# 第一种方法

参考地址：<http://isux.tencent.com/mobile-development-essential-knowledge.html>

**贡献者 [北京] 阿多**

### （1） weinre远程实时调试

Web开发者经常使用Firefox的firebug或者Chrome的开发人员工具进行Web调试，包括针对JavaScript，DOM元素和CSS样式的调试。但是，当我们期望为移动Web站点或应用进行调试时，这些工具就很难派上用场。

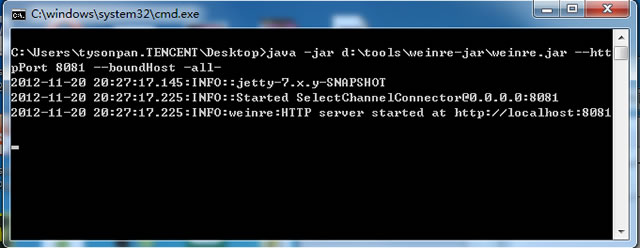
[weinre](http://people.apache.org/~pmuellr/weinre/docs/latest/Home.html)就是一个帮助我们在桌面来远程调试运行在移动设备浏览器内的Web页面或应用的调试工具。weinre是WEb INspector REmote的简写，现在是Apache的一个开源项目，托管在[github](https://github.com/apache/incubator-cordova-weinre)。

下面将介绍如此在日常工作使用它。

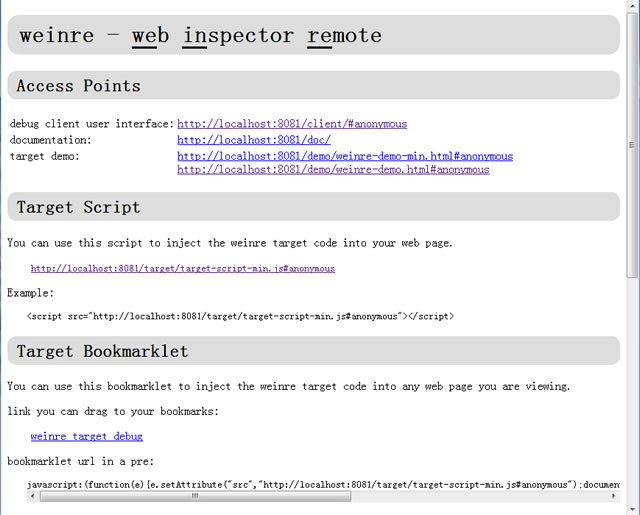
首先，我们要下载weinre的jar包——项目官方已经找不到该jar文件，网上能够找到，这里建议搭建个独立的web服务器，jar运行后是一个本地的服务器，和web服务器差不多~~

然后通过运行dos命令来启动它（**请注意在你的电脑上已经安装有JDK**）。运行命令如下，需要把路径改成你的实际文件位置：

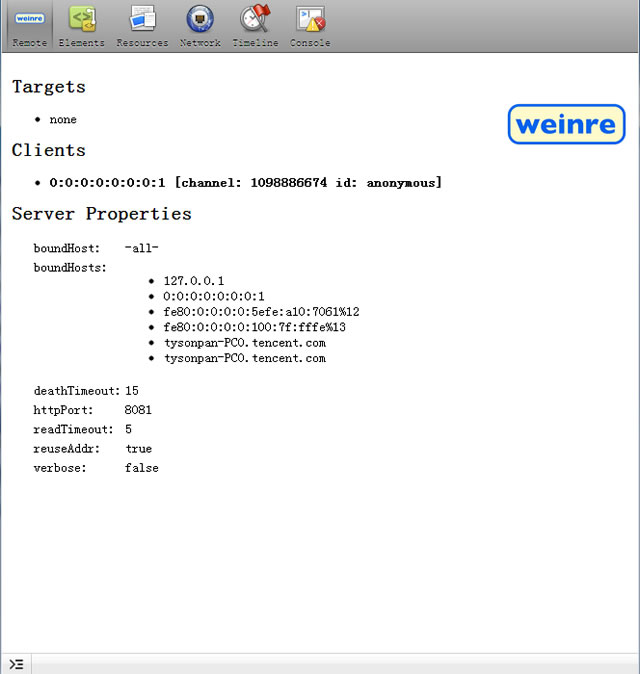
java -jar d:\tools\weinre-jar\weinre.jar –httpPort 8081 –boundHost -all- （httpPort是指定服务端口，boundHost参数说明可以使用IP访问，all参数代表支持所有的host）。



