你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

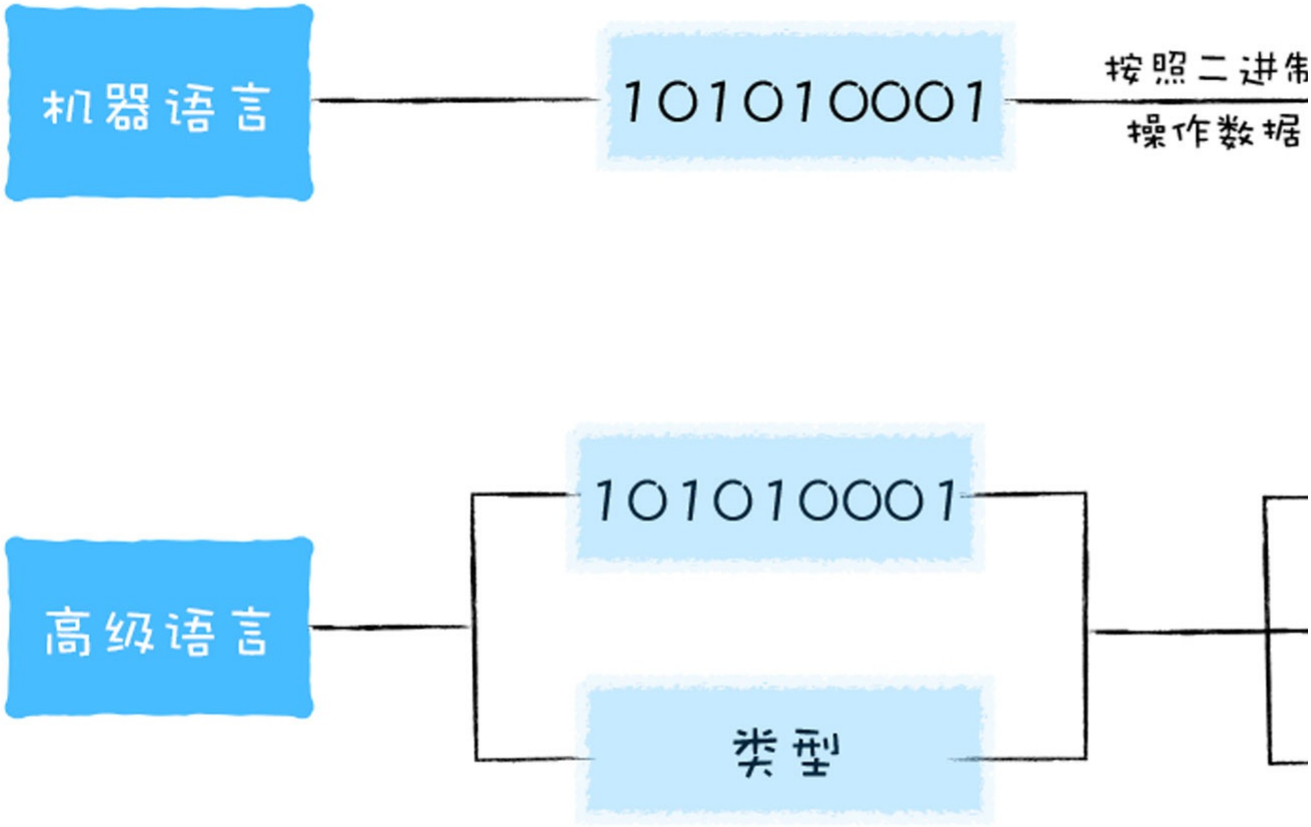
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

```
a+b
```

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

> AdditiveExpression : AdditiveExpression + MultiplicativeExpression

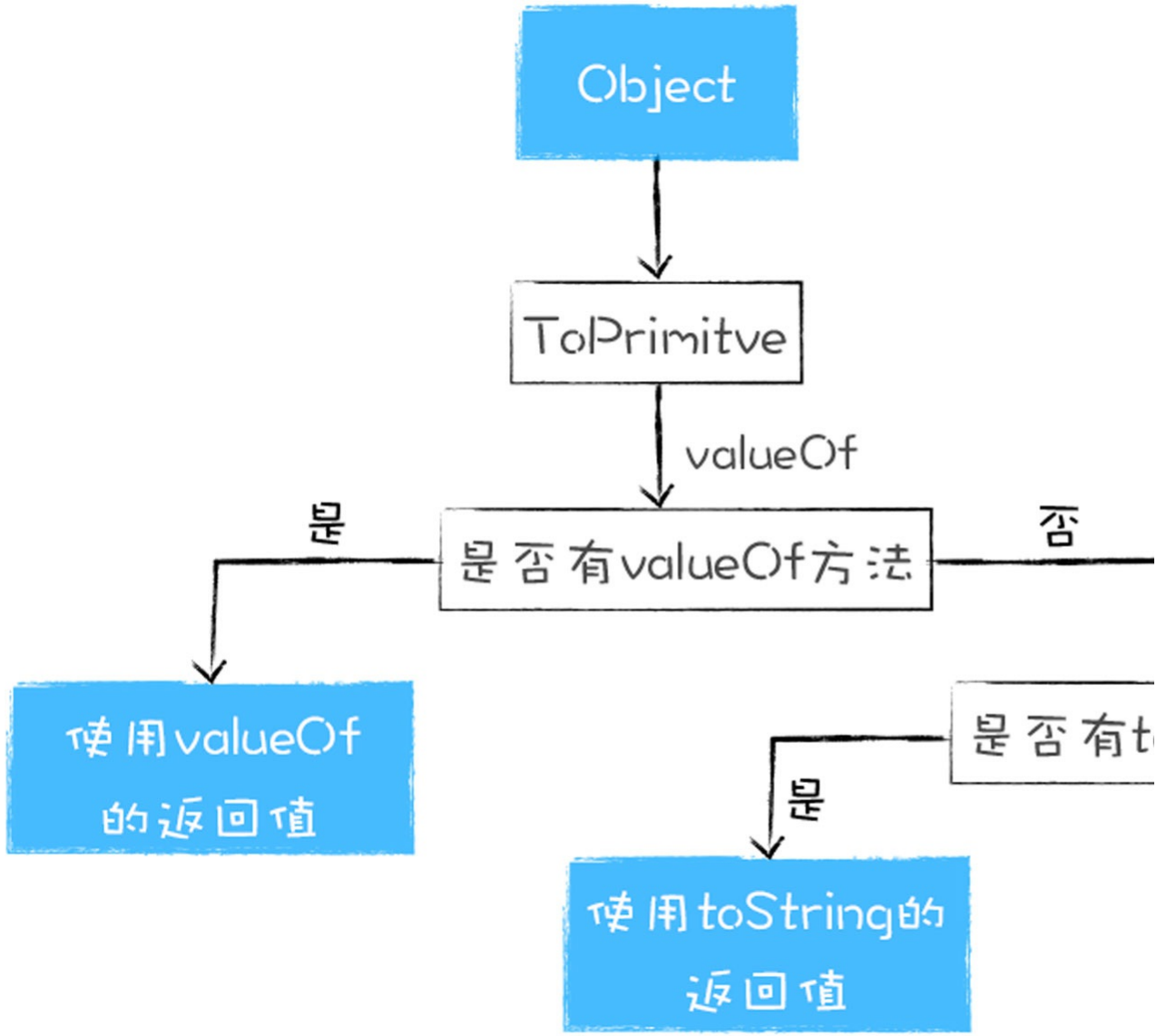
1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；

c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vauleOf和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
```

```
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulеOf和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

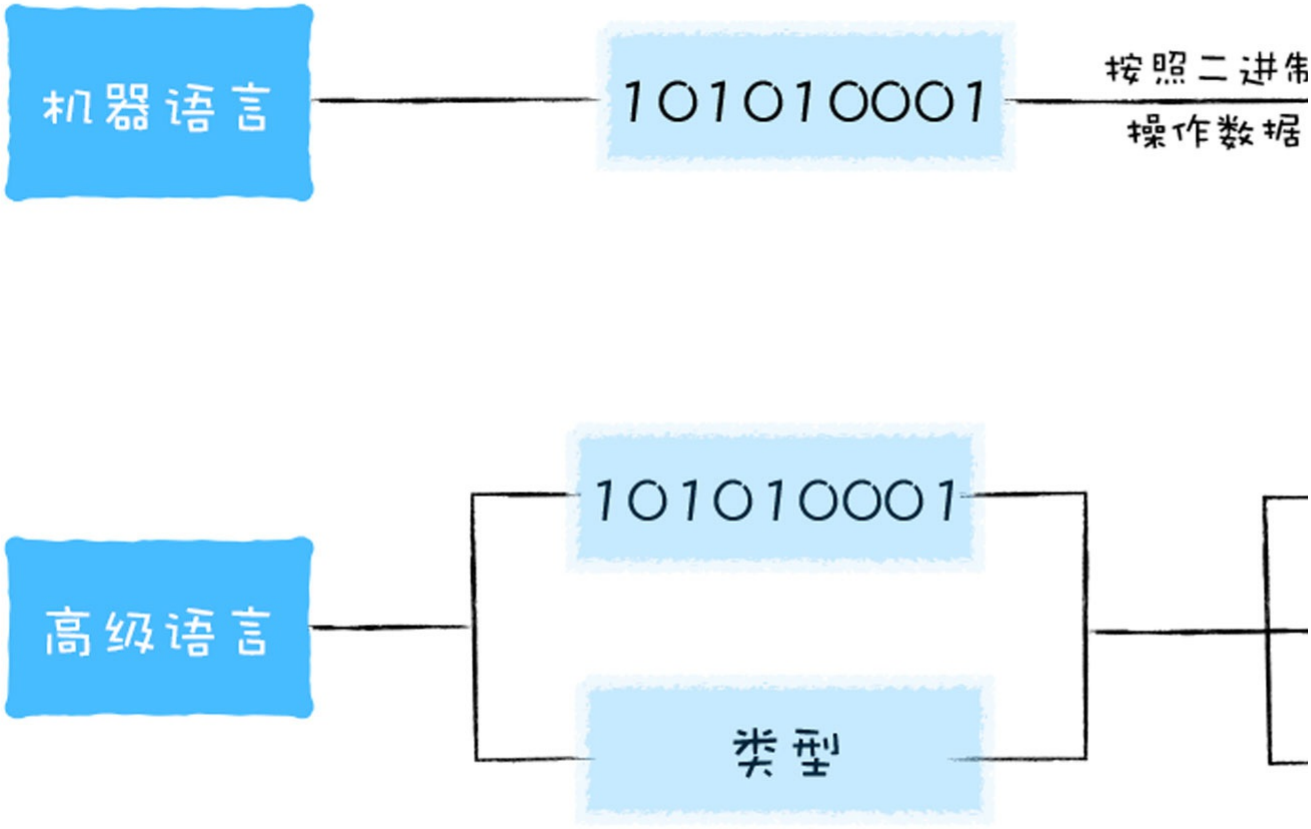
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

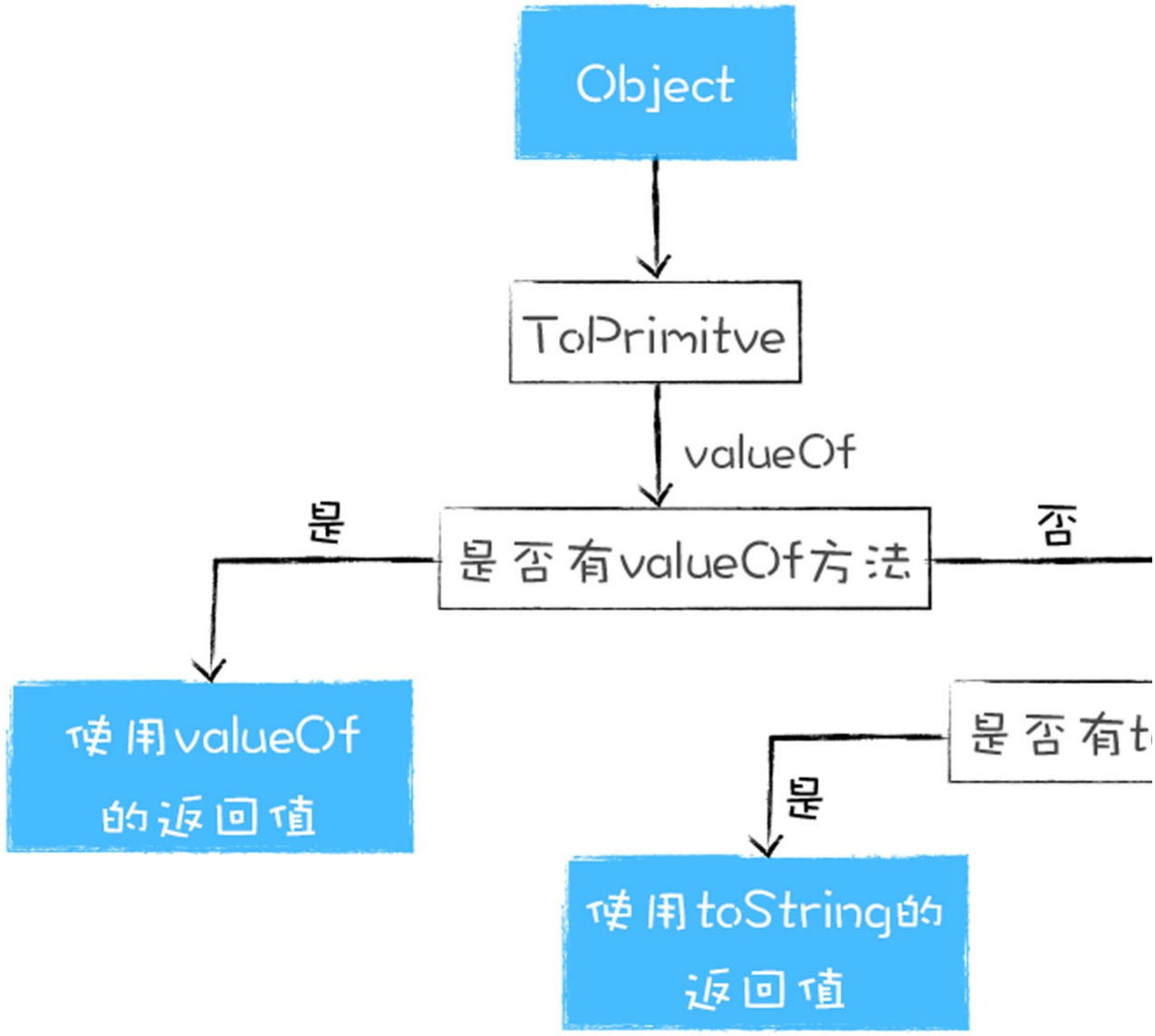
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

