目录

[ECMAScript 5.0 1](#_Toc235)

[复习： 3](#_Toc5477)

[一、 严格模式 6](#_Toc24501)

[1.1 开启严格模式 6](#_Toc28287)

[1.2 在严格模式下定义变量 7](#_Toc10785)

[1.3 八进制 8](#_Toc27561)

[1.4 arguments.callee 8](#_Toc1142)

[1.5 delete关键字 10](#_Toc12755)

[1.6 使用保留字定义变量 10](#_Toc21146)

[1.7 eval 11](#_Toc1312)

[1.8 with 12](#_Toc16563)

[1.9 函数的作用域 12](#_Toc8055)

[1.10 对象的属性 13](#_Toc28129)

[1.11 函数的参数 14](#_Toc9459)

[二、 对象的拓展 15](#_Toc12674)

[2.1 特性 15](#_Toc27766)

[2.2 Object.defineProperty 15](#_Toc30692)

[下午复习： 17](#_Toc17961)

[2.3 特性--配置值 18](#_Toc22876)

[2.3 特性--可修改性 18](#_Toc30388)

[2.4 特性--可枚举性 19](#_Toc26374)

[2.5 特性--可配置性 20](#_Toc5485)

[2.6 特性方法 20](#_Toc17104)

[2.7 设置多个属性特性 22](#_Toc11476)

[三、 原型方法 23](#_Toc1137)

[3.1 isPrototypeOf 23](#_Toc14353)

[3.2 getPrototypeOf 23](#_Toc15224)

[3.3 setPrototypeOf 24](#_Toc11608)

[四、 对象的拓展 24](#_Toc7849)

[4.1 preventExtensions 24](#_Toc1108)

[4.2 seal 25](#_Toc15149)

[4.3 freeze 26](#_Toc24842)

[五、 对象创建的新方式 26](#_Toc19047)

[5.1 Object.create 26](#_Toc3808)

# 复习：

JSON拓展：

ES5为了json字符串与js对象互转方便提供了JSON内置对象，分别提供了两个方法：

parse:

该方法用于将json字符串转为js对象

适用方式：

JSON.parse(str, fn)

str: 要处理的字符串

fn：处理的函数

有两个参数：属性名、属性值

作用域遍历到的该对象

stringify:

该方法用于将js对象解析为jsonz字符串

使用方式：

JSON.stirngify(obj, fn)

obj: 处理的对象

fn: 处理的函数

有两个参数：属性名、属性值

作用域遍历到的该对象

数组的拓展：

判断数组：

第一种判断对象的类型

Object.prototype.toString.call(obj)

第二种判断实例化对象

obj instanceof Array

第三种判断构造函数

obj.constructor === Array

第四种数组的静态方法

Array.isArray(obj)

forEach方法：

该方法是数组的迭代器方法，用于替代for循环，但是并没有将for循环移除，只是将for循环封装在了forEach方法的内部

接收一个参数，处理的函数

函数中三个参数：成员值、索引值、原数组

作用域window

返回值对forEach方法的执行结果没有影响

forEach方法的返回值始终是undefined

map:

数组的迭代器方法，使用方式与forEach类似，只不过它的返回值是有意义的

接收一个参数就是函数

函数中有三个参数: 成员值、 索引值、 原数组

返回值就是每次执行结果的成员

map方法的返回值是一个新的数组， 数组中的成员就是每次执行结果得到的新的成员

断言方法：

some:

用于判断数组中是否有成员满足条件

接收一个参数是函数

函数中有三个参数：成员值、 索引值、原数组

返回值就是判断的依据

some方法的返回值：

true: 至少有一个成员满足条件

false: 都不满足条件

every:

用于判断数组中是否都满足条件

接收一个参数是函数

函数中有三个参数：成员值、 索引值、原数组

返回值就是判断的依据

every:

true: 都满足条件

false: 至少有一个成员是不满足条件

filter:

该方法用于过滤数组

接收一个参数是函数

函数中有三个参数：成员值、 索引值、原数组

返回值就是过滤的条件

filter方法的返回值也是一个新的数组，数组中的成员就是满足过滤条件组成的成员

累加方法

reduce

reduceRight:

