第一部分：类型和语法

第一章：类型。

ES类型又进一步细分为语言类型和规范类型。ES语言中所有的值都有一个对应的语言类型。ES语言类型包括Undefined、Null、Boolean、String、Number、Object……

1. 类型
2. 内置类型：JS有7中内置类型。5种基本类型+对象（object）+符号（symbol，ES6中新增）。///符号也为基本类型。

通过typeof运算符来查看值的类型，返回类型的字符串值，有特殊为null🡪object；function a（）{ }🡪function，它实际上是object的一个子类型。

函数不仅是对象，还可以拥有属性。如：函数对象的length属性是其声明的参数的个数。

还有数组，数组也是对象。确切的说，她也是object的一个“子类型”，数组的元素按数字顺序来进行索引（而非普通对象那样通过字符串键值），其length属性是元素的个数。

1. 值和类型

JS中的变量是没有类型的，只有值才有。变量可以随时持有任何类型的值。

，typeof 运算符总是会返回一个字符串的 。

已经在作用域声明但是还没有赋值的变量，是undefined，相反，还没有在作用域中声明过的变量，是undeclared的。但是，请注意，对于undeclared（或者not defined）变量，typeof照样返回“undefined”。

还有一种不用通过typeof的安全防范机制的方法，就是检查所有全局变量是否是全局对象的属性，浏览器中的全局对象是window。

与undeclared变量不同，访问不存在的对象属性（甚至是在全局对象window上）不会产生ReferenceErroor错误。

一些开发人员不喜欢通过window来访问全局对象……

1. 小结

Js有7种内置类型，null、undefined、Boolean、number、string、object、symbol，可以使用typeof运算符来查看。

变量没有类型，但它们持有的值有类型。

在JS中它们是两码事，undefined是值的一种，而undeclared则表示变量还没有被声明过。但是极为遗憾，JS将它们混为一谈，typeof对这两种变量都会返回undefined。

第二章

数组（array）、字符串、和数字是一个程序最基本的组成部分。

1. 数组

和其它强类型语言不同，在JS中，数组可以容纳任何类型的值。可以是字符串、数字、对象，甚至是其他数组（多维数组的形成）

对数组声明后即可向其中加入值，不需要预先设定大小。

使用delete运算符可以将单元从数组中删除，但是，单元删除后，数组的length属性并不会发生变化。

在创建稀疏数组（即含有空白或空缺单元的数组）时要特别注意：……

数组通过数字进行索引，但有趣的是它们也是对象，所以也可以包含字符串键值和属性（但这些并不计算在数组长度内）

如果字符串键值能够被强制类型转换为十进制数字的话，它就会被当作数字索引来处理。

类数组：有时需要将类数组（一组通过数字索引的值）转换为真正的数组，这一般通过数组工具函数（如indexOf（）、concat（）、forEach（）等）来实现。

1. 字符串

字符串经常被当作字符数组。字符串和数组的确很相似，它们都是类数组，都有length属性以及indexOf（）和concat（）方法。

JS中字符串是不可变的，而数组是可变的。

字符串不可变是指字符串的成员函数不会改变其原始值，而是创建并返回一个新的字符串。而数组的成员函数都是在原始值上进行操作的。

许多数组函数用来处理字符串很方便。虽然字符串没有这些函数，但可以通过“借用”数组的非变更方法来处理字符串。

另一个不同点在于字符串反转。数组有一个字符串没有的可变更成员函数 reserve（）；

不过，我们无法借用数组的可变更成员函数，因为字符串是不可变的。

一个变通的方法是先将字符串转换为数组，待处理完后再将结果转换回字符串。

1. 数字

JS中的整数就是没有小数的十进制数，所以42.0=42；

数字的语法：JS中的数字常量一般用十进制表示；数字前面的0可以省略 0.24🡪.24；小数点后小数部分最后的0也可以省略；默认情况下大部分数字都以十进制显示，小数部分最后面的 0 被省略; 特别大和特别小的数字默认用指数格式显示，与 toExponential() 函数的输出结果相同;

tofixed(..) 方法可指定小数部分的显示位数; toPrecision(..) 方法用来指定有效数位的显示位数。

较小的数值：0.1 + 0.2 === 0.3; // false

简单来说，二进制浮点数中的 0.1 和 0.2 并不是十分精确，它们相加的结果并非刚好等于 0.3，而是一个比较接近的数字 0.30000000000000004，所以条件判断结果为 false。