访问localhost:8081，如果看到如下的页面，说明weinre已经启动成功：



输入debug client user interface地址（调试客户端UI地址）。本例中即：http://localhost:8081/client/#anonymous，其中#anonymous是默认的调试id（debug id）。如果这个weinre调试服务器只是由你一个人使用，那么你可以使用默认的debug id：anonymous。 启动的weinre调试客户端ui如下图：



在需要调试的页面加入中以下脚本：<script type=”text/javascript” src=”http://localhost:8081/target/target-script-min.js#anonymous”></script>，注意把localhost换成手机能够访问的真实IP地址。当手机访问这个页面时，weinre客户端就会检测到目标设备，然后就可以对它进行调试了。



因为手机上不方便截图，我这里就用两个浏览器窗口来展示效果，其实手机上的效果跟右边是一样的。

# 第二种方法

参考地址：http://blog.csdn.net/freshlover/article/details/42528643

**贡献者 [杭州]小夜\_is\_a\_Jock**

在智能手机还未普及时，移动设备的调试处处是alert的，这估计是最常用的办法了。以前很多时候为了预览页面在移动设备上的效果，需要先将页面上传到测试服务器，再将url输入到设备浏览器，或者使用第三方二维码扫码应用，通过移动设备访问打开浏览实际效果，每次换设备都要重复这些操作，页面多的话这些繁琐的事情就特别让人厌烦了。移动互联网的浪潮，伴随着智能硬件的兴起与移动设备的普及，让前端工程师这个职业变得更为专业，前端工程师要做的工作也越来越多，尤其是国内大大小小众多的手机厂商的兴起、手机型号的繁多，仅仅依靠Firebug与Chrome/Safari自带的debug工具已经远远难以满足需求了。繁琐重复性劳动占用了相当的时间，庆幸的是前端的自动化工具也越来越多。越来越多的便捷调试工具无非是前端工程师的一大福音。

近几年，浏览器厂商也纷纷推出自己的远程调试(RemoteDebugging)工具，比如Opera Mobile 可以借助其推出的跨设备跨平台桌面开发者工具Opera Dragonfly 实现远程调试，iOS Safari 可以开启Web检查器在 Mac OS X系统中实现远程调试。Android 4+已上系统的 Chrome for Android可以 配合 ADB（Android Debug Bridge）实现桌面远程调试，桌面版Chrome 32+已经支持免安装ADB即可实现远程调试移动设备页面/WebView 。国内的UC浏览器开发者版也推出了自己的远程调试工具RemoteInspector（简称RI）。

除了浏览器厂商之外，也涌现出许多第三方开发的远程调试工具，诸如支持全平台调试的Weinre 和Adobe Edge Inspect CC，国人自研的iOS专用工具MIHTool。

本篇主要说一下Chrome RemoteDebugging 的方法，之前也遇到一些坑，自己总结了一些经验，分享如下。

**Chrome DevTools调试移动设备Brower Page Tabs/WebViews**

**安卓远程调试目前支持所有操作系统（Windows,Mac, Linux, and Chrome OS.）中调试，支持**：

● 调试站点的页面

● 调试安卓原生App中的WebView

● 实时将安卓设备的屏幕图像同步显示到开发机器。

● 通过端口转发（port forwarding）与虚拟主机映射（virtual host mapping）实现安卓移动设备与开发服务器进行交互调试。

**调试要求：**

● 开发环境安卓桌面版Chrome32+

● 一条USB数据线，连接电脑与移动设备，安装相应机型的USB驱动。驱动程序下载地址：<http://developer.android.com/tools/extras/oem-usb.html>

   如果电脑上安装有百度手机助手、360手机助手这类软件，一般连接后可以自动安装相应的USB驱动程序。

● 如果是调试网页，移动设备需要安装Chrome forAndroid ，且安卓系统须为Android 4.0+

● 对于APP WebView的调试，需要系统为Android 4.4+ 并且原生应用内的Webview须进行相应的调试声明配置。

说明：远程调试要求桌面版Chrome浏览器版本要高于安卓移动设备的Chrome版本号。有条件的最好使用Chrome 的金丝雀特别版Chrome Canary (Mac/Windows)或者Chrome桌面开发版Chrome Dev channel release (Linux)。

