**1. 配置**

Socket.IO提供了4个配置的API：io.configure, io.set, io.enable, io.disable。其中io.set对单项进行设置，io.enable和io.disable用于单项设置布尔型的配置。io.configure可以让你对不同的生产环境（如devlopment，test等等）配置不同的参数。以下定义了development和release两种环境下Socket.IO的不同配置：

复制代码代码如下:

var io = require('socket.io').listen(80);

io.configure('development', function(){  
    io.enable('browser client etag');  
    io.set('log level', 1);  
});

io.configure('release', function(){  
    io.set('transports', ['websocket']);  
});

下面列举一些常用的配置项，具体配置参数参见官方WIKI

1).transports(默认['websocket', 'htmlfile', 'xhr-polling', 'jsonp-polling'])：一个包含通信方法类型的数组。Socket.IO支持多种实现在线即时通信的方式，如websocket、polling等等，该配置能让你自行选择备用的通信方式。  
2).log level(默认3)：日志输出的最低级别，0为error，1为warn，2为info，3为debug，默认即输出所有类型的日志。  
3).heartbeat interval(默认25秒)：心跳包发送间隔，客户端需要在此时间段之内向服务器发送一个心跳包才能保持通信。

**2. 房间**

房间是Socket.IO提供的一个非常好用的功能。房间相当于为指定的一些客户端提供了一个命名空间，所有在房间里的广播和通信都不会影响到房间以外的客户端。

在入门篇中，我们知道socket.join('room name')可用于客户端进入房间，socket.leave('room name')用于离开房间。当客户端进入一个房间之后，可以通过以下两种方式在房间里广播消息：

复制代码代码如下:

//1. 向my room广播一个事件，提交者会被排除在外（即不会收到消息）  
io.sockets.on('connection', function (socket) {  
    //注意：和下面对比，这里是从客户端的角度来提交事件  
    socket.broadcast.to('my room').emit('event\_name', data);  
}

//2. 向another room广播一个事件，在此房间所有客户端都会收到消息  
//注意：和上面对比，这里是从服务器的角度来提交事件  
io.sockets.in('another room').emit('event\_name', data);

//向所有客户端广播  
io.sockets.emit('event\_name', data);

除了向房间广播消息之外，还可以通过以下API来获取房间的信息。

复制代码代码如下:

//获取所有房间的信息  
//key为房间名，value为房间名对应的socket ID数组  
io.sockets.manager.rooms

//获取particular room中的客户端，返回所有在此房间的socket实例  
io.sockets.clients('particular room')

//通过socket.id来获取此socket进入的房间信息  
io.sockets.manager.roomClients[socket.id]

**3. 事件**

Socket.IO内置了一些默认事件，我们在设计事件的时候应该避开默认的事件名称，并灵活运用这些默认事件。

服务器端事件：

1).io.sockets.on('connection', function(socket) {})：socket连接成功之后触发，用于初始化  
socket.on('message', function(message, callback) {})：客户端通过socket.send来传送消息时触发此事件，message为传输的消息，callback是收到消息后要执行的回调  
2).socket.on('anything', function(data) {})：收到任何事件时触发  
3).socket.on('disconnect', function() {})：socket失去连接时触发（包括关闭浏览器，主动断开，掉线等任何断开连接的情况）

客户端事件：

1).connect：连接成功  
2).connecting：正在连接  
3).disconnect：断开连接  
4).connect\_failed：连接失败  
5).error：错误发生，并且无法被其他事件类型所处理  
6).message：同服务器端message事件  
7).anything：同服务器端anything事件  
8).reconnect\_failed：重连失败  
9).reconnect：成功重连  
10).reconnecting：正在重连

在这里要提下客户端socket发起连接时的顺序。当第一次连接时，事件触发顺序为：connecting->connect；当失去连接时，事件触发顺序为：disconnect->reconnecting（可能进行多次）->connecting->reconnect->connect。

**4. 授权**

1).向所有客户端广播：socket.broadcast.emit('broadcast message');

2).进入一个房间（非常好用！相当于一个命名空间，可以对一个特定的房间广播而不影响在其他房间或不在房间的客户端）：socket.join('your room name');

3).向一个房间广播消息（发送者收不到消息）：socket.broadcast.to('your room name').emit('broadcast room message');

4).向一个房间广播消息（包括发送者都能收到消息）（这个API属于io.sockets）：io.sockets.in('another room name').emit('broadcast room message');

5).强制使用WebSocket通信：（客户端）socket.send('hi')，（服务器）用socket.on('message', function(data){})来接收。

Socket.IO的进阶用法介绍基本就到这里。个人感觉在日常使用的时候这些基本API已经够用了，这也体现了Socket.IO极其简洁易用的设计哲学。本文只是抛砖引玉，当在实际运用中遇到解决不了的问题时，再去查看官方详细的WIKI会比较好。