这两个方法是累加方法，reduce是从前向后开始累加，reduceRight是从后向前开始累加

会对所有的成员逐一处理，并将结果返回

参数就是要处理的函数

函数中有四个参数: 上一次的累积结果、当前成员值、当前索引值、原数组

返回值就是当次累积的结果，将会在下一次遍历的时候作为第一个参数传递

reduce是从第二个成员开始遍历，第一个成员在第一次遍历的时候作为第一个参数传递

reduceRight是从倒数第二个成员开始遍历，倒数第一个成员在第一次遍历的时候作为第一个成员开始遍历

addNum(num1, num2) :

1 确定最大值和最小值

2 创建数组，确定长度

3 为了遍历数组，填充数组

4 为了得到一个从最小值到最大值之间的数组调用map方法， 在map方法中用min加上index

5 调用数组的累加方法， 并将结果返回

= bind：

为函数绑定作用域（执行函数，并改变函数的作用域）

第一个参数要改变的作用域

从第二个参数开始都是传递的参数

bind与call和apply之间的区别：

call和apply： 调用即执行

bind: 调用不执行，返回的是一个新的方法

日期拓展：

var date = new Date()

date.toJSON() (转为标准化格式)

# 严格模式

ES5拓展了严格模式，可以使我们的代码更加的安全可靠

在之前我们都是处于正常模式下

在正常模式下，定义一个变量可以不使用var声明， 严格模式下是不允许的

在正常模式下，定义一个数值的时候可以使用八进制，在严格模式下是不允许的

在正常模式下，可以使用arguments.callee，在严格模式下也是不允许的

## 开启严格模式

在js的第一行加入“use strict”

如果开启严格模式下之后， 就必须按照严格模式的规定进行书写

在js第一行加入“use strict, 此时代码将处于“全局严格模式”

如果在某个函数的第一行加入“use strict”，当函数执行的时候，此时函数处于“局部严格模式”

举例：

全局严格模式：

|  |
| --- |
| 1. // 开启严格模式 2. "use strict" 3. // 定义变量 使用var 4. var a = 10; 5. // 定义变量 不使用va 6. b = 20; |

结果：

|  |
| --- |
|  |

局部严格模式：

|  |
| --- |
| 1. // 定义函数 2. function fn() { 3. // 开启严格模式 4. "use strict" 5. // 定义变量 使用var 6. var c = 10; 7. // 定义变量 不使用var 8. d = 20; 9. } 10. fn(); |

结果：

|  |
| --- |
|  |

## 在严格模式下定义变量

在严格模式下定义变量必须使用var 进行声明

允许：

|  |
| --- |
| 1. var a = 10 |

不允许：

|  |
| --- |
| 1. b = 20; |

举例：

正常模式下：

|  |
| --- |
| 1. // 定义变量使用var 2. var a = 10; 3. // 定义变量 不使用var 4. b = 20; |

结果：

|  |
| --- |
|  |

严格模式下：

|  |
| --- |
| 1. // 开启严格模式 2. "use strict"; 3. // 定义变量使用var 4. var a = 10; 5. // 定义变量 不使用var 6. b = 20; |

结果：

|  |
| --- |
|  |

## 八进制

在js中的进制有： 8进制、 10进制、16进制

默认是10进制

以0开头是八进制

以0x、ox是16进制

允许：

|  |
| --- |
| 1. var num = 10; 2. var num1 = 0xff; |

不允许：

|  |
| --- |
| 1. var num2 = 011; |

正常模式下：

|  |
| --- |
| 1. // 定义数值 2. var num = 10; 3. var num1 = 0xff; 4. var num2 = 011; |

结果：

|  |
| --- |
|  |

严格模式下：

|  |
| --- |
|  |

## arguments.callee

代码写的越“具体”，越能提升性能，而arugments.callee在编译阶段是无法确定下来， 只有当执行到的时候才能够确定， 所以就无从提升性能

正常模式下：

|  |
| --- |
| 1. // 安全类 2. function People(name, age, sex) { 3. // 判断this指向 4. if (this instanceof arguments.callee) { 5. // 说明是使用new调用 6. this.name = name; 7. this.age = age; 8. this.sex = sex; 9. } else { 10. // 没有使用new 调用 11. return new arguments.callee(name, age, sex); 12. } 13. } 14. // 实例化对象 15. var p = new People("小明", 12, "男"); 16. var p1 = People("小红", 12, "女"); |

