String.prototype.charCodeAt()

Share: Twitter Facebook Google+

The **`charCodeAt()`** method returns the numeric Unicode value of the character at the given index (except for unicode codepoints > 0x10000).

Syntax

```
str.charCodeAt(index)
```

Parameters

`index`
An integer greater than or equal to 0 and less than the length of the string; if it is not a number, it defaults to 0.

Description

Unicode code points range from 0 to 1114111 (0x10FFFF). The first 128 Unicode code points are a direct match of the ASCII character encoding. For information on Unicode, see the JavaScript Guide.

Note that `charCodeAt()` will always return a value that is less than 65536. This is because the higher code points are represented by a pair of (lower valued) "surrogate" pseudo-characters which are used to comprise the real character. Because of this, in order to examine or reproduce the full character for individual characters of value 65536 and above, for such characters, it is necessary to retrieve not only `charCodeAt(i)`, but also `charCodeAt(i+1)` (as if examining/reproducing a string with two letters). See example 2 and 3 below.

`charCodeAt()` returns `NaN` if the given index is less than 0 or is equal to or greater than the length of the string.

Backward compatibilty: In historic versions (like JavaScript 1.2) the `charCodeAt()` method returns a number indicating the ISO-Latin-1 codeset value of the character at the given index. The ISO-Latin-1 codeset ranges from 0 to 255. The first 0 to 127 are a direct match of the ASCII character set.

Examples

Using `charCodeAt()`

The following example returns 65, the Unicode value for A.

```
'ABC'.charCodeAt(0); // returns 65
```

Fixing `charCodeAt()` to handle non-Basic-Multilingual-Plane characters if their presence earlier in the string is unknown

This version might be used in for loops and the like when it is unknown whether non-BMP characters exist before the specified index position.

```
function fixedCharCodeAt(str, idx) {
  // ex. fixedCharCodeAt('\uD800\uDC00', 0); // 65536
  // ex. fixedCharCodeAt('\uD800\uDC00', 1); // false
  idx = idx || 0;
  var code = str.charCodeAt(idx);
  var hi, low;

  // High surrogate (could change last hex to 0xDB7F to treat high
  // private surrogates as single characters)
  if (0xD800 <= code && code <= 0xDBFF) {
    hi = code;
    low = str.charCodeAt(idx + 1);
    if (isNaN(low)) {
      throw 'High surrogate not followed by low surrogate in fixedCharCodeAt()';
    }
    return ((hi - 0xD800) * 0x400) + (low - 0xDC00) + 0x10000;
  }
  if (0xDC00 <= code && code <= 0xDFFF) { // Low surrogate
    // We return false to allow loops to skip this iteration since should have
    // already handled high surrogate above in the previous iteration
    return false;
    /*hi = str.charCodeAt(idx - 1);
    low = code;
    return ((hi - 0xD800) * 0x400) + (low - 0xDC00) + 0x10000;*/
  }
  return code;
}
```

Fixing `charCodeAt()` to handle non-Basic-Multilingual-Plane characters if their presence earlier in the string is known

```
function knownCharCodeAt(str, idx) {
  str += '';
  var code,
      end = str.length;

  var surrogatePairs = /[\uD800-\uDBFF][\uDC00-\uDFFF]/g;
  while ((surrogatePairs.exec(str)) != null) {
    var li = surrogatePairs.lastIndex;
    if (li - 2 < idx) {
      idx++;
    }
    else {
      break;
    }
  }

  if (idx >= end || idx < 0) {
    return NaN;
  }

  code = str.charCodeAt(idx);

  var hi, low;
  if (0xD800 <= code && code <= 0xDBFF) {
    hi = code;
    low = str.charCodeAt(idx + 1);
    // Go one further, since one of the "characters" is part of a surrogate pair
    return ((hi - 0xD800) * 0x400) + (low - 0xDC00) + 0x10000;
  }
  return code;
}
```

Specifications

Specification	Status	Comment
ECMAScript 1st Edition (ECMA-262)	ST Standard	Initial definition. Implemented in JavaScript 1.2.
ECMAScript 5.1 (ECMA-262) The definition of 'String.prototype.charCodeAt' in that specification.	ST Standard	
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'String.prototype.charCodeAt' in that specification.	ST Standard	

Browser compatibility

Desktop Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	(Yes)	(Yes)	(Yes)	(Yes)	(Yes)

See also

- `String.fromCharCode()`
- `String.prototype.charAt()`
- `String.fromCodePoint()`
- `String.prototype.codePointAt()`