ES6 从基础到高级

讲师：张礼军

主要内容

- 为什么学 ES6+ ？
- ECMAScript 介绍
- ES6 兼容性问题
- ES6 开发环境搭建
- let 关键字
- const 关键字
- 解构赋值
- 模板字符串
- 对象的新语法
- for ... of
- 箭头函数
- 模块化
- promise
- async / await

为什么学 ES6+ ？

- 更高的开发效率
- 更少的出错可能
- 更人性化的语法

Object.assign　Array.from　……

let & const　Symbol　……

箭头函数　async / await　……

ECMAScript 介绍

- ECMAScript 是一种由 ECMA （前身为欧洲计算机制造商协会，英文名称是 European Computer Manufacturers Association ）通过 ECMA-262 标准化的脚本程序设计语言。
  - **官方网站：**https://github.com/tc39/ecma262

- ECMAScript 是JavaScript 的重要组成部分，是 JavaScript 中的语法规范。
  - **语法**——解析规则，关键字，语句，声明，操作等
  - **类型**——布尔型，数字，字符串，对象等
  - **原型和继承**
  - **内置对象和函数的标准库**——JSON，数字（Math）等等

ECMAScript 历史及版本

- 1996年11月，Netscape公司，决定将JavaScript提交给国际标准化组织ECMA。次年，ECMA发布ECMAScript1.0版。
- 1998年06月，ECMAScript2.0版发布。
- 1999年12月，ECMAScript3.0版发布，成为JavaScript的通行标准，得到了广泛支持。
- 2007年10月，ECMAScript4.0版草案发布，但以Yahoo、Microsoft、Google为首的大公司反对JavaScript的大幅升级，主张小幅改动，各方分歧太大，ECMA决定中止ECMAScript4.0的开发。
- 2009年12月，ECMAScript5.0版正式发布。
- 2015年06月，ECMAScript6（简称ES6，ES2015）正式通过，成为国际标准。

ES6 兼容性问题

- 各大浏览器的最新版本，对 ES6 的支持情况可查看下面这个网址：
  - http://kangax.github.io/compat-table/es6/
  - 虽然 ES6 提供了许多新特性，但并不是所有的浏览器都能够完美支持。好在目前各大浏览器自身也加快速度兼容 ES6 的新特性，其中对 ES6 新特性最友好的是 Chrome 和 Firefox 浏览器，但完全兼容还需要些时日。

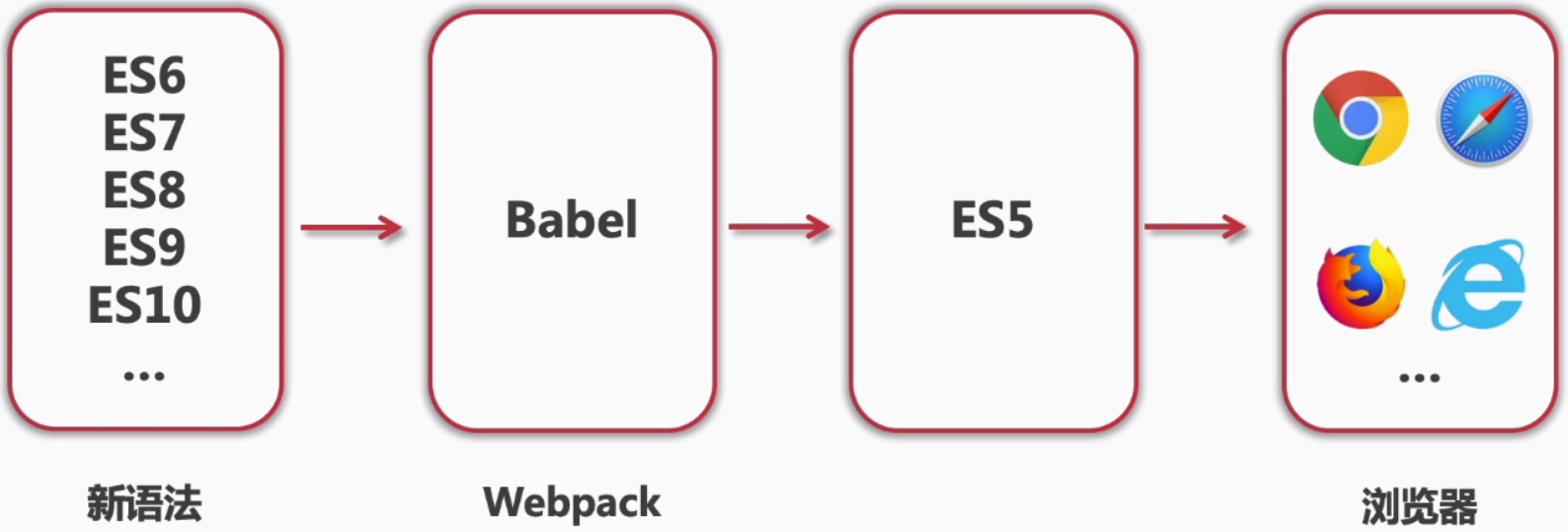
- 所以，既能使用 ES6 的新特性，又能保证各个浏览器的兼容，就成了当务之急！

