JavaScript极速入门 —— 数据类型

主讲人与课程设计：耕耕

数据类型 —— 8种

Object：用于更复杂的数据结构
Undefined：用于未定义的值 —— 只有一个 undefined 值的独立类型
Null：用于未知的值 —— 只有一个 null 值的独立类型
Number：用于任何类型的数字：整数或浮点数，在 $\pm 2^{53}$ 范围内的整数
BigInt：用于任意长度的整数
Boolean：用于 true 和 false
String：用于字符串：一个字符串可以包含一个或多个字符，所以没有单独的单字符类型
Symbol：用于唯一的标识符

记忆：OUNNBBSS （哦，牛逼死）

注意：**BigInt** 和 **Symbol** 是es6 的内容，先不用掌握

“null” 值

Null 类型是只有一个值的数据类型，这个特殊的值是**null**

JavaScript 中的 null 仅仅是一个代表“无”、“空”或“值未知”的特殊值

如果定义的变量准备在将来用于保存对象，那么最好将该变量初始化为null（习惯上用来标示一个空对象指针） 而不是其他值

注意：null 和undefined 之间的相等操作符（==）总是返回true

“undefined” 值

Undefined 类型只有一个值，即特殊的**undefined**

在使用var 声明变量但未对其加以初始化时，这个变量的值就是undefined

```
var message = undefined;
alert(message == undefined); //true
```

object 类型

ECMAScript 中的对象其实就是一组数据和功能的集合。对象可以通过执行**new** 操作符后跟要创建的对象类型的名称来创建。而创建Object 类型的实例并为其添加属性和（或）方法，就可以创建自定义对象

```
var o = new Object();
```

对象我们将在后续进行详细讲解，这里做了解

Symbol 类型

Symbol是由ES6规范引入的一项新特性，它的功能类似于一种标识唯一性的ID

```
var s1 = Symbol();
```

对象我们将在后续进行详细讲解，这里做了解

BigInt 类型 （了解）

在 JavaScript 中，“number” 类型无法代表大于 2^{53}（或小于-2^{53}）的整数，这是其内部表示形式导致的技术限制。

BigInt 类型是最近被添加到 JavaScript 语言中的，用于表示**任意长度的整数**。

通过将 **n** 附加到整数字段的末尾来创建 BigInt

```
var n = 123n;
```

Boolean 类型（逻辑类型，小白们）

boolean 类型仅包含两个值：**true** 和 **false**。

```
var nameFieldChecked = true; // yes, name field is checked
var ageFieldChecked = false; // no, age field is not checked
```

拓展：任何值可以通过Boolean函数转化为boolean值

```
var message = "Hello world!";
var messageAsBoolean = Boolean(message); //true
```

结论：**0, ‘’, null, undefined, NaN, false** 这6个值全部转化为false，其他都为true

```
var message = "Hello world!";
if (message) {
    alert("Value is true");
}
```

String 类型（小白们）

String 类型用于表示由零或多个16 位Unicode 字符组成的字符序列，即字符串。字符串可以由**双引号**（“）或**单引号**（‘）或**反引号**（``）表示， 使用toString方法可以转为字符串

```
var firstName = "Nicholas";
var lastName = 'Zakas';
var middleNmae = `Niu`; //es6用法
```

拓展：数值、布尔值、对象和字符串值（没错，每个字符串也都有一个toString()方法，该方法返回字符串的一个副本）都有**toString**()方法。但**null 和undefined** 值没有这个方法

String类型转换，一般情况是看到什么转化什么，只有对象**一般情况**转换为[object Object], 数组转为逗号分隔的字符串, [1,2,3] => “1,2,3”

String 类型（对象转换）

如果转换是对象：

1. 如果没有toString()方法，直接返回[object Object]；

2. 看两者是否其中一个返回原始值，如果是，则返回转换，否则报错， 先toString()后valueOf()

3. toString()是否能返回原始值，如果不能返回，看valueOf()是否能返回原始值，如果不能返回就报错。如果有返回就转化为字符串。两个方法如果都不能返回原始值，就报错

补充说明：

Number、Stirng、Boolean、Null、Underfined, Symbol这些基本数据类型（原始值），他们的值直接保存在栈中；
Object、Function、Array、Date、RegExp这些引用类型，他们的引用变量储存在栈中，通过指针指向储存在堆中的实际对象

Number 类型，小白们

number 类型代表**整数**和**浮点数(小数)**

数字可以有很多操作，比如，乘法 *、除法 /、加法 +、减法 - 等等

除了常规的数字，还包括所谓的“特殊数值（“special numeric values”）”也属于这种类型：
Infinity、**-Infinity** 和 **NaN**

```
var n = 123;
n = 12.345;
```

拓展：第一位是0标示八进制，如果前两位是0x标示16进制（**了解一下**）

```
var shi = 98; // 十进制
var ba = 070; // 八进制的56
var shi6 = 0xA; // 十六进制的10
```

Number 类型(转化规则，**拓展**)

如果是Boolean 值，**true** 和**false** 将分别被转换为**1** 和**0**。
　　如果是**数字值**，只是简单的传入和**返回**。
　　如果是**null** 值，返回**0**。
　　如果是**undefined**，返回**NaN**。
　　如果是**[]**，返回**0**。
　　如果是**[9]**，返回**9**。
　　如果是**function(){}**，返回**NaN**。
　　如果是字符串，遵循下列规则:
　　如果字符串中只包含数字（包括前面带正号或负号的情况），则将其转换为十进制数值，即"1"会变成1，"123"会变成123，而"011"会变成11（注意：前导的零被忽略了）；
　　如果字符串中包含有效的浮点格式，如"1.1"，则将其转换为对应的浮点数值（同样，也会忽略前导零）；
　　如果字符串中包含有效的十六进制格式，例如"0xf"，则将其转换为相同大小的十进制整数值;
　　如果字符串是空的（不包含任何字符），则将其转换为0;
　　如果字符串中包含除上述格式之外的字符，或者其他情况的数组，则将其转换为**NaN**。
　　如果是对象，则调用对象的**valueOf**()方法，然后依照前面的规则转换返回的值。如果没有valueOf()，调用对象的**toString**()方法，然后再次依照前面的规则转换返回的字符串值

Number 类型(对象类型转换)

1. 如果对象没有valueOf()和toString()，返回NaN

2. 看两者是否其中一个返回原始值，如果是，则返回并转换，否则报错， 先valueOf()后toString()

3. valueOf()是否能返回原始值，如果不能返回，看toString()是否能返回原始值，如果不能返回就报错。如果有返回就转化为数字。两个方法如果都不能返回原始值，就报错

typeof操作符，小白们

鉴于ECMAScript 是松散类型的，因此需要有一种手段来检测给定变量的数据类型——**typeof** 就是负责提供这方面信息的操作符

```
typeof undefined //  "undefined",包括没有定义的
typeof 0 // "number"
typeof 10n // "bigint"
typeof true // "boolean"
typeof "foo" // "string"
typeof Symbol("id") // "symbol"
typeof Math // "object"  (1)
typeof null // "object"  (2)
typeof alert // "function"  (3)
```

注意：8种数据类型7种typeof 返回都是自己的类型；然而null返回object，还多了一种function类型，返回结果都是**字符串**