUserList();
    })
    .fail(function () {
        alert("error");
    });
```

Optimistic update

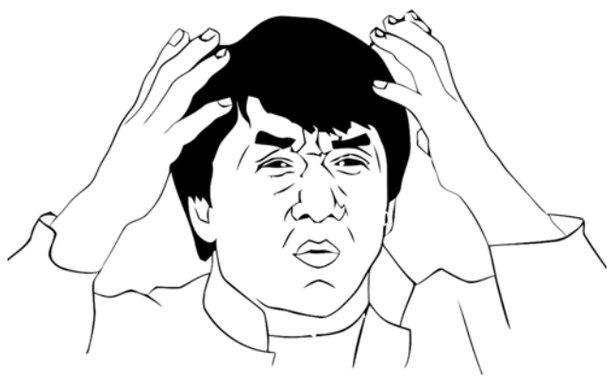
```
// I believe update will success
userList.push(user);
renderUserList();

$.post("/api/user", user)
    // but what if it fails?
    .fail(function () {
        alert("error");

        // undo push user

        renderUserList();
    });
```

Optimistic update will need to ‘undo’ when fail.

How to undo?

The Command Pattern

- Add -> Delete
- Insert -> Reinsert
- Delete -> Add
- Update -> ????



How to undo? (cont.)

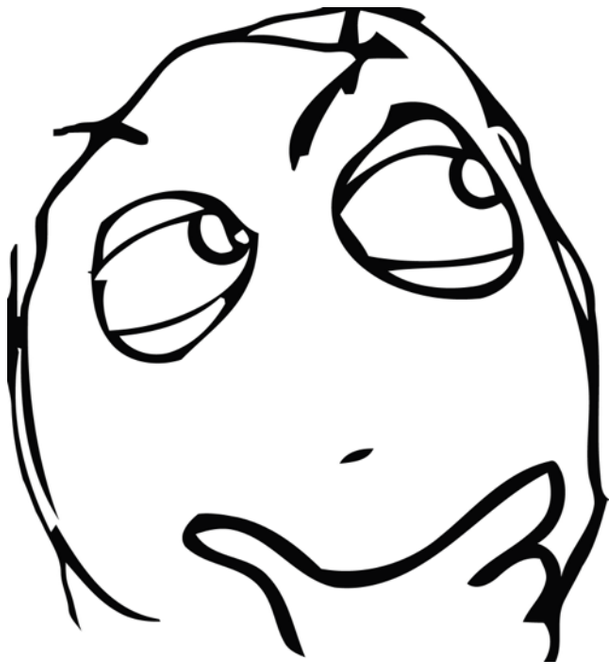
The state management way

- oldData = copy(data)
- if (fail) data = oldData

How to undo? (cont.)

The state management way

- oldData = **copy**(data)
- if (fail) data = oldData



- There is no efficient way to deep copy javascript objects (maybe *JSON.stringify()* is the simplest way to do that, but it doesn't handle Date very well).
- Data only change a little bit, but you copy all the data, that wastes space and time.

The problem of render performance: Deep compare

UI is generated based on data

```
[{
    name: "Jas Chen",
    email: "a@g.com",
    birthday: "2014-11-30T00:00:00.000Z",
    habits: [
        "Movies",
        "Basketball",
        "Coding"
    ]
}, {
    name: "David",
    email: "a@g.com",
    birthday: "2014-11-30T00:00:00.000Z",
    habits: [
        "Take pictures",
        "Football",
        "Novels"
    ]
}]
```

Jas Chen

a@g.com

2014/11/30

- Movies
- Basketball
- Coding

David

a@g.com

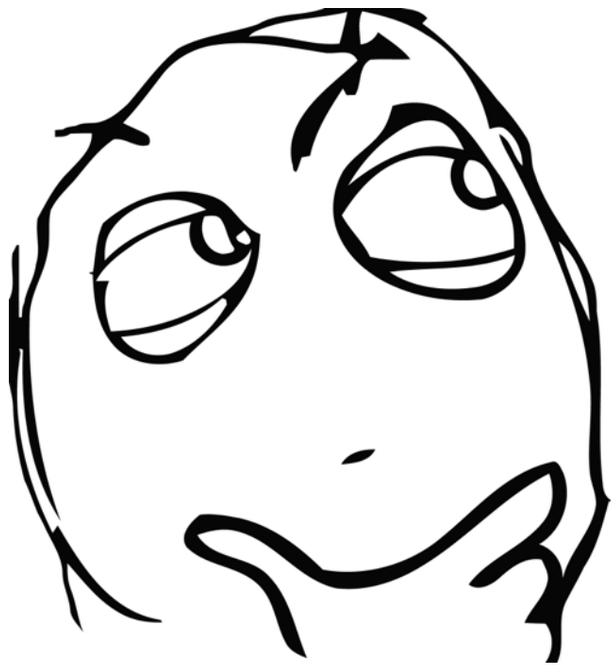
2014/11/30

- Take pictures
- Football
- Novels

Single source of truth: you change the data, UI will re-render

But render is expensive

- Don’t manipulate DOM when data is the same.
- That means you have to deeply compare new data and old data.



But… there is no efficient way to deep compare two javascript objects.

The solution:

Immutable data

What is Immutable data?

It is a kind of data structure, it provides data structures like Map, List, etc, and all of these data structures are immutable.

```
> var imList = Immutable.List( [1, 'hello'] )
<- undefined
> imList
<- ▶ Object {size: 2, _origin: 0, _capacity: 2, _level: 5, _root: null…}
> imList.get(0)
<- 1
> imList.get(1)
<- "hello"
>
```

How to modify immutable data?

- You can’t modify immutable data
- It will return a new immutable data to you

```
> var imList2 = imList.push('wow')
< undefined
> imList.size
< 2
> imList2.size
< 3
> imList2.get(2)
< "wow"
```

The good

- Save memory space: Immutable data is not a deep copied javascript object, new immutable data will reference values not changed from old immutable data.

- Efficient compare: If you want to find out whether two data structures is the same, just compare their reference.

```
> imList === imList2
< false
```

The bad

- Not native data structures: This meant you cant iterate them like native ones.

```
> imList[0]
< undefined
> var arr = imList.toArray()
< undefined
> arr[0]
< 1
```

Immutable Data Implementations

- Immutable.js
  https://facebook.github.io/immutable-js/
- mori
  https://swannodette.github.io/mori/

They are ported from Clojure

Who use it?

React / Angular Meeting

Attendees

Christopher Chedeau (react)
Igor Minar (angular)
Lee Byron (react)
Miško Hevery (angular)
Sebastian Markbage (react)
Yegor Jbanov (angular)
Victor Savkin (angular)
Matias Niemela (angular)
John Lindquist (egghead.io)

Collaboration points

- Best practices for NPM as package manager
  - AI: Sebastian to send model for ensuring smooth upgrades
- Performance measurement tooling and techniques
  - AI: Brad to share Benchpress techniques/arcana (this link contains most of it, but will be updated with more info soon)
- Using "other renderers" like iOS and Web Worker support to prove model for updates to Web platform
  - AI: Brad to nominate Angular team members to collaborate with Christopher
- Immutable data structures: identify nice patterns in Angular apps as support for inclusion as part of TC39 spec
  - AI: Victor to add specific support for immutable-js to Angular 2 change detection

https://docs.google.com/document/d/1QZxArgMwidgCrAbuSikcB2iBxkffH6w0YB0C1qCsuH0/edit?pli=1

Thanks