webpack如何配置？

概念：本质上，webpack是一个现代JavaScript应用程序的静态模块打包器（module bundler）。当webpack处理应用程序时，它会递归地构建一个依赖关系图（dependency graph），其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个bundle。

**4个核心概念：**

* 入口（entry）
* 输出（output）
* loader
* 插件（plugins）

入口（entry）

入口起点（entry point）指示webpack应该使用哪个模块，来作为构建其内部依赖图的开始。进入入口起点后，webpack会找出有哪些模块和库是入口起点（直接和间接）依赖的。

每个依赖项随即被处理，最后输出到称之为bundles的文件中。

可以通过在webpack配置中配置entry属性，来指定一个入口起点（或多个入口起点）。默认值为./src。

出口（output）

output属性告诉webpack在哪里输出它所创建的bundles，以及如何命名这些文件，默认值为./dist。基本上，整个应用程序结构，都会被编译到你指定的输出路径的文件夹中。你可以通过在配置中指定一个output字段，来配置这些处理过程。

loader

loader让webpack能够去处理那些非JavaScript文件（webpack自身只能理解JavaScript）。loader可以将所有类型的文件转换为webpack能够处理的有效模块，然后你就可以利用webpack的打包能力，对它们进行处理。

本质上，webpack loader将所有类型的文件，转换为应用程序的依赖图（和最终的bundle）可以直接引用的模块。

注意，loader能够import导入任何类型的模块（例如.css文件），这是webpack特有的功能，其它打包程序或任务执行器的可能并不支持。我们认为这种语言扩展是很有必要的，因为这可以使开发人员创建出更准确的依赖关系图。

在更高层面，在webpack的配置中loader有两个目标：

1. test属性，用于标识出应该被对应的loader进行转换的某个或某些文件
2. use属性，表示进行转换时，应该使用哪个loader

webpack.config.js

const path=require(‘path’);

const config={

output:{

filename: ’my-first-webpack.bundle.js’

},

module:{

rules: [

{test:/\.txt$/, use:’raw-loader’}

]

}

};

module.exports=config;

以上配置中，对一个单独的module对象定义了rules属性，里面包含两个必须属性：test和use。这告诉webpack编译器（compiler）如下信息：

“嘿，webpack编译器，当你碰到[在require() / import语句中被解析为’.txt’的路径]时，在你对它打包之前，先使用raw-loader转换一下。”

重要的是要记得，在webpack配置中定义loader时，要定义在modules.rules中，而不是rules。然而，在定义错误时webpack会给出严重的警告。为了使你受益于此，如果没有按照正确方式去做，webpack会“给出严重的警告”

插件（plugins）

loader被用于转换某些类型的模块，而插件则可以用于执行范围更广的任务。插件的范围包括，从打包优化和压缩，一直到从新定义环境中的变量。插件接口功能极其强大，可以用来处理各种各样的任务。

想要使用一个插件，你只需要require()它，然后把它添加到plugins数组中。多数插件可以通过选项（option）自定义。你也可以在一个配置文件中因为不同目的而多次使用同一个插件，这时需要通过使用new操作符来创建它的一个实例。

**有用过promise吗？请写出下列代码的执行结果，并写出你的理解思路：**

* 首先

虚拟DOM就是放在JS和HTML中间的一个层。它可以通过对比新旧DOM，来获取差异，然后只把差异部分真正渲染到页面上，从而减少实际DOM操作，提升网页性能。

全面理解Javascript闭包和闭包的几种写法及用途：

1. 什么是闭包？闭包的几种写法和用法
2. 什么是闭包

官方对闭包的解释是：一个拥有许多变量和绑定了这些变量的环境的表达式（通常是一个函数），因而这些变量也是该表达式的一部分。

闭包的特点：①作为一个函数变量的一个引用，当函数返回时，其处于激活状态②一个闭包就是当一个函数返回时，一个没有释放资源的栈区。

简单地说，Javascript允许使用内部函数---即函数定义和函数表达式位于另一个函数的函数体内。而且，这些内部函数可以访问它们所在的外部函数中声明的所有局部变量、参数和声明的其它内部函数。当其中一个这样的内部函数在包含它们的外部函数之外被调用时，就会形成闭包。

function A(){

function B(){

console.log(“Hello!”);

