

# Lecture 10

## Talking to The Web: Extroverted Apps (2)

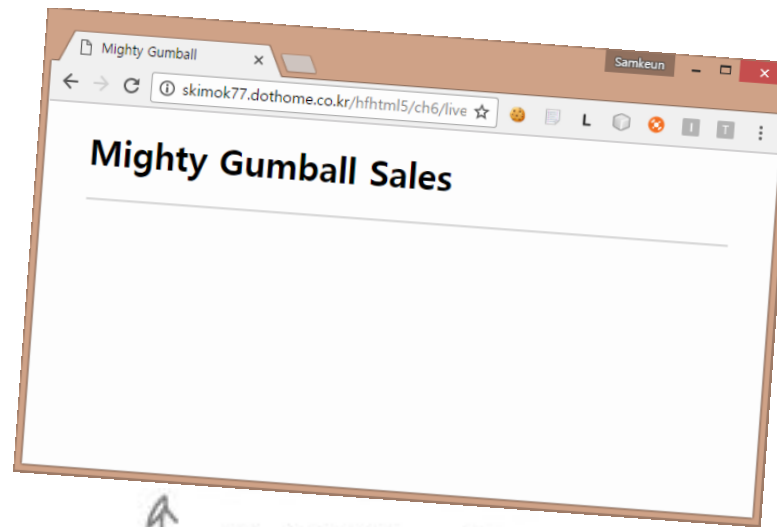
Samkeun Kim <skim@hknu.ac.kr>

<http://cyber.hknu.ac.kr/>

# 실습과제 9-3: A Live Test Drive...



Make sure your URL change is saved in your mightygumball.js file on your server, if you want to keep retrieving your HTML from there, or to your local hard drive if you are using localhost. From there you know what to do: point your browser to your HTML file and watch the live, beautiful, real data from all those people around the world buying Mighty Gumballs!



↑ What?! We're not seeing any data!

[http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/live\\_mightygumball.html](http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/live_mightygumball.html)

Houston, we have a problem!  
Come quick, we're getting no  
sales data since we changed  
to the live servers!



↑  
Ajay, the Upset Quality Assurance Guy

## It's a cliffhanger!

We're not seeing any data in our page. It was all working fine until we moved to the live server...

Will we *find* the problem?

Will we *fix* it?



Stay tuned... we'll answer these questions, and more...

I don't know what's going on with this code, Jim, but it just isn't working for me.



Guys, where were you on the Starbuzz Coffee project? Remember we had a problem with the same behavior. I bet you've got cross-domain issues because you're requesting data from a server that is different than where your page came from. The browser thinks that is a security issue.



# What Browser Security Policy?

Browser does enforce some security around your **XMLHttpRequest** HTTP requests and that can cause some issues.



## What is this policy?

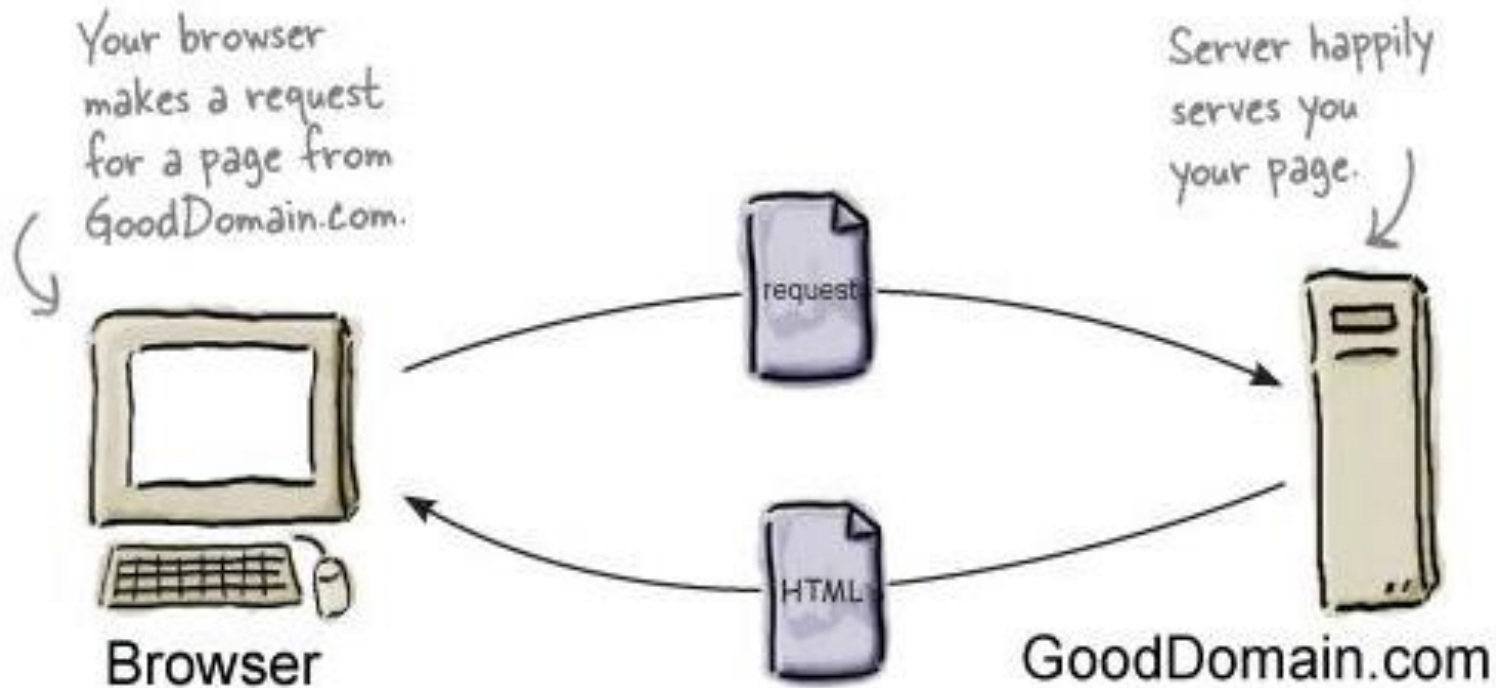
Well, it's a browser policy, and it says you can't retrieve data from a domain that is **different from** the domain the page itself was served from.

