Question about resourceCache in _load(url) function in resources.js

Courses
Object-Oriented JavaScript

---

mimi_183382 2015-03-22 23:06:04 UTC #1

Why **resourceCache[url]** is assigned to **false**, and not to **true** in the code below:

(I am a little confused about this technique, and where it is assigned to **true**?

Please explain for me, thank a lot.

I am still unfamiliar with the **engine function** and the **update(dt), render** functions in **engine.js** , when do they run and how can they repeatedly run?)

```
<code>function _load(url) {
    if(resourceCache[url]) {
        return resourceCache[url];
    } else {
        var img = new Image();
        img.onload = function() {
            resourceCache[url] = img;

            if(isReady()) {
            readyCallbacks.forEach(function(func) { func(); });
        }
    };

    resourceCache[url] = false;
    img.src = url;
}
}
</code>
```

Michalicious 2015-03-23 08:12:30 UTC #2

The idea is that `resourceCache[url]` should contain `false` until the image for the specified `url` has finished loading, upon which `resourceCache[url]` should point to **the loaded image**:

- Prior to loading the resource (which occurs in the line `img.src = url;`, which **starts loading the image from the url**), `resourceCache[url]` is set to `false`.
- Only when the image has **finished** loading is the `load` event triggered and the event handler (that has been **registered** for the `img` element with `img.onload = ...`) is executed, setting `resourceCache[url]` to the loaded image with `resourceCache[url] = img;`. At this point, `resourceCache[url]` is "**truthy**" rather than `true`.

As long as `resourceCache[k]` is `false` for any (own) property `k` of the `resourceCache` object, the function call `isReady()` will return `false`. This ensures that `readyCallbacks.forEach(function(func) { func(); });` will only be executed after all images that have **started** loading have also **finished** loading.

As for the logic behind the animation loop, check out

- this article: **Anatomy of a video game**,
- and in particular, this paragraph: **What most browser games should look like**.

Lesson 2 of Udacity's **HTML5 Canvas** course also covers this technique to some extent.

For your orientation:

`Resources.onReady(init);` schedules `init` to be called after all resources have finished loading.

(More precisely, it pushes `init` onto the `readyCallbacks` array so that, at the appropriate time, `readyCallbacks.forEach(function(func) { func(); });` can invoke it.)

`init` in turn calls `main` for the first time.

The `main` function does the following:

- It calls `update` (which in turn calls the user-defined `update` methods on all enemies and on the player) and passes it information about how much time `dt` has passed since `main` was last invoked. You need this information in order to determine how far an enemy (with a given velocity) has moved in the meantime.
- It calls `render` (which draws all background images, the enemies, and the player on the screen. For example, the `render` method for enemies just does `ctx.drawImage(Resources.get(this.sprite), this.x, this.y);`).
- It tells the browser (with `win.requestAnimationFrame(main);`) to call the `main` function **again** when starting a new frame, i.e. after the next **VSync** (while the previous frame is handled by the display). So each call to `main` commissions the subsequent call to `main`, hence the loop.

---

mimi_183382 2015-03-23 17:51:39 UTC #3

Thank you a lot. Your answer is very helpful for me.

ethan1 2015-04-15 01:53:18 UTC #4

Very happy to see this asked and answered. Such a good answer.
