talk is cheap,show me the code.

# Js

https://cloud.tencent.com/developer/article/1621572

1. Undefined和null有什么区别

在理解undefined和null之间的差异之前，我们先来看看它们的相似

他们都属于JavaScript的7种基本类别

let primitiveTypes = ['string', 'number', 'null', 'undefined', 'boolean', 'symbol', 'object']

他们都属于虚值，可以使用Boolean（value）或!!value将其转换为布尔值时，值为false

console.log(!!null); *// false*

console.log(!!undefined); *// false*

console.log(Boolean(null)); *// false*

console.log(Boolean(undefined)); *// false*

\*简单的来说虚值就是在转换为布尔值时变为false的值

const falsyValues = ['', 0, null, undefined, NaN, false];

接着来看看他们的区别。

Undefined是未指定特定值的变量的默认值，或者没有显示返回值的函数，如console.log(1)，还包括对象中不存在的属性，这些js引擎都会为其分配undefined的值

let \_thisIsUndefined

const doNothing = (a) => {a=a\*a}

const someObj = {

    a : 'ay',

    b : 'bee',

    c : 'see'

}

console.log(\_thisIsUndefined); *// undefined*

console.log(doNothing(2)); *// undefined*

console.log(someObj["d"]); *// undefined*

null是“不代表任何值的值”。Null是已明确定义给变量的值。在此示例中，当fs.readFile方法未引发错误时，我们将获得null值

var fs = require('fs');

fs.readFile('./tmp.js', (e,data) => {

    console.log(e); *// 当没有错误发生时，打印 null*

    if(e){

      console.log(e);

    }

    console.log(data);

});

在比较null和undefined时，我们使用==时得到true，使用===时得到false：

console.log(null == undefined); *// true*

console.log(null === undefined); *// false*

1. &&运算符能做什么

&&也叫逻辑与，在其操作数中找到第一个虚值表达式并返回它，如果没有找到任何虚值表达式，则返回最后一个真值表达式。它采用短路来防止不必要的工作

console.log(false && 1 && []) *// false*

console.log(" " && true && 5) *// 5*

false && console.log('this is false SHORT') *// dont print any*

true && console.log('this is true') *// this is true*

true && console.log('this is true') && false && console.log('this is false SHORT') *// this is true*

使用if语句

const router: Router = Router();

router.get("/endpoint", (req: Request, res: Response) => {

  let conMobile: PoolConnection;

  try {

*//do some db operations*

  } catch (e) {

    if (conMobile) {

      conMobile.release();

    }

  }

});

使用&&操作符

const router: Router = Router();

router.get("/endpoint", (req: Request, res: Response) => {

  let conMobile: PoolConnection;

  try {

*//do some db operations*

  } catch (e) {

    conMobile && conMobile.release();

  }

});

1. ||操作符能做什么

||也叫逻辑或，在其操作数中找到第一个真值表达式并返回它。这也使用了短路来防止不必要的工作。在支持ES6默认函数之前，它用于初始化函数中的默认参数值

console.log(null || 1 || undefined); *// 1*

function logName(name) {

  var n = name || "Mark";

  console.log(n);

}

logName(); *// "Mark"*

1. 使用+或一元运算符是将字符串转换为数字的最快方法吗？

根据MDN文档，+是将字符串转换为数字的最快方法，因为如果值已经是数字，他不会执行任何操作

console.log(typeof(+"45"))

1. DOM是什么？

DOM代表文档对象模型，是HTML和XML文档的接口（API）。当浏览器第一次读取（解析）HTML文档时，他会创建一个大对象，一个基于HTML文档的非常大的对象就是DOM。他是一个从HTML文档中建模的树状结构，DOM用于交互和修改DOM结构或特定元素或节点。

假设我们有这样的HTML结构：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document Object Model</title>