**第一步：首先在移动设备上开启USB调试模式。**方法：

● Android 3.2+，打开设置 – 应用程序 – 开发，在“USB调试”处打钩选上

● Android 4.0~ Android 4.1 ，打开设置-开发者选项-进入在“USB调试”处打钩选上。

● Android 4.2+，打开设置-关于手机-手机配置信息-连点“版本号”7次，返回上层就可以看到“开发者选项”显示出来了，在“USB调试”处打钩选上。

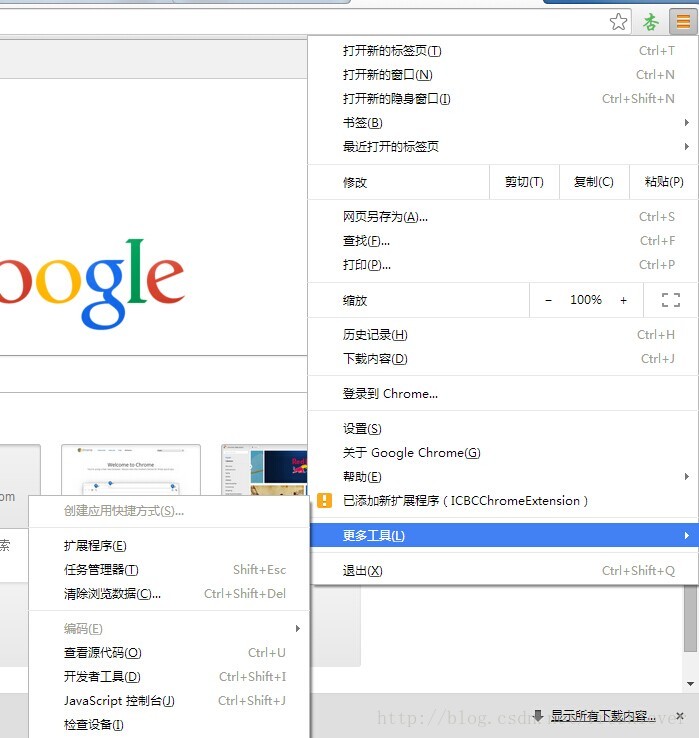


**第二步：用USB数据线连接设备**，驱动装好连接成功后，你可能会在设备上看到一个弹框请求允许使用这台计算机通过usb调试

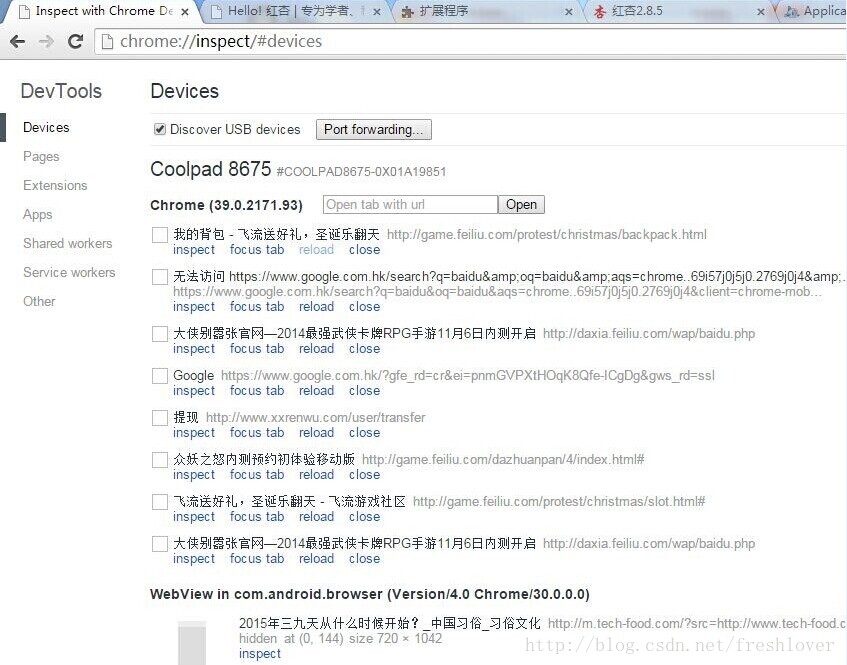


勾选后点击“确定”。

**第三步：在电脑上打开Chrome浏览器的菜单– 更多工具 – 检查设备**（Chromemenu > More tools > Inspect Devices），或者直接在浏览器地址栏输入chrome://inspect 或者about:inspect



