

# XSS対策としての ES6テンプレートリテラル




Shibuya.XSS  
Yosuke HASEGAWA

# ES6テンプレートリテラル



Chromium Blog: Chromium x

blog.chromium.org/2015/01/chrome-41-beta-new-es6-features-and.html




## Chromium Blog

News and developments from the open source browser project

### Chrome 41 Beta: New ES6 Features and Improved DevTools for Service Workers and Web Animations

Posted: Thursday, January 22, 2015

 455  471  いいね!

Today's Chrome Beta channel release includes new Javascript ES6 features and improved workflows for debugging [Service Workers](#) and [Web Animations](#). Unless otherwise noted, changes described below apply to Chrome for Android, Windows, Mac, Linux, and Chrome OS.

#### ES6 Template Literals

There are many pitfalls of working with strings on the modern web. The Javascript we know today lacks basic string formatting features, doesn't support multi-line strings, and makes it difficult to protect users from [XSS attacks](#) when inserting user-generated content into pages.

Template Literals, introduced in this release, aims to solve these problems. Its basic form adds string



# ES6テンプレートリテラル

♥バッククォートで囲って改行も含められる

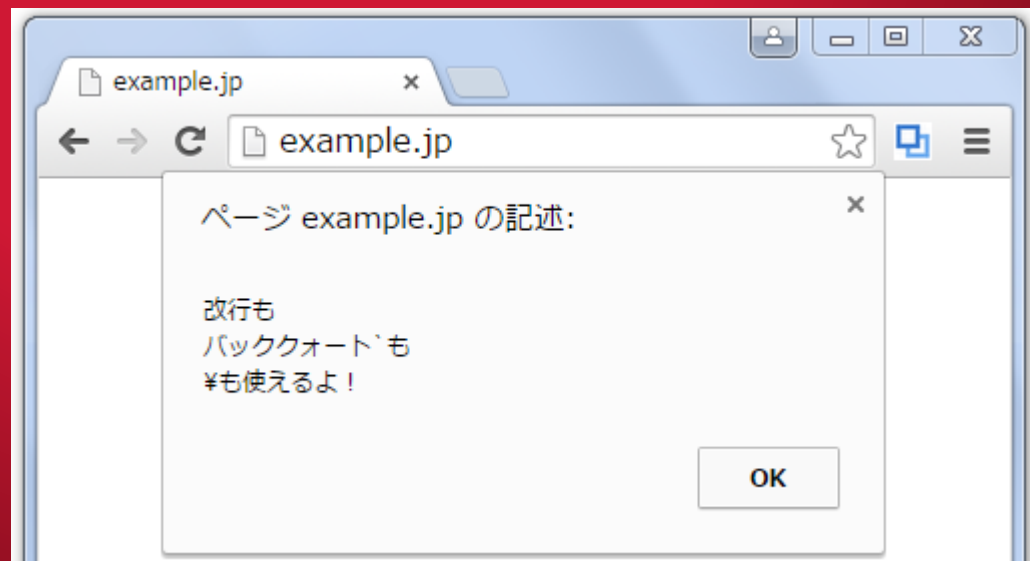
```
var x = `改行も  
ダブルクォート「"」も  
シングルクォート「'」も  
使えるよ!`;  
alert( x );
```



# ES6テンプレートリテラル

- ♥ ヒアドキュメントというには少し残念
- ♥ 円記号でのエスケープが生きてる…

```
var x = `改行も ¥nバッククォート ¥`も  
¥ ¥も使えるよ!`;  
alert( x );
```



# ES6テンプレートリテラル

♥ リテラル内に\${...}で埋め込んだ式を評価

```
var name = "hasegawa";  
var x = `Hello, ${name}-san`;  
console.log( x ); // "Hello, hasegawa-san"  
  
x = `abc${  
  (function(){ return "def"; })()  
}`;  
console.log( x ); // "abcdef"
```

# ES6テンプレートリテラル

- ♥ タグ関数 – テンプレート展開の際に呼び出されるリテラル内に`${...}`で埋め込んだ式を評価

```
function foo( /* strings, ...values */ ){  
  var strings = arguments[ 0 ];  
  var values = [].slice.call( arguments, 1 );  
  console.log( strings );      // [ "a¥nb", "cd", "" ]  
  console.log( strings.raw ); // [ "a¥¥nb", "cd", "" ]  
  console.log( values );      // [ 123, 456 ]  
}
```

```
foo`a¥nb${123}cd${456}`;
```

# ES6テンプレートリテラル

- ♥ タグ関数 – 「foo`string`」の形式で呼び出し可能
  - ♥ 「()」が不要

```
alert`xss`;  
eval.call`${alert}``1`;
```




# ES6テンプレートリテラル



Chromium Blog: Chromium x

blog.chromium.org/2015/01/chrome-41-beta-new-es6-features-and.html



## Chromium Blog

News and developments from the open source browser project

### Chrome 41 Beta: New ES6 Features and Improved DevTools for Service Workers and Web Animations

Posted: Thursday, January 22, 2015

 455  471  いいね!

