Timing races in Selenium 2: implicit waits vs explicit waits

Jan 14, 2012 // by Daniel Kranowski // Algorithms // 8 comments

The most common debugging experience in automated Selenium testing is the ubiquitous **timing race**, where your code is rushing to access a WebElement that hasn't been fully loaded into the driver yet. If you're stepping through the test one line at a time it will work just fine, but when you let it run in batch mode at its own speed, then it crashes with a complaint like NoSuchElementException. When you've got a suite of a hundred or more Selenium testcases you definitely don't want to step through them one line at a time, it needs to be automated.

The timing race stems from the fact that `WebDriver.get()`, `driver.navigate().to()`, `driver.navigate().refresh()` are nonblocking invocations that start a page load and return immediately, before the load thread is done. Your followup call to driver.findElement(...) will throw NoSuchElementException if the load thread hasn't populated the desired element yet. There are two ways to wait for the element to be ready:

1. **Explicit wait**
   `WebDriverWait.until(condition-that-finds-the-element)`
2. **Implicit wait**
   `driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);`

With the explicit wait you have to insert a WebDriverWait.until(condition) before each findElement(). With the implicit wait it's done for you, globally, after you make the driver timeouts setting. In general I like implicit wait, because the explicit WebDriverWait calls turn into clutter. I can't think of a good reason why implicit wait is not the default.

Here's what the explicit wait looks like:

```
1  public void testExplicitWait(WebDriver driver) {
2     // Assume we're already on a fully loaded webpage.
3     driver.findElement(By.id("myForm")).submit();
4     // Now the next page starts loading.
5
6     // Here's the explicit wait.
7     WebDriverWait wdw = new WebDriverWait(driver, 10);
8     ExpectedCondition<Boolean> condition = new ExpectedCondition<Boolean>() {
9        @Override
10       public Boolean apply(WebDriver d) {
11          WebElement result = d.findElement(By.className("myResult"));
12          return "The Next Page".equals(result.getText());
13          // Returns true as soon as an element of class 'myResult' is found
14          // where the element's text value is "The Next Page".
15       }
16    };
17    wdw.until(condition); // Won't get past here til timeout or element is found
18
19    // It is safe to operate on the element now.
20    WebElement result = driver.findElement(By.className("myResult"));
21    Assert.assertEquals("The Next Page", result.getText());
22 }
```

Compare that to the implicit wait, which is much more concise:

```
1  public void testImplicitWait(WebDriver driver) {
2     // This setup would be done once per driver execution.
3     driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
4
5     driver.findElement(By.id("myForm")).submit();
6     // Now the next page starts loading.
7
8     // An attempt to find the element implicitly waits til it is ready.
9     WebElement result = driver.findElement(By.className("myResult"));
10    Assert.assertEquals("The Next Page", result.getText());
11 }
```

implicitlyWait only affects `findElement`, it does not affect other WebDriver methods like `driver.getCurrentUrl()` or `driver.manage().deleteAllCookies()`. So you can very easily and accidentally get back the old url and/or delete cookies on the old subdomain, not the new url/subdomain you were expecting... unless you call findElement first to force an implicit wait:

```
1  private void forceImplicitWait(WebDriver driver) {
2     driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
3
4     driver.findElement(By.id("myForm")).submit();
5     // Now the next page starts loading ... but it's not done yet.
6
7     printCurrentUrl(driver); // Timing race!  Might print "/oldPath".
8
```