使用 Babel 把 ES6 编译成 ES5

- 初始化项目
  - npm init -y
- 全局安装 babel-cli
  - npm install –g babel-cli
- 本地安装 babel-preset-es2015 和 babel-cli
  - npm install babel-preset-es2015 babel-cli
- 新建 .babelrc 文件
  - 添加配置项： {"presets": ["es2015"]}
- 执行转化命令
  - babel src -d dist

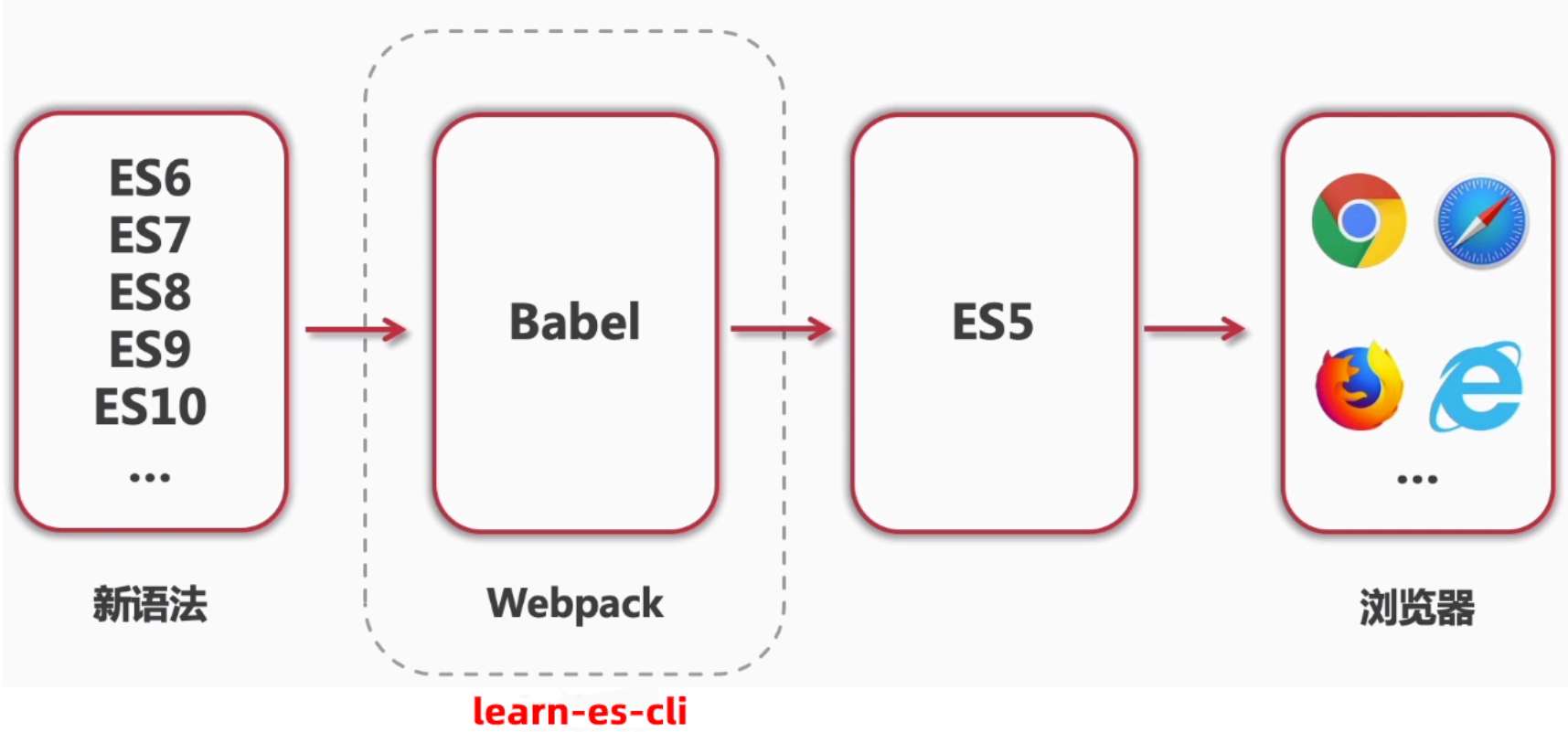
ES6 开发环境搭建

- Webpack 配合 Babel 将 ES6 转成 ES5（适合老手）
  - 针对 ES6 兼容性问题，很多团队为此开发出了多种语法解析转换工具，把我们写的 ES6 语法转换成 ES5，相当于在 ES6 和浏览器之间做了一个翻译官。比较通用的工具方案有 Babel。
  - Babel 是编写下一代 JavaScript 的编译器，可以将 ES6 代码转为 ES5 代码，让低端运行环境（如浏览器）能够认识并执行。Babel 中文站点： https://www.babeljs.cn/ 。



ES6 开发环境搭建

- 配置脚手架工具将 ES6 转成 ES5（适合新手）
  - 为了大家可以把更多的精力放在 ES6 新语法的学习上，而不是纠结于环境配置，我已经为大家准备好了开发环境，利用脚手架只需要一步就可以完成安装。



let 关键字

- 用 var 的不足之处
  - 没有提供**块级作用域**
  - 用 var 声明变量的时候会出现“**变量提升**”的现象

- 块级作用域
  - 任何一对花括号（{}）中的语句都属于一个块，在花括号里面用let定义的所有变量在花括号外都不可见，我们称之为块级作用域。

- 变量提升：先执行声明，后执行定义。

let 关键字

- let 确实能弥补一些 var 的不足之处，那么使用 let 的时候还有什么要注意的吗？
  - 同一个块级作用域内，不允许重复声明同一个变量
  - 函数内不能用 let 重新声明函数的参数

- 所以，平时记得养成变量先声明后使用的好习惯！

const 关键字

- const 关键字是专门用来声明常量的。

- 常量的特点
  - 不可更改
  - 只在块级作用域起作用
  - 不存在变量提升，必须先声明后使用
  - 不可重复声明同一个常量
  - 声明后必须同时赋值

如果常量是一个对象呢？

- 用 const 来声明一个对象类型的常量，就是传址赋值。不可修改的是对象在内存中的地址，而不是对象本身。