}

return B();

}

var C=A();

这就是最简单的闭包。

上述代码翻译成自然语言如下：

* + - 定义普通函数A
    - 在A中定义普通函数B
    - 在A中返回B
    - 执行A，并把A的返回结果赋值给变量C
    - 执行C

把这5步操作总结成一句话就是：函数A的内部函数B被函数A外的一个变量C引用。

把这句话再加工一下就变成了闭包的定义：当一个内部函数被其外部函数之外的遍历引用时，就形成了一个闭包。

Javascript中的GC机制：

在Javascript中，如果一个对象不再被引用，那么这个对象就会被GC回收，否则这个对象一直会保存在内存中。

垃圾回收机制­---GC

Javascript具有自动垃圾回收机制（GC：Garbage Collection），也就是说，执行环境会负责管理代码执行过程中使用的内存。原理：**垃圾收集器会定期（周期性）**找出那些不再继续使用的变量，然后释放其内存。找出不再使用的变量，然后释放掉其占用的内存，但是这个过程不是实时的，因为其开销比较大，所以垃圾回收器会按照固定的时间间隔周期性地执行。

1. 闭包的几种写法和用法

首先要明白，在JS中一切都是对象，函数是对象的一种。下面先来看一下闭包的5种写法，简单理解一下什么是闭包。后面具体解释。

1. 闭包的用途

闭包最大的2个用处：一个是可以读取函数内部的变量，另一个是让这些变量的值始终保持在内存中。

call、bind、apply的用法和区别

call、apply、bind的作用是改变函数运行时this的指向。

所以先说清楚this。

以下是函数的调用方法：

1. 方法调用模式

当一个函数被保存为对象的一个方法时，如果调用表达式包含一个提取属性的动作，那么它就是被当做一个方法来调用，此时的this被绑定到这个对象。

var a=1;

var obj1={

a:2,

fn:function(){

console.log(this.a);

}

}

obj1.fn();//2

此时的this是指obj1这个对象，obj1.fn()实际上是obj1.fn.call(obj1)。事实上，谁调用这个函数，this就是谁。补充一下，DOM对象绑定事件也属于方法调用模式，因此它绑定的this就是事件源DOM对象。如：

document.addEventListener(‘click’,function(e){

console.log(this);

setTimeout(function(){

console.log(this);

},200);

},false);

点击页面，依次输出：document和window对象

解析：点击页面监听click事件属于方法调用，this指向事件源DOM对象，即obj.fn.apply(obj)，setTimeout内的函数属于回调函数，可以这么理解：f1.call(null,f2)，所以this指向window。

1. 函数调用模式

就是普通函数的调用，此时的this被绑定到window

**1—最普通的函数调用**

function fn1(){

console.log(this);//window

}

fn1();

**2—函数嵌套**

function fn1(){

function fn2(){

console.log(this);//window

}

fn2()

}

fn1();

3—把函数赋值以后再调用

1. 构造器调用模式

Call

call方法第一个参数是要绑定给this的值，后面传入的是一个参数列表。当第一个参数为null、undefined的时候，默认指向window。

1. bind是返回对应函数，便于稍后调用，apply、call是立即调用
2. call的第二部分参数要一个一个传，apply要把这些参数放到数组中

css有哪些实现布局的方式

* table布局

display:table; =>相当于“table”标签

display:table-row; =>相当于”tr”标签

display:table-cell; =>相当于”td”标签

* flex布局

如果需要两边定宽，中间自适应，可以这样写:

<div class=”container”>

<div class=”left”></div>

<div class=”middle”></div>

<div class=”right”></div>

</div>

.left{width:20px;}

.middle{flex:1};

.right{width:20px;}

* float布局

解决父元素高度塌陷的方法：

父元素设置：overflow:auto; 或者 overflow:hidden;

给父元素加一个after伪类：

.container::after{

content:””;

clear:both;

display:block;

visibility:hidden;

height:0;