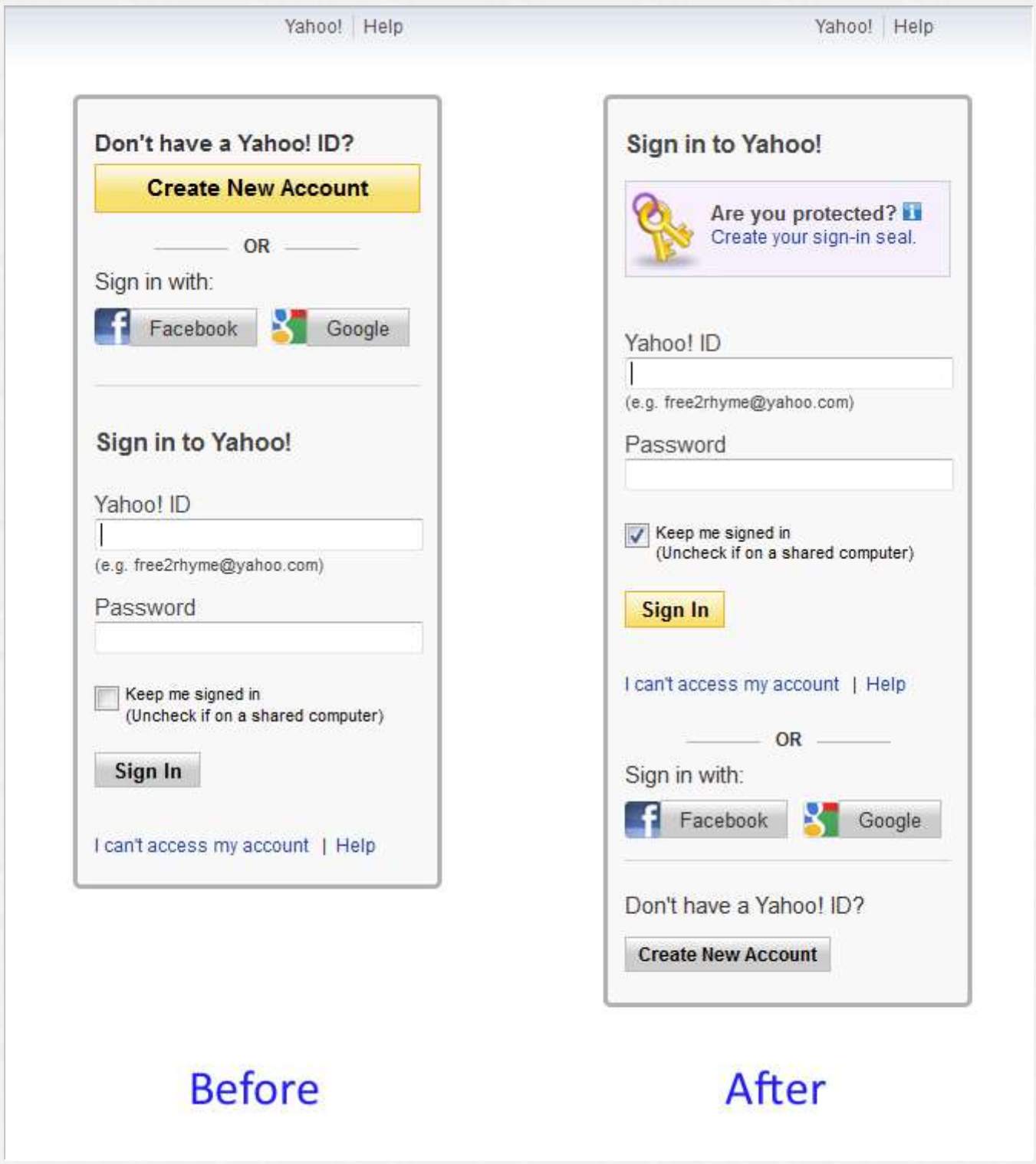
```
    // An attempt to find an element on the next page forces an implicit wait
    // til it is ready.  At that point the driver will have the new URL.
    driver.findElement(By.className("myResult"));

    printCurrentUrl(driver); // Safely prints "/newPath".
}

private void printCurrentUrl(WebDriver driver) {
    URL currentUrl = null;
    try {
        currentUrl = new URL(driver.getCurrentUrl());
    }
    catch (MalformedURLException e) {
        System.out.println("malformedUrl");
    }
    System.out.println(currentUrl.getPath());
}
```

How long to wait?

One question is “what element should I wait for?” It depends on what your test is trying to do, because if the next step is to operate on element X then of course just wait for findElement(By.id(“X”)). If you’re about to operate on X, Y, and Z then you could add three explicit waits, or just add one explicit wait for Z if you know it’s the last one to load, but trying to do “just one wait” will probably make your test more brittle. That’s why I like implicit waits, it lets Selenium do the polling for you, and it waits for exactly the element you want.

There’s a train of thought that sometimes comes up here, “let’s just wait for the whole page to load” and be done with it. Perhaps by hooking into the window.onLoad event. The theory is that you’ll intentionally lose some test performance to gain safety, robustness and reliability in your test suite. It’s a nice idea, but ultimately unhelpful, primarily because onLoad is only the beginning. onLoad is an event that countless webpages are already listening for, whether in `<body onload="doSomething()"/>`, or its jQuery equivalent `$(document).ready(function() { doSomething(); })`, and “doSomething()” often means “modify the DOM.” It’s better to just wait for the specific element on which your test operates, since it’s faster and more accurate.

