QA Notes

Hot Code Push Reload

Run the leaderboard example, and click on one of the names. Make a change to the leaderboard.html file, see the client reload, and see that the name is still selected.

AUTOUPDATE_VERSION

Set the `AUTOUPDATE_VERSION` environment variable when running the application:

```
$ AUTOUPDATE_VERSION=abc meteor
```

Now when you make an HTML change, it won't appear in the client automatically. (Note the leader list flickers when the server subscription restarts, but that's not a window reload).

Conversely, you can force a client reload (even without making any client code changes) by restarting the server with a new value for `AUTOUPDATE_VERSION`.

No Client Reload on Server-only Change

Revert previous changes and run the example without setting AUTOUPDATE_VERSION.

Note that it might look like the browser is reloading because the page content in the leaderboard example will flicker when the server restarts because the example is using autopublish, but that the window won't actually be reloading.

In the browser console, assign a variable such as `a = true` so that you can easily verify that the client hasn't reloaded.

In the leaderboard example directory, create the `server` directory and add `foo.js`. See in the browser console that `a` is still defined, indicating the browser hasn't reloaded.

Test with the appcache

Add the appcache package:

```
$ meteor add appcache
```

And do the above tests again.

Note that if 1) AUTOUPDATE_VERSION is set so the client doesn't automatically reload, 2) you make a client change, and 3) you manually reload the browser page, you usually *won't* see the updated HTML the *first* time you reload (unless the browser happened to check the app cache manifest between steps 2 and 3). This is normal browser app cache behavior: the browser populates the app cache in the background, so it doesn't wait for new files to download before displaying the web page.

Autoupdate.newClientAvailable

Undo previous changes made, such as by using `git checkout .` Reload the client, which will cause the browser to stop using the app cache.

It's hard to see the `newClientAvailable` reactive variable when the client automatically reloads. Remove the `hot-code-push` package so you can see the variable without having the client also reload.

```
$ meteor remove meteor-base
$ meteor add meteor webapp ddp autoupdate
```

Add to leaderboard.js:

```
Template.leaderboard.helpers({
  available: function () {
    return Autoupdate.newClientAvailable().toString();
  }
});
```

And add `{{available}}` to the leaderboard template in leaderboard.html.

Initially you'll see `false`, and then when you make a change to the leaderboard HTML you'll see the variable change to `true`. (You won't see the new HTML on the client because you disabled reload).

Amusingly, you can undo the addition you made to the HTML and the "new client available" variable will go back to `false` (you now don't have client code available on the server different than what's running in the browser), because by default the client version is based on a hash of the client files.

DDP Version Negotiation Failure

A quick way to test DDP version negotiation failure is to force the client to use the wrong DDP version. At the top of livedata_connection.js:

```
 var Connection = function (url, options) {
    var self = this;
+   options.supportedDDPVersions = ['abc'];
```

You will see the client reload (in the hope that new client code will be available that can successfully negotiation the DDP version). Each reload takes longer than the one before, using an exponential backoff.

If you remove the `options.supportedDDPVersions` line and allow the client to connect (or manually reload the browser page so you don't have to wait), this will reset the exponential backoff counter.

You can verify the counter was reset by adding the line back in a second time, and you'll see the reload cycle start over again with first reloading quickly, and then again taking longer between tries.