Say you're running the site for **DaddyWarBucksBank.com** and someone has hacked into your systems and inserted a bit of JavaScript that takes the user's personal information and does all kinds of interesting things with it by communicating with the server **HackersNeedMoreMoney.com**.

- ✓ 웹 페이지가 서비스되고 있는 오리지널 도메인과 다른 도메인에 XMLHttpRequest 요청 금지

# Acceptable Behavior for JavaScript code

1. Makes a **request** for an HTML page:

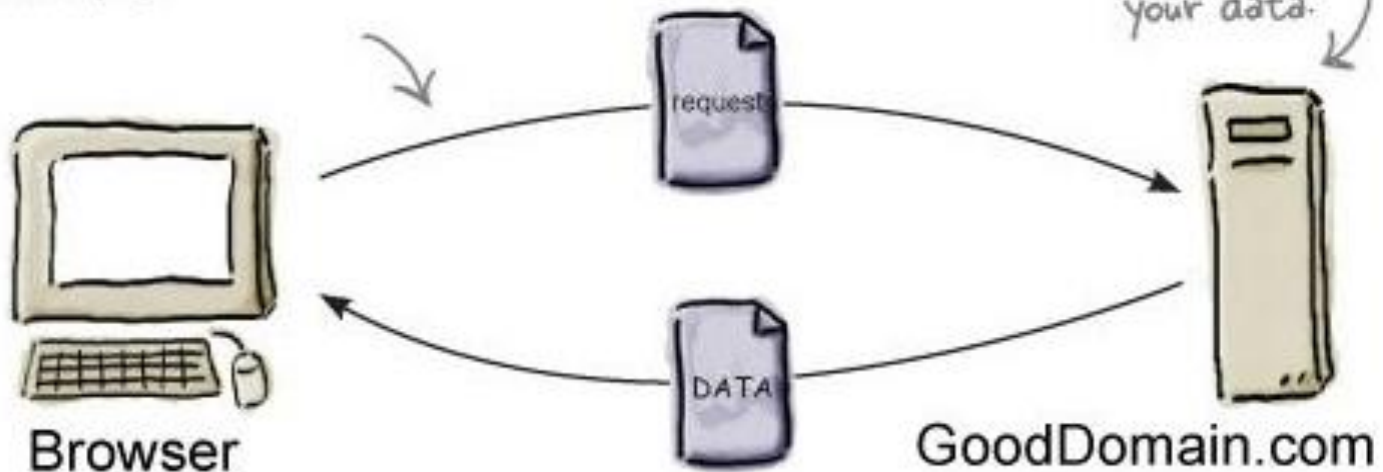




## 2. Makes a **XMLHttpRequest** for the data

This request to get data from GoodDomain.com succeeds because the page and the data are at the same domain.

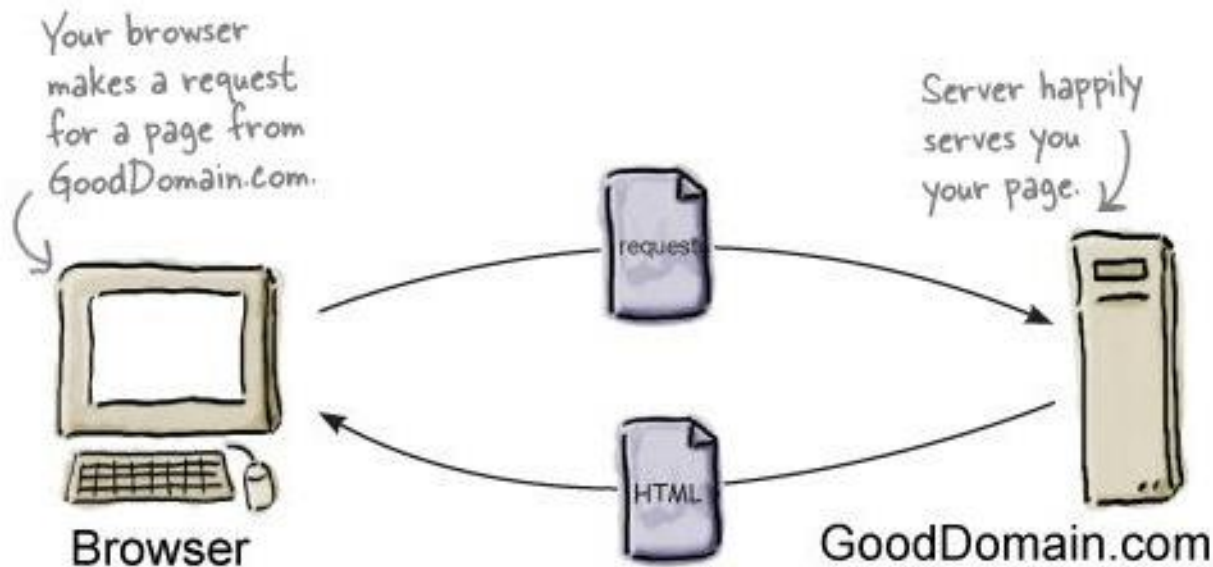
Server happily serves you your data.



# Unacceptable Behavior for JavaScript code

Now let's see what happens when your page hosted at GoodDomain.com tries to make a request for data using XMLHttpRequest to **BadDomain.com** instead.

1. Makes a **request for a page** on GoodDomain.com.



2. But now we have **code that wants data** from BadDomain.com.



그럼, 우리가 만든 파일을 마이티  
검볼 서버에 복사할 수 있나??



## Usually the answer is **yes**.

Say you were a developer working on code for Mighty Gumball, then you'd typically have access to their servers (or to people who could deploy files to the servers for you), and you could place all your files there and avoid any cross-domain issues. In this case, however (and we do hate to break your suspension of disbelief), you're not actually working for Mighty Gumball, you're readers of this book, and **we can't think of a way to have a couple hundred-thousand people copy their files to the Mighty Gumball servers.**

# So, what are our options?

## Plan 1: Use our hosted files.

We've already **put files on our server for you** and placed the files at:

<http://gumball.wickedlysmart.com/gumball/gumball.html>

*Go ahead and give it a try by pointing your browser to this URL and you'll **be able to see the same code you typed in so far** in action and working.*

## Plan 2: Use another way to get the data.

So, **XMLHttpRequest** is a great way to get data into your apps when that data is hosted at the same domain as your app,

But what if you need to really get data from a third party?

Say you need data from **Google** or **Twitter** for instance?

Finding another approach Based on JSON:

⇒ **JSONP** ("JSON with Padding").

JSONP, guys, this is  
our chance to get ahead  
of Judy, for once.





# Meet JSONP

You've probably figured out that **JSONP** is a way to retrieve JSON objects by using the **<script>** tag

It's also a way of retrieving data (again, in the form of JSON objects) that **avoids** the same-origin security issues we saw with XMLHttpRequest

Same origin policy (SOP):

A web page served from server1.example.com cannot normally connect to or communicate with a server other than server1.example.com.

A few exceptions include the HTML **<script>** element

Exploiting the **open policy** for `<script>` elements, some pages use them to retrieve JavaScript code that operates on dynamically generated JSON-formatted data from other origins

## JSONP

Requests for JSONP retrieve not JSON, but **JavaScript code**

⇒ Are evaluated by the JavaScript interpreter, not parsed by a JSON parser

## On the Gumball Server (Same domain)

```
<!doctype html>
<html lang="en">
  ...
  <body>
    <h1>Mighty Gumball Sales</h1>
    <div id="sales">
      </div>
      <script src="http://gumball.wickedlysmart.com/"></script>
    </body>
  </html>
```

The Browser

데이터를 JSON 형태로 제  
공할 웹 서비스 URL

- ① In our HTML we include a `<script>` element. The source for this script is actually the URL of a web service that is going to supply us with JSON for our data, like our Mighty Gumball sales data.

`<script>` 태그를 만나면  
HTTP 요청 !!

String 형태

- ④ The JSON response is in the form of a string, which is parsed and interpreted by the browser. Any data types are turned into real JavaScript object and values, and any code will be executed.

Remember this is just a string  
representation of the object  
at this point!

서버는 요청을 마치  
HTTP 요청처럼 취급 !

- ③ The server treats the request like any HTTP request, and sends back JSON in its response.

JSON

request



Web Service

# But what is the “P” in JSONP for?

OK, the first thing you need to know about JSONP is it has a dumb and non-obvious name: “**JSON with Padding**.”

If we had to name it, we’d call it something like

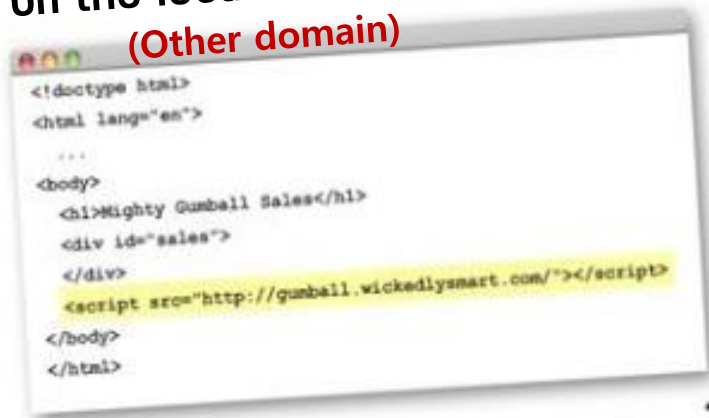
“**JSON with a Callback**” or “get me some JSON and execute it when you get it back”

or, well, really just about anything other than JSON with Padding.

But, all the padding amounts to is wrapping a function around the JSON before it comes back in the request.

On the localhost  
(Other domain)

The Browser



① Same as before, we include a `<script>` element. The source for this script is the URL of a web service that is going to supply us with JSON data.

② Same as before, the browser encounters the `<script>` element in the page and sends an HTTP request to the src URL.

④ This time when the JSON response is parsed and interpreted, it is wrapped in a function call. And so that function is called, and the object created from the JSON string is passed to it.

This time the JSON is wrapped in a function call.

updateSales ( JSON )

③ And as before the server treats the request as normal and sends back JSON, but...this part is a little different.

Before the server sends back the JSON string, it first wraps it in a function call, like a call to `updateSales`.

Web Service

## Web services let you specify a callback function.

- Web services allow you to specify what you want the function to be named
- Mighty Gumball is already supporting a way to do this

## When you specify your URL, add a parameter on the end:

`http://gumball.wickedlysmart.com/?callback=updateSales`

↑  
Here's the usual URL  
we've been using.

And here we've added a URL  
parameter, `callback`, that says to use  
the function `updateSales` when the  
JavaScript is generated.

- ✓ MightyGumball will then use `updateSales` to wrap the JSON formatted object before sending it back to you
- ✓ Typically, web services name this parameter **callback**

# Let's update the Mighty Gumball web app

- It's time to **update** your Mighty Gumball code **with JSONP**.
- Other than removing the existing code that deals with the XMLHttpRequest call, all the changes are minor. Let's make those changes now:

## What we need to do:

1. Remove our XMLHttpRequest code.
2. Make sure the **updateSales** function is ready to receive an object, not a string (as it was with the XMLHttpRequest).
3. Add the `<script>` element to do the actual data retrieval.

# Let's update the Mighty Gumball web app

1. Delete all the code involved in the XMLHttpRequest.  
Open up your **mightygumball.js** file and make these changes:

```
window.onload = function() {  
    var url = "http://gumball.wickedlysmart.com";  
    var request = new XMLHttpRequest();  
    request.open("GET", url);  
    request.onload = function() {  
        if (request.status == 200) {  
            updateSales(request.responseText);  
        }  
    },  
    request.send(null);  
}
```

For now, just delete all the  
code in this function.





2. When we use the <script> element, the browser retrieves JavaScript, parses it and evaluates it.

JSON이 updateSales 함수에 실려서 전달될 때 이제는 스트링 형태가 아니라 **JavaScript 객체 형태**이다.

(XMLHttpRequest에서는 String 형태로 돌려 받았다)

**따라서 스트링이 아닌 객체를 다룰 수 있도록 콜백 함수를 변경하자.**

```
function updateSales(responseText) {
```

```
function updateSales(sales) {
```

```
    var salesDiv = document.getElementById("sales");
```

```
    var sales = JSON.parse(responseText);
```

```
    for (var i = 0; i < sales.length; i++) {
```

```
        var sale = sales[i];
```

```
        var div = document.createElement("div");
```

```
        div.setAttribute("class", "saleItem");
```

```
        div.innerHTML = sale.name + " sold " + sale.sales + " gumballs";
```

```
        salesDiv.appendChild(div);
```

```
    }
```

```
}
```

← Remove responseText and rewrite the line with a parameter named sales.

← And we can delete the JSON.parse call too.

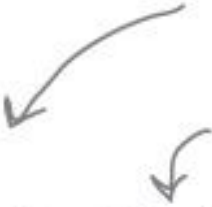
↑ And that's it: we've now got a function ready to handle our data.

Requests for JSONP retrieve **not JSON**, but JavaScript code.

3. And finally, let's add the <script> element to do the actual data retrieval.

```
<!doctype html>
<html lang="en">
<head>
  <title>Mighty Gumball</title>
  <meta charset="utf-8">
  <script src="mightygumball.js"></script>
  <link rel="stylesheet" href="mightygumball.css">
</head>
<body>
  <h1>Mighty Gumball Sales</h1>
  <div id="sales">
  </div>
  <script src="http://gumball.wickedlysmart.com/?callback=updateSales"></script>
</body>
</html>
```

This is the link to the Mighty Gumball web service. We're using the callback parameter and specifying our function, updateSales, so the web service wraps the JSON in a function call to updateSales.



# 실습과제 10-1: Test JSONP-charged code



[http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/jsonp\\_charged\\_mightygumball.html](http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/jsonp_charged_mightygumball.html)

You came up a little short. I thought I was going to see a constantly updated stream of sales from my gumball machines. Sure, I could hit refresh on my browser, but then I see only the newest reports, and only when I manually refresh. That's not what I want!



# Improving Mighty Gumball

As you can see we have a little more work to do, but it's not going to be too bad. Basically, we wrote our first version so that it grabs the latest sales reports from Mighty Gumball and displays them, **once**. Our bad, because almost any web app these days should continuously monitor data and update the app in (near) real time.

Here's what we need to do:

1. Removes the JSONP `<script>` element from the Mighty Gumball HTML.
2. Sets up a handler to handle making the JSONP request every few seconds: JavaScript's **setInterval** method.
3. Implements our JSONP code in the handler:  
Makes a request to get the latest Mighty Gumball sales reports.

# Improving Mighty Gumball

## Step 1: Taking care of the script element...

- Using a new way to invoke our JSONP requests:  
**Remove** the `<script>` element from our HTML.

```
<!doctype html>
<html lang="en">
<head>
  <title>Mighty Gumball</title>
  <meta charset="utf-8">
  <script src="mightygumball.js"></script>
  <link rel="stylesheet" href="mightygumball.css">
</head>
<body>
  <h1>Mighty Gumball Sales</h1>
  <div id="sales">
    </div>
  <script src="http://gumball.wickedlysmart.com/?callback=updateSales"></script>
</body>
</html>
```

You can go ahead and delete this element from your HTML file.

## Step 2: Now it's time for the timer

- Progresses from retrieving the sales reports once, to retrieving them every so often, say every three seconds:
- Has a function we can call every three seconds: **setInterval** method



```
setInterval (handleRefresh, 3000) ;
```

↑  
The `setInterval` method takes a handler and a time interval.

↑  
Here's our handler function, which we'll define in a sec.

↑  
And here's our time interval, expressed in milliseconds.  
3000 milliseconds = 3 seconds.



- Every 3,000 milliseconds JavaScript will invoke your handler (**handleRefresh**)

```
function handleRefresh() {  
    alert("I'm alive");  
}
```

← Every time this is called (which will be every three seconds), we'll throw up the alert "I'm alive."

- Sets up the **setInterval** call, which we'll add to the **onload** function so it gets set up right after the entire page is loaded:

```
window.onload = function() {  
    setInterval(handleRefresh, 3000);  
}
```

← This is our old onload function, which had nothing in it after we deleted the XMLHttpRequest code.

↑ And all we need to do is add our call to setInterval, which, when the init function is run, will start a timer that fires every three seconds and calls our function handleRefresh.

# 실습과제 10-2: I'm Alive

This should be fun. Make sure you've typed in the handleRefresh function and also made the changes to the onload handler. Save everything and load it into your browser. You'll see a stream of alerts, and you'll have to close your browser window to stop it!



[http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/improving\\_mightygumball.html](http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/improving_mightygumball.html)



## Step 3: Reimplementing JSONP

- Can insert new elements into the DOM at any time, even <script> elements

### **First, let's set up the JSONP URL**

- Assigns it to a variable for later use
- Deletes the alert out of your handler and add this code:

We're back in our  
handleRefresh function.



```
function handleRefresh() {
```

```
    var url = "http://gumball.wickedlysmart.com?callback=updateSales";
```

```
}
```

Here, we're setting up the JSONP URL  
and assigning it to the variable url.