……

整数的安全范围：

整数检测：要检测一个值是否是整数，可以使用 ES6 中的 Number.isInteger(..) 方法。

要检测一个值是否是安全的整数，可以使用 ES6 中的 Number.isSafeInteger(..) 方法

32位有符号整数：

a | 0 可以将变量 a 中的数值转换为 32 位有符号整数，因为数位运算符 | 只适用于 32 位 整数（它只关心 32 位以内的值，其他的数位将被忽略）。因此与 0 进行操作即可截取 a 中 的 32 位数位。

1. 特殊数值

不是值的值：undefined 类型只有一个值，即 undefined。null 类型也只有一个值，即 null。它们的名称既是类型也是值。null 指空值，undefined 指没有值。

undefined 指从未赋值 ， null 指曾赋过值，但是目前没有值。

null是一个特殊关键字，不是标识符，我们不能将其当作变量来使用和赋值。然而 undefined 却是一个标识符，可以被当作变量来使用和赋值。

Undefined：……

总之，如果要将代码中的值（如表达式的返回值）设为 undefined，就可以使用 void。

特殊的数字：不是数字的数字, NaN 是一个特殊值，它和自身不相等，是唯一一个非自反的值。判断是否为NaN，使用内建的全局工具函数 isNaN(..) 来判断一个值是否是 NaN。要使用Number.isNaN(..)来判断。

无穷数：

零值：加法和减法运算不会得到负零，根据规范，对负零进行字符串化会返回 "0"。……

特殊等式：NaN 和 -0 在相等比较时的表现有些特别。由于 NaN 和自身不相等，所以必须使 用 ES6 中的 Number.isNaN(..)。而 -0 等于 0（对于 === 也是如此），因此我们必须使用 isNegZero(..) 这样的工具函数。

ES6 中新加入了一个工具方法 Object.is(..) 来判断两个值是否绝对相等，可以用来处理 上述所有的特殊情况。

能使用 == 和 === 时就尽量不要使用 Object.is(..)，因为前者效率更高、 更为通用。Object.is(..) 主要用来处理那些特殊的相等比较。

1. 值和引用

…………

1. 小结

null 类型只有一个值 null，undefined 类型也只有一个值 undefined。所有变量在赋值之 前默认值都是 undefined。void 运算符返回 undefined。

简单标量基本类型值（字符串和数字等）通过值复制来赋值 / 传递，而复合值（对象等） 通过引用复制来赋值 / 传递。

第四章

强制类型转换

1. 值的类型转换

将值从一种类型转换为另一种类型通常称为类型转换，这是显式的情况；隐式的情况称为强制类型转换。

类型转换发生在静态类型语言的编译阶段，而强制类型转换则发生在强制类型转换动态类型语言的运行时。

1. 抽象值操作

ToString：它负责处理非字符串到字符串的强制类型转换。

基本类型值的字符串化规则为：null 转换为 "null"，undefined 转换为 "undefined"，true 转换为 "true"。数字的字符串化则遵循通用规则。不过那些极小和极大的数字使用指数形式。

如果对象有自己的 toString() 方法，字符串化时就会调用该方法并使用其返回值。

toString() 可以被显式调用，或者在需要字符串化时自动调用。

JSON字符串化

工具函数 JSON.stringify(..) 在将 JSON 对象序列化为字符串时也用到了 ToString。 请注意，JSON 字符串化并非严格意义上的强制类型转换。

所有安全的 JSON 值都可以使用 JSON.stringify(..) 字符串化。不安全的值包括：。undefined、function、symbol （ES6+）和包含循环引用（对象之间相互引用，形成一个无限循环）的对象都不符合 JSON 结构标准，支持 JSON 的语言无法处理它们。