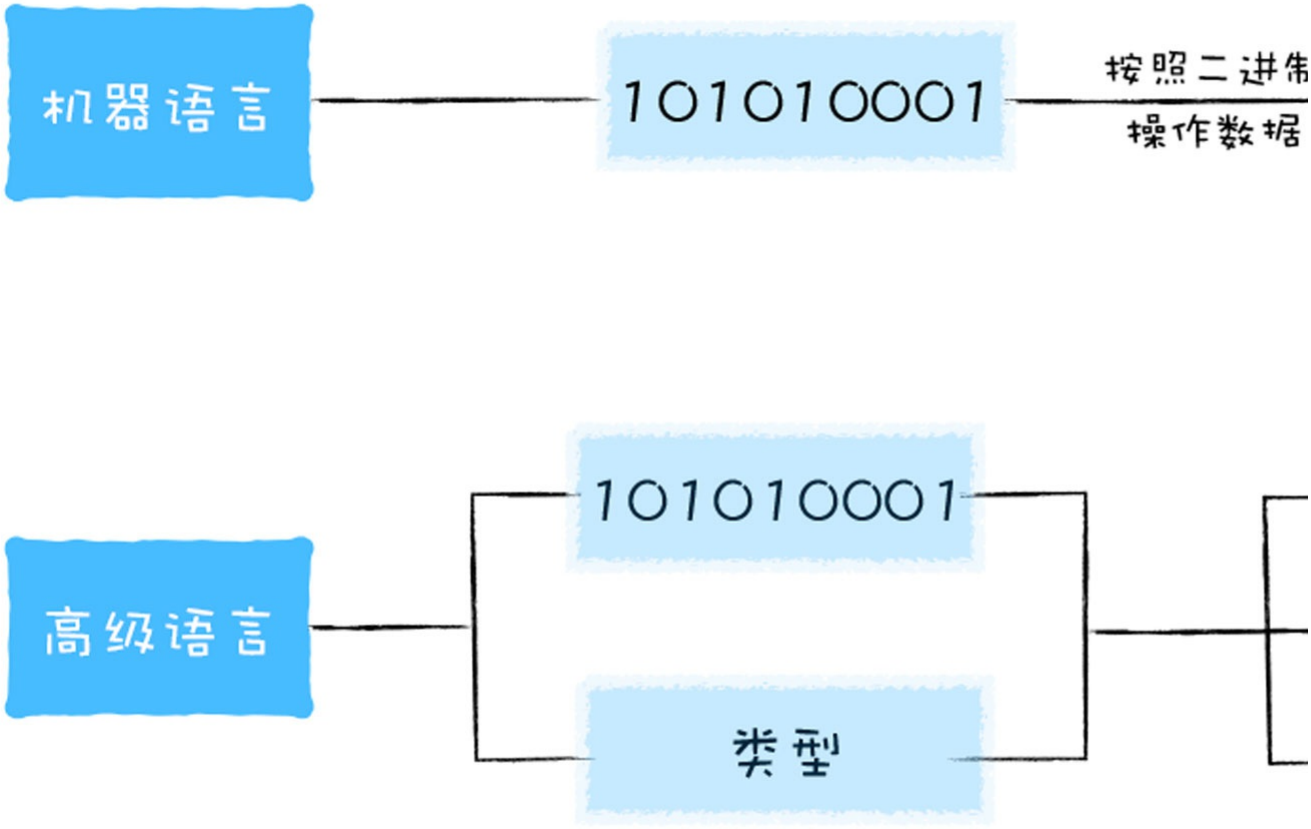
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

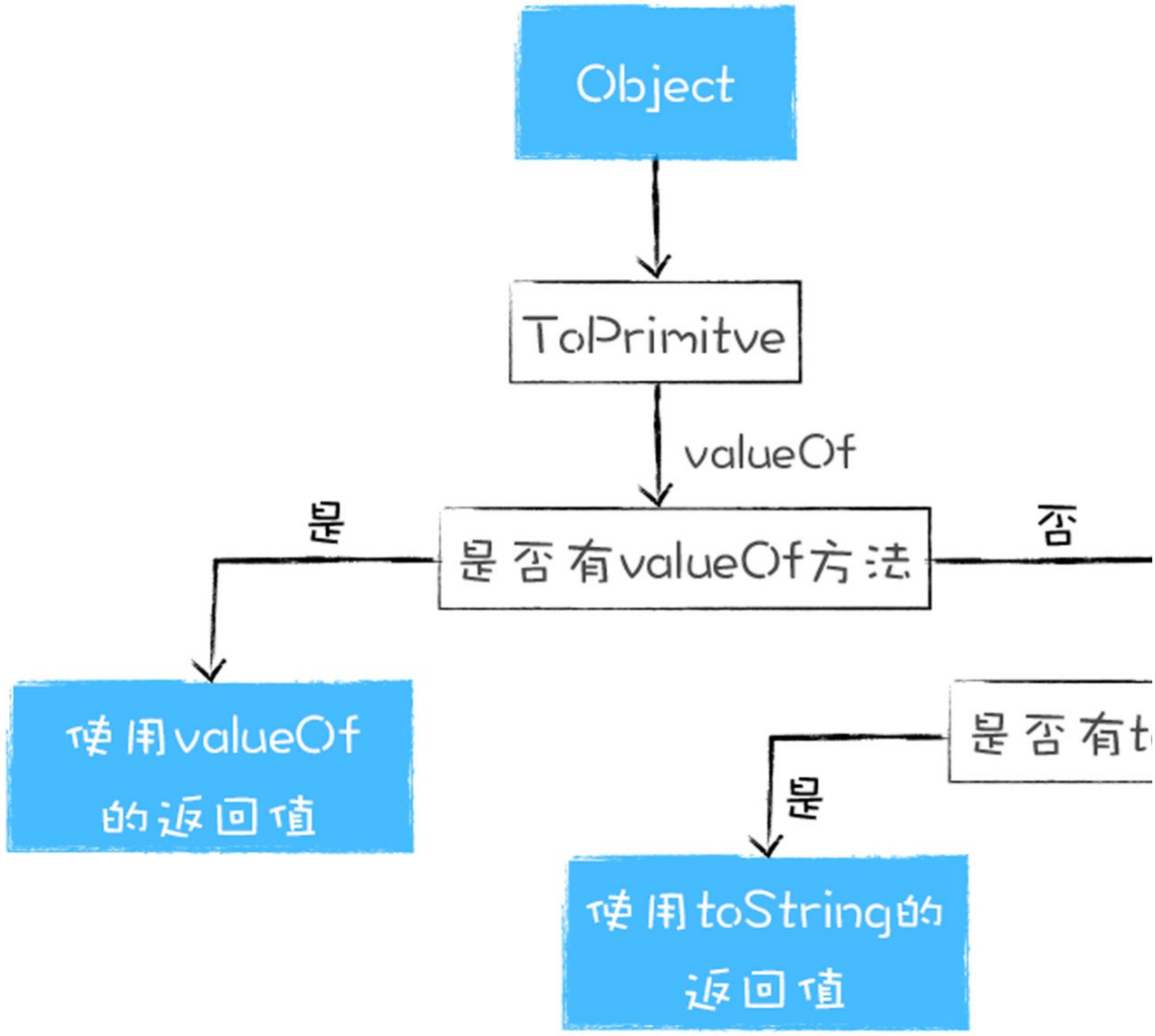
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

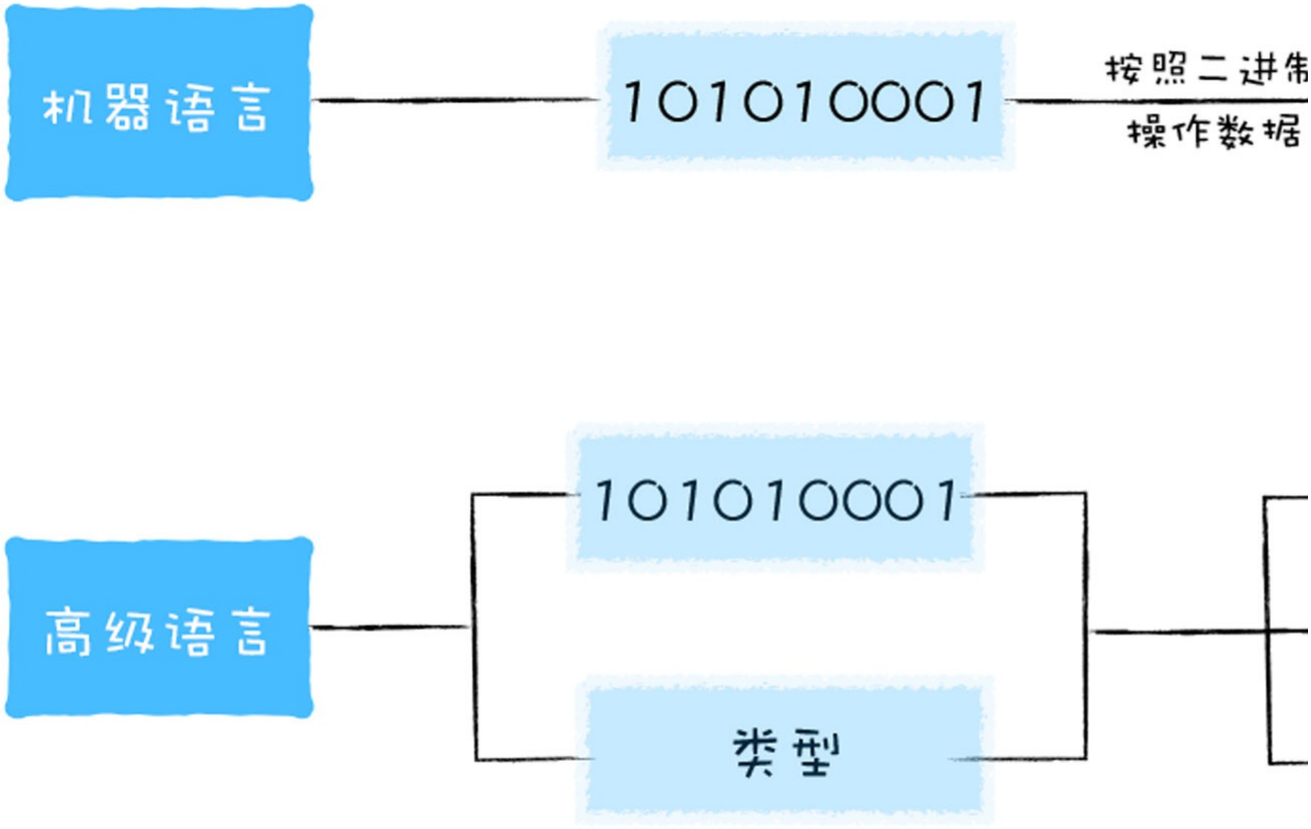
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

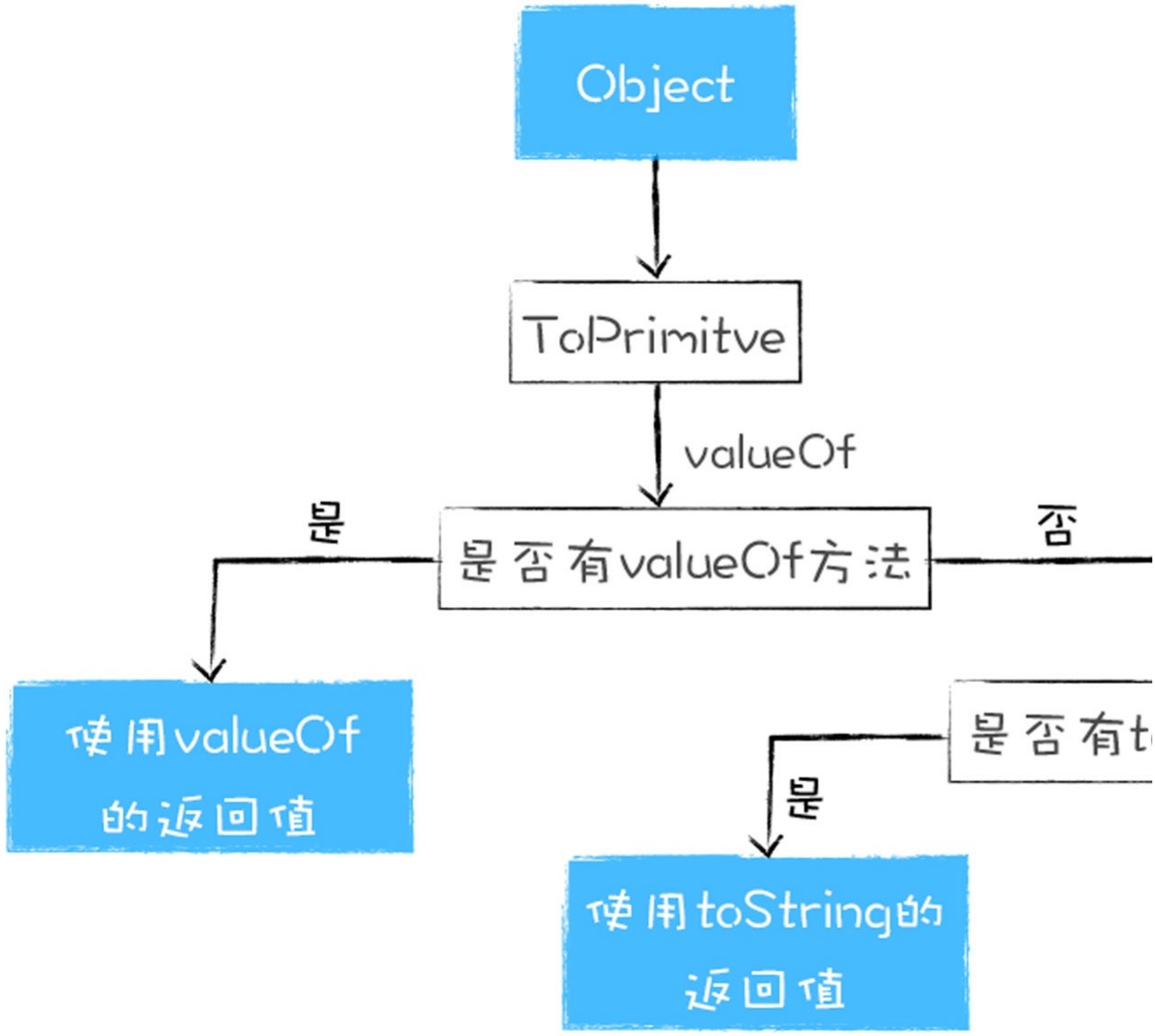
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

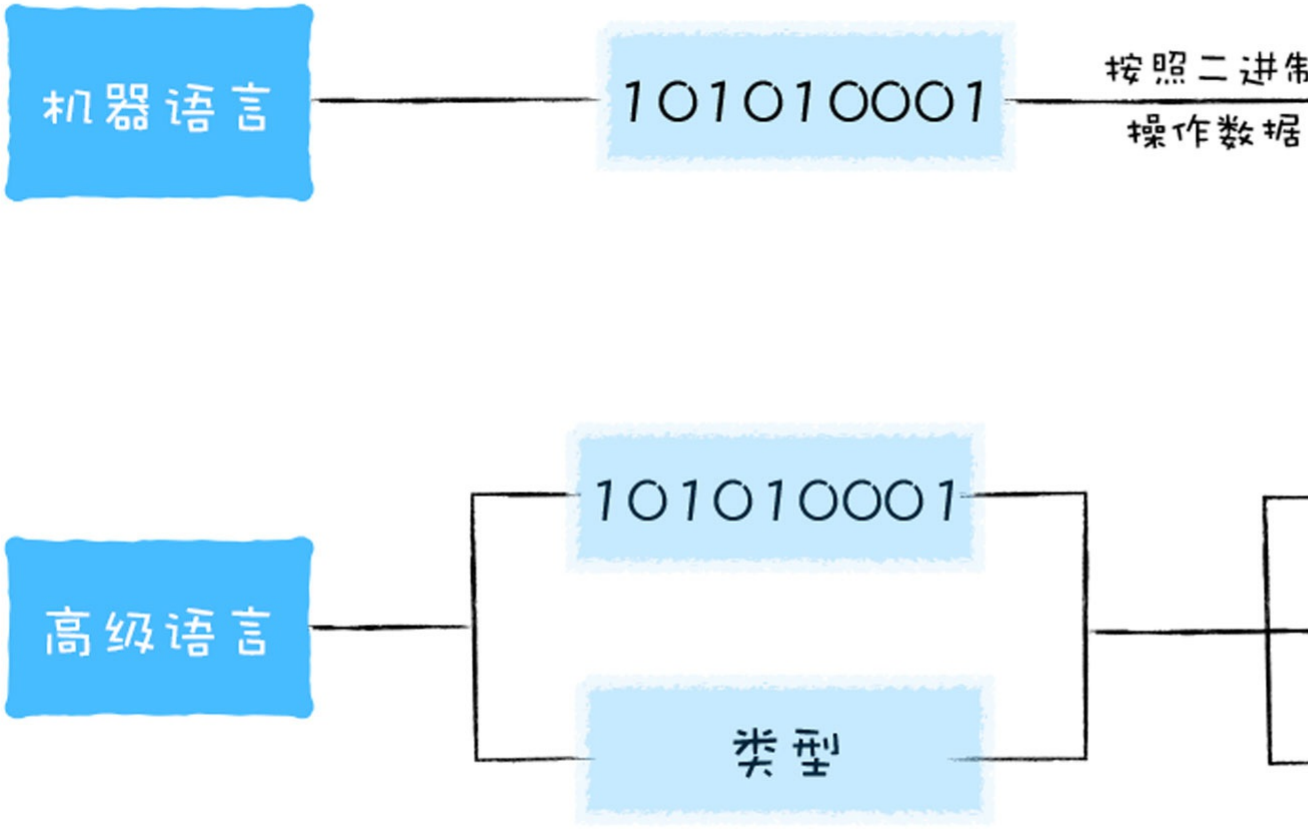
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