打开后DevTools后，确保打钩选中Discover USB devices



如果USB连接成功，这时候我们可以看到移动设备的型号和设备上运行的页面和允许调试的WebView列表。找到需要调试的目标页面，点击inspect即可打开DevTools，点击reload可重新加载当前的调试页面，点击focus tab可将标签页置顶，close为关闭当前页面。更可以通过在输入框中键入网址新开一个页面。

点击inspect打开DevTools后，你可以选中页面中的DOM元素，同时设备中对应元素也高亮显示，也可使用DevTools中的Inspect Element 选中目标元素，可以实时与移动设备页面交互，方便的定位问题所在，进行代码调试。

在输入框中输入一个新的网址，点击Opoen可打开你想要调试的新页面。

说明：由于Chrome版本不同，DevTools也可能有些差别，比如我的Chrome38.0.2125.104 m版竟然没有打开新页面的url输入框，升级到v39后恢复正常，知道原因的欢迎留言。

1. DevTools窗口使用F5快捷键（CMD+R for Mac）重新加载页面

2. 使用Network面板可以实时观察页面在手机实际网络环境中资源的加载情况。

3. Timeline面板可以用来分析页面渲染和CPU使用情况，通常情况下移动设备的性能会比电脑上要低一些。

4.  在DevTools的Console控制台中编写脚本执行，会同步表现在移动设备中检查的页面。

5. 如果要调试本地搭建的服务器程序，需要用到端口转发和虚拟主机映射，以使设备上可以呈现你本地环境下的页面内容。

调试WebView需要满足安卓系统版本为Android 4.4+已上。并且需要再你的APP内配置相应的代码，在WebView类中调用静态方法setWebContentsDebuggingEnabled，如下：

**[java]** [view plaincopy](http://blog.csdn.net/freshlover/article/details/42528643)[在CODE上查看代码片](https://code.csdn.net/snippets/575398)

1. **if** (Build.VERSION.SDK\_INT >=Build.VERSION\_CODES.KITKAT) {
3. WebView.setWebContentsDebuggingEnabled(**true**);
5. }

以上配置方法适用于安卓应用内所有的WebView情形。

安卓WebView是否可调试并不取决于应用中manifest的标志变量debuggable，如果你想只在debuggable为true时候允许WebView远程调试，请使用以下代码段：

**[java]** [view plaincopy](http://blog.csdn.net/freshlover/article/details/42528643)[在CODE上查看代码片](https://code.csdn.net/snippets/575398)

1. **if** (Build.VERSION.SDK\_INT>= Build.VERSION\_CODES.KITKAT) {
3. **if** (0 != (getApplicationInfo().flags &=ApplicationInfo.FLAG\_DEBUGGABLE))
5. { WebView.setWebContentsDebuggingEnabled(**true**);}
7. }

WebView的可用列表如下：



所列信息其中包括页面的title，url地址，页面尺寸大小及其相对于设备屏幕的相对位置。

**实时屏幕投影（Live screencasting）**

调试移动设备时视线在两个设备的屏幕之间来回切换太不方便了，screencast实现了移动设备屏幕与开发环境DevTools的同步，你可以通过screencast与移动设备上的内容进行交互式的操作。

Screencast只呈现页面的内容，而不显示工具条地址栏、设备键盘等其他设备接口，这些在Screencase中表现为透明部分。

Android 4.4，是由Google公司制作和研发的代号为KitKat的手机操作系统，于北京时间2013年9月4日凌晨对外公布了该Android新版本的名称，为Android 4.4（代号 KitKat 奇巧）。据悉，该代号来自雀巢的KitKat巧克力。对于KitKat 4.4.3，screencast不仅可以在Tab网页实现，在WebView上也可以同步交互操作。

点击 DevTools右上角的Screencast图标，可以开启Screencast视图，移动设备屏幕会实时展现在左侧的Screencast面板中。

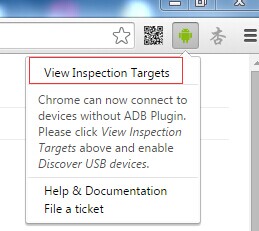


Chrome32+版本的DevTools对连接的设备支持native USB debugging。而版本号稍低的一些则需要通过安装ADB或ADB plugin调试移动设备上的Chrome网页。由于之前一直是使用ADB plugin方式调试，如今一番折腾发现高版本更为方便实用的特性，由于网上的资料比较零散，关键的东西都没有提及，坑也不少，这也正是写此文的缘由。

