# Underscore关于函数

# 参数类型参考

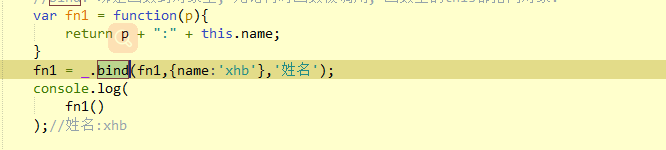
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **数据类型** | **数组类型** | **字符串类型** | **数字类型** | **对象类型** | **函数类型** | **元素选择器** | **未定义类型** | **空类型** | **无参数类型** | **事件类型** | **或者** | **所有参数** |
| **参数指向** | Arr/arr | Str/str | N/n | Obj/obj | Fn/fn | Ele/ele | underfind | null |  | event | / | param |

## 详细内容参考：https://www.cnblogs.com/zzsdream/p/6843864.html

### bind: 绑定函数到对象上, 无论何时函数被调用, 函数里的this都指向对象.

如：\_.bind(fn,obj,param)

例子：



### defer: 延迟调用函数, 直到当前调用栈被清空为止, 跟使用 setTimeout 赋予0毫秒的延时很像. 对执行高消耗算法或大型HTML呈现而不阻碍UI更新线程很有用.

如：\_.defer(fn)

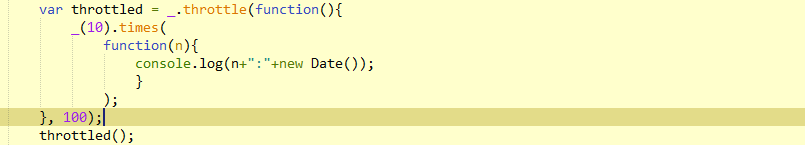
例子：

\_.defer(function(){ console.log('deferred'); });

### throttle:返回一个类似于节流阀一样的函数, 当高频率的调用函数, 实际上会每隔 wait 毫秒才会调用一次. 对于高到您感觉不到的高频率执行的函数时非常有用

如：\_.throttle(fn,n)

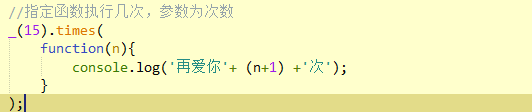
例子：

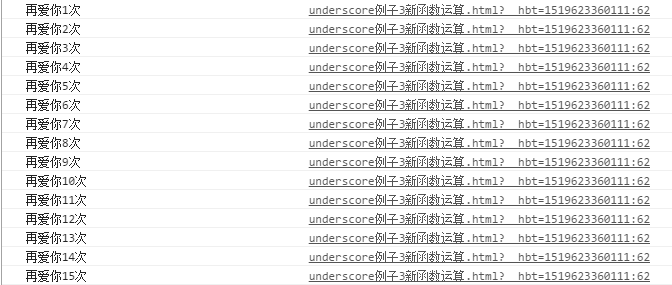


### 指定函数执行几次，参数为次数

如：\_(n).times(fn);

例子：





### debounce: 返回函数的防反跳版本, 将延迟函数的执行(真正的执行)在函数最后一次调用时刻的等待xx毫秒之后，可以实现延迟加载(应用场景：在有分页的页面中不断点击下一页时，间隔不超过设定时间则响应最后一次点击)。

如：\_.debounce(fn,n)

例子：



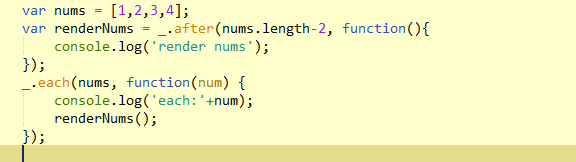
### once: 创建一个只能运行一次的函数. 重复调用此修改过的函数会没有效果, 只会返回第一次执行时返回的结果。单例模式。

如：\_.once(fn)

### after: 对循环计数，只有超过计数，才会调用指定的函数

如：\_.after(n,fn)

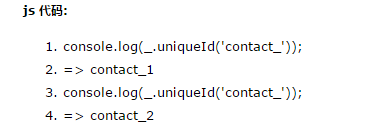
例子：





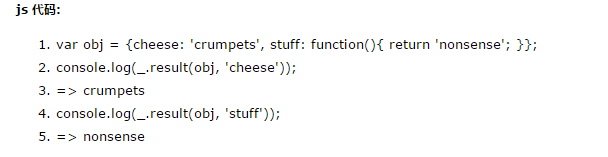
### ****uniqueId:产生一个全局的唯一id，以参数作为前缀****

如：\_.**uniqueId**(str/n)



### ****result: 通过字符串调用对象的函数，或返回属性值****

如：\_.result(obj,param)



### ****chain: 返回一个封装的对象. 在封装的对象上调用方法会返回封装的对象本身, 直到value() 方法调用为止.(链式语法)****

如：\_.**chain**(param)

....

.value();