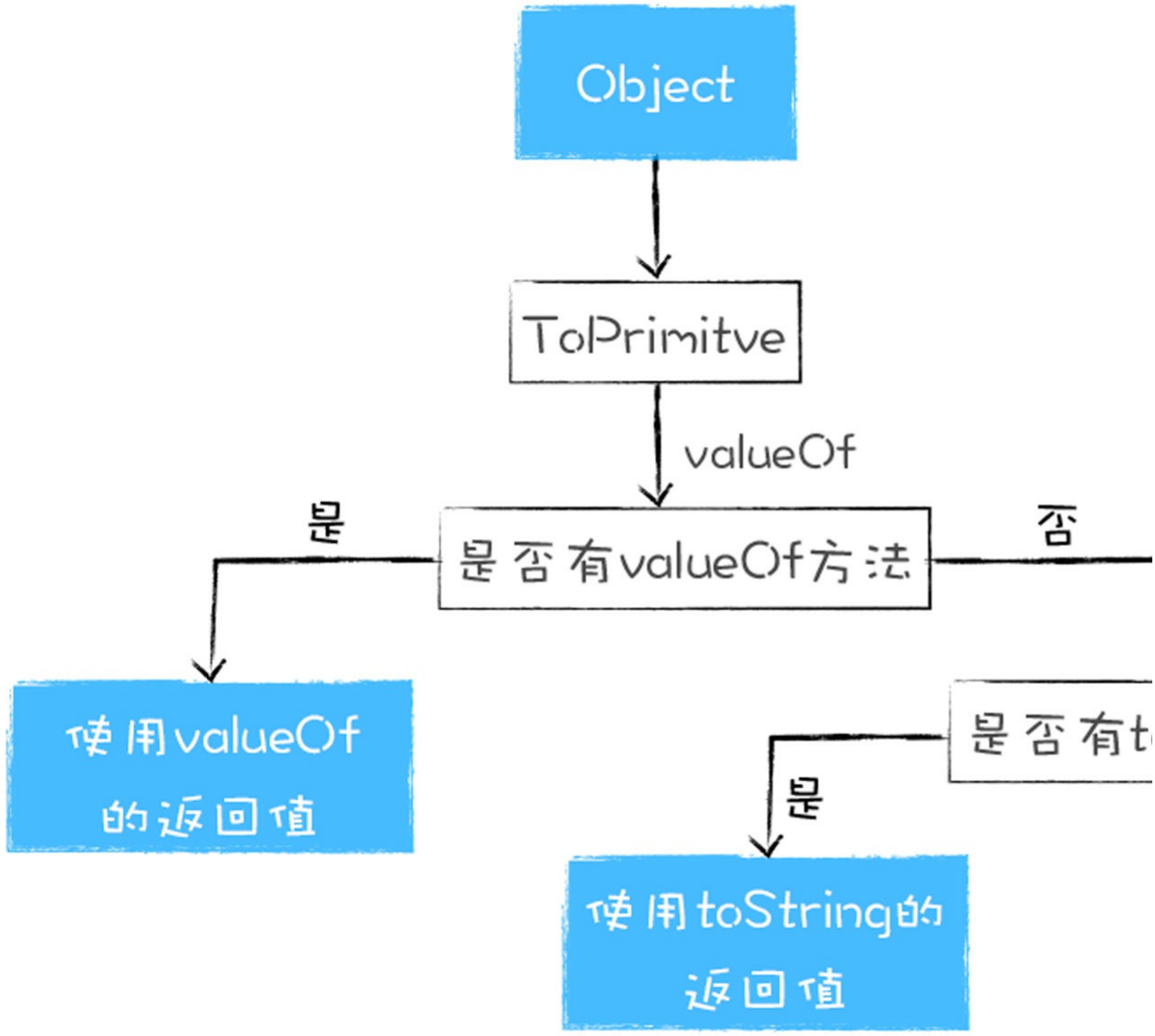
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

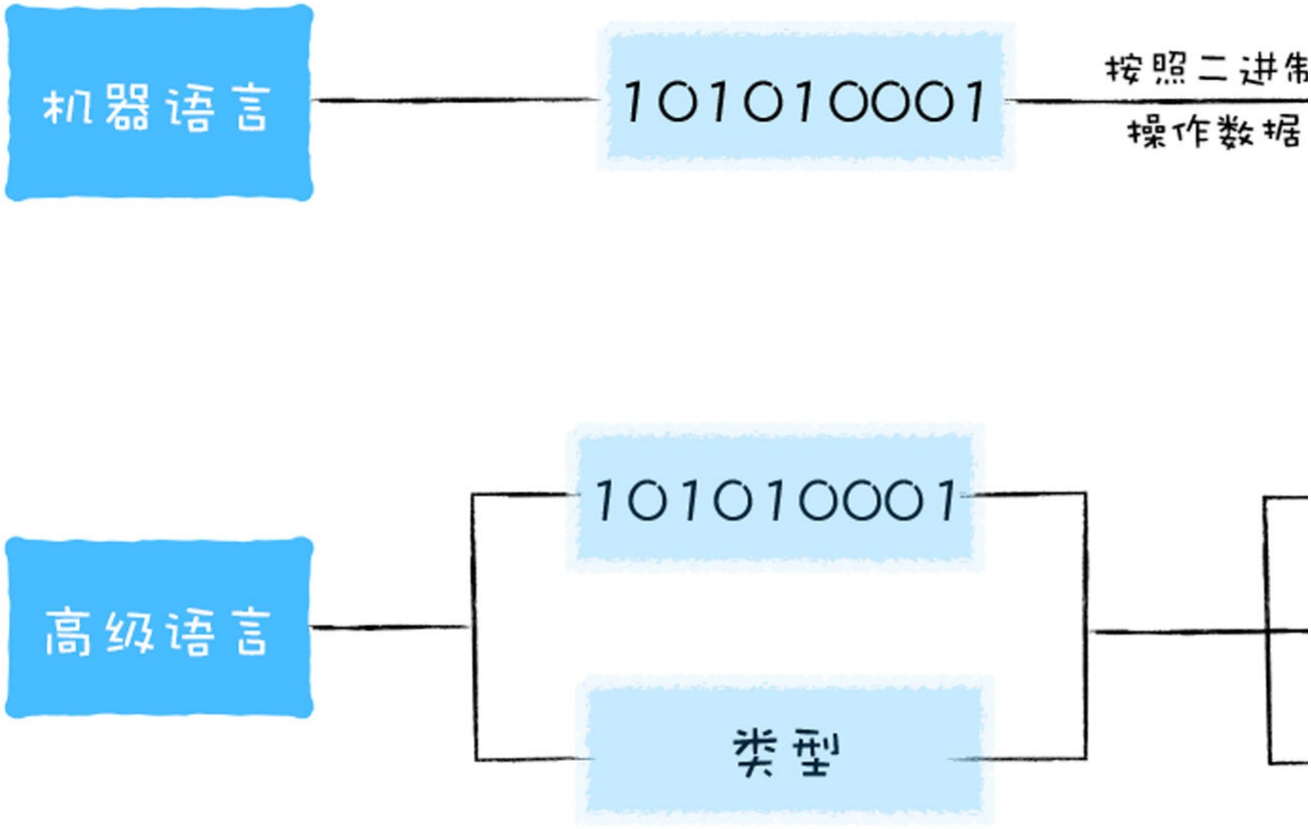
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

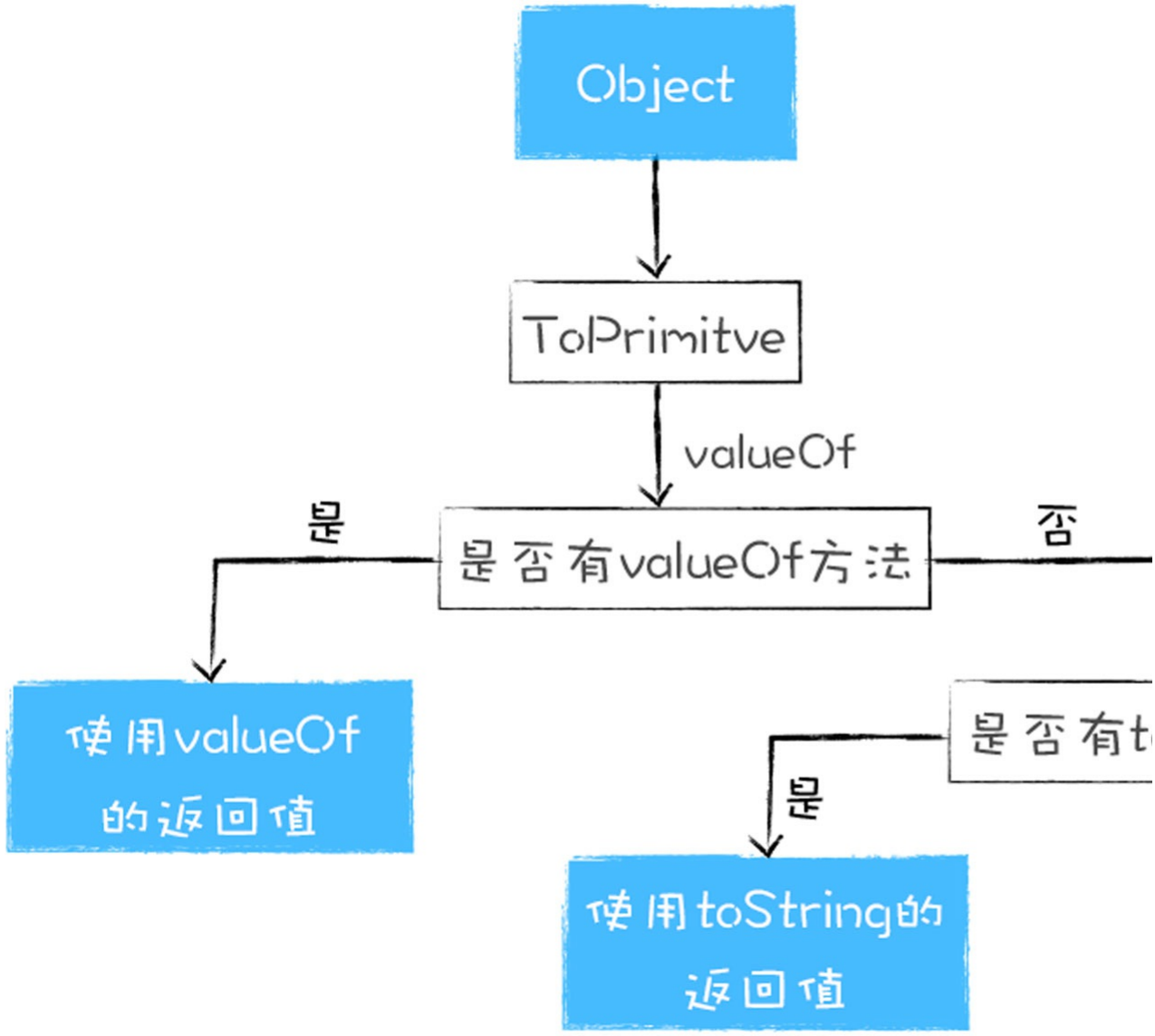
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

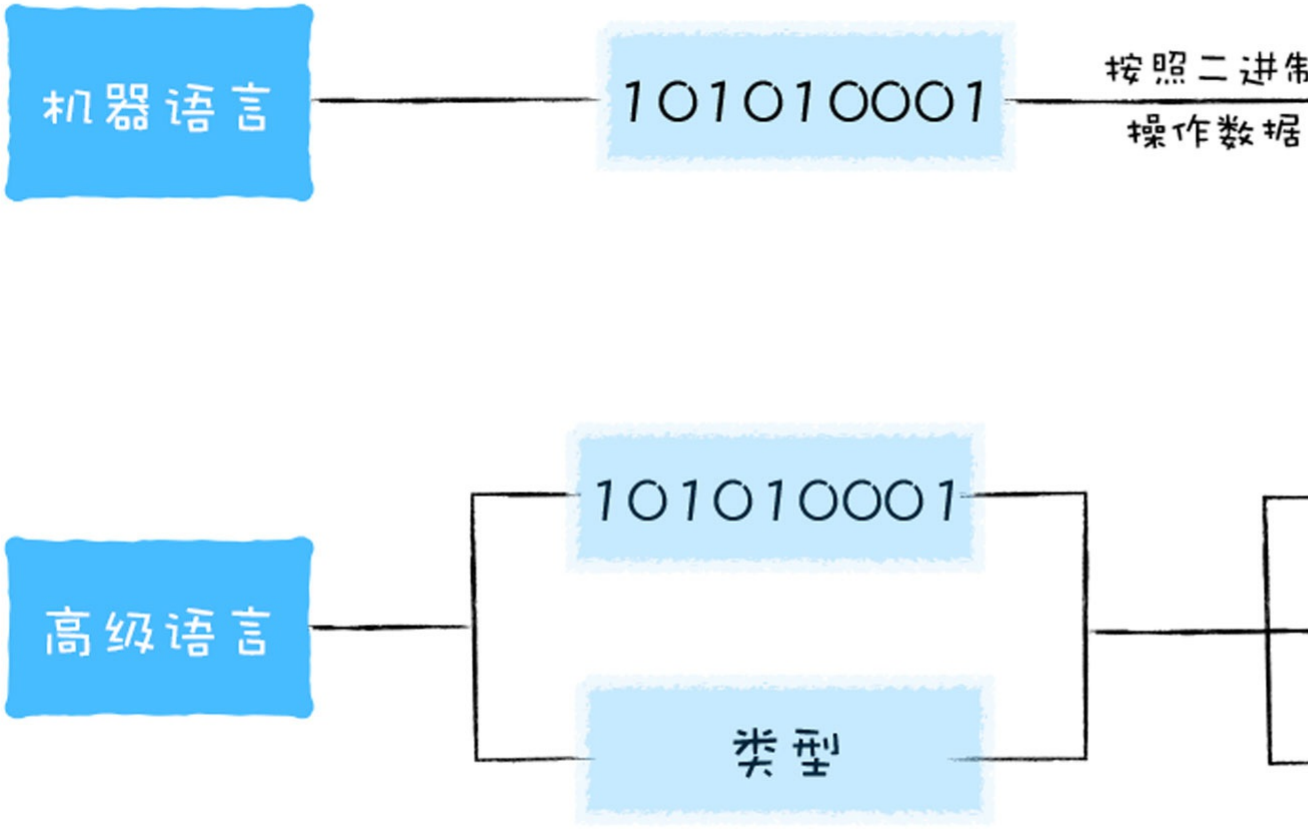
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