JSON.stringify(..) 在对象中遇到 undefined、function 和 symbol 时会自动将其忽略，在 数组中则会返回 null（以保证单元位置不变）。对包含循环引用的对象执行 JSON.stringify(..) 会出错。

如果要对含有非法 JSON 值的对象做字符串化，或者对象中的某些值无法被序列化时，就 需要定义 toJSON() 方法来返回一个安全的 JSON 值。

toJSON() 返回的应该是一个适当的值，可以是任何类型，然后再由 JSON.stringify(..) 对其进行字符串化。 也就是说，toJSON() 应该“返回一个能够被字符串化的安全的 JSON 值”，而不是“返回一个 JSON字符串”。

我们可以向 JSON.stringify(..) 传递一个可选参数 replacer…… 看不懂！！！！！

请记住，JSON.stringify(..) 并不是强制类型转换。在这里介绍是因为它涉及 ToString 强 制类型转换，具体表现在以下两点。 (1) 字符串、数字、布尔值和 null 的 JSON.stringify(..) 规则与 ToString 基本相同。 (2) 如果传递给 JSON.stringify(..) 的对象中定义了 toJSON() 方法，那么该方法会在字符 串化前调用，以便将对象转换为安全的 JSON 值。

ToNumber：

其中 true 转换为 1，false 转换为 0。undefined 转换为 NaN，null 转换为 0。

ToNumber 对以 0 开头的十六进制数并不按十六进制处理，而是按十进制。

ToBoolean：

假值：

JS中的值可以分为一下两类：(1) 可以被强制类型转换为false 的值 (2) 其他（被强制类型转换为 true 的值）。

以下这些是假值： • undefined • null • false • +0、-0 和 NaN • ""

假值的布尔强制类型转换结果为 false。

从逻辑上说，假值列表以外的都应该是真值（truthy）。

假值对象：并不是封装了假值的对象，而是浏览器在某些特定情况下，在常规 JavaScript 语法基础上自己创建了一些外来值。

假值对象看起来和普通对象并无二致（都有属性，等等），但将它们强制类型转换为布尔 值时结果为 false。

真值：

真值就是假值列表之外的值。

1. 显示强制类型转换

字符串和数字之间的显示转换：

字符串和数字之间的转换是通过 String(..) 和 Number(..) 这两个内建函数（原生构造函 数）来实现的，请注意它们前面没有 new 关键字，并不创建封装对象。

a.toString() 是显式的（“toString”意为“to a string”），不过其中涉及隐式转换。因为 toString() 对 42 这样的基本类型值不适用，所以 JavaScript 引擎会自动为 42 创建一个封 装对象（参见第 3 章），然后对该对象调用 toString()。这里显式转换中含有隐式转换。

日期显示转换为数字：一元运算符 + 的另一个常见用途是将日期（Date）对象强制类型转换为数字，返回结果为 Unix 时间戳，以微秒为单位（从 1970 年 1 月 1 日 00:00:00 UTC 到当前时间）。

将日期对象转换为时间戳并非只有强制类型转换这一种方法，或许使用更显式的方法会更好一些。

不过最好还是使用 ES5 中新加入的静态方法 Date.now()。

我们不建议对日期类型使用强制类型转换，应该使用 Date.now() 来获得当前的时间戳，使 用 new Date(..).getTime() 来获得指定时间的时间戳。

奇特的 ~ 运算符：？？？？？？

字位截除：？？？

显式解析数字字符串:

解析字符串中的数字和将字符串强制类型转换为数字的返回结果都是数字。但解析和转换 两者之间还是有明显的差别。

解析允许字符串中含有非数字字符，解析按从左到右的顺序，如果遇到非数字字符就停 止。而转换不允许出现非数字字符，否则会失败并返回 NaN。

解析和转换之间不是相互替代的关系。它们虽然类似，但各有各的用途。如果字符串右边 的非数字字符不影响结果，就可以使用解析。而转换要求字符串中所有的字符都是数字， 像 "42px" 这样的字符串就不行。