结果：

|  |
| --- |
|  |

在严格模式下:

|  |
| --- |
| 1. // 开启严格模式 2. "use strict" 3. // 安全类 4. function People(name, age, sex) { 5. // 判断this指向 6. if (this instanceof arguments.callee) { 7. // 说明是使用new调用 8. this.name = name; 9. this.age = age; 10. this.sex = sex; 11. } else { 12. // 没有使用new 调用 13. return new arguments.callee(name, age, sex); 14. } 15. } 16. // 实例化对象 17. var p = new People("小明", 12, "男"); 18. var p1 = People("小红", 12, "女"); |

结果：

|  |
| --- |
|  |

## delete关键字

在正常模式下，使用delete删除一个变量， 可以执行通过，但是不能删除成功

在严格模式下，会报错

正常模式下：

|  |
| --- |
| 1. // 定义变量 2. var a = 10; 3. window.b = 20; 4. // 使用delete关键字 删除变量 5. delete a; 6. delete b; |

结果：

|  |
| --- |
|  |

在严格模式下：

|  |
| --- |
| 1. // 开启严格模式 2. "use strict" 3. // 定义变量 4. var a = 10; 5. window.b = 20; 6. // 使用delete关键字 删除变量 7. delete a; 8. delete b; |

结果:

|  |
| --- |
|  |

## 使用保留字定义变量

在正常模式下可以使用保留字定义变量，

在严格模式中是不允许的

正常模式下：

|  |
| --- |
| 1. // 使用保留字定义变量 2. var public = "hehe"; 3. console.log(public); |

结果：

|  |
| --- |
|  |

在严格模式下：

|  |
| --- |
|  |

## eval

eval是一个函数，作用就是可以将字符串当做代码执行

但是会污染全局作用域，在严格模式是不允许使用， 因此可以避免全局作用域的污染

正常模式下：

|  |
| --- |
| 1. // 定义字符串 2. var str = "var a = 10"; 3. // eval函数接收一个参数，是字符串， 结果可以将字符串当做代码执行 4. eval(str); |

结果：

|  |
| --- |
|  |

在严格模式下：

|  |
| --- |
| 1. // 开启严格模式 2. "use strict" 3. // 定义字符串 4. var str = "var a = 10"; 5. // eval函数接收一个参数，是字符串， 结果可以将字符串当做代码执行 6. eval(str); |

结果：

|  |
| --- |
|  |

## with

with是一个方法，也会污染全局作用域

在严格模式下不允许使用

with(target) {

console.log(a) // 如果target中存在a 优先输出a, 如果没有会沿着词法作用域往上找

}

举例：

正常模式下：

|  |
| --- |
| 1. // 定义对象 2. var obj = { 3. a: 10, 4. b: 20 5. } 6. // 定义数值 7. var a = 100; 8. var b = 200; 9. // 使用With 10. with (obj) { 11. // 因为obj中存在a和b属性，那么优先输出obj中的a和b 12. // console.log(a); 13. // console.log(b); 14. // 由于obj中存在a, 所以优先改变Obj中的a 15. a = 123; 16. // obj中没有存在该属性， 将定义在全局中 17. c = 111; 18. } |

结果：

|  |
| --- |
|  |

严格模式下：

|  |
| --- |
|  |

## 函数的作用域

全局函数作用域中的this指向window, 会污染全局作用域，

在严格模式下，this是undefined, 因此避免了对全局作用域的污染  
正常模式下：

|  |
| --- |
| 1. function fn() { 2. console.log(this); 3. // 添加属性 4. this.color = "red"; 5. } 6. // 执行 |

结果：

|  |
| --- |
|  |

严格模式下:

|  |
| --- |
| 1. // 开启严格模式 2. "use strict" 3. function fn() { 4. console.log(this); 5. // 添加属性 6. this.color = "red"; 7. } 8. // 执行 |

结果：

|  |
| --- |
|  |

## 对象的属性

在正常模式下定义对象中的属性可以是同名的，

在严格模式下，是不允许的，但是浏览器检测不出来

正常模式下:

|  |
| --- |
| 1. // 定义对象 2. var obj = { 3. a: 2, 4. a: 1 5. } 6. console.log(obj.a); |