- **传址赋值**：在赋值过程中，变量实际上存储的是数据的地址（对数据的引用），而不是原始数据或者数据的拷贝。

解构赋值

- 解构赋值语法是 ES6 的一种表达式，可以方便的从数组或者对象中快速提取值赋给定义的变量。解构赋值好处就是代码简短，可读性和表现力更强。

- 解构赋值的用途
  - 交换变量的值
  - 提取函数返回的多个值
  - 定义函数的参数
  - 函数参数的默认值

模板字符串

- ES6 支持模板字符串，使得字符串的拼接更加的简洁、直观。

- 通过 `${}` 就可以完成字符串的拼接，只需要将变量放在大括号之中——`${变量名}`
  - ${} 中可以是运算表达式
  - ${} 中可以是对象的属性
  - ${} 中可以是函数的调用

对象的新语法

- 简写
  - 属性和值名字一样可以简写
  - 方法可以简写

for ... of

- 传统遍历数组的方式以及各自的缺陷
  - for 循环（不足：代码不够简洁）
  - forEach()方法（不足：无法中断停止整个循环）
  - for ... in（不足：每次循环得到的值是字符串类型，常用于遍历 json 对象）

- for...of：一种新增的用于遍历数据结构的方法。它可以遍历数组、对象、字符串等数据结构。
  - 写法比 for 循环简洁
  - 可以用 break 来终止整个循环，或者 continute 来跳出当前循环，继续后面的循环
  - 结合 keys() 获取到循环的索引，并且是数字类型，而不是字符串类型

箭头函数

- ES6 有一种全新的定义函数的方式，就是用箭头符号（=>），故得名为箭头函数。
  - 如果参数超过1个的话，需要用小括号 () 括起来，函数体语句超过1条的时候，需要用大括号 {} 括起来。例如：let fn = (a, b) => { return a + b }

- 使用箭头函数的注意事项
  - 箭头函数的 this 指向的是定义时的 this 对象，而不是执行时的 this 对象
  - 箭头函数里面没有 arguments ，建议用...剩余参数
  - 箭头函数不能当构造函数使用

ES6 模块化的实现

- 基本用法
- 默认导出（常用）

ES6 模块化的注意事项

- 模块化定义与使用，需要在服务器环境下。
- 声明的变量，对外都是只读的，但并不是所有导出的变量都不可修改。比如，对象类型的值就可修改。

什么是同步和异步？

- JavaScript 是一个单线程的编程语言
  - 所谓单线程，就是指一次只能完成一件任务。如果有多个任务，就必须排队，前一个任务完成，才会执行后一个任务，依次类推。如果前一个任务耗时过长，后一个任务就不得不一直等待下去，这样会拖延整个程序的执行。
  - 为了解决这个问题，JS 语言将任务的执行模式分成两种：同步和异步。