There are more pitfalls here. Some webpages are so dynamic that an element identified by selector X will render multiple times. A classic example is the login page for Yahoo! Mail, where the username/password form is initialized from HTML, then the page’s JavaScript checks your cookies and then may choose to rip out and replace the username/password form with a brand new element hierarchy. The script operates at human visible speed so you can watch the page adjusting itself. The username/password fields have the same CSS ids before and after the replacement, so there’s a timing race if you write Selenium code to try to automate this login. The test code can’t tell whether it has found the “before” element or the “after” element.



If the Selenium test calls sendKeys() on the Before element, it could get interrupted when the After element loads. It’s almost comical, you’ve written the first half of your username in Before and the second half in After. This of course breaks the test and breaks your automation. The solution is site-specific, not systematic: with intimate

knowledge of the DOM and scripts on this page you can find other clues that tell you whether the element is Before or After. For example you'll wait for the existence of an element that occurs only when After has loaded, or you'll make cookie settings to avoid having After load in the first place. Unfortunately this kind of intimate knowledge leads to brittle tests. You'll want to abstract out this knowledge in a PageObject.

Tags: Selenium

[+] Share & Bookmark

8 Comments

james lapointe June 12, 2013 at 11:56 am

Reply

Great article – it helped me with my understanding

Adam December 20, 2013 at 5:36 pm

Reply

Great post.

Do you happen to know if there's any negative effects to just waiting forever?

I'm working on an interface that will tie into a web application front-end based on a card swipe and prepopulate a few fields (basically a systray app that will manipulate the DOM of Firefox). All the fields are on page 3 of the kiosk web-app, so I would just wait for one of them to appear. But the kiosk could potentially sit there with a FF window open for the weekend until Monday morning when someone comes in and swipes their card.

Thanks!

Daniel Kranowski December 21, 2013 at 1:33 pm

Reply

Looks like all the WebDriverWait constructors have a timeout parameter (long timeOutInSeconds). You could set it to a very large number to avoid timeout over the weekend.

It sounds like you are trying to use Selenium for core (non-test) functionality? You plan to have the card-swipe/systray app use Selenium to launch the browser, then wait for a swipe, then the app populates fields in the browser? Or the swipe unlocks a keyboard and the Selenium instance waits for the user key input? Just wondering.

Umesh May 27, 2014 at 4:57 pm

Reply

This is a good article but it does a favor to Implicit wait. Here are the 2 advantages of using Explicit wait over implicit wait:

1. Elements may require different load time. Let's say one element can be loaded in 30 seconds and second element may take upto 2 minutes to load and a third one can take upto 3 minutes. We can set default timeout to 3 minutes, but is it advisable?

2. Sometimes we have to wait for element to hide. This happens in AJAX when a 'loading' icon is visible on the top and we wait for that icon to hide before moving to next action. This is not achievable using Implicit wait because implicit wait checks for element to enable on DOM and in this scenario, DOM already has that element enabled.

Warn Regards,
Umesh

Vin August 11, 2015 at 8:23 am

Reply

Am a newbie…
my need is similar to your description.
Regarding using 'Explicit', noticed that the declaration is at the beginning of a class. Now we have a verification class, wherein we have methods from different pages. How to do you think we can have different wait times for any unique scenario?

Anand August 21, 2014 at 11:53 pm

Reply

wdw.until(condition); // Won't get past here til timeout or element is found

I am geeting error on this line could you please explain me

D December 17, 2014 at 10:04 am

Reply

Hi,nicely written. Thank you

Please tell me if there is a default polling time for

1. Implicit waits
2.Webdriver wait

If yes, what is it.

Regards
D

Maxim August 5, 2015 at 7:09 am

Reply

Great post.

I don’t have words.
Thank you and write more interesting about webdriver.

Please share your thoughts

Name (required)

Mail (will not be published) (required)

Website

Comment guard: What is 2 + 3? (required)

Comment

POST COMMENT

Notify me of followup comments via e-mail

Technology

- Applications
- Automated Test
- High Performance

Services

- Application Development
- Design and Architecture
- Quality Assurance

Business Value

We're dedicated to strong customer relationships, understanding your business, and delivering only the highest quality software. Our approach to IT excellence is designed to make your organization more effective, more cost-efficient, and more secure.

Site

- Home
- Contact
- RSS Feed