结果：

|  |
| --- |
|  |

严格模式下：

|  |
| --- |
| 1. // 开启严格模式 2. "use strict" 3. // 定义对象 4. var obj = { 5. a: 2, 6. a: 1 7. } 8. console.log(obj.a); |

结果：

|  |
| --- |
|  |

注意: 不是没有开启严格模式，而是浏览器没有实现

## 函数的参数

在正常模式下定义函数的参数可以是同名的， 但是后面的会覆盖掉前面

在严格模式下是不允许的

正常模式下：

|  |
| --- |
| 1. // 定义函数 2. function fun(color, color) { 3. console.log(color); 4. } 5. // 执行 6. fun("red", "blue"); |

结果：

|  |
| --- |
|  |

严格模式下：

|  |
| --- |
| 1. // 开启严格模式 2. "use strict" 3. // 定义函数 4. function fun(color, color) { 5. console.log(color); 6. } 7. // 执行 8. fun("red", "blue"); |

结果：

|  |
| --- |
|  |

# 对象的拓展

## 特性

对象是什么，我们用属性来描述

属性是什么， 我们用特性来描述

特性是用来描述属性的

## Object.defineProperty

该方法用于设置单一属性特性

使用方式：

Object.defineProperty(obj, property, descript)

obj：要设置特性的对象

property: 要设置的特性属性

descript：描述特性对象

举例：

在没有设置特性之前:

|  |
| --- |
| 1. // 在ES5中 2. var obj = { 3. num: 100, 4. color: "red" 5. } 6. // 枚举 7. for (var i in obj) { 8. console.log(i); 9. } |
|  |

结果：

|  |
| --- |
|  |

在设置了特性之后：

|  |
| --- |
| 1. // 设置对象的特性 2. Object.defineProperty(obj, "color", { 3. // 修改可枚举性 4. enumerable: false 5. }) 6. // 再次枚举 7. for (var i in obj) { 8. console.log(i); 9. } |

结果：

|  |
| --- |
|  |

## 下午复习：

严格模式：

ES5提供了严格模式，可以使我们的代码更加的安全可靠

开启方式：

在js第一行加入“use stict”, 此时将处于全局严格模式

如果在函数中的第一行加入“use strict”，当函数执行的时候，此时函数将处于“局部严格模式”

严格模式与正常模式之间的区别：

在严格模式下定义变量必须使用var进行声明

在严格模式下定义数值的时候不能使用八进制

在严格模式下不能使用arguments.callee

在严格模式下不能使用delete删除变量

在严格模式下不能使用保留字定义变量

在严格模式下不能使用eval函数

在严格模式下不能使用with方法

在严格模式下全局函数作用域变成undefined

在严格模式下定义对象中的属性不能是同名的

在严格模式下定义函数中的参数也不能是同名的

特性：

设置单一属性特性：

Object.defineProperty(obj, property, descript)

obj: 要设置的对象

property: 要设置的属性

descript: 描述特性对象

## 2.3 特性--配置值

使用方式：

Object.defineProperty(obj, property, {

// 配置值

value: “”

})

举例：

|  |
| --- |
| 1. // 在ES5中 2. var obj = { 4. } 5. // 设置特性 6. Object.defineProperty(obj, "color", { 7. // 配置值 8. value: "red" 9. }) |

结果：

|  |
| --- |
|  |

## 特性--可修改性

使用方式：

Object.defineProperty(obj, property, {

// 配置可修改性

writable: true | false

})

举例：

|  |
| --- |
| 1. // 在ES5中 2. var obj = { 3. num: 100 4. } 5. // 设置特性 6. Object.defineProperty(obj, "num", { 7. // 不可修改 8. writable: false 9. }) 10. // 尝试修改属性值 11. obj.num = 200; |

结果：

|  |
| --- |
|  |

## 特性--可枚举性

使用方式：

Object.defineProperty(obj , property，{

// 不可枚举

enumerable: true | false

})

举例：

|  |
| --- |
| 1. // 在ES5中 2. var obj = { 3. num: 100, 4. color: "red" 5. } 6. // 默认情况下对象中的属性是可以被枚举的 7. for (var i in obj) { 8. console.log(i); 9. } |

结果：

|  |
| --- |
|  |

设置特性之后：

|  |
| --- |
| 1. // 设置特性 2. Object.defineProperty(obj, "color", { 3. // 不可枚举 4. enumerable: false 5. }) 6. // 再次枚举 7. for (var i in obj) { 8. console.log(i); 9. } |