}

* 响应式布局

1. meta标签

最简单的处理方式是加上一个meta标签

<meta name=”viewport” content=”width=device-width,initial-scale=1”>

其中，width=device-width这一句的意思是让页面的宽度等于屏幕的宽度

1. 使用rem

rem指的是html元素的font-size，html元素的font-size默认是16px，所以1rem=16px；然后根据rem来计算各个元素的宽高

1. media query

css2允许用户根据特定的media类型定制样式，基本语法如下：

@media screen and (max-width:360px){

html{font-size:12px;}

}

意思就是：当屏幕的宽度小于等于360px的时候html的字体大小为12px。

width:calc(100% - 10px);

height:calc(100% - 10px);

JS文件上传和下载？

react组件间的通信：

1. 父组件向子组件通信
2. 子组件向父组件通信
3. 兄弟组件之间通信

**父组件向子组件通信**

在react中数据是单向传递的，父组件可以向子组件通过传props的方式，子组件拿到props之后做相应的处理，这就是父组件向子组件进行通信方式。

**子组件向父组件通信**

子组件向父组件传递数据（通信）也是要通过props传递一个函数，子组件调用这个函数，并将子组件需要传递的数据作为参数，传递给父组件。

兄弟组件之间通信

两个兄弟组件之间的数据传递，我们可以通过他们的共同父组件来实现。Child1将要传递的信息传递给Parent然后Parent再将从Child1拿到的信息传递给Child2，当然我们同样是利用props。

面试中可能还会有更深层次更开发性的问题：

* 如果能够改进REACT的一样功能，那会是哪一个功能？（react的缺点）
* immutable.js原理是什么?Immutable详解及React中实践
* react性能优化有哪些？
* react diff算法
* react虚拟dom原理
* react是什么
* react和vue的区别

对于react技术栈react-router、redux当然也有很多：

* redux react-redux分别负责哪些功能
* provider connect的用法
* store数据流向
* redux的三个原则
* append在元素内部的尾部加上元素
* prepend在元素内部的前部加上元素
* after在元素外部的尾部加上元素
* before在元素外部的头部加上元素
* html用当前内容替换掉原内容

**React的工作原理**：React会创建一个虚拟DOM（Virtual DOM）。当一个组件中的状态改变时，React首先会通过”diffing”算法来标记虚拟DOM中的改变，第二步是调节（reconciliation），会用diff的结果来更新DOM。

**React的优点**：

1. **React速度很快**

与其它框架相比，React采取了一种特立独行的操作DOM的方式，它不直接对DOM进行操作。它引入了一个叫做虚拟DOM的概念，安插在Javascript逻辑和实际的DOM之间。这一概念提高了Web性能。在UI渲染过程中，React通过在虚拟DOM中的微操作来实现对实际DOM的局部更新。

1. **跨浏览器兼容**

虚拟DOM帮助我们解决了跨浏览器问题，它为我们提供了标准化的API，甚至是IE8中都没问题。

1. **模块化**

为你程序编写独立的模块化UI组件，这样当某个或者某些组件出现问题时，可以方便地进行隔离。每个组件都可以进行独立的开发和测试，并且它们可以引入其它组件。这等同于提高了代码的可维护性。

1. **单向数据流让事情一目了然**

Flux是一个用于在Javascript应用中创建单向数据层的架构，它随着React视图库的开发而被Facebook概念化。它只是一个概念，而非特定工具的实现。它可以被其它框架吸纳。例如，Alex Rattray有个很好的Flux实例，在React中使用了Backbone的集合和模型。

1. **纯粹的JavaScript**

现代Web应用程序与传统的Web应用有着不同的工作方式。

例如，视图层的更新需要通过用户交互而不需要请求服务器。因此视图和控制器非常依赖彼此。

许多框架使用Handlebars或Mustache等模板引擎来处理视图层。但React相信视图和控制器应该相互依存在一起，而不是使用第三方模板引擎，而且最重要的是，它是纯粹的JavaScript程序。

