# 80%应聘者都不及格的JS面试题

|  |
| --- |
| for (*var* i = 0; i < 5; i++) {      setTimeout(*function* () {  *console*.log(new Date, i);      }, 1000);  }  *console*.log(new Date, i);  /\*  只要你对 JS 中同步和异步代码的区别、变量作用域、闭包等概念有正确的理解，就知道正确答案  2017-03-18T00:43:45.873Z 5  2017-03-18T00:43:46.866Z 5  2017-03-18T00:43:46.868Z 5  2017-03-18T00:43:46.868Z 5  2017-03-18T00:43:46.868Z 5  2017-03-18T00:43:46.868Z 5  \*/ |

## 闭包

|  |
| --- |
| for (*var* i = 0; i < 5; i++) {      setTimeout(() *=>* {  *console*.log(new Date, i);      }, 1000);  }  *console*.log(new Date, i);  /\*   5 -> 5,5,5,5,5，即第 1 个 5 直接输出，1 秒之后，输出 5 个 5；   循环执行过程中，几乎同时设置了 5 个定时器，一般情况下，这些定时器都会在 1 秒之后触发，   而循环完的输出是立即执行的  \*/  // ==============================================================  // 闭包  for (*var* i = 0; i < 5; i++) {      (*function* (*j*) { // j = i          setTimeout(*function* () {  *console*.log(new Date, j);          }, 1000);      })(i);  }  *console*.log(new Date, i);  // 5 -> 0,1,2,3,4  // IIFE（Immediately Invoked Function Expression：声明即执行的函数表达式）来解决闭包造成的问题  // ==============================================================  for (*var* i = 0; i < 5; i++) {      setTimeout(*function* (*j*) {  *console*.log(new Date, j);      }, 1000, i);  }  *console*.log(new Date, i);  // 5 -> 0,1,2,3,4  // ==============================================================  // 利用参数按值传递  *var* output = *function* (*i*) {      setTimeout(*function* () {  *console*.log(new Date, i);      }, 1000);  };  for (*var* i = 0; i < 5; i++) {      output(i); // 这里传过去的 i 值被复制了  }  *console*.log(new Date, i);  // 5 -> 0,1,2,3,4  // ==============================================================  // ES6块级作用域  for (*let* i = 0; i < 5; i++) {      setTimeout(*function* () {  *console*.log(new Date, i);      }, 1000);  }  *console*.log(new Date, i);  // 报错  // 0,1,2,3,4 |

## ES6

|  |
| --- |
| // 代码执行时，立即输出 0，之后每隔 1 秒依次输出 1,2,3,4，循环结束后在大概第 5 秒的时候输出 5  // 这里使用大概，是为了避免钻牛角尖的同学陷进去，因为 JS 中的定时器触发时机有可能是不确定的  // 虽粗暴有效  for (*var* i = 0; i < 5; i++) {      (*function* (*j*) {          setTimeout(*function* () {  *console*.log(new Date, j);          }, 1000 \* j); // 这里修改 0~4 的定时器时间      })(i);  }  setTimeout(*function* () { // 这里增加定时器，超时设置为 5 秒  *console*.log(new Date, i);  }, 1000 \* i);  // ==============================================================  // Promise  *const* tasks = [];  for (*var* i = 0; i < 5; i++) { // 这里 i 的声明不能改成 let，如果要改该怎么做？      ((*j*) *=>* {          tasks.push(new Promise((*resolve*) *=>* {              setTimeout(() *=>* {  *console*.log(new Date, j);                  resolve(); // 这里一定要 resolve，否则代码不会按预期 work              }, 1000 \* j); // 定时器的超时时间逐步增加          }));      })(i);  }  *Promise*.all(tasks).then(() *=>* {      setTimeout(() *=>* {  *console*.log(new Date, i);      }, 1000); // 注意这里只需要把超时设置为 1 秒  });  // ==============================================================  *const* tasks = []; // 这里存放异步操作的 Promise  *const* output = (*i*) *=>* new Promise((*resolve*) *=>* {      setTimeout(() *=>* {  *console*.log(new Date, i);          resolve();      }, 1000 \* i);  });  // 生成全部的异步操作  for (*var* i = 0; i < 5; i++) {      tasks.push(output(i));  }  // 异步操作完成之后，输出最后的 i  *Promise*.all(tasks).then(() *=>* {      setTimeout(() *=>* {  *console*.log(new Date, i);      }, 1000);  }); |

## ES7

|  |
| --- |
| // async/await  // 模拟其他语言中的 sleep，实际上可以是任何异步操作  *const* sleep = (*timeountMS*) *=>* new Promise((*resolve*) *=>* {      setTimeout(resolve, timeountMS);  });  (async () *=>* { // 声明立即执行的 async 函数表达式      for (*var* i = 0; i < 5; i++) {          if (i > 0) {              await sleep(1000);          }  *console*.log(new Date, i);      }      await sleep(1000);  *console*.log(new Date, i);  })(); |

## 其他

|  |
| --- |
| for (*var* i = 0; i < 5; i++) {  *console*.log(i);  }  *console*.log(i);  //这个大家应该很快就知道了，012345  for (*var* i = 0; i < 5; i++) {      setTimeout(*function* () {  *console*.log(i);      }, 1000);  }  *console*.log(i);  //这个大家就要小心一点了，答案是5 55555  //在setTimeout执行之前，for循环早就执行完了，i的值早已经是5了，所以一开始是执行，最后面的console.log(i);  //在for循环的时候一下子自定义5个setTimeout，大概一秒后，就是输出55555  for (*var* i = 0; i < 5; i++) {      (*function* (*j*) { // j = i          setTimeout(*function* () {  *console*.log(j);          }, 1000);      })(i);  }  *console*.log(i);  //这里的解析和上面基本一样，只是用闭包来记录每一次循环的i,  //所以答案是5 01234  *var* output = *function* (*i*) {      setTimeout(*function* () {  *console*.log(i);      }, 1000);  };  for (*var* i = 0; i < 5; i++) {      output(i); // 这里传过去的 i 值被复制了  }  *console*.log(i);  //这里的解析和上面基本一样，把i当参数传进output，记录每一次循环的i,  //所以答案是5 01234  for (*let* i = 0; i < 5; i++) {      setTimeout(*function* () {  *console*.log(i);      }, 1000);  }  *console*.log(i);  //结果是 报错 01234  //注意i是用let定义的，不是var |

## 运行一秒后打印输出0-9

|  |
| --- |
| // 这道题涉及到作用域  for (*var* i = 0; i < 10; i++) {      setTimeout((*function* (*i*) {          return *function* () {  *console*.log(i);          }      })(i), 1000);  }  // 1s后打印0到9  // ==============================================================  for (*let* i = 0; i < 10; i++) {      setTimeout(*function* () {  *console*.log(i);      }, 1000);  }  // 1s后打印0到9 |