结果：

|  |
| --- |
|  |

## 特性--可配置性

使用方式：

Object.defineProperty(obj, property, {

// 可配置性

configurable: true | false

})

举例：

|  |
| --- |
| 1. // 在ES5中 2. var obj = { 3. num: 100, 4. color: "red" 5. } 6. // 设置特性 7. Object.defineProperty(obj, "num", { 8. // 配置值 9. value: 500, 10. // 不可修改 11. writable: false, 12. // 不可配置 13. configurable: false 14. }) 15. // 设置特性 16. Object.defineProperty(obj, "num", { 17. // 配置值 18. value: 500, 19. // 不可修改 20. writable: true 21. }) 22. obj.num = 1000; |

结果：

|  |
| --- |
|  |

## 特性方法

get方法用来获取值的

没有参数

返回值就是要获取的属性值

作用域是当前对象

千万不要在方法中获取该属性值， 否则将会递归死循环

通常，我们是获取该属性值的一个备用值

set方法用来设置值的

参数就是要设置的值

没有返回值

作用域是当前对象

千万不要在方法中设置该属性值， 否则将会递归死循环

通常，我们设置该属性的一个备用属性

举例：

|  |
| --- |
| 1. // 定义对象 2. var obj = { 3. color: "red" 4. } 5. // 一旦设置了特性方法， 当obj.color(读取属性值)的时候，触发get方法，当obj.color = xx(设置属性值)的时候， 会触发set方法 6. // 设置特性 7. Object.defineProperty(obj, "color", { 8. // 取值器 9. get: function() { 10. console.log(1111, this, arguments); 11. // 不要在方法中获取属性值 12. // return this.color; 13. // 通常， 我们是获取该属性值的一个备用值 14. return this.\_color; 15. }, 16. // 赋值器 17. set: function(value) { 18. console.log(2222, this, arguments); 19. // 不要在方法中设置该属性 20. // this.color = value; 21. // 通常，我们设置该属性的备用属性 22. this.\_color = value; 23. } 24. }) 25. obj.color = "blue"; 26. console.log(obj.color); |

结果：

|  |
| --- |
|  |

## 设置多个属性特性

使用方式：

Object.defineProperties(obj, options)

obj: 要设置的对象

options: 配置对象

key: 设置的特性属性

value: 描述特性对象

我们可以设置特性属性以及方法

举例：

|  |
| --- |
| 1. // 定义对象 2. var obj = { 3. num: 100, 4. color: "red" 5. } 6. // 设置特性 7. Object.defineProperties(obj, { 8. // 配置num 9. num: { 10. // 配置值 11. value: 500, 12. // 不可修改 13. writable: false, 14. // 不可枚举 15. enumerable: false, 16. }, 17. // 特性方法 18. color: { 19. get: function() { 20. return this.\_color; 21. }, 22. set: function(value) { 23. this.\_color = value; 24. }, 25. // 配置不可枚举 26. enumerable: false, 27. // 一旦设置了特性方法， 将无法再次配置writable以及value 28. // 配置可修改性 29. // writable: false 30. // 配置值 31. // value: "orange" 32. } 33. }) 34. obj.num = 1000; 35. obj.color = "blue"; |

# 原型方法

ES5中为原型拓展了几个方法

## isPrototypeOf

原型对象的方法

判断原型对象是否是参数对象的原型

参数就是实例化对象

在查找的过程中， 会查找整个原型链

举例：

|  |
| --- |
| 1. <div>1</div> 2. <div>2</div> 3. <div>3</div> 4. <script type="text/javascript"> 5. // 获取类数组对象 6. var obj = document.getElementsByTagName('div'); 7. // 定义数组 8. var arr = []; 9. // isPrototypeOf 10. // 判断Array的原型是哪一个对象的原型 11. console.log(Array.prototype.isPrototypeOf(obj)); 12. console.log(Array.prototype.isPrototypeOf(arr)); 13. // 查找过程中，会查找整个原型链 14. console.log(Object.prototype.isPrototypeOf(obj)); 15. console.log(Object.prototype.isPrototypeOf(arr)); 16. </script> |

