**第六章**

**&&与| |基本理解和应用**

* 最常见情况（运算符两边的操作数都是布尔类型）  
  - 对于&&来说， 除了两侧都为真时为真，其他情况都为假  
  - 对于 | | 来说，除了两侧都为假时为假，其他情况都为真

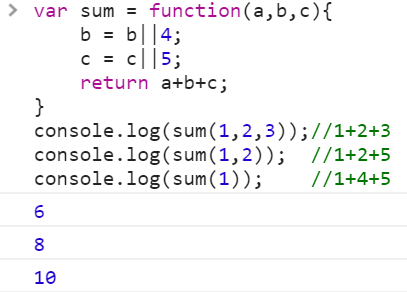
**&&与||深层次理解**

* 当逻辑运算符&&和||两侧的操作数不是布尔类型时  
  - 首先将左操作数转换成布尔类型  
  - 对转换后的左操作数进行逻辑判断（true or false）  
  - 根据短路原则返回原始左操作数或原始右操作数
* 短路原则（忽略对右操作数的判断）  
  - 对于&&，转换后的左操作数若为true，则直接返回原始右操作数，若为false则  
   直接返回原始左操作数  
  - 对于| |，转换后的左操作数若为true，则直接返回原始左操作数，若为false则直  
   接返回原始右操作数  
  - 通过短路原则，可以用&&和| |来实现复杂的条件语句来代替if-else

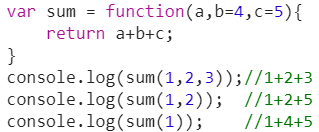
**&&与||在实际中的应用**

**遵循短路特性，使用&&和||可用来实现条件语句**

**使用||来设置函数参数的默认值**- 函数定义时可以给参数指定默认值，调用时若未传参数则该参数的值取它定义时的默认值  
- JS（ES6之前）不能直接为函数的参数指定默认值，可以通过 | | 来实现



案例中：未传实参的话，形参初始为undefined，undefined转为布尔类型为false，根据||短路原则直接返回右操作数，等效代码如下



**sum函数的问题及完善**- 若实参转换为布尔类型为false，返回值则可能不是预期结果，如sum(1,0,0)为10  
- 增加判断，确保实参转换为布尔类型时为true

* ！运算符与逻辑运算符  
  ！（A&&B） === ！A ||！B  
  ！（A||B） === ！A &&！B  
   用!!实现布尔类型转换
* 使用||实现返回值的限制  
  function foo(a,b){  
   return (a+b)||"和不能为0";  
  }  
  foo(-2,2);//和不能为0