## Next, let's create a new script element

- **Builds a <script> element using JavaScript:**  
Creates the element and then sets its src and id attributes:

```
function handleRefresh() {  
    var url = "http://gumball.wickedlysmart.com?callback=updateSales";  
  
    var newScriptElement = document.createElement("script");  
    newScriptElement.setAttribute("src", url);  
    newScriptElement.setAttribute("id", "jsonp");  
}
```

First, we create a new script element...

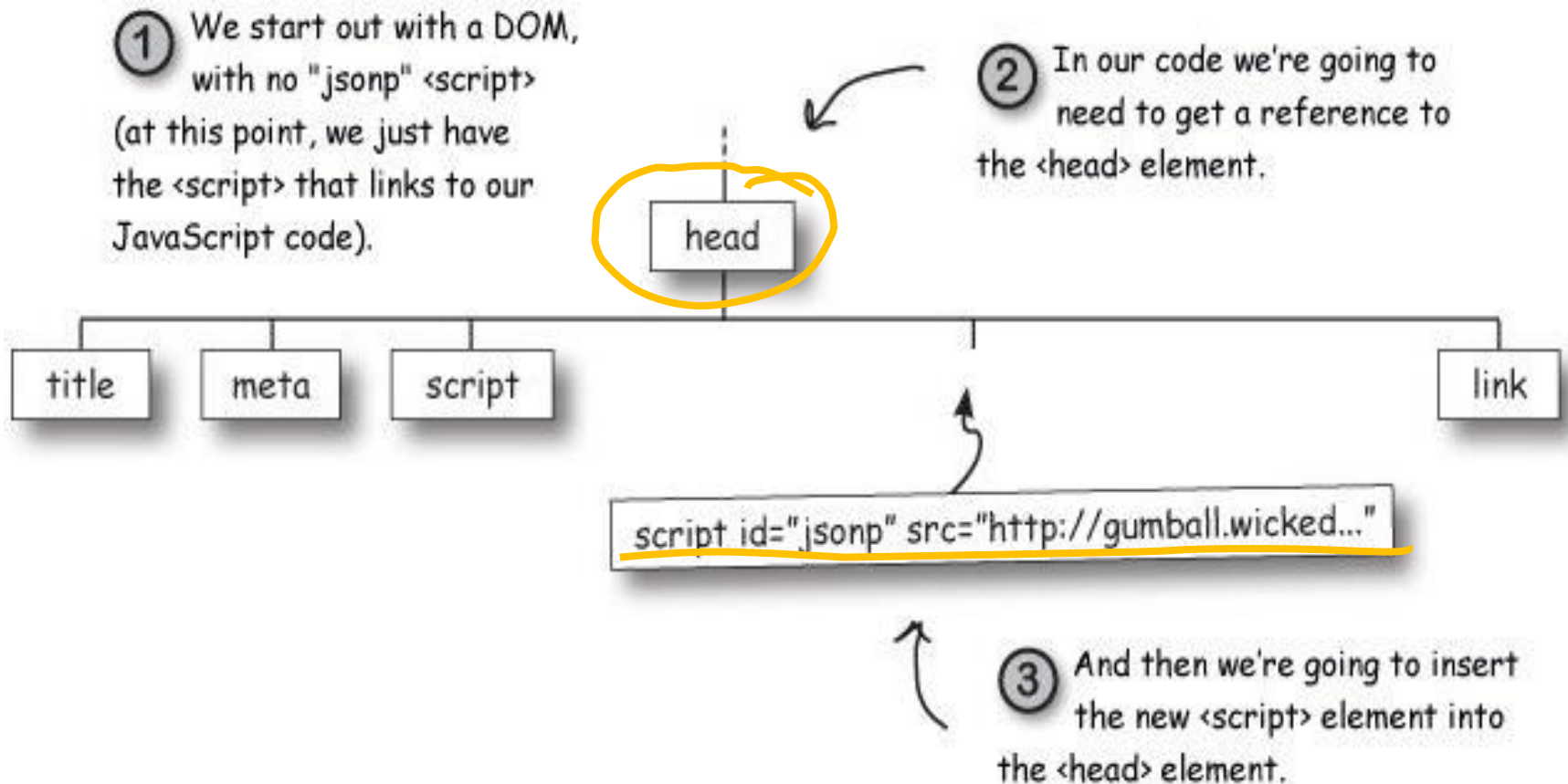
... and set the src attribute of the element to our JSONP URL.

And we'll give this script an id so we can easily get it again, which we'll need to, as you'll see.

