Mojo JS Bindings Rename

ortuno@ (heavily based on rockot@'s Mojo & the Naming of Things)
March 2019

THIS DOCUMENT IS PUBLIC

Motivation

- Make Mojo JS bindings easier to understand and use
- Reduce differences with C++ bindings to make it easier for C++ developers to work on WebUI and WebUI developers to work on the C++ side. (This doesn't mean adopting C++ patterns but rather using the same nomenclature for the same Mojo concepts)

Summary of C++ Changes

The C++ bindings are in the process of migrating from InterfacePtr, InterfaceRequest, InterfacePtrInfo, etc. to two new names `Remote` and `Receiver`. `Remote` corresponds to the endpoint where interface methods are called and `Receiver` corresponds to the endpoint where interface methods are received. There is a general consensus that these names are clearer, easier to remember, and more accurate.

Summary of Proposed JS changes

- Rename `FooProxy` to `FooRemote`.
- Rename `Foo` to `FooReceiver`.
- Rename `getProxy()` to `getRemote()`.
- Rename `createProxy()` to `bindNewPipeAndPassRemote()`.
- Rename `createRequest()` to `bindNewPipeAndPassReceiver()`.

All changes proposed are simple renames. The shape of the API will stay the same. See "Proposed JS Changes" below for more details about the usage of these concepts, the reasons for renaming them, and some alternatives considered.

Proposed JS changes

Get an interface proxy and use it

```
// Current:
let widget = Widget.getProxy();
```

```
widget.click();

// Proposed:
let widget = Widget.getRemote();
widget.click();
```

Rename `Proxy` to `Remote`: There is consensus that Remote is more accurate. Quoting from the C++ doc: It's short, simple, and conveys the most important information about the object: it calls methods on an implementation of the interface. Also if the name matches C++ then it's one less name to remember.

Passing unbound Proxies (common case)

```
// Current
const callbackRouter = new WidgetObserverCallbackRouter();
widget.addObserver(callbackRouter.createProxy());

// Proposed:
const callbackRouter = new WidgetObserverCallbackRouter();
// |pendingRemote| is a WidgetObserverPendingRemote; PendingRemote indicates that
// it's the "Remote" end of the pipe that needs to be bound.
const pendingRemote = callbackRouter.$.bindNewPipeAndPassRemote();
widget.addObserver(pendingRemote);
```

Rename `createProxy()` to `bindNewPipeAndPassRemote()`: `bindNewPipeAndPassRemote()` is more explicit about what is happening.

Passing unbound Proxies (uncommon case)

Note that this will be uncommon for WebUI since most of WebUI will use CallbackRouter.

```
// Current:
const widgetObserver = new WidgetObserver(this);
widget.addObserver(widgetObserver.createProxy());

// Proposed:
const widgetObserver = new WidgetObserverReceiver(this);
// |pendingRemote| is a WidgetObserverPendingRemote; PendingRemote indicates that
// it's the "Remote" end of the pipe that needs to be bound.
const pendingRemote = widgetObserver.bindNewPipeAndPassRemote();
widget.addObserver(pendingRemote);
```

Two changes:

Abandon `new Foo` in favor of `new FooReceiver`: At first glance, it's hard to tell what `new Foo(this)` is meant to do. Renaming to `FooReceiver(this)` more clearly indicates that this is the receiver end of a pipe and that it forwards calls to `this`. Also matches the C++ name.

Rename `createProxy()` to `bindNewPipeAndPassRemote()`: Similarly, it's unclear what exactly `createProxy()` does. `bindNewPipeAndPassRemote()` more clearly indicates that a new pipe will be bound to that receiver and that a remote will be returned.

Passing Requests

```
// Current:
const widget = new WidgetProxy();
widgetFactory.CreateWidget(widget.$.close());

// Proposed:
const widget = new WidgetRemote();
const pendingReceiver = widget.$.bindNewPipeAndPassReceiver();
widgetFactory.createWidget(pendingReceiver);
```

Rename `createRequest` to `bindNewPipeAndPassReceiver`: With `createRequest()` it's unclear what end of the pipe is being passed to `createWidget`. `bindNewPipeAndPassReceiver()` makes this clearer.