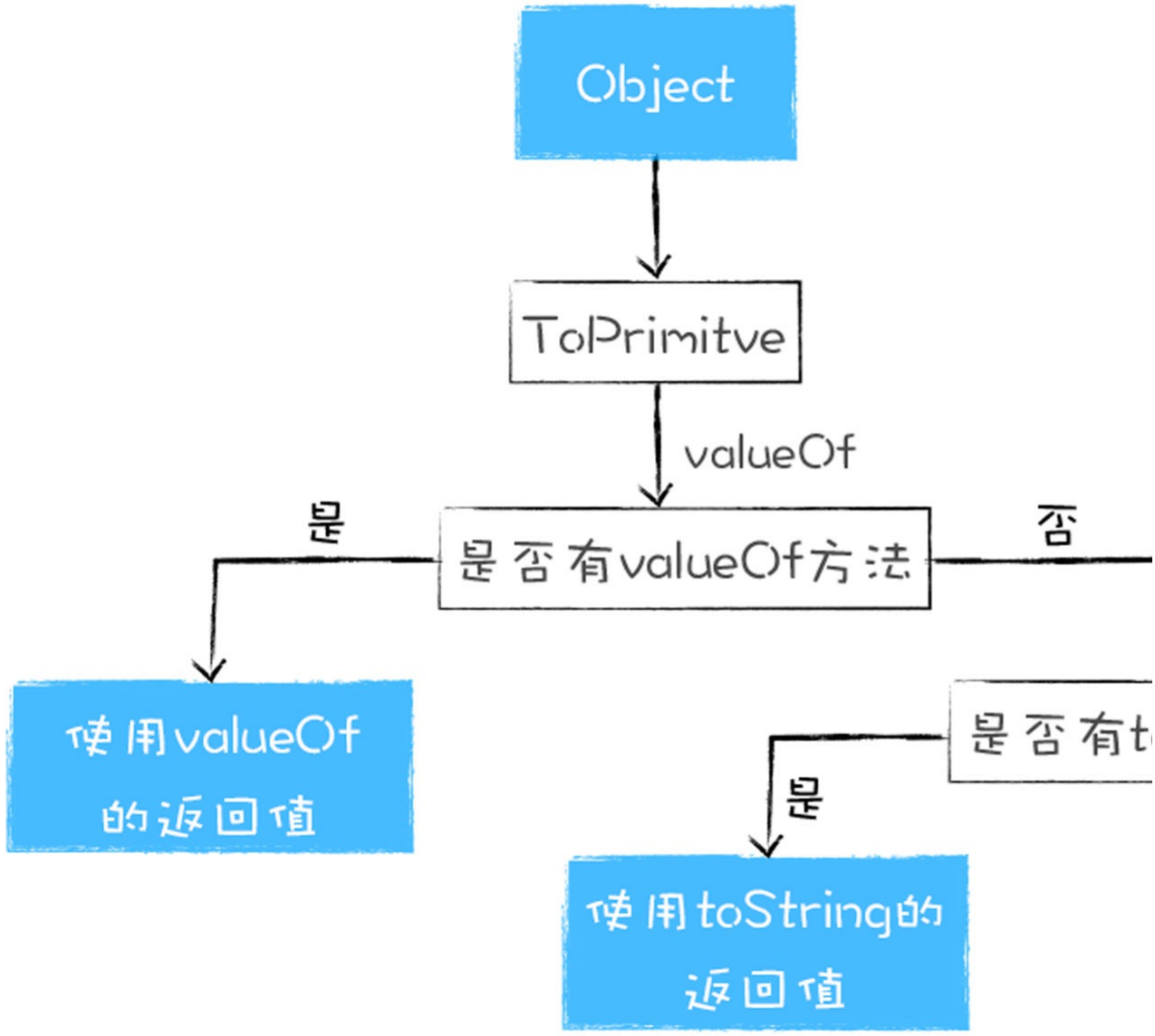
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

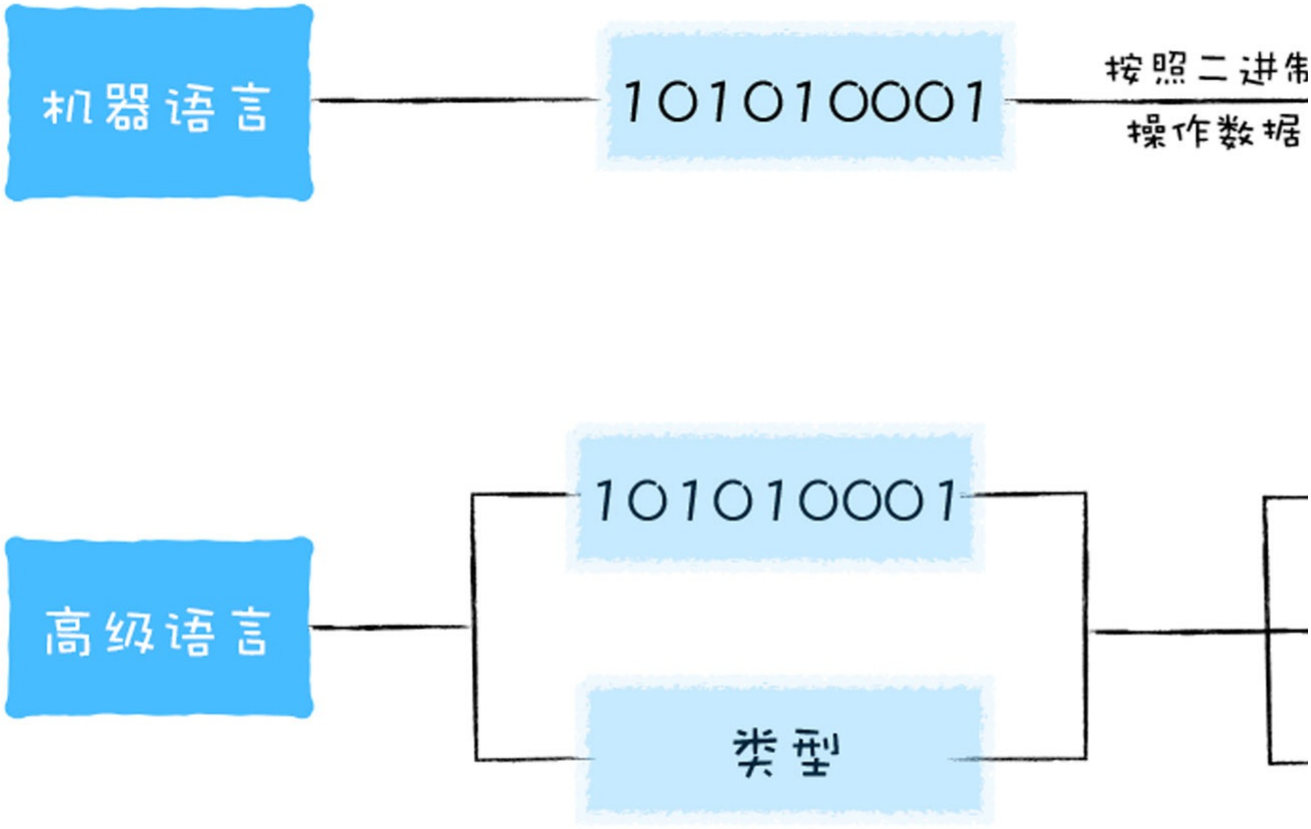
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

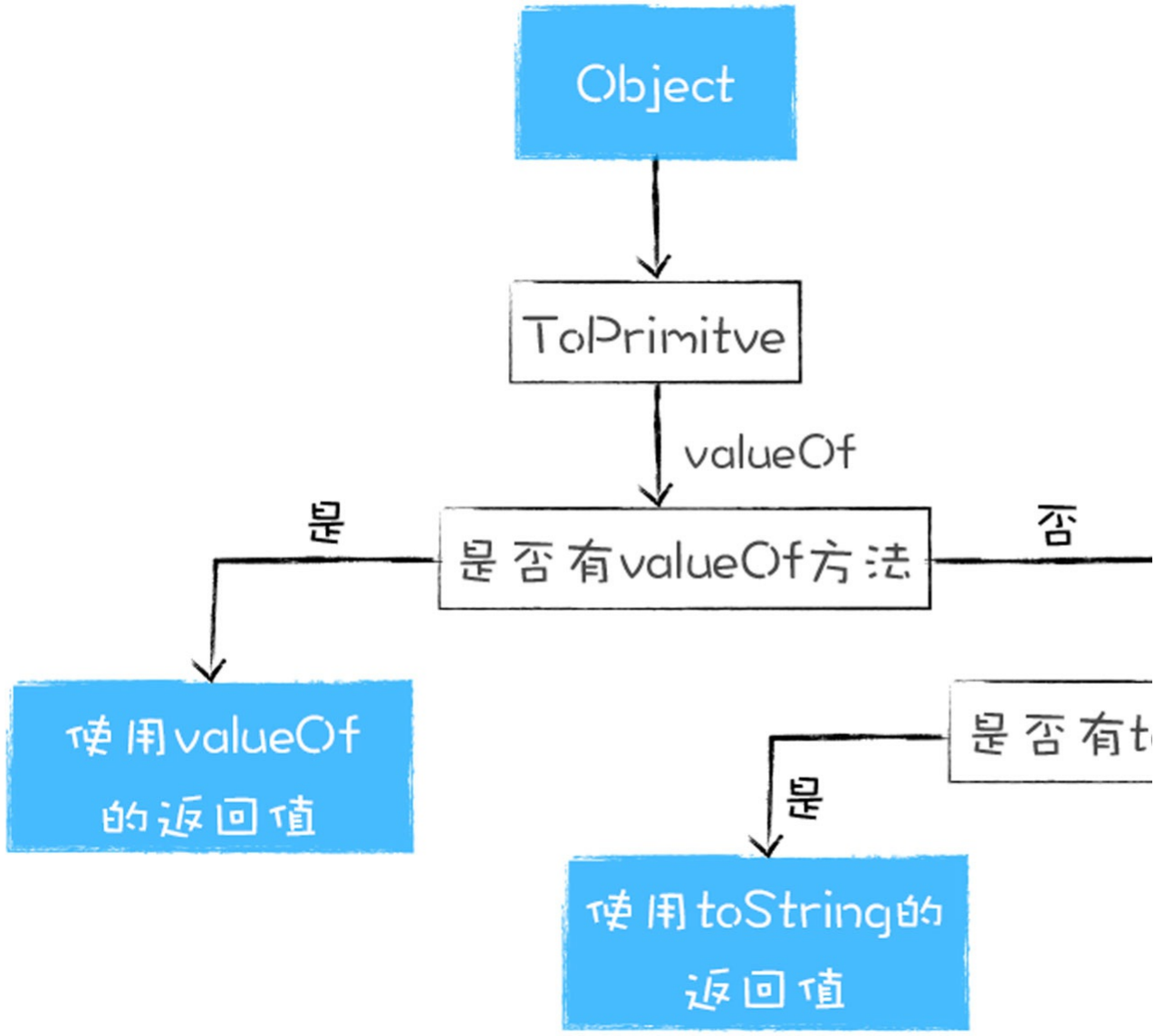
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

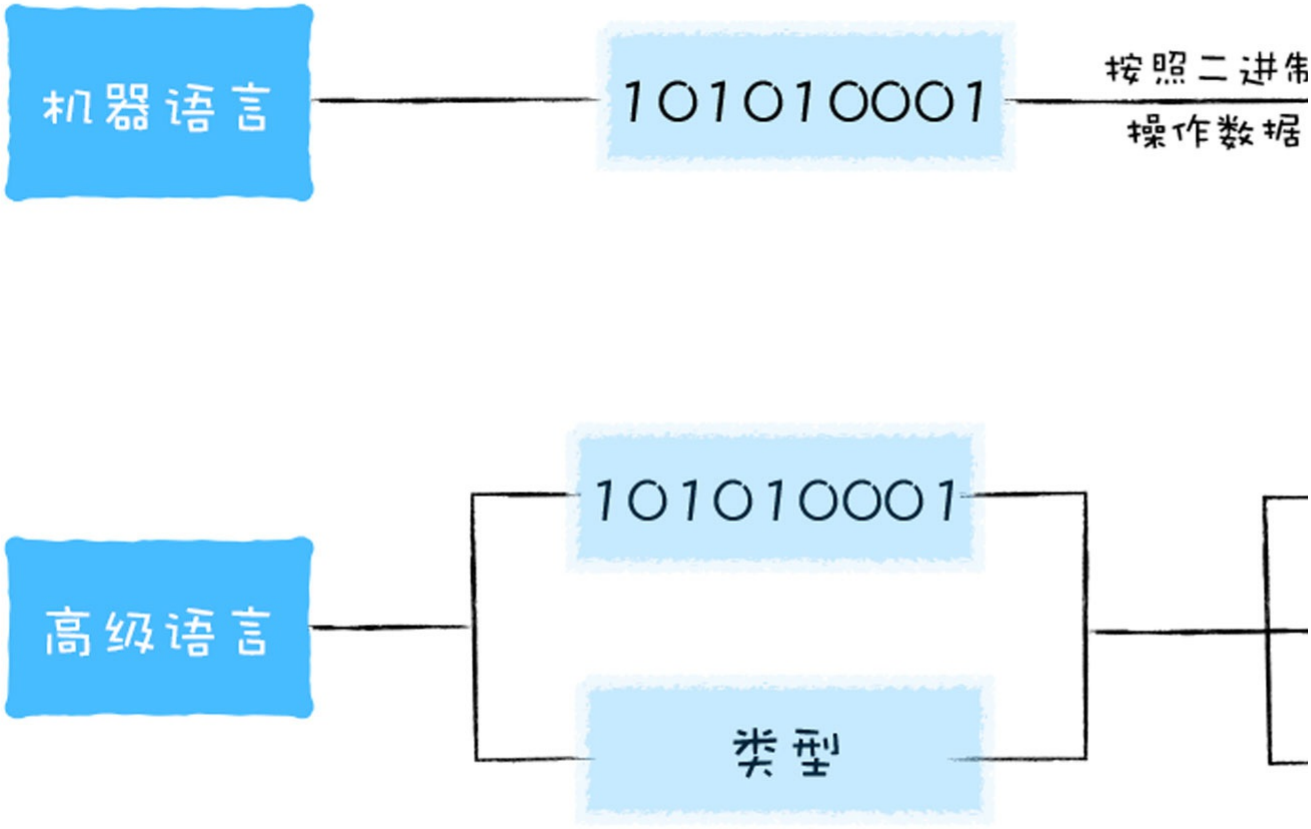
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