曾经靠安装Android SDK套件，需要执行类似 adb forward tcp:9001 localabstract:chrome\_devtools\_remote 这样的命令，才能开始调试，现在我们完可以摆脱这些繁琐的命令了。

开发机器安装ADB plugin是Chrome 28+之后的方式，区别于以前需要安装整个Android SDK套件，同时移动设备要安装Chrome for Android 28+。ADB扩展程序安装成功后，Chrome菜单旁边会出现一个灰色的Andr​​oid启动图标。

点击Android的图标，然后单击ADB开始。一旦ADB已经开始，菜单图标变成绿色，并显示当前连接的设备的数量，如果有连接设备的话。



点击View inspection Targets打开：列表中会显示每个连接的设备及其选项卡页面。查找要调试的页面，并点击页面链接URL旁边的inspect启动DevTools调试窗口。

**使用用DevTools特别重要的一点是：如果你点击inspect打开的DevTools窗口一片空白，且刷新无效时，那极有可能是由于被墙的缘故，你可以尝试appspot.com是否可以ping的通，如果无法ping通，那你现在就先翻墙吧**（goagent 或红杏都不错，将appspot.com加入白名单），当然你还需要有google账户。

补充说明一下，**如果接上USB数据线后，没有显示任何连接的设备，请按照以下步骤排查：**

● 检查您的设备连接到USB，检查USB数据线。

● 确保您的设备发出ADB设备命令列为可用。如果没有，检查是否有您的设备上是否启用USB调试。

● 若是Windows系统，检查你的USB驱动是否安装正确。参考<http://developer.android.com/tools/extras/oem-usb.html>

● 桌面Chrome浏览器打开 chrome://inspect检查Discover USB devices是否选中

● 确保桌面浏览器版本要高于移动设备的Chrome版本号

● 如果Chrome for Android版本较低，请检查移动设备上Chrome浏览器设置，确保设置中启用USB调试。高版本中无此设置项，无须设置。

● 如果依然无法显示，尝试重新拔插USB数据线。

**端口转发（Port forwarding）**

你的手机和开发机器有时会处于两个不同的网络（如本地服务器和线上服务器两个不同的网络环境），手机上可能获取不到开发环境的页面内容。况且，有时候你的开发环境处于公司出于安全原因管控限制下的网络中。

Chrome For Android中的端口转发解决了这一困难，在手机上测试你开发的网站瞬间变得容易。它的工作原理是在移动设备上创建一个监听TCP端口，该端口映射到开发机器特定的TCP端口，两个端口通过USB线路通信，所以这种连接并不依赖于所处网络环境的配置。

在进入正题前必须要提前说明搭建本地服务器需要注意的一些点。安装本地服务器，这里安装的是wamp集成包。一键安装完成后，需要修改Apache的一些设置。

**1. 打开Apache安装目录下的配置文件httpd.conf增加监听多个端口。**

**[plain]** [view plaincopy](http://blog.csdn.net/freshlover/article/details/42528643)[在CODE上查看代码片](https://code.csdn.net/snippets/575398)

1. # Listen: Allows you to bindApache to specific IP addresses and/or
2. # ports, instead of the default. See also the <VirtualHost>
3. # directive.
4. #
5. # Change this to Listen on specific IP addresses as shown below to
6. # prevent Apache from glomming onto all bound IP addresses.
7. #
8. #Listen 12.34.56.78:80
9. Listen 80
10. Listen 8888
12. Listen 8999
13. Listen 1818

开启Apache服务后可通过命令netstat –n –a查看所配置的端口是否开启。

如果你本地环境也开启了IIS服务，为了避免二者都是用80端口发生冲突，可以按照上述方法修改Apache的默认端口。修改设置完成后别忘了重启Apache服务。

如何改变IIS端口：若保留Apache服务与IIS服务同时运行，不想改变Apache默认80端口，那么只能改变IIS端口，方法如下：开始->运行->输入:inetmgr->Internet信息服务->本地计算机->网站->右键“默认网站”->属性->选择“网站”标签->修改TCP端口即可。最后重新启动IIS服务。

**2. 在httpd.conf文件中开启虚拟主机配置**

**[plain]** [view plaincopy](http://blog.csdn.net/freshlover/article/details/42528643)[在CODE上查看代码片](https://code.csdn.net/snippets/575398)