- 编程中的同步和异步与现实生活中的同步和异步正好相反
  - **同步**：在主线程上排队执行的任务，只有前一个任务执行完成，才能执行后一个任务。
  - **异步**：不进入主线程，而进入“任务队列”的任务，只有“任务队列”通知主线程，某个异步任务可以执行了，该任务才会进入主线程执行。

JS 异步编程的几种方法

- 回调函数
  - 回调就是一个在另外一个函数执行完成后要执行的函数。如果多个嵌套回调，就会引发“回调地狱”。
- Promise 对象
  - 虽然解决了“回调地狱”问题，但代码冗余。
- async / await
  - Promise 的语法糖，Promise 的进一步优化。

什么是 Promise ？

- Promise 简单说就是一个全局对象，是异步编程的一种解决方案。它可以将异步操作以同步操作的方式表达出来，避免了层层嵌套的回调函数，所以它比传统的解决方案回调函数更加的优雅。此外，Promise 对象提供统一的接口，使得控制异步操作更加容易。

Promise 两个特点

- **Promise 对象的状态不受外界影响**
  - Promise 对象有三种状态：**Pending（进行中）**、**Resolved（已完成，又称Fulfilled）**和**Rejected（已失败）**。只有异步操作的结果，可以决定当前是哪一种状态，任何其他操作都无法改变这个状态。
- **Promise 对象一旦状态改变，就不会再变，任何时候都可以得到这个结果**
  - Promise 对象的状态改变，只有两种可能：从 Pending 变为 Resolved 和从 Pending 变为 Rejected 。只要这两种情况发生，状态就不会再变了，会一直保持这个结果。

Promise——基本用法

- Promise 对象是全局对象，可以理解为一个类，参数是一个匿名函数，其中有两个参数方法，分别是 resolve 和 reject 。

```
new Promise(function(resolve, reject) {
    ......
})
```

resolve的作用是，将Promise对象的状态从“未完成”变为“成功”（即从Pending变为Resolved），在异步操作成功时调用，并将异步操作的结果，作为参数传递出去。

reject的作用是，将Promise对象的状态从“未完成”变为“失败”（即从Pending变为Rejected），在异步操作失败时调用，并将异步操作报出的错误，作为参数传递出去。

Promise——实例方法

- then 方法：用于指定发生状态改变时的回调函数。它的第一个参数是 Resolved 状态的回调函数，第二个参数（可选）是 Rejected 状态的回调函数。
- catch 方法：用于指定发生错误时的回调函数。它的参数是 Rejected 状态的回调函数。

什么是 async / await ？

- async 的本质是 Promise 的语法糖。只要函数标记为 async ， 就表示里头可以编写 await 的同步语法，而 await 顾名思义就是“等待”的意思，它会确保一个 Promise 的状态发生改变（不管是成功还是失败）后才会进行下一步。
- 一句话简单总结：async 表示函数里有异步操作，await 表示紧跟在后面的表达式需要等待结果。

async / await　VS　Promise

- Promise 主要是 then 方法的链式调用，是一种从左向右的横向写法。
- async / await 是从上到下，顺序执行，这更像在写同步代码，也更符合编写代码的习惯（异步编程的最高境界，就是根本不用关心它是不是异步）。
- async / await 是基于 Promise 的，是对 Promise 的进一步优化。

async / await 基本语法

- async 函数返回一个 Promise 对象
  - async 函数内部 return 语句返回的值，会成为 then 方法回调函数的参数。
  - async 函数内部抛出错误，会导致返回的 Promise 对象变为 reject 状态。抛出的错误对象会被 catch 方法回调函数接收到。

async / await 基本语法

- async 函数返回的 Promise 对象，必须等到内部所有 await 命令的 Promise 对象执行完，才会发生状态改变。也就是说，只有 async 函数内部的异步操作执行完，才会执行 then 方法指定的回调函数。

async / await 基本语法

- 正常情况下，await 命令后面是一个 Promise 对象。如果不是，会被转成一个立即 resolve 的 Promise 对象。
- 只要一个 await 语句后面的 Promise 变为 reject ，那么整个 async 函数都会中断执行。
- await 必须写在 async 函数中，但 async 函数中可以没有 await。
- 如果 await 的 Promise 失败了，就会抛出异常，需要通过 try...catch 来捕获处理。