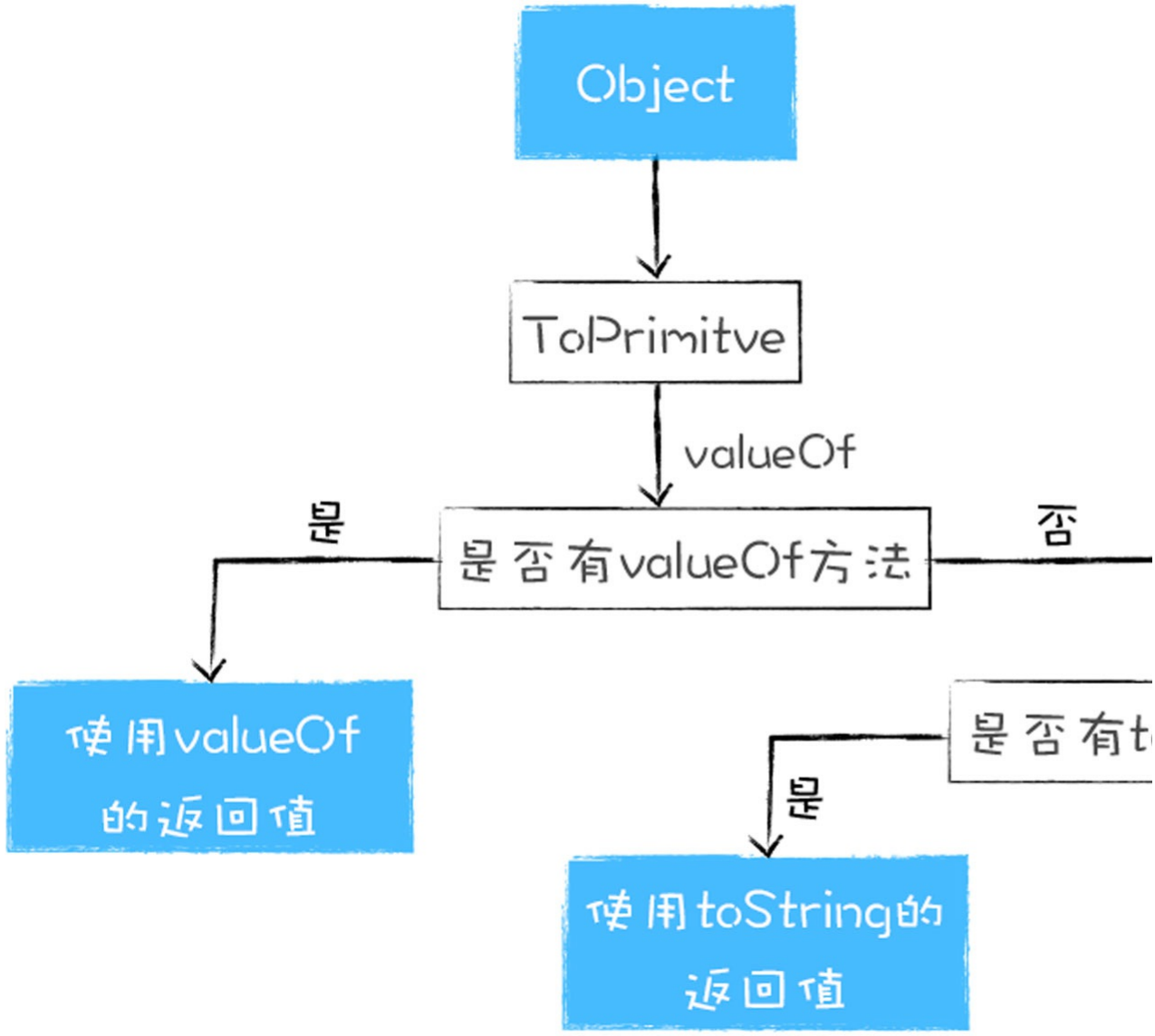
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

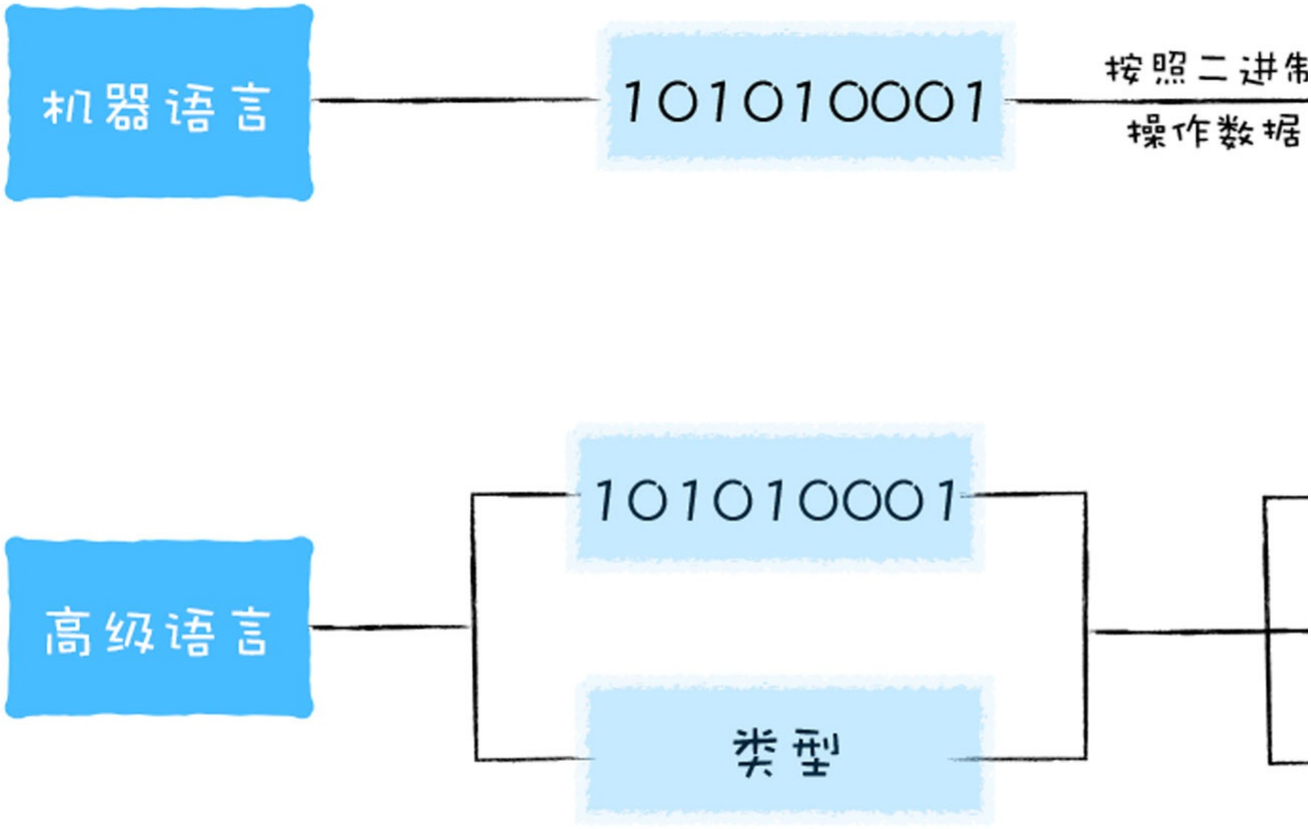
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

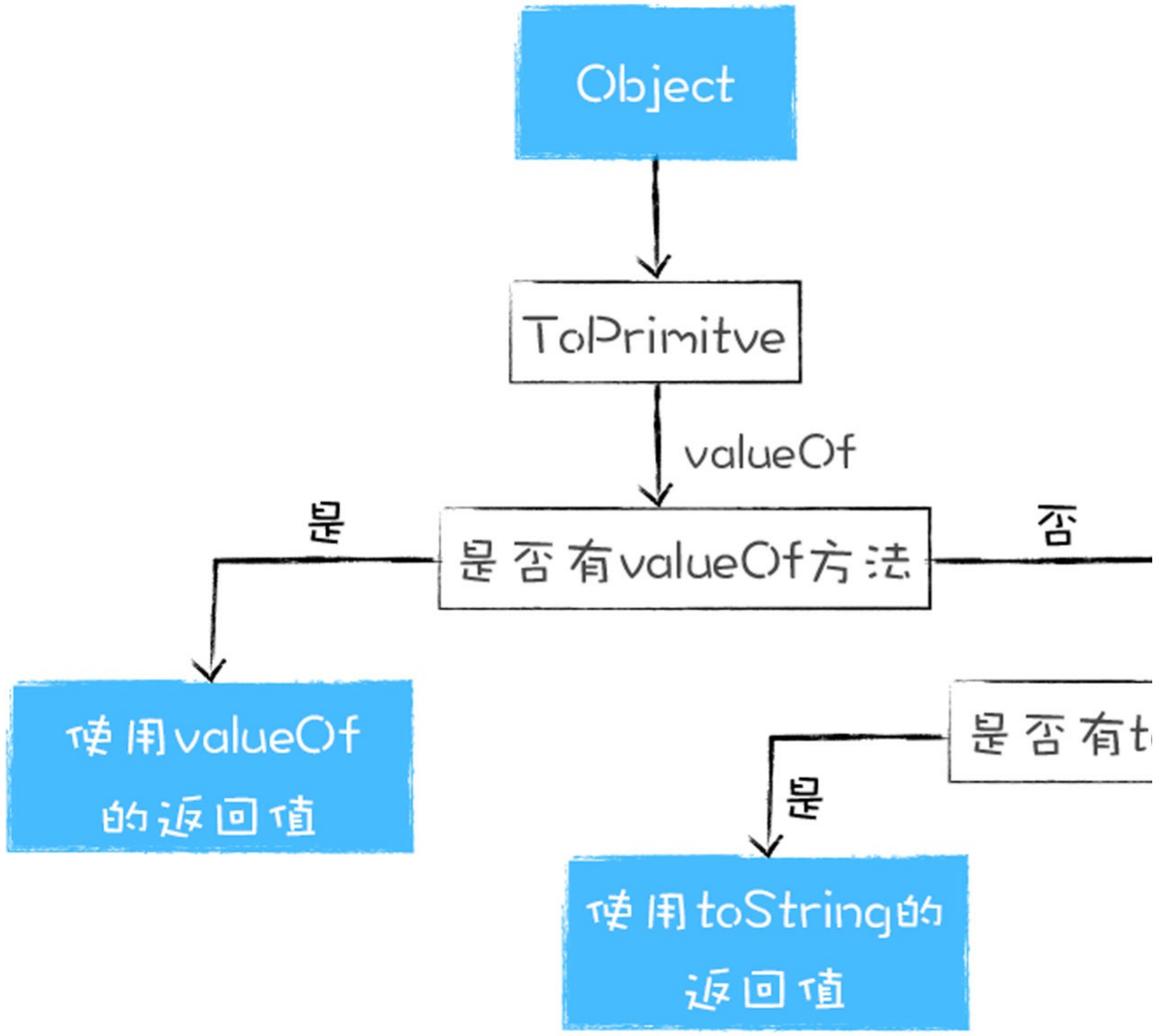
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

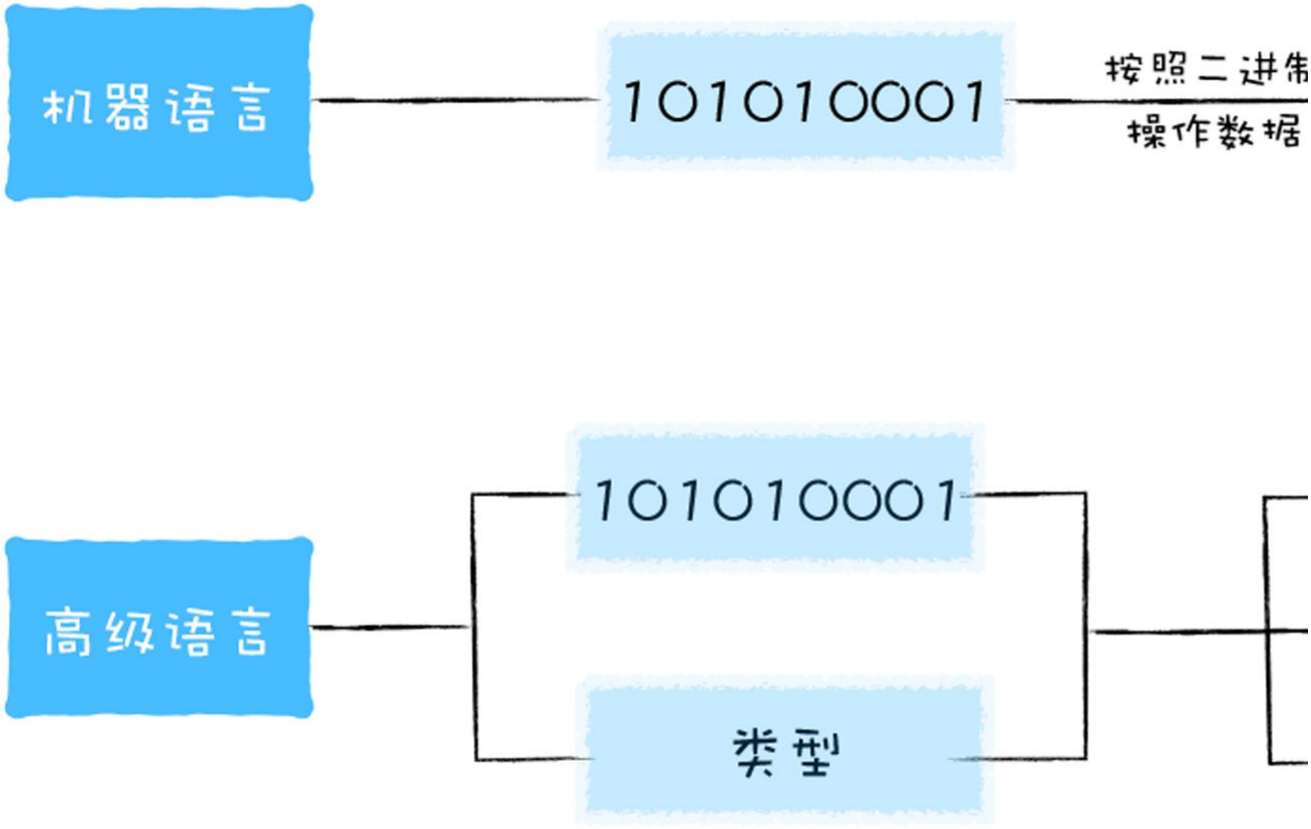
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

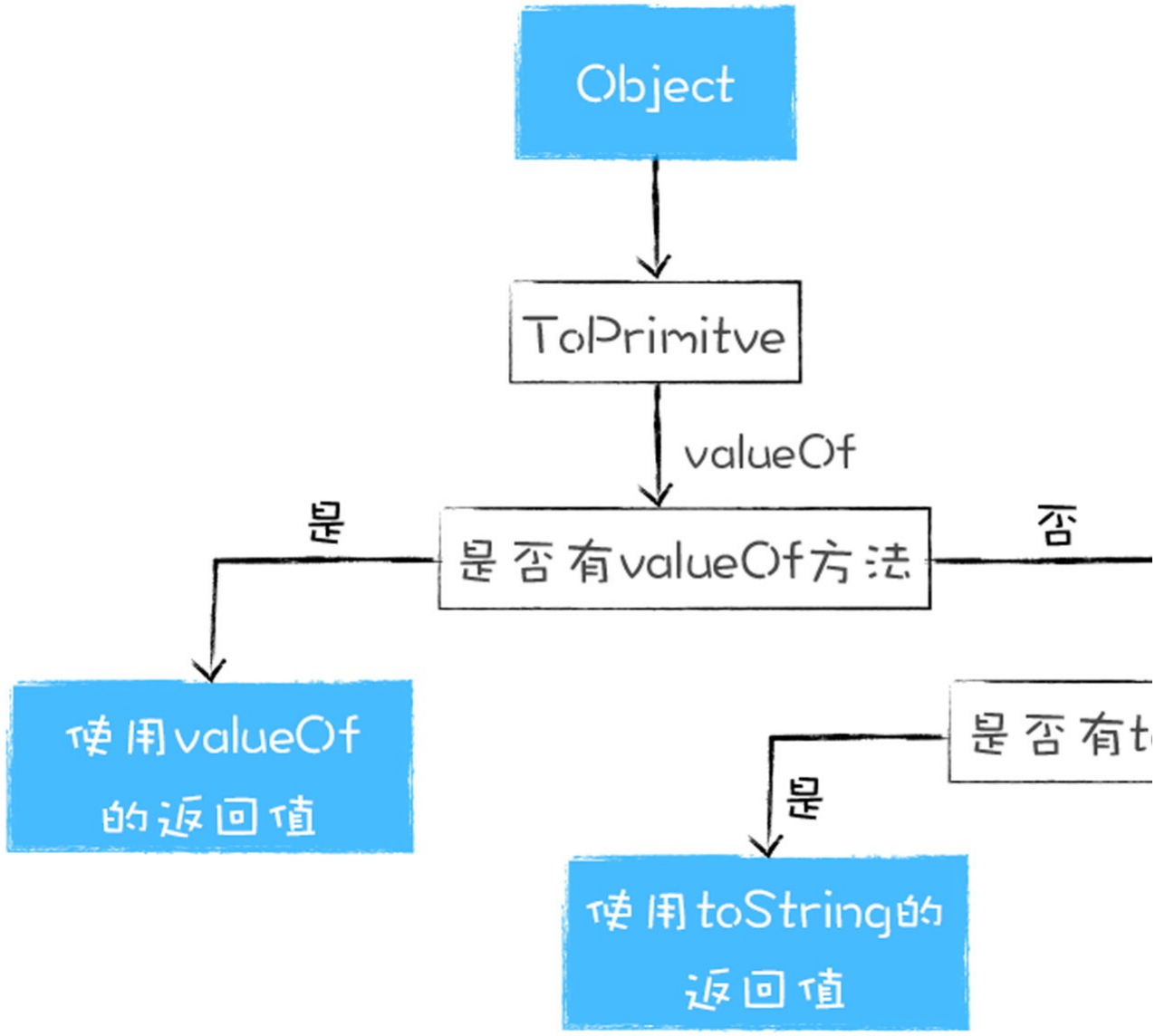
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