解析字符串中的浮点数可以使用 parseFloat(..) 函数。

parseInt(..) 针对的是字符串值。向 parseInt(..) 传递数字和其他类型的参数是没有用的。

显示转换为布尔值：

与前面的 String(..) 和 Number(..) 一样，Boolean(..)（不带 new）是显式的 ToBoolean 强 制类型转换。

显式强制类型转换为布尔值最常用的方法是！！，因为第二个 ! 会将结果反转回原值。

在 if(..).. 这样的布尔值上下文中，如果没有使用 Boolean(..) 和 !!，就会自动隐式地进 行 ToBoolean 转换。

显式 ToBoolean 的另外一个用处，是在 JSON 序列化过程中将值强制类型转换为 true 或 false。 ？？？？

var a = 42;

var b = a ? true : false;

三元运算符 ? : 判断 a 是否为真，如果是则将变量 b 赋值为 true，否则赋值为 false。

1. 隐式强制类型转换

隐式的简化：

字符串和数字之间的隐式强制类型转换：

- 是数字减法运算符，因此 a - 0 会将 a 强制类型转换为数字。也可以使用 a \* 1 和 a / 1，因为这两个运算符也只适用于数字，只不过这样的用法不太常见。

布尔值到数字的隐式强制类型转换：

隐式强制类型转换为布尔值：相对布尔值，数字和字符串操作中的隐式强制类型转换还算比较明显。

if (..) 语句中的条件判断表达式、 for ( .. ; .. ; .. ) 语句中的条件判断表达式（第二个）、while (..) 和 do..while(..) 循环中的条件判断表达式、? : 中的条件判断表达式、逻辑运算符 ||（逻辑或）和 &&（逻辑与）左边的操作数（作为条件判断表达式）。

||和&&：&& 和 || 运算符的返回值并不一定是布尔类型，而是两个操作数其中一个的值。

对于 || 来说，如果条件判断结果为 true 就返回第一个操作数（a 和 c）的值，如果为 false 就返回第二个操作数（b）的值。

&& 则相反，如果条件判断结果为 true 就返回第二个操作数（b）的值，如果为 false 就返 回第一个操作数（a 和 c）的值。

符号的强制类型转换：

1. 宽松相等和严格相等

宽松相等（loose equals）== 和严格相等（strict equals）=== 都用来判断两个值是否“相 等”，但是它们之间有一个很重要的区别，特别是在判断条件上。 常见的误区是“== 检查值是否相等，=== 检查值和类型是否相等”。听起来蛮有道理，然而 还不够准确。

正确的解释是：“== 允许在相等比较中进行强制类型转换，而 === 不允许。”

相等比较操作的性能：

抽象相等：

有几个非常规的情况需要注意。 • NaN 不等于 NaN。 • +0 等于 -0。

== 在比较两个不同类型的值时会发生隐式强制类型转换，会将其中之 一或两者都转换为相同的类型后再进行比较。

字符串和数字之间的相等比较、

其他类型和布尔类型之间的相等比较、

null 和 undefined 之间的相等比较：如果 x 为 null，y 为 undefined，则结果为 true。如果 x 为 undefined，y 为 null，则结果为 true。 在 == 中 null 和 undefined 相等（它们也与其自身相等），除此之外其他值都不存在这种情况。 这也就是说在 == 中 null 和 undefined 是一回事，可以相互进行隐式强制类型转换：

条件判断 a == null 仅在非 null 和 undefined 时才成立，除此之外其他值都不成立，包括 0、false 和 "" 这样的假值。

对象和非对象之间的相等比较：(1) 如果 Type(x) 是字符串或数字，Type(y) 是对象，则返回 x == ToPrimitive(y) 的结果；(2) 如果 Type(x) 是对象，Type(y) 是字符串或数字，则返回 ToPromitive(x) == y 的结果。

比较少见的情况：1. 返回其他数字、2. 假值的相等比较、（懵、看不懂、记不住）、3、极端情况、4.完整性检查

1. 抽象关系比较

比较双方首先调用 ToPrimitive，如果结果出现非字符串，就根据 ToNumber 规则将双方强 制类型转换为数字来进行比较。

如果比较双方都是字符串，则按字母顺序来进行比较

？？？？？？？？？？？？？？？？？看不懂

1. 小结