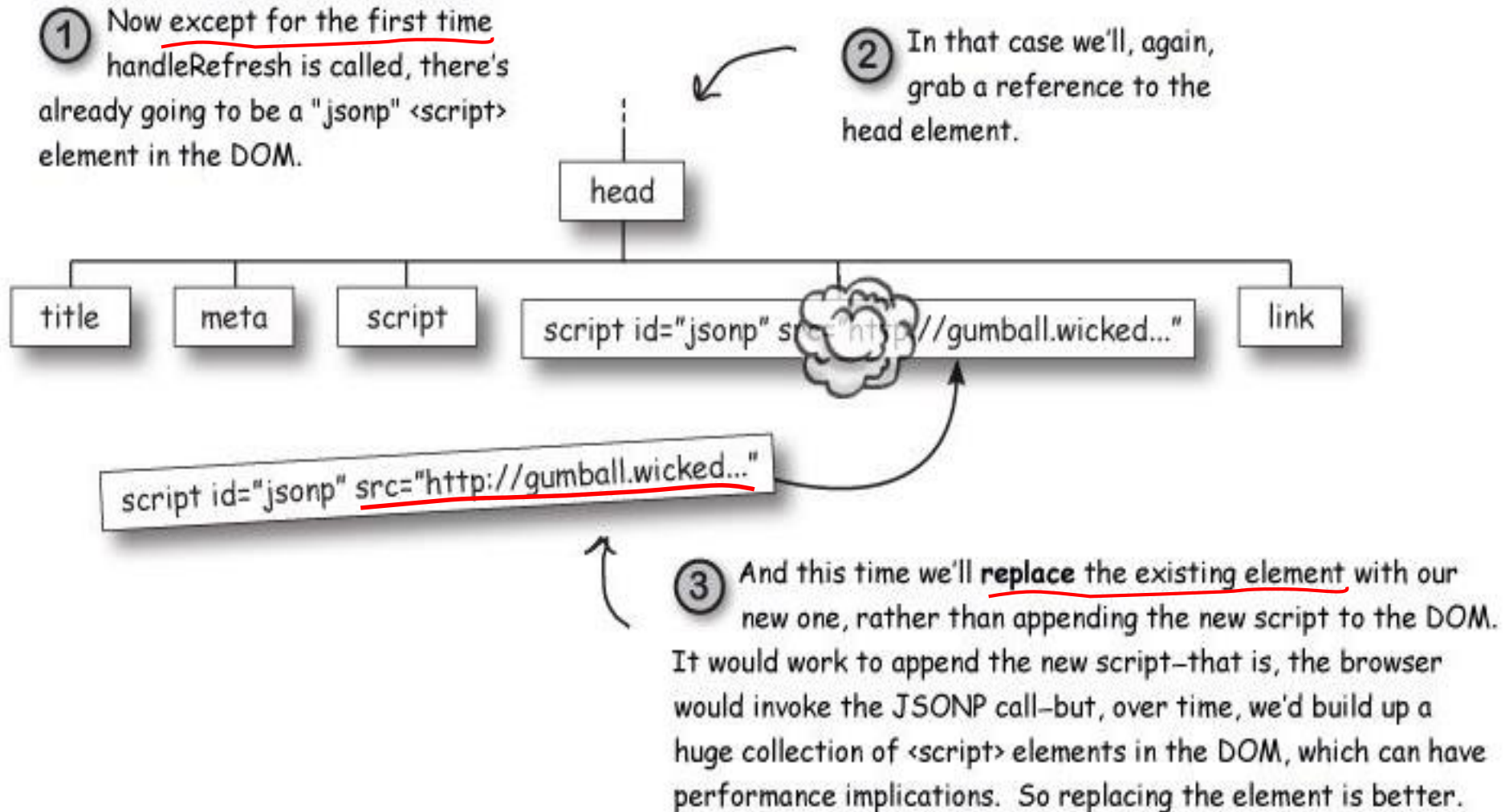
The `setAttribute` method might look new to you (we've only mentioned it in passing so far), but it's pretty easy to see what it does. The `setAttribute` method allows you to set the attributes of an HTML element, like the `src` and `id` attributes or a number of others including `class`, `href` and so on.

# How do we insert the script into the DOM?

- Inserts our newly created script element
- Causes the JSONP request to be made



Once we've inserted the script, the browser will see the new script in the DOM and go retrieve what's at the URL in the **src** attribute.





# Now let's write the code to insert the script into the DOM

- Does this in two steps:  
Code to add a new script and code to replace a script:

```
function handleRefresh() {  
    var url = "http://gumball.wickedlysmart.com?callback=updateSales";
```

```
    var newScriptElement = document.createElement("script");  
    newScriptElement.setAttribute("src", url);  
    newScriptElement.setAttribute("id", "jsonp");
```

We're first going to get the reference to the `<script>` element. If doesn't exist, we'll get back null. Notice we're counting on it having the id "jsonp."

```
    var oldScriptElement = document.getElementById("jsonp");  
    var head = document.getElementsByTagName("head")[0];  
    if (oldScriptElement == null) {  
        head.appendChild(newScriptElement);  
    }
```

Next we're going to get a reference to the `<head>` element using a new document method. We'll come back to how this works, but for now just know it gets a reference to the `<head>` element.

Now that we have a reference to the `<head>` element, we check to see if there is already a `<script>` element, and if there isn't (if its reference is null) then we go ahead and append the new `<script>` element to the head.



- Check out the code that replaces the script element if it already exists:

Just first time!

Here's our conditional again, remember it is just checking to see if a `<script>` element already exists in the DOM.

```
if (oldScriptElement == null) {  
    head.appendChild(newScriptElement);  
} else {  
    head.replaceChild(newScriptElement, oldScriptElement);  
}
```

If there is already a `<script>` element in the head, then we just replace it. You can use the `replaceChild` method on the `<head>` element and pass it the old and new scripts to do that. We'll look a little more closely at this new method in a sec.

## getElementsByTagName method:

Returns an array of elements that match a given tag name.

getElementsByTagName returns all the  
elements in the DOM with this tag.



```
var arrayOfHeadElements = document.getElementsByTagName("head");
```

In this case it returns an array  
of head elements.



Can get the first item in it using index 0:

```
var head = arrayOfHeadElements[0];
```



Returns the first head element  
in the array (and there should  
be only one, right?).