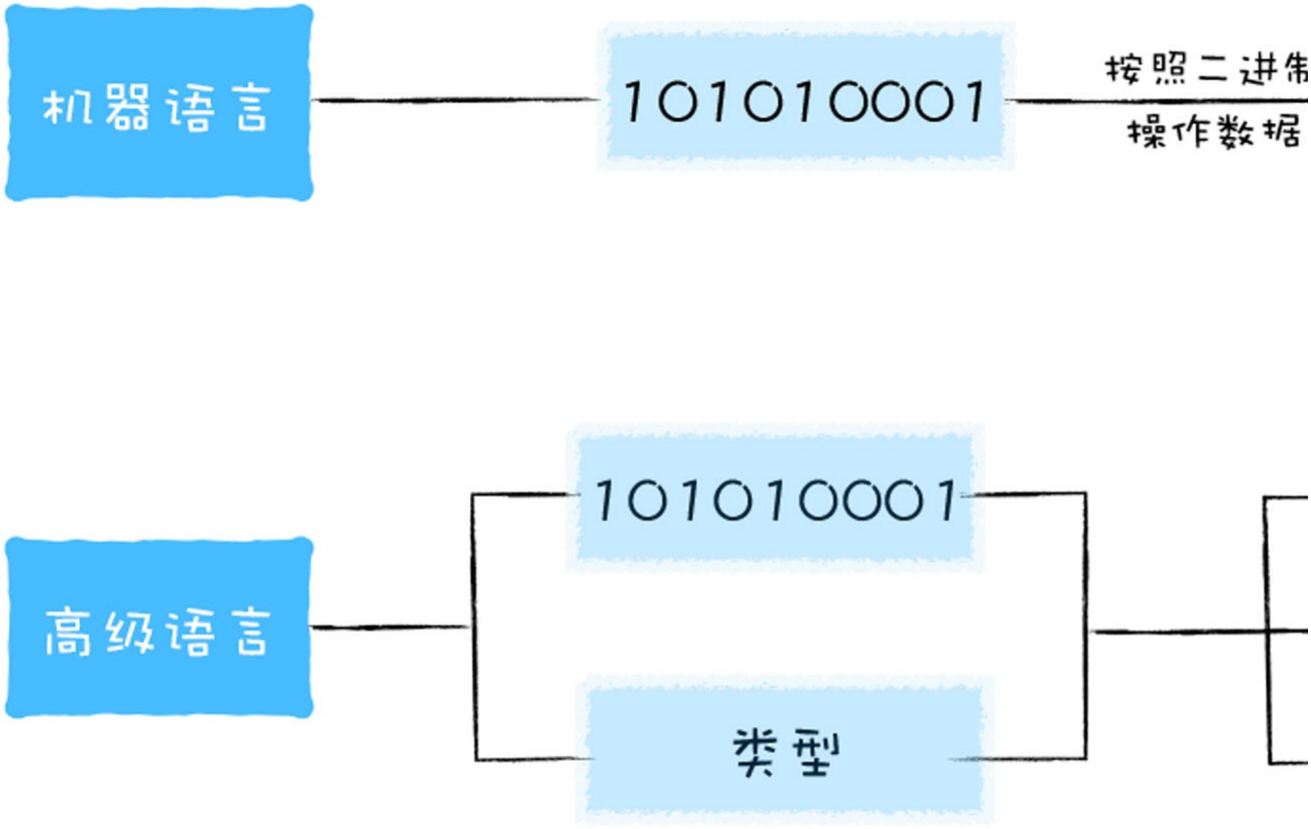
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

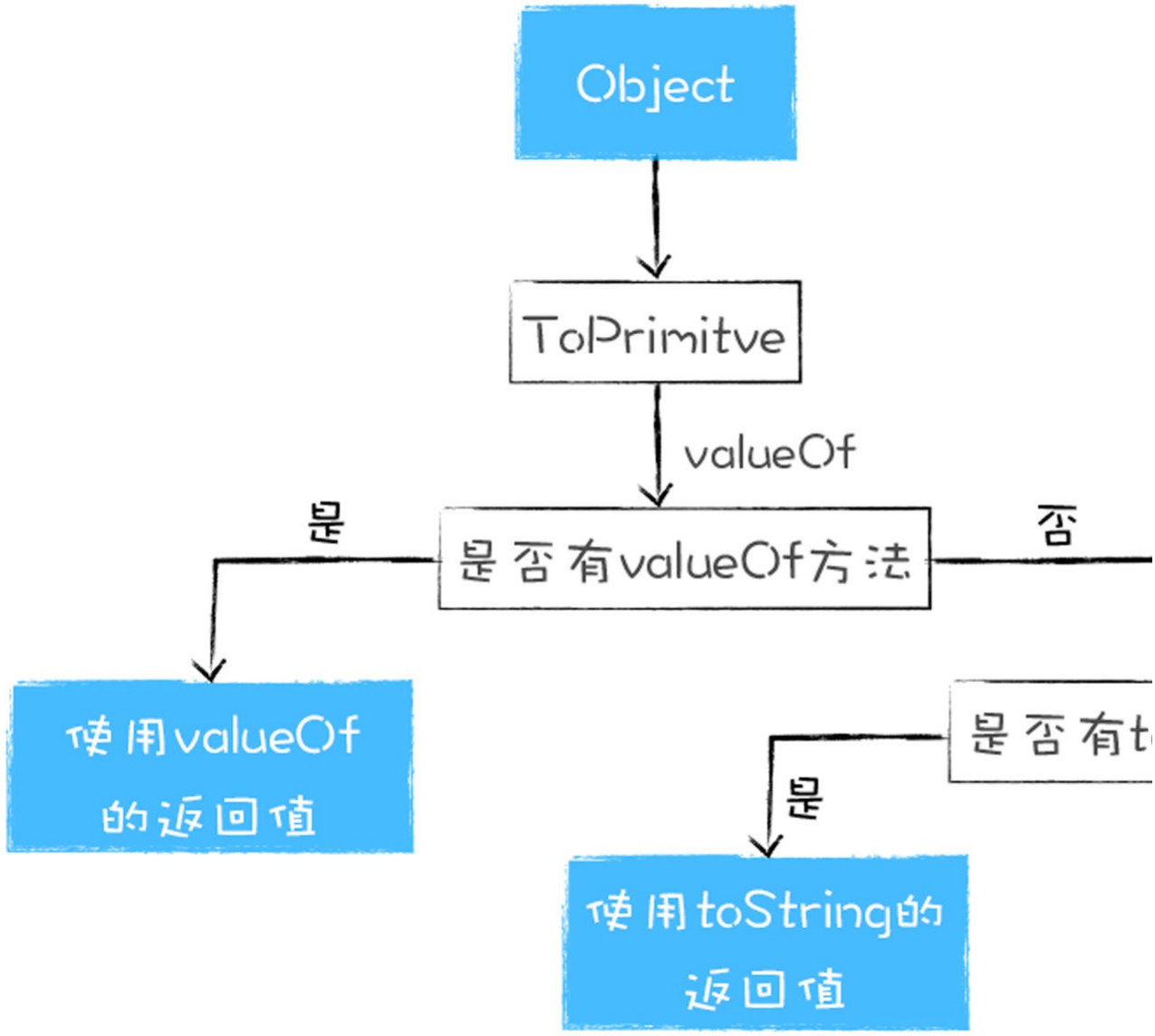
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

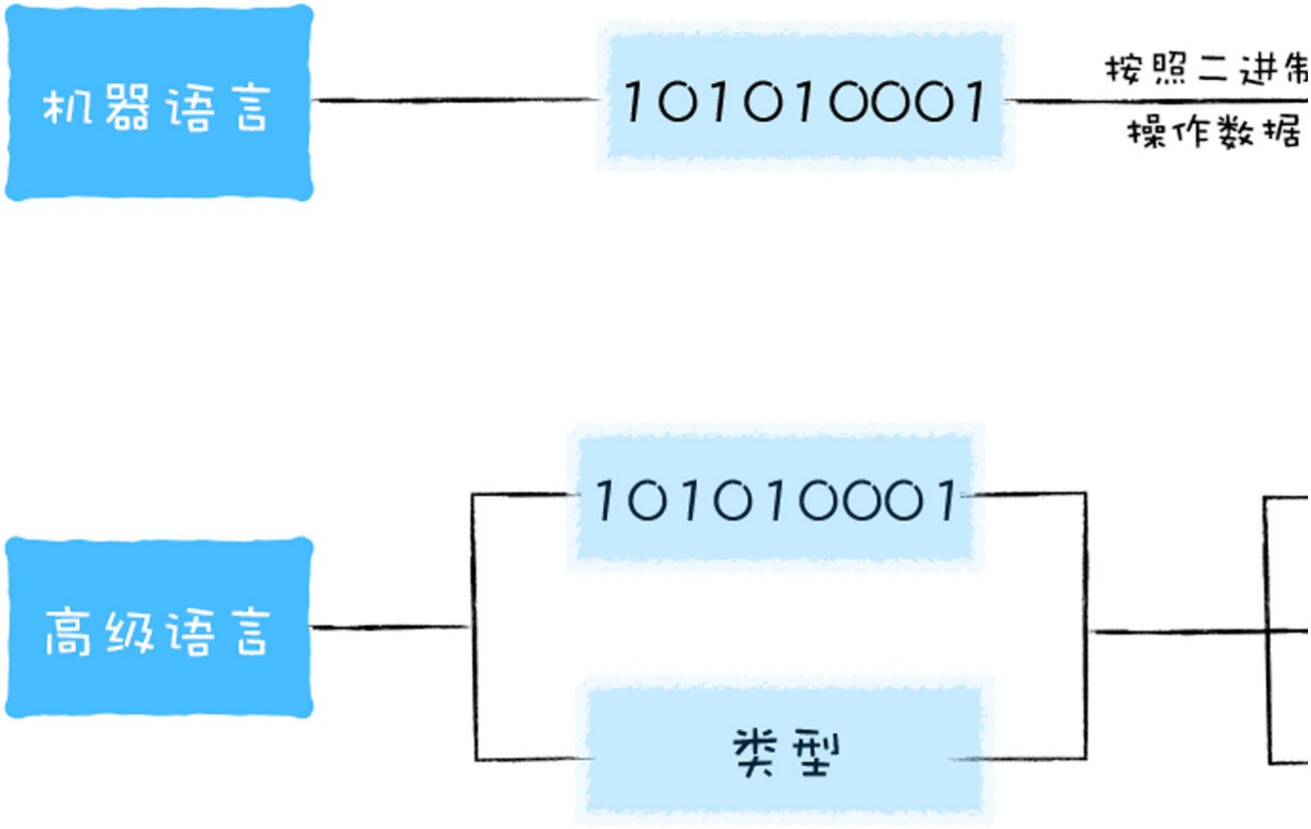
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

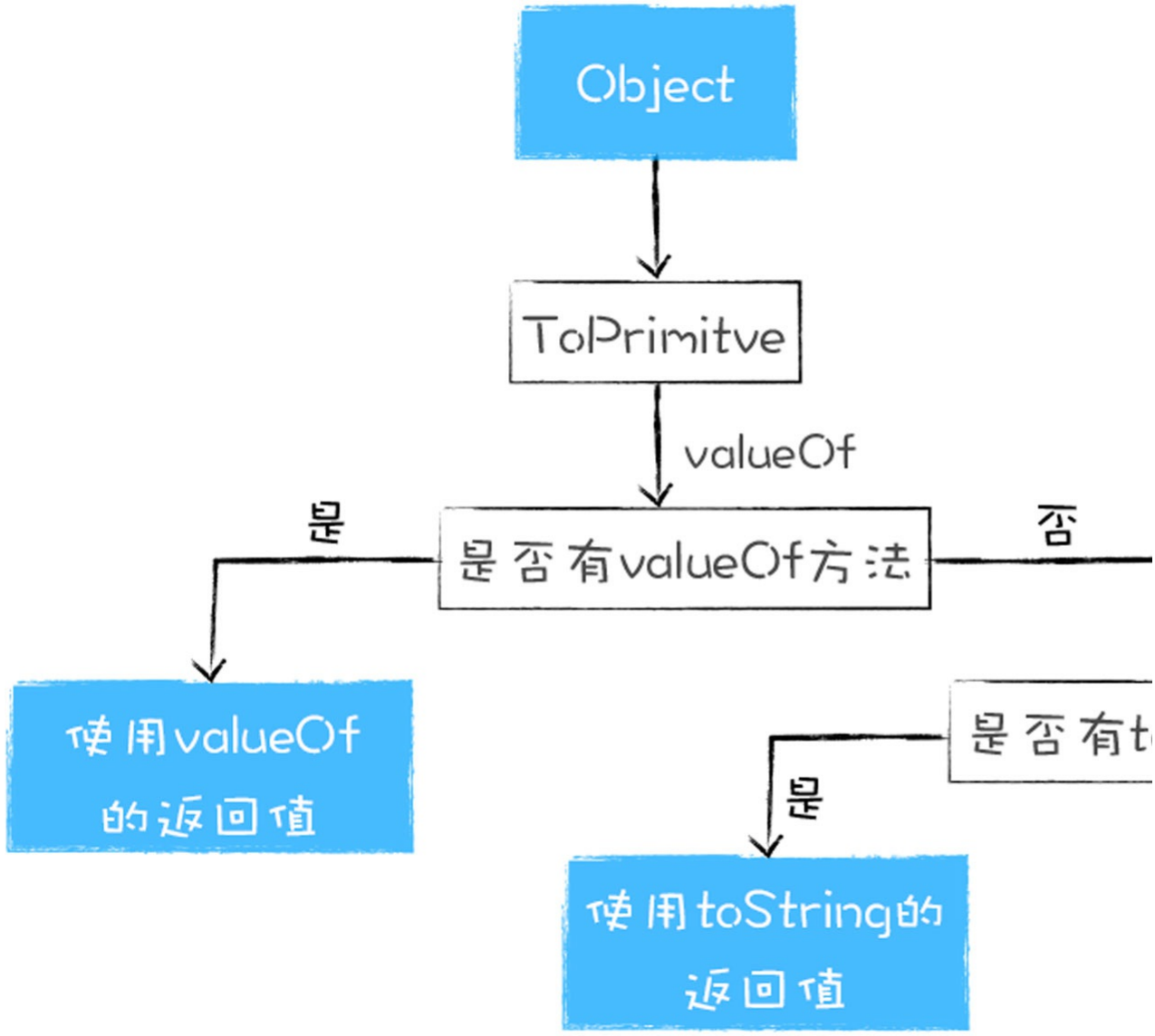
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