1. **同构的JavaScript**

单页面JS应用程序的最大缺陷在于对搜索引擎的索引有很大限制。React对此有了解决方案。

React可以在服务器上预渲染应用再发送到客户端。它可以从预渲染的静态内容中恢复一样的记录到动态应用程序中。

因为搜索引擎的爬虫程序依赖的是服务端响应而不是JavaScript的执行，预渲染你的应用有助于搜索引擎优化。

1. **React与其它框架/库兼容性好**

比如使用RequireJS来加载和打包，而Browserify和Webpack适用于构建大型应用。它们使得那些艰难的任务不再让人望而生畏。

缺点：React本身只是一个V而已，所以如果是大型项目想要一套完整的框架的话，也许还需要引入Flux和routing相关的东西。

this指向

JS中的DOM与BOM

Javascript组成：

1. ECMAScript基本语法
2. BOM（浏览器对象模型）
3. DOM（文档对象模型）

一）BOM（browser Object Model）

浏览器对象模型：使用对象描述了浏览器的各个部分的内容

1. window：当前的窗口

window常用的方法：

open() 打开一个新的资源

moveTo() 将窗口左上角的屏幕位置移动到指定的x和y位置

moveBy() 相对于当前的窗口移动指定的x和y偏移值（左上角）

setInterval() 每隔指定的毫秒数执行代码（无限次）

setTimeout() 经过指定毫秒数执行一次代码（一次）

open()方法接收4个参数，分别是：打开资源名、打开方式、打开的窗口大小位置、资源找不到是否用其它代替

window.open(“image.html”,”\_blank”,”height=500pt,width=350pt,top=50pt,left=450pt”,false);

1. 地址栏对象

常用的方法有：

href：设置或获取整个URL为字符串

reload()：重新加载页面地址

location.href=”http://www.baidu.com”; 设置地址栏地址

location.reload(); 重新加载地址栏地址

1. 屏幕对象

screen（屏幕对象）：获取电脑的屏幕的一些数据

常用方法：

availHeight 获取系统屏幕的工作区域高度，排除Microsoft® Windows® 任务栏(可用的屏幕高度)

availWidth 获取系统屏幕的工作区域宽度，排除Windows任务栏（可用的屏幕宽度）

height width 表示的是显示器的宽和高，也就是电脑的分辨率

二）DOM编程（Document Object Model）

文档对象模型：当一个html页面加载到浏览器的时候，那么浏览器会为每个标签都创建一个对应的对象描述该标签的所有信息，那么我们看到的网页信息实际上就是看到了这些标签对象的信息，如果我们需要操作页面的数据，那么我们就可以通过这些标签对象进行操作。

document.getElementById(“id1”);

document.getElementsByTagName(“div”);

document.getElementsByName(“Button”);

var trNode=document.createElement(“tr”); //创建tr标签节点

JS面向对象、继承、原型链

prototype（原型）是什么？它是怎么使用的？

每个函数都有一个prototype属性，它是一个引用变量，默认指向一个空Object对象，有备用的意思，当调用一个对象的函数或者属性的时候，如果在当前对象里面找不到，就到原型里面去找。

Redux

1. 为什么要用redux

在React中，数据在组件中是单向流动的，数据从一个方向父组件流向子组件（通过props），所以两个非父子组件之间通信就相对麻烦，redux的出现就是为了解决state里面的数据问题。

webpack plugin有哪些？

REACT父子组件通信

在React中，父组件向子组件通过传递props的方式，向子组件进行通讯

父组件中使用了子组件，然后子组件的props属性被父组件的state赋值，这样，我们在子组件中使用this.props就能获得父组件用来赋值的state，也就获取了父组件要传递的信息。

父组件更新组件状态  -----props----->　子组件更新

子组件更新父组件状态   -----需要父组件传递回调函数----->  子组件调用触发

子组件更新父组件就需要：父组件通过props传递一个回调函数到子组件中，这个回调函数可以更新父组件，子组件通过触发这个回调函数，从而使父组件得到更新。

兄弟组件沟通：

方式一：按照React单向数据流方式，我们需要借助父组件进行传递，通过父组件回调函数改变兄弟组件的props。

方式二：