Now we can combine these two lines, like this:

```
var head = document.getElementsByTagName("head")[0];
```

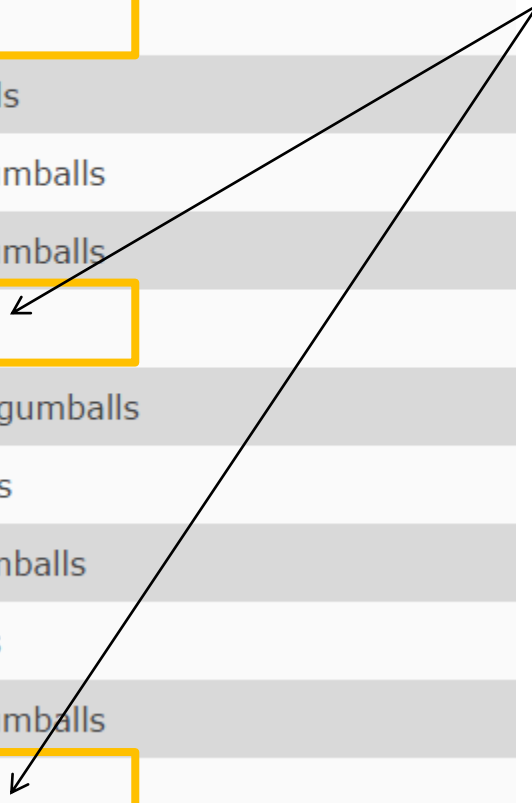
We get the array and then index into the array to get the first item in one step.

In our code example, we're always using the first <head> element but you can use this method on any tag, like <p>, <div> and so on.

## Mighty Gumball Sales

|                               |
|-------------------------------|
| SACRAMENTO sold 7 gumballs    |
| APTOS sold 4 gumballs         |
| OXNARD sold 6 gumballs        |
| SACRAMENTO sold 9 gumballs    |
| SACRAMENTO sold 7 gumballs    |
| APTOS sold 4 gumballs         |
| POLLOCK PINES sold 3 gumballs |
| COYOTE sold 8 gumballs        |
| MILL VALLEY sold 1 gumballs   |
| PIEDRA sold 5 gumballs        |
| SACRAMENTO sold 7 gumballs    |
| APTOS sold 4 gumballs         |

From cache



[http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/JSONP/mightygumball\\_cache.html](http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/JSONP/mightygumball_cache.html)

# Watch out for the **dreaded browser cache**

- Browser ends up caching it for efficiency, and so you just get the same cached file (or data) back over and over
- Luckily there is an easy and old-as-the-Web cure for this:
  - All we do is add a **random number** onto the end of the URL
  - Then the browser is tricked into a new URL the browser's never seen before:

You'll find this code at the top of your `handleRefresh` function.



Change your URL declaration above to look like this.



```
var url = "http://gumball.wickedlysmart.com/?callback=updateSales" +  
          "&random=" + (new Date()).getTime();
```



We're adding a new, "dummy" parameter on the end of the URL. The web server will just ignore it, but it's enough to fake out the browser.



We create a new `Date` object, use the `getTime` method of the `Date` object to get the time in milliseconds, and then add the time to the end of the URL.

With this new code, the generated URL will look something like this:

This part should look familiar...  
↓  
`http://gumball.wickedlysmart.com?callback=updateSales&random=1309217501707`  
↓  
And here's the random parameter.  
↑  
This part will change each time to defeat caching.

Go ahead and replace the **url** variable declaration in your **handleRefresh** function with the code and then we'll be ready for a test drive!

# 실습과제 10-3 One more TIME test drive (Duplicated)

Alright, surely we've thought of everything this time. We should be all ready. Make sure you've got all the code in since the last test drive, and reload the page. Wow, we're seeing continuous updates!

Wait a sec... are you seeing what we're seeing? What are these duplicates? That's not good. Hmm. Maybe we're retrieving too fast and getting reports we've already retrieved?

Per 3 seconds!

Duplicates!

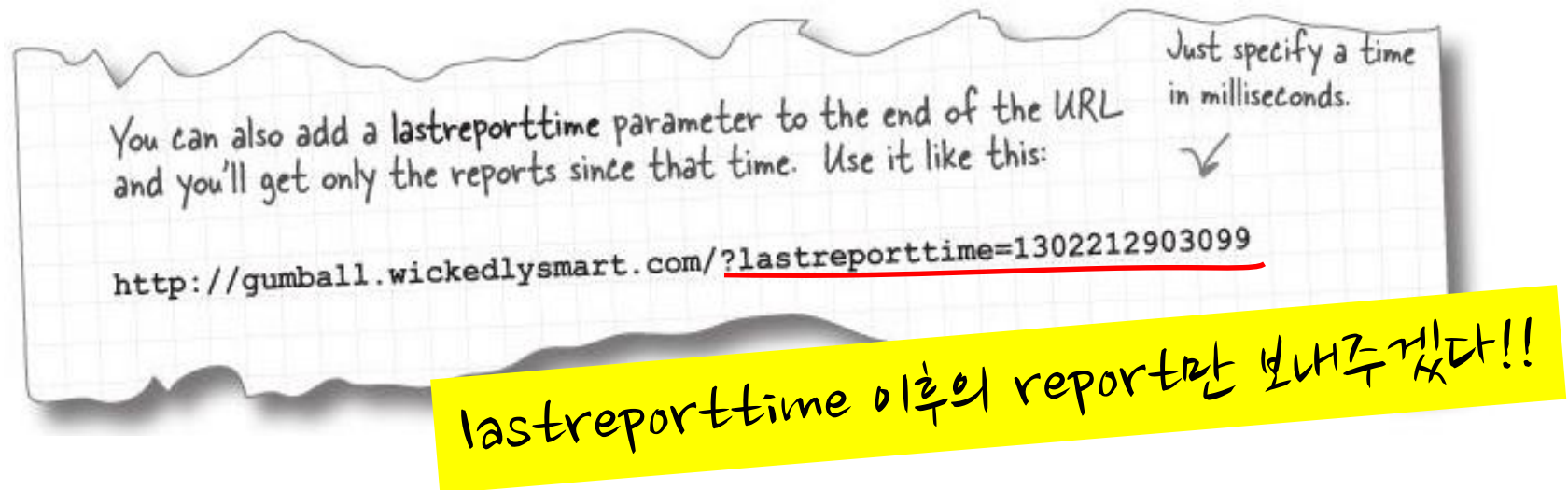
The screenshot shows a web browser window with the address bar displaying 'll\_timeplus.html'. The page title is 'Mighty Gumball Sales'. The content is a list of gumball sales by city. The list contains 18 entries, with several cities appearing multiple times. Handwritten notes and arrows highlight the problem of duplicates. A note on the left says 'Wait a sec... are you seeing what we're seeing? What are these duplicates? That's not good. Hmm. Maybe we're retrieving too fast and getting reports we've already retrieved?' with 'fast' and 'already retrieved' underlined. A red arrow points from this note to the 'Duplicates!' text. Another red arrow points from 'Duplicates!' to the list of sales. Blue and yellow arrows point from the note to specific entries in the list: 'HAYWARD sold 9 gumballs', 'ORLAND sold 3 gumballs', 'HAYWARD sold 9 gumballs', 'ORLAND sold 3 gumballs', 'ORLAND sold 3 gumballs', 'LOS ANGELES sold 2 gumballs', 'ORLAND sold 3 gumballs', 'LOS ANGELES sold 2 gumballs', 'LOS ANGELES sold 2 gumballs'. Red arrows point from the 'Duplicates!' text to the 'LOS ANGELES sold 2 gumballs' entries. The list of sales is as follows:

