题目：红灯三秒亮一次，绿灯一秒亮一次，黄灯2秒亮一次；如何让三个灯不断交替重复亮灯？（用Promse实现）

三个亮灯函数已经存在：

function red(){

console.log('red');

}

function green(){

console.log('green');

}

function yellow(){

console.log('yellow');

}

这道题首先考察Promise的应用，Promise的详细说明请看我的这篇文章：[闲话Promise机制](http://www.cnblogs.com/dojo-lzz/p/4340897.html)。首先我们需要一个函数来实现时间控制：

var tic = function(timmer, cb){

return new Promise(function(resolve, reject) {

setTimeout(function() {

cb();

resolve();

}, timmer);

});

};

如果把问题简化一下，如果只需要一个周期，那么利用Promise应该这样写：

var d = new Promise(function(resolve, reject){resolve();});

var step = function(def) {

def.then(function(){

return tic(3000, red);

}).then(function(){

return tic(2000, green);

}).then(function(){

return tic(1000, yellow);

});

}

现在一个周期已经有了，剩下的问题是如何让他无限循环。说道循环很容易想到for while do-while这三个，比如：

var d = new Promise(function(resolve, reject){resolve();});

var step = function(def) {

while(true) {

def.then(function(){

return tic(3000, red);

}).then(function(){

return tic(2000, green);

}).then(function(){

return tic(1000, yellow);

});

}

}

如果你是这样想的，那么恭喜你成功踩了坑！这道题的第二个考查点就是setTimeout相关的异步队列会挂起知道主进程空闲。如果使用while无限循环，主进程永远不会空闲，setTimeout的函数永远不会执行！

正确的解决方法就是这道题的第三个考查点——递归！！！解决方案如下：

var d = new Promise(function(resolve, reject){resolve();});

var step = function(def) {

def.then(function(){

return tic(3000, red);

}).then(function(){

return tic(2000, green);

}).then(function(){

return tic(1000, yellow);

}).then(function(){

step(def);

});

}

整体代码如下：

function red(){

console.log('red');

}

function green(){

console.log('green');

}

function yellow(){

console.log('yellow');

}

var tic = function(timmer, cb){

return new Promise(function(resolve, reject) {

setTimeout(function() {

cb();

resolve();

}, timmer);

});

};

var d = new Promise(function(resolve, reject){resolve();});

var step = function(def) {

def.then(function(){

return tic(3000, red);

}).then(function(){

return tic(2000, green);

}).then(function(){

return tic(1000, yellow);

}).then(function(){

step(def);

});

}

step(d);

同时可以看到虽然Promise可以用来解决回调地狱问题，但是仍然不可避免的会有回调出现，更好的解决方案是利用Generator来减少回调:

var tic = function(timmer, str){

return new Promise(function(resolve, reject) {

setTimeout(function() {

console.log(str);

resolve(1);

}, timmer);

});

};

function \*gen(){

yield tic(3000, 'red');

yield tic(1000, 'green');

yield tic(2000, 'yellow');

}

var iterator = gen();

var step = function(gen, iterator){

var s = iterator.next();

if (s.done) {

step(gen, gen());

} else {

s.value.then(function() {

step(gen, iterator);

});

}

}

step(gen, iterator);