**Basic Async**

* setTimeout(callback,time)
  + 只运行一次，达到触发时间后自动触发callback
* setInterval(callback,time)
  + 按照间隔不间断运行callback
* clearTimeout(timeId)
  + 清除timeout超时运行callback,timeId 由setTimeout返回获得
* clearInterval(timeId)
  + 清除interval超时运行callback,timeId 由setInterval返回获得
* Queue(setTimeout,setInterval) 必需在当前的Stack(普通函数)结束才会再运行
* Promise 约定模式(Promise对象接收两个回调，成功与失败，可以利用这种机制来避开Queue Stack运行先后的问题)
  + var p1 = new Promise(function(resolve,reject){

//成功

Resolve(“data”); 运行p1.then(function(data){})

//失败

Reject(“data”); 运行p1.catch(function(data){})

});

* 使用Promise Chain构建多级Promise(在then中return new Promise再进行控制从而实现多级Promise)

**Ajax**

* JSON转换对象的基本功能
  + JSON.parse() 从字符串转换为对象
  + JSON.stringify() 从对象转换为字符串
* 实例化XmlHttpRequest对象实现，又称为XHR方法
  + var xmlHttpRequest = new XMLHttpRequest();  
    xmlHttpRequest.onreadystatechange = function () {  
     if(this.readyState == 4){ //1.状态变更，总共有4种  
     var answer = this.responseText;  
     var data = JSON.parse(answer); //2.解析JSON数据  
     img\_picture.setAttribute("src",data.message);  
     console.log(data.message);  
     }  
    }  
      
    xmlHttpRequest.open("get","https://dog.ceo/api/breeds/image/random"); //3.设置请求地址及请求类型  
    xmlHttpRequest.send();//4.发送请求
  + readyState的四种状态
    - 0 UNSENT，客户端已经建立，但是还没有使用open()
    - 1 OPENED，使用open()后
    - 2 HEADERS\_RECEIVED，使用send()后
    - 3 LOADING, responseText已经有部分数据了
    - 4 DONE, 操作已完成，responseText可以提供所有数据了
* FETCH API实现
  + XHR不能实现streaming（流处理）功能，语法不清晰，FETCH API很好的解决了这两个问题。
    - fetch(URL).then(function (res) {

//请求成功  
   
 }).catch(function (error) {  
 //请求错误  
 });

* FETCH API 支持的选项
  + Fetch(url,{method:post,body:JSON.stringify({name:’xx’}),xxx:xxx});