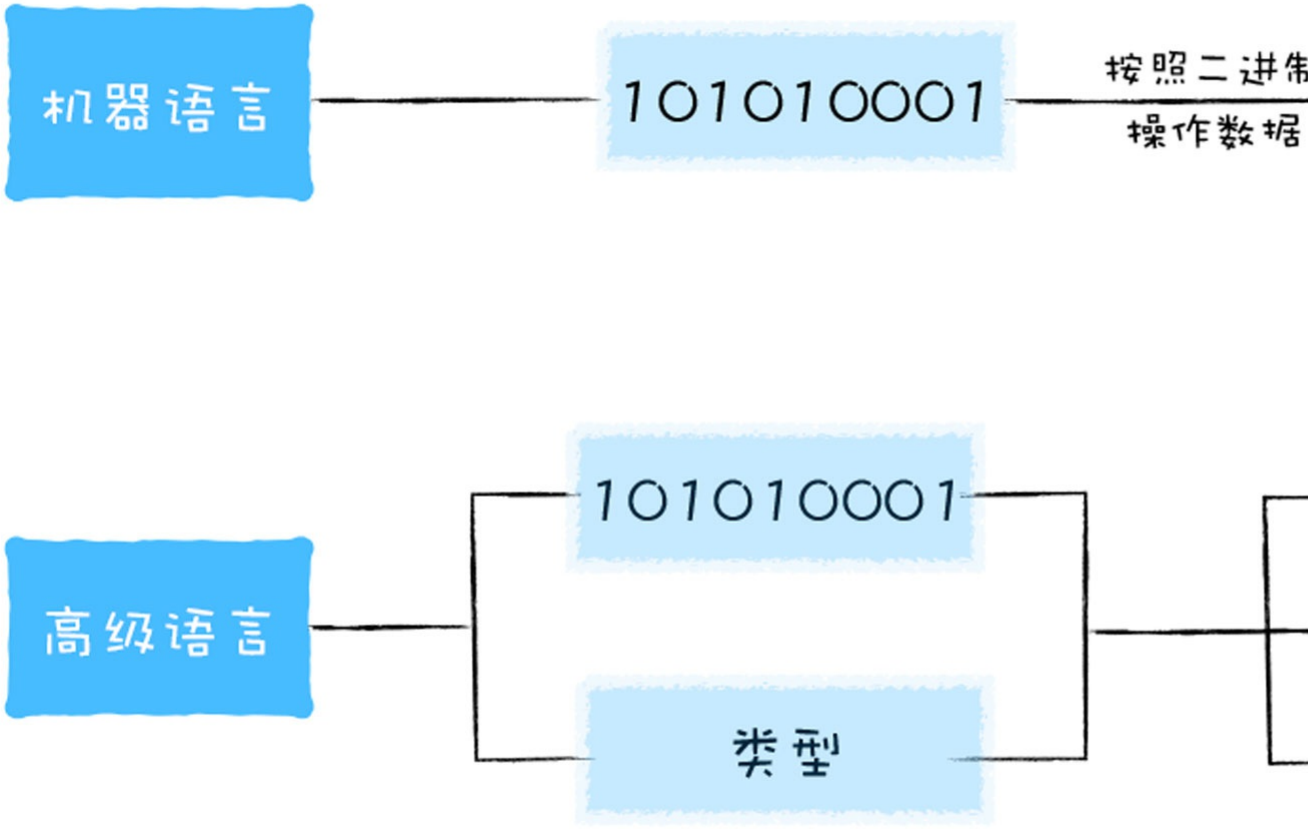
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

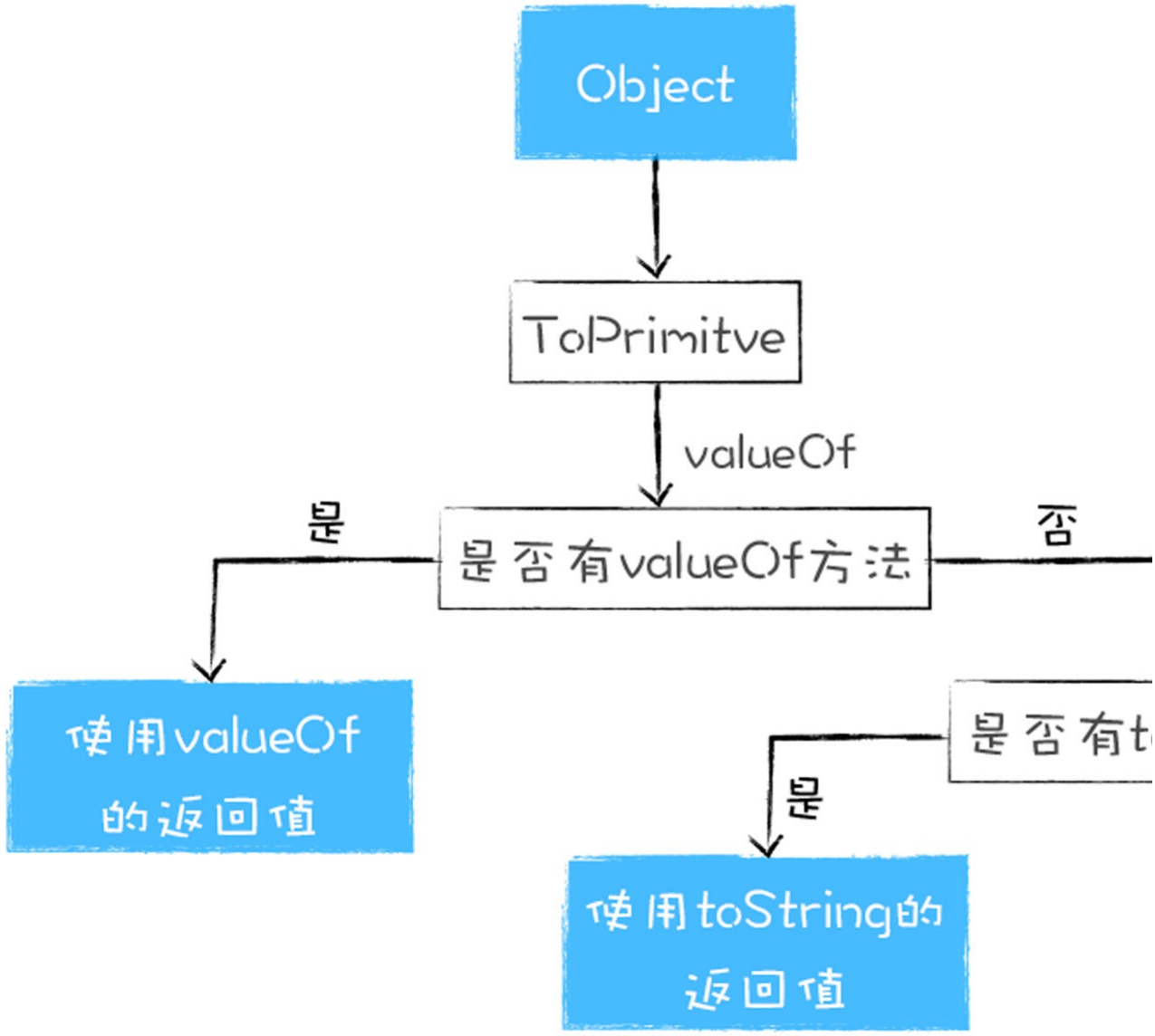
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

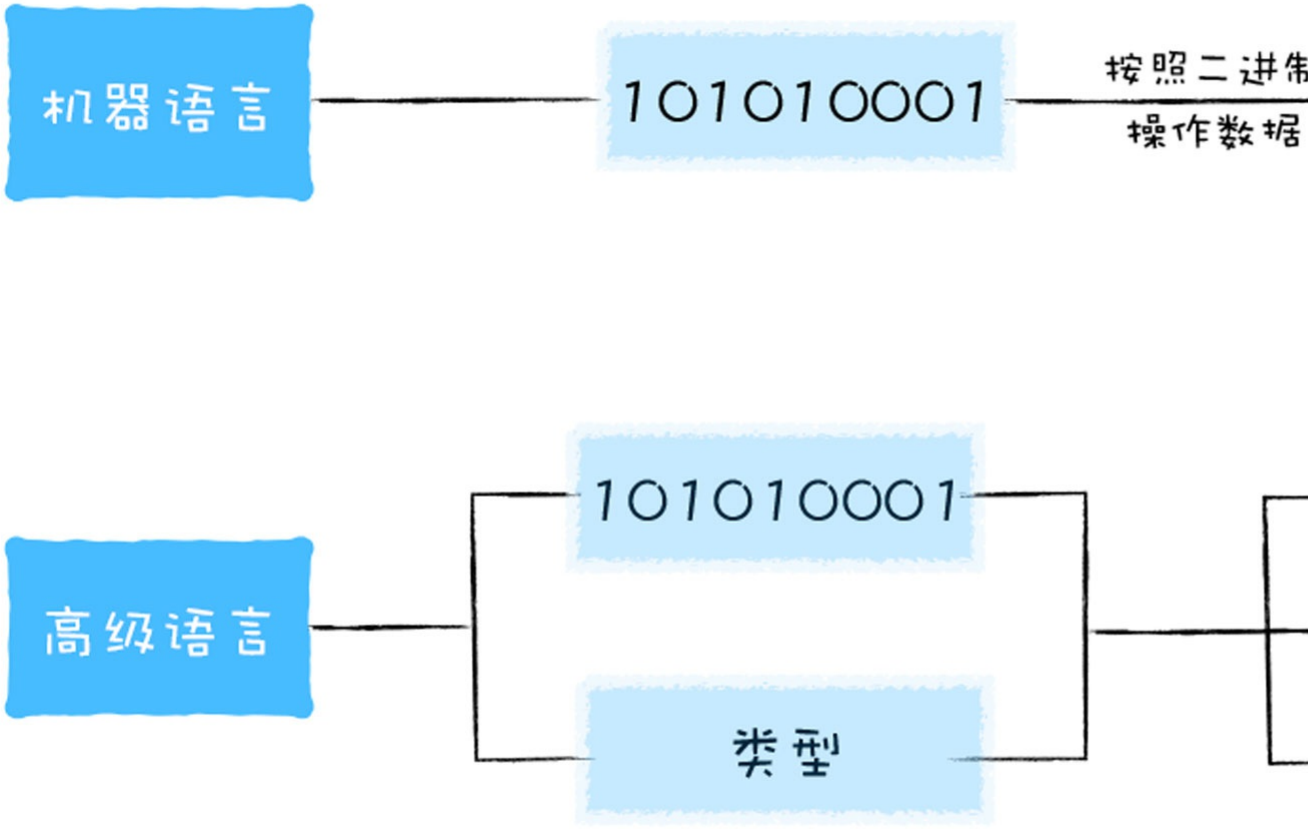
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

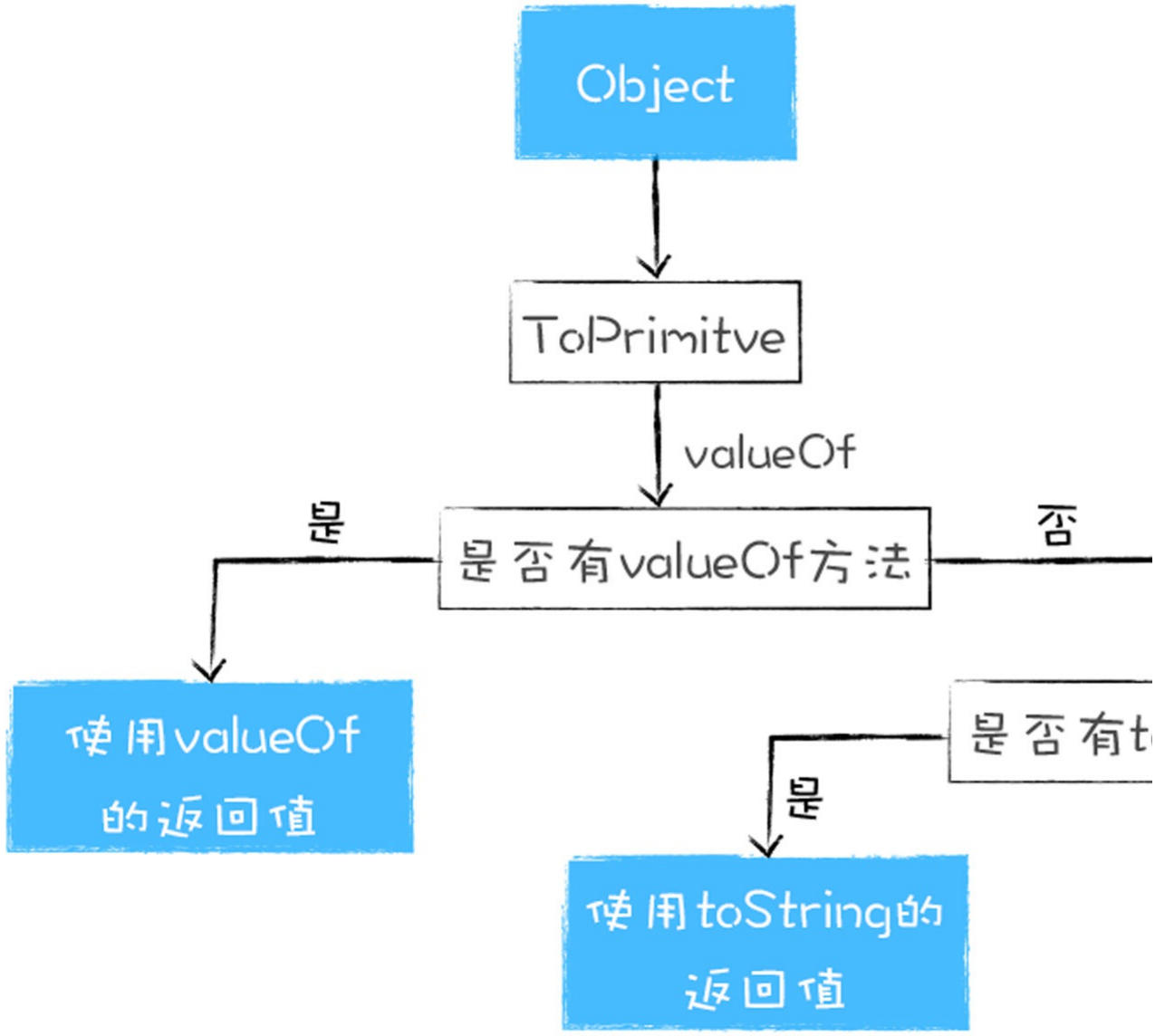
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringj两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。

你好，我是李兵。

前面我们花了很多篇幅聊了JavaScript中最基础却很容易被忽略的“对象”，以及V8是怎么处理“对象”的，今天我们继续来聊另一个非常基础，同时也是很容易被大家忽略的问题，那就是JavaScript中的“类型系统”。

在理解这个概念之前，你可以先思考一个简单的表达式，那就是在JavaScript中，“1+‘2’等于多少？”

其实这相当于是在问，在JavaScript中，让数字和字符串相加是会报错，还是可以正确执行。如果能正确执行，那么结果是等于数字3，还是字符串“3”，还是字符串“12”呢？

如果你尝试在Python中使用数字和字符串进行相加操作，那么Python虚拟机会直接返回一个执行错误，错误提示是这样的：

```
>>> 1+'2'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

这段错误代码提示了这是个类型错误，表明Python并不支持数字类型和字符串类型相加的操作。

不过在JavaScript中执行这段表达式，是可以返回一个结果的，最终返回的结果是字符串“12”。

最终结果如下所示：