| City          | Sales      |
|---------------|------------|
| HAYWARD       | 9 gumballs |
| ORLAND        | 3 gumballs |
| HAYWARD       | 9 gumballs |
| ORLAND        | 3 gumballs |
| ORLAND        | 3 gumballs |
| LOS ANGELES   | 2 gumballs |
| ORLAND        | 3 gumballs |
| LOS ANGELES   | 2 gumballs |
| LOS ANGELES   | 2 gumballs |
| REDWOOD CITY  | 7 gumballs |
| REDWOOD CITY  | 7 gumballs |
| STOCKTON      | 7 gumballs |
| REDWOOD CITY  | 7 gumballs |
| STOCKTON      | 7 gumballs |
| STOCKTON      | 7 gumballs |
| SAN FRANCISCO | 6 gumballs |
| SAN FRANCISCO | 6 gumballs |
| AUBURN        | 1 gumballs |
| AUBURN        | 1 gumballs |
| NORTHBRIDGE   | 5 gumballs |

[http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/JSONP/mightygumball\\_timeplus.html](http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/JSONP/mightygumball_timeplus.html)

# How to remove duplicate sales reports

If you take a quick look back at the Gumball Specs in A quick example using JSON, you'll see that you can specify a **lastreporttime** parameter in the request URL, like this:



That's a great, but how do we know the time of the last report we've retrieved? Let's look at the format of the sales reports again:

```
[{"name": "LOS ANGELES", "time": 1309208126092, "sales": 2},  
 {"name": "PASADENA", "time": 1309208128219, "sales": 8},  
 {"name": "DAVIS CREEK", "time": 1309214414505, "sales": 8},  
 ...]
```

Each sales report has the time it was reported.





I see where you're going with this; we can keep track of the time of the last report, and then use that when we make the next request so that the server doesn't give us reports we've already received?

## You got it.

And to keep track of the last sales report received we're going to need to **make some additions to the updateSales function**, where all the processing of the sales data happens. First, though, we should declare a variable to hold the time of the most recent report:

```
var lastReportTime = 0;
```

← The time can't be less than zero, so let's just set it to 0 for starters.

Put this at the top of your JavaScript file, outside any function.

And let's grab the time of the most recent sale in **updateSales**:

```
function updateSales(sales) {  
    var salesDiv = document.getElementById("sales");  
    for (var i = 0; i < sales.length; i++) {  
        var sale = sales[i];  
        var div = document.createElement("div");  
        div.setAttribute("class", "saleItem");  
        div.innerHTML = sale.name + " sold " + sale.sales +  
            " gumballs";  
        salesDiv.appendChild(div);  
    }  
    if (sales.length > 0) {  
        lastReportTime = sales[sales.length-1].time;  
    }  
}
```

If you look at the sales array, you'll see that the most recent sale is the last one in the array. So here we're assigning that to our variable lastReportTime.

We need to make sure there IS an array though; if there are no new sales, then we'd get back an empty array and our code here would cause an exception.

# Updating the JSON URL to include the **lastreporttime**

Edits the handleRefresh function, and add the **lastreporttime** query parameter like this:

```
function handleRefresh() {  
    var url = "http://gumball.wickedlysmart.com" +  
        "?callback=updateSales" +  
        "&lastreporttime=" + lastReportTime +  
        "&random=" + (new Date()).getTime();  
    var newScriptElement = document.createElement("script");  
    newScriptElement.setAttribute("src", url);  
    newScriptElement.setAttribute("id", "jsonp");  
    var oldScriptElement = document.getElementById("jsonp");  
    var head = document.getElementsByTagName("head")[0];  
    if (oldScriptElement == null) {  
        head.appendChild(newScriptElement);  
    }  
    else {  
        head.replaceChild(newScriptElement, oldScriptElement);  
    }  
}
```

We've split up the URL into several strings that we're concatenating together...

... and here's the lastreporttime parameter with its new value.

# 실습과제 10-4 Test drive lastReportTime



Let's take the `lastreporttime` query parameter for a test run and see if it solves our duplicate sales reports problem. Make sure you type in the new code, reload the page, and click that refresh button.

| Mighty Gumball Sales |            |
|----------------------|------------|
| BELLFLOWER           | 8 gumballs |
| LA MESA              | 1 gumballs |
| VAN NUYS             | 7 gumballs |
| VALLEJO              | 4 gumballs |
| SAN FRANCISCO        | 5 gumballs |
| IRVINE               | 8 gumballs |
| VENTURA              | 1 gumballs |
| SAN JOSE             | 8 gumballs |
| VAN NUYS             | 8 gumballs |
| SAN BERNARDINO       | 4 gumballs |

<http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch6/JSONP/mightygumball.html>

# Q & A