Today's Chrome Beta channel release includes new Javascript ES6 features and improved workflows for debugging Service Workers and Web Animations. Unless otherwise noted, changes described below apply to Chrome for Android, Windows, Mac, Linux, and Chrome OS.

#### ES6 Template Literals

There are many pitfalls of working with strings on the modern web. The Javascript we know today lacks basic string formatting features, doesn't support multi-line strings, and makes it difficult to protect users from XSS attacks when inserting user-generated content into pages.

Template Literals, introduced in this release, aims to solve these problems. Its basic form adds string



Google Chrome Developers

google.com/+GoogleChromeDevelopers

A place for Chrome developers to



# ES6テンプレートリテラル



The image shows a screenshot of a Chromium Blog article. The browser window title is "Chromium Blog: Chrom...". The address bar shows the URL "blog.chromium.org/2015/01/chrome-41-beta-new-es6-features-and.html". The page header includes the Chromium logo and the text "Chromium Blog" and "News and developments from the open source browser project". The main article title is "Chrome 41 Beta: New... for Service Workers a...". Below the title, it says "Posted: Thursday, January 22, 2015". The article text starts with "Today's Chrome Beta channel release... debugging Service Workers and Web... to Chrome for Android, Windows, Mac, Linux, and Chrome OS." A yellow arrow points to the section title "ES6 Template Literals". The text below it says "There are many pitfalls of working with strings on the modern web. The Javascript we know today lacks basic string formatting features, doesn't support multi-line strings, and makes it difficult to protect users from XSS attacks when inserting user-generated content into pages." A red box highlights a Japanese text block that explains the difficulty of XSS prevention with template literals and the introduction of a new tag.

文字列を処理するにはたくさんの落とし穴がある。  
…略…ユーザーの生成したコンテンツをページに挿入する際にXSSを防ぐのは難しい。…略…テンプレートリテラルにはXSSを防ぐためにHTMLをエスケープするのに便利なタグ付きテンプレートが導入される。

**ES6 Template Literals**

There are many pitfalls of working with strings on the modern web. The Javascript we know today lacks basic string formatting features, doesn't support multi-line strings, and makes it difficult to protect users from XSS attacks when inserting user-generated content into pages.

Template Literals, introduced in this release, aims to solve these problems. Its basic form adds string

Google Chrome Developers  
google.com/+GoogleChromeDevelopers  
A place for Chrome developers to

# テンプレートリテラルでXSS対策

## ♥ タグ関数でエスケープ

```
function escapeHtml( strings, ...values ){
  var result = strings[ 0 ];
  for( var i = 0; i < values.length; i++ ){
    result += values[i].replace( /&/g, "&amp;" )
      .replace( /</g, "&lt;" )
      .replace( />/g, "&gt;" )
      .replace( /"/g, "&quot;" )
      .replace( /'/g, "&#x27;" )
    + strings[ i + 1 ];
  }
  return result;
}
```

```
var name = "<img src='#' onerror='alert(1)'\>";
elm.innerHTML = escapeHtml`<div>${name}</div>`;
```

# テンプレートリテラルでXSS対策

## ♥ タグ関数でエスケープ

```
function escapeHtml( strings, ...values ){
  var result = strings[ 0 ];
  for( var i = 0; i < values.length; i++ ){
    result += values[i].replace( /&/g, "&amp;" )
      .replace( /</g, "&lt;" )
      .replace( />/g, "&gt;" )
      .replace( /"/g, "&quot;" )
      .replace( /'/g, "&#x27;" )
      + strings[ i + 1 ];
  }
}
```

```
<div>&lt;img src=&#x27;#&#x27;
onerror=&#x27;alert(1)&#x27;&gt;</div>
```

```
var name = "<img src='#' onerror='alert(1)'\>";
elm.innerHTML = escapeHtml`<div>${name}</div>`;
```



# テンプレートリテラルでXSS対策

- ♥HTMLテンプレートの的な使い方ができる
- ♥ただしJSのコードとテンプレート部分が不可分
  - ♥あくまでもリテラル

```
var str = escapeHtml`<div>${name}</div>`;  
elm.innerHTML = str;
```



# テンプレートリテラルでXSS対策

## ♥あるいはeval等が必要

```
<template id="tpl">
  <div>${name}</div>
</template>
<script>
  var t =
document.getElementById("tpl").innerHTML;
  var name = "<img src='#' onerror='alert(1)'>";
  elm.innerHTML = eval( "escapeHtml`" + t + "`" );
</script>
```

♥ 「XSS対策」といいつつevalはない

♥ CSP使いたいし



# まとめ

- ♥ テンプレートリテラルはHTMLテンプレートとは無関係
- ♥ テンプレートリテラルをXSS対策に利用することは不可能ではないがあまり綺麗ではない
- ♥ 文字列置換ではない、DOM-basedなHTMLテンプレートエンジン欲しい…