1. # Virtual hosts
3. #Includeconf/extra/httpd-vhosts.conf

将第二行前的#号去掉，即是为了允许加载扩展配置。

1.  Apache服务器默认的路径为安装目录下的htdocs，如果想改变默认目录，可以修改httpd.conf文件下的DocumentRoot和Directory，修改两者路径到你指定的目录即可。

**[plain]** [view plaincopy](http://blog.csdn.net/freshlover/article/details/42528643)[在CODE上查看代码片](https://code.csdn.net/snippets/575398)

1. #
3. # DocumentRoot: The directoryout of which you will serve your
5. # documents. By default, allrequests are taken from this directory, but
7. # symbolic links and aliasesmay be used to point to other locations.
9. #
11. DocumentRoot "E:/wamp/www/"


15. #
17. # This should be changed towhatever you set DocumentRoot to.
19. #
21. <Directory "E:/wamp/www/">

打开extra目录下的httpd-vhosts.conf配置文件，在文件最后位置分别为监听的端口设置虚拟主机目录

**[plain]** [view plaincopy](http://blog.csdn.net/freshlover/article/details/42528643)[在CODE上查看代码片](https://code.csdn.net/snippets/575398)

1. #
3. # Use name-based virtual hosting.
5. #
7. NameVirtualHost \*:80


11. #
13. # VirtualHost example:
15. # Almost any Apache directive may go into a VirtualHostcontainer.
17. # The first VirtualHost section is used for all requests that donot
19. # match a ServerName or ServerAlias in any <VirtualHost>block.
21. #
23. <VirtualHost \*:80>
25. ServerAdminwebmaster@dummy-host.example.com
27. DocumentRoot "E:/wamp/www /dummy-host.example.com"
29. ServerName dummy-host.example.com
31. ServerAliaswww.dummy-host.example.com
33. ErrorLog"logs/dummy-host.example.com-error.log"
35. CustomLog"logs/dummy-host.example.com-access.log" common
37. </VirtualHost>


41. <VirtualHost \*:80>
43. ServerAdminwebmaster@dummy-host2.example.com
45. DocumentRoot"E:/wamp/www"
47. ServerName localhost
49. ErrorLog"logs/dummy-host2.example.com-error.log"
51. CustomLog"logs/dummy-host2.example.com-access.log" common
53. </VirtualHost>


57. <VirtualHost \*:8888>
59. ServerAdminprograms@dummy-host2.example.com
61. DocumentRoot"E:/wamp/www/programs"
63. ServerName localhost
65. ErrorLog"logs/programs-host2.example.com-error.log"
67. CustomLog"logs/programs-host2.example.com-access.log" common
69. </VirtualHost>


73. <VirtualHost \*:8999>
75. ServerAdminprograms@dummy-host2.example.com
77. DocumentRoot"E:/wamp/www/site"
79. ServerName localhost
81. ErrorLog"logs/programs-host2.example.com-error.log"
83. CustomLog"logs/programs-host2.example.com-access.log" common
85. </VirtualHost>

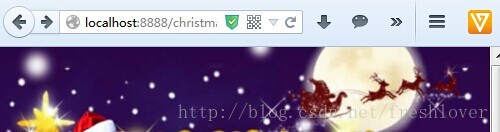

89. <VirtualHost \*:1818>
91. ServerAdmin webapp@dummy-host2.example.com
93. DocumentRoot"E:/wamp/www/webapp"
95. ServerName a.test.com
97. ErrorLog"logs/webapp-host2.example.com-error.log"
99. CustomLog"logs/webapp-host2.example.com-access.log" common
101. </VirtualHost>

其中a.test.com若要生效还需要在本地host文件中增加映射关系：127.0.0.1 a.test.com

然后我们测试一下，在浏览器输入你配置的主机名和端口号，url指向你要打开的文件。



若出现上述提示，检查你的虚拟主机目录是否正确。如果目录设置正确，即可打开页面。



一般情况下目录访问权限默认为：

**[plain]** [view plaincopy](http://blog.csdn.net/freshlover/article/details/42528643)[在CODE上查看代码片](https://code.csdn.net/snippets/575398)

1. #
3. # First, we configure the"default" to be a very restrictive set of
5. # features.
7. #
9. <Directory />
11. Options FollowSymLinks
13. AllowOverride All
15. Order deny,allow
17. Deny from all
19. </Directory>