```
>>> 1+'2'
>>> "12"
```

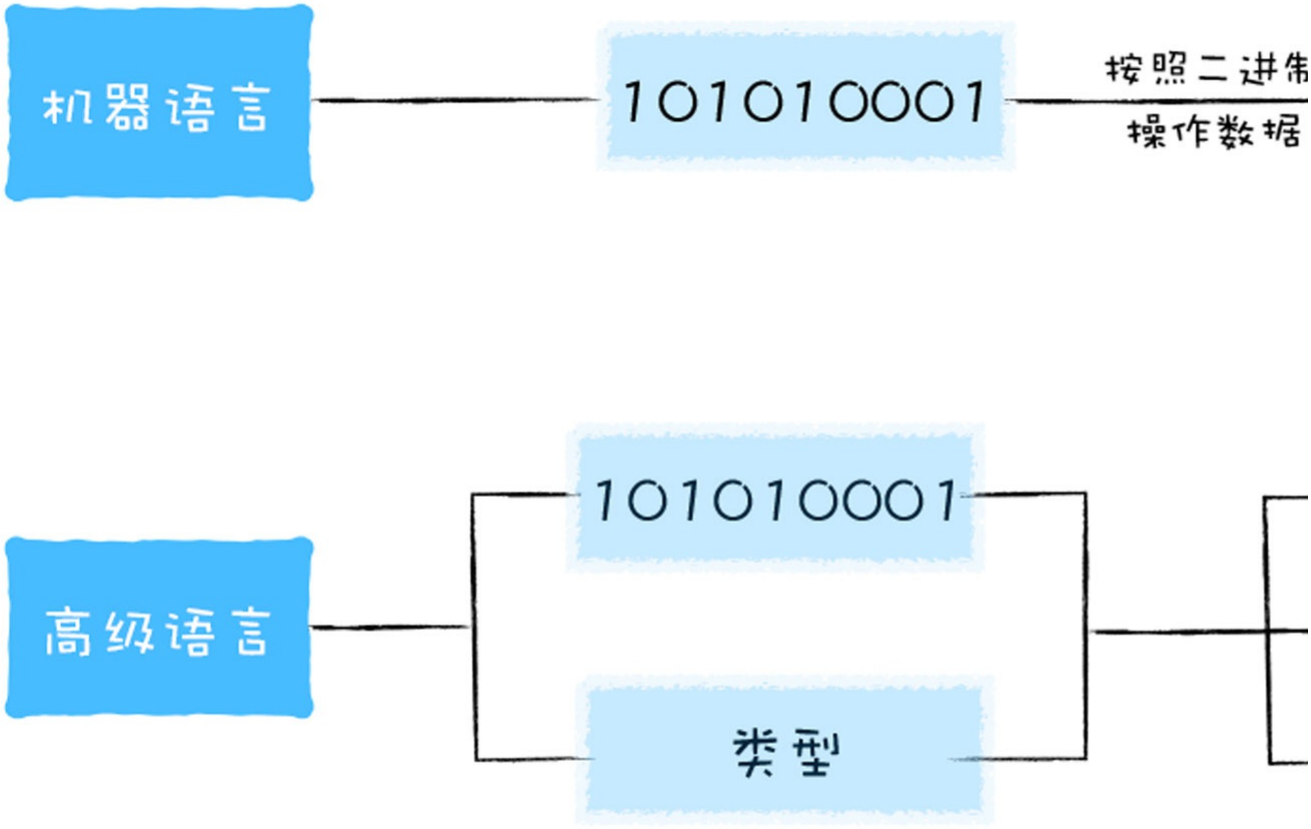
为什么同样一条的表达式，在Python和JavaScript中执行会有不同的结果？为什么在JavaScript中执行，输出的是字符串“12”，不是数字3或者字符串“3”呢？

什么是类型系统(Type System)？

在这个简单的表达式中，涉及到了两种不同类型的数据的相加。要想理清以上两个问题，我们就需要知道类型的概念，以及JavaScript操作类型的策略。

对机器语言来说，所有的数据都是一堆二进制代码，CPU处理这些数据的时候，并没有类型的概念，CPU所做的仅仅是移动数据，比如对其进行移位，相加或相乘。

而在高级语言中，我们都会为操作的数据赋予指定的类型，类型可以确认一个值或者一组值具有特定的意义和目的。所以，类型是高级语言中的概念。



比如在C/C++中，你需要为要处理的每条数据指定类型，这样定义变量：

```
int counter = 100 # 赋值整型变量
float miles = 1000.0 # 浮点型
char* name = "John" # 字符串
```

C/C++编译器负责将这些数据片段转换为供CPU处理的正确命令，通常是二进制的机器代码。

在某些更高级的语言中，还可以根据数据推断出类型，比如在Python或JavaScript中，你就不必为数据指定专门的数据类型，在Python中，你可以这样定义变量：

```
counter = 100 # 赋值整型变量
miles = 1000.0 # 浮点型
name = "John" # 字符串
```

在JavaScript中，你可以这样定义变量：

```
var counter = 100 # 赋值整型变量
let miles = 1000.0 # 浮点型
const name = "John" # 字符串
```

虽然Python和JavaScript定义变量的方式不同，但是它们都不需要直接指定变量的类型，因为虚拟机会根据数据自动推导出类型。

通用的类型有数字类型、字符串、Boolean类型等等，引入了这些类型之后，编译器或者解释器就可以根据类型来限制一些有害的或者没有意义的操作。

比如在Python语言中，如果使用字符串和数字相加就会报错，因为Python觉得这是没有意义的。而在JavaScript中，字符串和数字相加是有意义的，可以使用字符串和数字进行相加的。再比如，你让一个字符串和一个字符串相乘，这个操作是没有意义的，所有语言几乎都会禁止该操作。

每种语言都定义了自己的类型，还定义了如何操作这些类型，另外还定义了这些类型应该如何相互作用，我们就把这称为**类型系统**。

关于类型系统，wiki百科上是这样解释的：

> 在计算机科学中，类型系统（type system）用于定义如何将编程语言中的数值和表达式归类为许多不同的类型，如何操作这些类型，这些类型如何互相作用。

直观地理解，一门语言的类型系统定义了各种类型之间应该如何相互操作，比如，两种不同类型相加应该如何处理，两种相同的类型相加又应该如何处理等。还规定了各种不同类型应该如何相互转换，比如字符串类型如何转换为数字类型。

一个语言的类型系统越强大，那编译器能帮程序员检查的东西就越多，程序员定义“检查规则”的方式就越灵活。

V8是怎么执行加法操作的？

了解了JavaScript中的类型系统，接下来我们就可以来看看V8是怎么处理1+“2”的了。

当有两个值相加的时候，比如:

a+b

V8会严格根据ECMAScript规范来执行操作。ECMAScript是一个语言标准，JavaScript就是ECMAScript的一个实现，比如在ECMAScript就定义了怎么执行加法操作，如下所示：

12.7.3.1 Runtime Semantics: Evaluation

AdditiveExpression : *AdditiveExpression* + *MultiplicativeExpression*

1. Let *lref* be the result of evaluating *AdditiveExpression*.
2. Let *lval* be GetValue(*lref*).
3. ReturnIfAbrupt(*lval*).
4. Let *rref* be the result of evaluating *MultiplicativeExpression*.
5. Let *rval* be GetValue(*rref*).
6. ReturnIfAbrupt(*rval*).
7. Let *lprim* be ToPrimitive(*lval*).
8. ReturnIfAbrupt(*lprim*).
9. Let *rprim* be ToPrimitive(*rval*).
10. ReturnIfAbrupt(*rprim*).
11. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Let *lstr* be ToString(*lprim*).
    b. ReturnIfAbrupt(*lstr*).
    c. Let *rstr* be ToString(*rprim*).
    d. ReturnIfAbrupt(*rstr*).
    e. Return the String that is the result of concatenating *lstr* and *rstr*.
12. Let *lnum* be ToNumber(*lprim*).
13. ReturnIfAbrupt(*lnum*).
14. Let *rnum* be ToNumber(*rprim*).
15. ReturnIfAbrupt(*rnum*).
16. Return the result of applying the addition operation to *lnum* and *rnum*. See the

具体细节你也可以参考规范，我将标准定义的内容翻译如下：

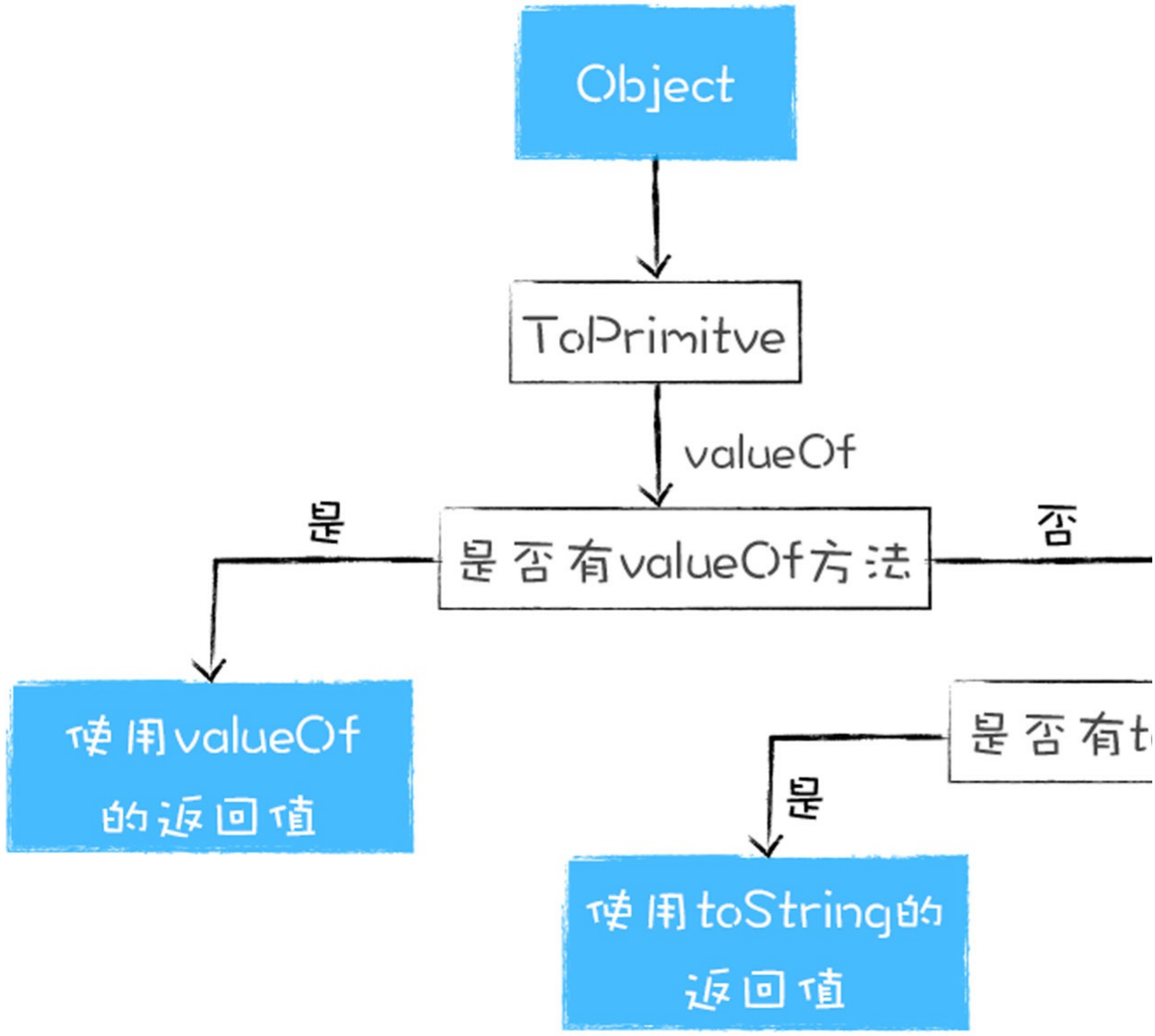
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

1. 把第一个表达式(AdditiveExpression)的值赋值给左引用(lref)。
2. 使用GetValue(lref)获取左引用(lref)的计算结果，并赋值给左值。
3. 使用ReturnIfAbrupt(lval)如果报错就返回错误。
4. 把第二个表达式(MultiplicativeExpression)的值赋值给右引用(rref)。
5. 使用GetValue(rref)获取右引用(rref)的计算结果，并赋值给rval。
6. 使用ReturnIfAbrupt(rval)如果报错就返回错误。
7. 使用ToPrimitive(lval)获取左值(lval)的计算结果，并将其赋值给左原生值(lprim)。
8. 使用ToPrimitive(rval)获取右值(rval)的计算结果，并将其赋值给右原生值(rprim)。
9. 如果Type(lprim)和Type(rprim)中有一个是String，则：
   a. 把ToString(lprim)的结果赋给左字符串(lstr)；
   b. 把ToString(rprim)的结果赋给右字符串(rstr)；
   c. 返回左字符串(lstr)和右字符串(rstr)拼接的字符串。
10. 把ToNumber(lprim)的结果赋给左数字(lnum)。
11. 把ToNumber(rprim)的结果赋给右数字(rnum)。
12. 返回左数字(lnum)和右数字(rnum)相加的数值。

通俗地理解，V8会提供了一个ToPrimitive方法，其作用是将a和b转换为原生数据类型，其转换流程如下：

- 先检测该对象中是否存在valueOf方法，如果有并返回了原始类型，那么就使用该值进行强制类型转换；
- 如果valueOf没有返回原始类型，那么就使用toString方法的返回值；
- 如果vaule Of和toString两个方法都不返回基本类型值，便会触发一个TypeError的错误。

将对象转换为原生类型的流程图如下所示：



当V8执行1+“2”时，因为这是两个原始值相加，原始值相加的时候，如果其中一项是字符串，那么V8会默认将另外一个值也转换为字符串，相当于执行了下面的操作：

```
Number(1).toString() + "2"
```

这里，把数字1偷偷转换为字符串“1”的过程也称为强制类型转换，因为这种转换是隐式的，所以如果我们不熟悉语义，那么就很容易判断错误。

我们还可以再看一个例子来验证上面流程，你可以看下面的代码：

```
var Obj = {
    toString() {
      return '200'
    },
    valueOf() {
      return 100
    }
  }
  Obj+3
```

执行这段代码，你觉得应该返回什么内容呢？

上面我们介绍过了，由于需要先使用ToPrimitive方法将Obj转换为原生类型，而ToPrimitive会优先调用对象中的valueOf方法，由于valueOf返回了100，那么Obj就会被转换为数字100，那么数字100加数字3，那么结果当然是103了。

如果我改造下代码，让valueOf方法和toString方法都返回对象，其改造后的代码如下：

```
var Obj = {
    toString() {
      return new Object()
    },
    valueOf() {
      return new Object()
    }
  }
  Obj+3
```

再执行这段代码，你觉得应该返回什么内容呢？

因为ToPrimitive会先调用valueOf方法，发现返回的是一个对象，并不是原生类型，当ToPrimitive继续调用toString方法时，发现toString返回的也是一个对象，都是对象，就无法执行相加运算了，这时候虚拟机就会抛出一个异常，异常如下所示：

```
VM263:9 Uncaught TypeError: Cannot convert object to primitive value
    at <anonymous>:9:6
```

提示的是类型错误，错误原因是无法将对象类型转换为原生类型。

所以说，在执行加法操作的时候，V8会通过ToPrimitive方法将对象类型转换为原生类型，最后就是两个原生类型相加，如果其中一个值的类型是字符串时，则另一个值也需要强制转换为字符串，然后做字

符串的连接运算。在其他情况时，所有的值都会转换为数字类型值，然后做数字的相加。

总结

今天我们主要学习了JavaScript中的类型系统是怎么工作的。类型系统定义了语言应当如何操作类型，以及这些类型如何互相作用。因为Python和JavaScript的类型系统差异，所以当处理同样的表达式时，返回的结果是不同的。

在Python中，数字和字符串相加会抛出异常，这是因为Python认为字符串和数字相加是无意义的，所以限制其操作。

在JavaScript中，数字和字符串相加会返回一个新的字符串，这是因为JavaScript认为字符串和数字相加是有意义的，V8会将其中的数字转换为字符，然后执行两个字符串的相加操作，最终得到的是一个新的字符串。

在JavaScript中，类型系统是依据ECMAScript标准来实现的，所以V8会严格根据ECMAScript标准来执行。在执行加法过程中，V8会先通过ToPrimitive函数，将对象转换为原生的字符串或者是数字类型，在转换过程中，ToPrimitive会先调用对象的valueOf方法，如果没有valueOf方法，则调用toString方法，如果vaulleOf和toStringg两个方法都不返回基本类型值，便会触发一个TypeError的错误。

思考题

我们一起来分析一段代码：

```
var Obj = {
    toString() {
      return "200"
    },
    valueOf() {
      return 100
    }
  }
  Obj+"3"
```

你觉得执行这段代码会打印出什么内容呢？欢迎你在留言区与我分享讨论。

感谢你的阅读，如果你觉得这一讲的内容对你有所启发，也欢迎把它分享给你的朋友。