</head>

<body>

<div>

<p>

<span></span>

</p>

<label></label>

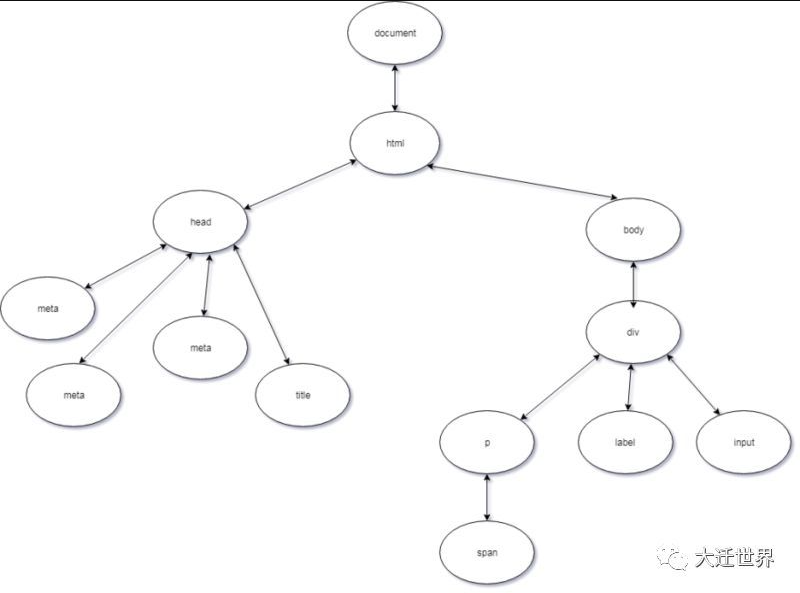
<input>

</div>

</body>

</html>

等价的DOM是这样的：



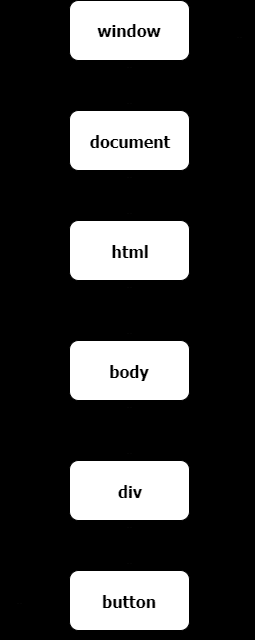
JS中的document对象表示DOM。他为我们提供了许多方法，我们可以使用这些方法来选择元素来更新元素内容等等

1. 什么是事件传播？

当事件发生在DOM元素上时，该事件并不完全发生在那个元素上。在“冒泡阶段”中，事件冒泡或向上传播至父级，祖父级，直到到达window为止；而在“捕获阶段”中，事件从window开始向下触发元素事件或event.target

事件传播有三个阶段：

1. 捕获阶段 – 事件从window开始，然后向下到每个元素，直到到达目标元素
2. 目标阶段 – 事件已达到目标元素
3. 冒泡阶段 – 事件从目标元素冒泡，然后上升到每个元素，直到到达window



1. 什么是事件冒泡？

当事件发生在DOM元素上时，该事件并不完全发生在那个元素上。在冒泡阶段，事件冒泡，或者事件发生在它的父代，祖父母，祖父母的父代，直到到达window为止。

假设有如下的 HTML 结构：

<div class="grandparent">

<div class="parent">

<div class="child">1</div>

</div>

</div>

对应的 JS 代码:

function addEvent(el, event, callback, isCapture = false) {

if (!el || !event || !callback || typeof callback !== 'function') return;

if (typeof el === 'string') {

el = document.querySelector(el);

};

el.addEventListener(event, callback, isCapture);

}

addEvent(document, 'DOMContentLoaded', () => {

const child = document.querySelector('.child');

const parent = document.querySelector('.parent');

const grandparent = document.querySelector('.grandparent');

addEvent(child, 'click', function (e) {

console.log('child');

});

addEvent(parent, 'click', function (e) {

console.log('parent');

});

addEvent(grandparent, 'click', function (e) {

console.log('grandparent');

});

addEvent(document, 'click', function (e) {

console.log('document');

});

addEvent('html', 'click', function (e) {

console.log('html');

})

addEvent(window, 'click', function (e) {

console.log('window');

})

});

addEventListener方法具有第三个可选参数useCapture，其默认值为false，事件将在冒泡阶段中发生，如果为true，则事件将在捕获阶段中发生。如果单击child元素，它将分别在控制台上记录child，parent，grandparent，html，document和window，这就是事件冒泡。

1. 什么是事件捕获？

当事件发生在 DOM 元素上时，该事件并不完全发生在那个元素上。在捕获阶段，事件从window开始，一直到触发事件的元素。

假设有如下的 HTML 结构：

<div class="grandparent">

<div class="parent">

<div class="child">1</div>

</div>

</div>

对应的JS代码：

function addEvent(el, event, callback, isCapture = false) {

if (!el || !event || !callback || typeof callback !== 'function') return;

if (typeof el === 'string') {

el = document.querySelector(el);

};

el.addEventListener(event, callback, isCapture);

}

addEvent(document, 'DOMContentLoaded', () => {

const child = document.querySelector('.child');

const parent = document.querySelector('.parent');

const grandparent = document.querySelector('.grandparent');

addEvent(child, 'click', function (e) {

console.log('child');

});

addEvent(parent, 'click', function (e) {

console.log('parent');

});

addEvent(grandparent, 'click', function (e) {

console.log('grandparent');

});

addEvent(document, 'click', function (e) {

console.log('document');

});

addEvent('html', 'click', function (e) {

console.log('html');

})

addEvent(window, 'click', function (e) {

console.log('window');

})

});

addEventListener方法具有第三个可选参数useCapture，其默认值为false，事件将在冒泡阶段中发生，如果为true，则事件将在捕获阶段中发生。如果单击child元素，它将分别在控制台上打印window，document，html，grandparent和parent，这就是事件捕获。