许多人为了方便直接设置为

**[plain]** [view plaincopy](http://blog.csdn.net/freshlover/article/details/42528643)[在CODE上查看代码片](https://code.csdn.net/snippets/575398)

1. #
3. # First, we configure the"default" to be a very restrictive set of
5. # features.
7. #
9. <Directory />
11. Options FollowSymLinks
13. AllowOverride All
15. Order allow ,deny
17. Allow from all
19. </Directory>

出于安全考虑，根据调试的人员和实际需要，我个人倾向于为特定目录设置不同的权限，示例如下：

**[plain]** [view plaincopy](http://blog.csdn.net/freshlover/article/details/42528643)[在CODE上查看代码片](https://code.csdn.net/snippets/575398)

1. #
3. # First, we configure the"default" to be a very restrictive set of
5. # features.
7. #
9. <Directory />
11. Options FollowSymLinks
13. AllowOverride All
15. Order deny,allow
17. Deny from all
19. </Directory>


23. <Directory "E:/wamp/www/webapp">
25. Options FollowSymLinks
27. AllowOverride All
29. Order deny,allow
31. Deny from all
33. Allow from 192.8
35. </Directory>


39. <Directory "E:/wamp/www/programs">
41. Options -Indexes FollowSymLinks
43. AllowOverride All
45. Order deny,allow
47. Deny from all
49. Allow from 192.8.102. 192.8.104. 192.8.22.
51. </Directory>

设置完成后重启Apache服务生效。

安全是安全了，不过有一个问题，访问localhost下的页面会出现这样的情形：



其实我们再增加一句 Allow from127.0.0.1 ::1 localhost 即可解决此问题。如下：

**[plain]** [view plaincopy](http://blog.csdn.net/freshlover/article/details/42528643)[在CODE上查看代码片](https://code.csdn.net/snippets/575398)

1. <Directory "E:/wamp/www/programs">
3. Options -Indexes FollowSymLinks
5. AllowOverride All
7. Order deny,allow
9. Deny from all
11. Allow from 192.8.102. 192.8.104. 192.8.22.
13. Allow from 127.0.0.1 ::1 localhost
15. </Directory>

重启再试，页面正常显示出来了。如下图：

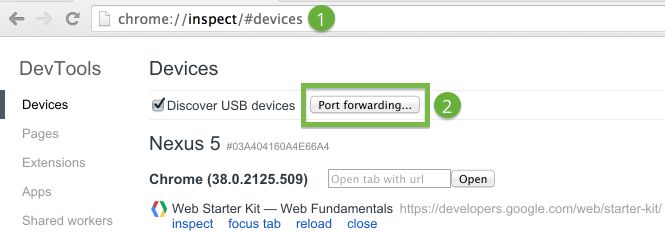


现在搭建好了本地服务器，开启了若干监听端口，进入正题，接着说端口转发。

**要使用端口转发（Port forwarding）功能，需要满足以下条件：**

1.      在开发机器的Chrome浏览器打开chrome://inspect

2.      点击PortForwarding，弹出设置窗口

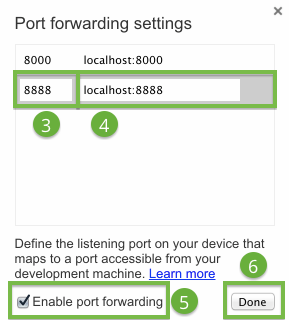


3.      在设备端口输入框，填写移动设备要监听的端口号（默认为8080）

4.      在Host主机输入域，填写运行web应用的当前ip地址（或主机名称，如localhost）加端口号。IP地址可以填写开发机器可访问的任何本地地址。当前，端口号范围必须在1024~65535之间（包括）

5.      选中Enableport forwarding.

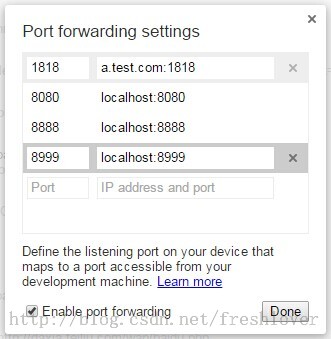
6.      点击Done完成。



当chrome://inspect窗口的端口号闪动为绿色时，表明该端口转发配置已生效。此时你可以在移动设备上打开本地页面进行调试了。

在Chrome29之前端口转发功能有限，如果遇到问题，请确保你的Chrome版本升级到29以上再使用。在Chrome31+端口转发已经不是试验性功能了，而是正式发布的功能了。