结果:

|  |
| --- |
|  |

## getPrototypeOf

在ES5之前，获取某个对象的原型对象可以通过\_\_proto\_\_获取， 但是在ES5中， 不推荐以\_\_开头，所以提供了getPrototypeOf方法用来获取原型对象

举例：

|  |
| --- |
| 1. // 获取原型对象 2. console.log(arr.\_\_proto\_\_); 3. console.log(Object.getPrototypeOf(arr)); 4. console.log(arr.\_\_proto\_\_ === Object.getPrototypeOf(arr)); |

结果：

|  |
| --- |
|  |

## setPrototypeOf

用于设置某个对象的原型

使用方式：

Object.setPrototypeOf(obj, prototype)

obj: 要设置的对象

prototype: 新的原型对象 （可以是对象 还可以是null）

举例：

|  |
| --- |
| 1. // 设置arr的原型 2. Object.setPrototypeOf(arr, {"name": "王老五"}); 3. Object.setPrototypeOf(arr, null); 4. // 会修改整个原型链 5. console.log(arr.\_\_proto\_\_); |

结果：

|  |
| --- |
|  |

# 对象的拓展

## preventExtensions

作用: 用于取消对象的可拓展性

举例：

|  |
| --- |
| 1. // 定义对象 2. var obj = { 3. a: 1 4. } 5. // 默认情况下对象是可以被拓展的 6. obj.b = 2; 7. // 取消对象的可拓展性 8. Object.preventExtensions(obj); 9. // 再次尝试拓展属性 10. obj.c = 3; 11. // 尝试修改属性值 12. obj.a = 11; 13. obj.b = 22; 14. // 尝试删除属性 15. delete obj.a; |

结果：

|  |
| --- |
|  |

**总结:**

**当对象取消了可拓展性之后， 对象不能被拓展，但是可以修改属性值， 还可以删除属性**

**查看对象是否取消了可拓展性：**

**Object.isExtensible(obj) 返回值bool 如果是true 表示对象可以拓展，反之不可拓展**

## seal

该方法用于封闭对象

举例：

|  |
| --- |
| 1. // 定义对象 2. var obj = { 3. a: 1 4. } 5. // 封闭对象 6. Object.seal(obj); 7. // 尝试拓展属性 8. obj.b = 2; 9. // 尝试删除属性 10. delete obj.a; 11. // 尝试修改属性值 12. obj.a = 11; |

结果：

|  |
| --- |
|  |

**总结:**

**当一个对象被封闭之后，此时不可再次拓展， 不可删除属性， 可以修改属性值**

**查看对象是否被封闭：**

**Object.isSealed(obj) 返回值bool 如果是true 表示被封闭了， 反之没有被封闭**

## freeze

作用： 冻结对象

举例：

|  |
| --- |
| 1. // 定义对象 2. var obj = { 3. a: 1 4. } 5. // 冻结对象 6. Object.freeze(obj); 7. // 尝试拓展属性 8. obj.b = 2; 9. // 尝试修改属性值 10. obj.a = 11; 11. // 尝试删除属性 12. delete obj.a; |

结果：

|  |
| --- |
|  |

**总结：**

**当一个对象冻结之后，此时对象不可拓展属性、不可删除属性、 不可修改属性值**

**查看对象是否被冻结：**

**Object.isFrozen(obj) 返回值bool 如果是true表示被冻结了， 反之没有被冻结**

# 对象创建的新方式

## Object.create

使用方式：

Object.create(prototype, options)

方法接收两个参数

第一个参数是一个对象 （可以是null）

该对象是通过Object.create方法创建出来的原型对象

第二个参数也是一个对象 （可以省略）

该对象是通过Object.create方法创建出来的特性对象

举例：

|  |
| --- |
| 1. var prototype = { 2. // 添加方法 3. sayHello: function() { 4. console.log("sayHello"); 5. }, 6. sayNihao: function() { 7. console.log("sayNihao"); 8. }, 9. sayHehe: function() { 10. console.log("sayHehe"); 11. } 12. } 13. var options = { 14. num: { 15. // 配置值 16. value: 500, 17. // 不可修改 18. writable: false, 19. // 不可枚举 20. enumerable: false 21. }, 22. color: { 23. // 配置值 24. value: "red", 25. // 不可枚举 26. enumerable: false 27. } 28. } 29. var obj = Object.create(prototype, options); |

结果：

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |