How to find whether a particular string has unicode characters (esp. Double Byte characters)

Asked 16 years, 2 months ago Modified 6 years, 1 month ago

Viewed 55k times

40

To be more precise, I need to know whether (and if possible, how) I can find whether a given string has double byte characters or not. Basically, I need to open a pop-up to display a given text which can contain double byte characters, like Chinese or Japanese. In this case, we need to adjust the window size than it would be for English or ASCII. Anyone has a clue?

javascript unicode double-byte

Share

Improve this question

Follow

edited Oct 14, 2014 at 13:42

brasofilo

26k ● 15 ● 93 ● 184

asked Sep 29, 2008 at 7:39

Jay

469 ● 1 ● 4 ● 7

Well, I expected this to work. But it didn't work in IE. I guess some layout problems. Anyways, since the code to compute

the text-to-be-shown length and height/width was already there, I went ahead with the code that just finds whether there is a double byte character or not. And this solved.
– Jay Sep 30, 2008 at 5:08

With HTML5, you can use the context of a Canvas element (`var ctx = canvas.getContext('2d')`) to obtain the width text metric. `var text_width = ctx.measureText(text).width;` I'm not sure how well this method works with unicode characters, and its a shame that all the `measureText` method currently returns is width.
– WebWanderer Dec 2, 2015 at 21:14

6 Answers

Sorted by: Highest score (default)

53

I used mikesamuel answer on this one. However I noticed perhaps because of this form that there should only be one escape slash before the `u`, e.g. `\u` and not `\\u` to make this work correctly.

```
function containsNonLatinCodepoints(s) {
    return /[^\u0000-\u00ff]/.test(s);
}
```

Works for me :)

Share Improve this answer

Follow

edited Nov 9, 2009 at 5:14

sth

229k ● 56 ● 286 ● 368

answered Nov 8, 2009 at 20:06

james

531 ● 4 ● 2

3 Your function is much better than the ticked answer, regex is always better – AmerllicA May 29, 2017 at 12:59

this works for me too, using regrex is better in performance than using a loop as well. – Tai Vu Apr 19, 2023 at 4:29

34

JavaScript holds text internally as UCS-2, which can encode a fairly extensive subset of Unicode.

But that's not really germane to your question. One solution might be to loop through the string and examine the character codes at each position:

```
function isDoubleByte(str) {
    for (var i = 0, n = str.length; i < n; i++) {
        if (str.charCodeAt( i ) > 255) { return true;
    }
    return false;
}
```

This might not be as fast as you would like.

Share Improve this answer

Follow

edited Jan 13, 2012 at 11:48

Cheers and hth. - Alf

145k ● 15 ● 214 ● 339

answered Sep 29, 2008 at 13:18

pcorcoran

8,052 ● 6 ● 30 ● 26

I don't know JavaScript, but don't you mean UTF-16? There is no such thing as UCS-16; there were UCS-x encoding forms, now obsolete, in the ISO/IEC 10646 standard that's equivalent to Unicode. UCS-2 used exactly two bytes and could thus represent the first 2^16 Unicode characters. UTF-16, on the contrary, uses 16-bit units, but not necessarily a single one of those. All Unicode characters can be represented as UTF-16 byte sequences. – Arthur Reutenauer Nov 8, 2009 at 20:21

16

I have benchmarked the two functions in the top answers and thought I would share the results. Here is the test code I used:

```
const text1 = `The Chinese Wikipedia was established a
Wikipedias in May 2001. 中文維基百科的副標題是「海納百川，有
則徐（1785年－1850年）於1839年為`;

const regex = /[^\u0000-\u00ff]/; // Small performance
the regex
function containsNonLatinCodepoints(s) {
    return regex.test(s);
}

function isDoubleByte(str) {
    for (var i = 0, n = str.length; i < n; i++) {
        if (str.charCodeAt( i ) > 255) { return true;
    }
```

```
    return false;
}

function benchmark(fn, str) {
    let startTime = new Date();
    for (let i = 0; i < 10000000; i++) {
        fn(str);
    }
    let endTime = new Date();

    return endTime.getTime() - startTime.getTime();
}

console.info('isDoubleByte => ' + benchmark(isDoubleBy
console.info('containsNonLatinCodepoints => ' +
benchmark(containsNonLatinCodepoints, text1));
```

When running this I got:

```
isDoubleByte => 2421
containsNonLatinCodepoints => 868
```

So for this particular string the regex solution is about 3 times faster.

However note that for a string where the first character is unicode, `isDoubleByte()` returns right away and so is much faster than the regex (which still has the overhead of the regular expression).

For instance for the string `中国`, I got these results:

```
isDoubleByte => 51
containsNonLatinCodepoints => 288
```

To get the best of both world, it's probably better to combine both:

```
var regex = /[^\u0000-\u00ff]/; // Small performance g
the regex
function containsDoubleByte(str) {
    if (!str.length) return false;
    if (str.charCodeAt(0) > 255) return true;
    return regex.test(str);
}
```

In that case, if the first character is Chinese (which is likely if the whole text is Chinese), the function will be fast and return right away. If not, it will run the regex, which is still faster than checking each character individually.

Share Improve this answer

Follow

answered Oct 12, 2017 at 21:30

laurent

90.6k ● 82 ● 306 ● 438

7

Here is benchmark test: http://jsben.ch/NKjKd

This is much faster:

```
function containsNonLatinCodepoints(s) {
    return /[^\u0000-\u00ff]/.test(s);
}
```

than this:

```
function isDoubleByte(str) {
    for (var i = 0, n = str.length; i < n; i++) {
        if (str.charCodeAt( i ) > 255) { return true;
```

```
    }
    return false;
}
```

Share Improve this answer Follow

answered Nov 21, 2018 at 21:29

David Dehghan
24.7k ● 11 ● 110 ● 101

Awesome! So many thanks! It helped in making a crypto library sodium free – jolly Jan 7, 2019 at 5:41

2 @jolly Sodium-free? – Cog Nov 5, 2020 at 23:04

6

Actually, all of the characters are Unicode, at least from the Javascript engine's perspective.

Unfortunately, the mere presence of characters in a particular Unicode range won't be enough to determine you need more space. There are a number of characters which take up roughly the same amount of space as other characters which have Unicode codepoints well above the ASCII range. Typographic quotes, characters with diacritics, certain punctuation symbols, and various currency symbols are outside of the low ASCII range and are allocated in quite disparate places on the Unicode basic multilingual plane.

Generally, projects that I've worked on elect to provide extra space for all languages, or sometimes use javascript to determine whether a window with auto-

scrollbar css attributes actually has content with a height which would trigger a scrollbar or not.

If detecting the presence of, or count of, CJK characters will be adequate to determine you need a bit of extra space, you could construct a regex using the following ranges: [\u3300-\u9fff\uf900-\ufaff], and use that to extract a count of the number of characters that match. (This is a little excessively coarse, and misses all the non-BMP cases, probably excludes some other relevant ranges, and most likely includes some irrelevant characters, but it's a starting point).

Again, you're only going to be able to manage a rough heuristic without something along the lines of a full text rendering engine, because what you really want is something like GDI's MeasureString (or any other text rendering engine's equivalent). It's been a while since I've done so, but I think the closest HTML/DOM equivalent is setting a width on a div and requesting the height (cut and paste reuse, so apologies if this contains errors):

```
o = document.getElementById("test");

document.defaultView.getComputedStyle(o,"").getPropert
```


edited Sep 29, 2008 at 8:48

community wiki

3 revs, 2 users 91%
JasonTrue

0

Why not let the window resize itself based on the runtime height/width?

Run something like this in your pop-up:

```
window.resizeTo(document.body.clientWidth, document.bo
```

Share Improve this answer

Follow

answered Sep 29, 2008 at 7:53

Oli

240k ● 65 ● 226 ● 303

Something like this should work in non-pathological cases; of course you'd need to make sure you're not exceeding the available screen space, or at least assume reasonable limits.
– JasonTrue Sep 29, 2008 at 8:12