前面我们已经搭建好本地服务器环境。接上USB线，打开chrome://inspect/#devices，根据Apache所设的监听端口，设置相应DevTools的端口转发参数，如下图：



完成了端口转发的设置后，这时候我们就可以进行调试本地环境下的页面了。

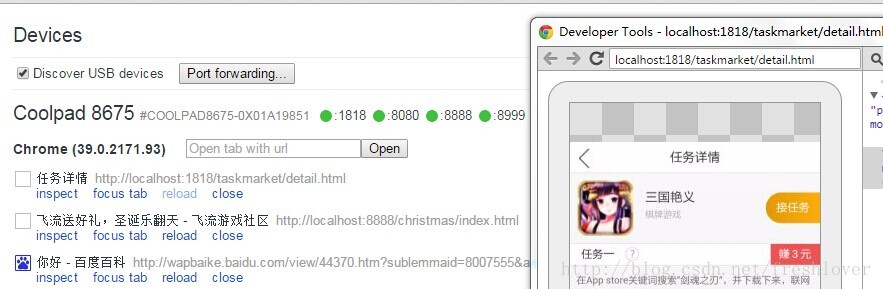
输入url，打开本地的页面。

例如

[http://localhost:1818/taskmarket/index.html](http://a.test.com:1818/taskmarket/index.html)

<http://localhost:8888/christmas/index.html>

如下图所示：



**虚拟主机映射（Virtual hostmapping）**

当你在localhost域名（或你的本地开发机器IP）下调试，端口转发很灵验。不过，如果你要使用自己定制的本地域名来调试那就不管用了。

如果你在使用一个仅能运行于特定白名单中的域名下的JavaScriptSDK，因此你在host文件中设置了如127.0.0.1 xxx.com这样的对应关系。或许你在Web服务器（MAMP，wamp，xampp等）用虚拟主机配置了个性的域名。

若是想让移动设备显示特定域名下页面内容，你需要使用上述的端口转发技术和代理服务器技术来实现。代理服务器可以将移动设备上的请求映射到主机的正确位置。

**搭建代理**

目前有许多搭建代理服务器的工具，比如Mac上的Charles Proxy，在Windows系统下的Fiddler，Linux下的Squid cache，Squid作为一款开源的代理服务软件，也可以用作Web缓存服务器，实现高速的Web访问需求。主机上安装运行代理服务器，所有来自安卓移动设备的请求都将会转发到代理服务器。

代理服务的搭建方法这里不细说看，感兴趣的可以参考我的文章或自行查找。

**端口转发代理设置的步骤：**

1.  安装代理服务软件，在主机上搭建代理服务器。

2.  开启代理服务，并确保代理服务器运行的端口与Chrome端口转发所设端口不相同。

3.  Chrome浏览器打开“检查设备”chrome://inspect

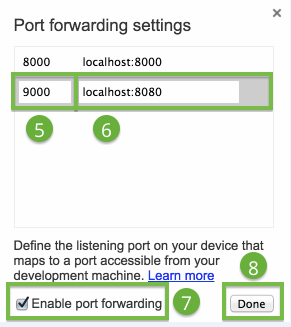
4.  点击Port forwarding 弹出设置窗口。

5.  设备端口处填上安卓设备要监听的端口，如9000。

6.  主机IP处可以填写localhost:xxxx，比如localhost:9000，也可以填写我上面配置过的本地域名a.test.com:1818

7.  勾选 Enableport forwarding

8.  点击Done按钮完成参数配置。



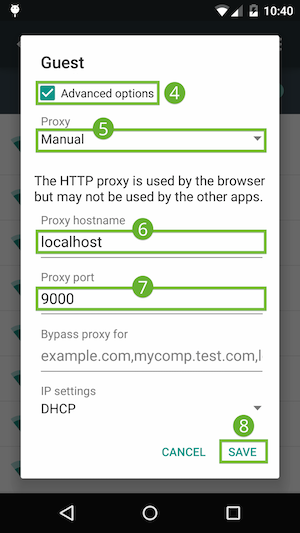
**移动设备代理服务参数设置：**

1.  打开设置-WLAN（**Settings> Wi-Fi**）

2.  长按当前连接的无线网络（代理服务设置适用于每个无线网络）

3.  点击修改网络（**Modify network**）

4.  勾选高级选项（**Advanced options**），将显示设置项：



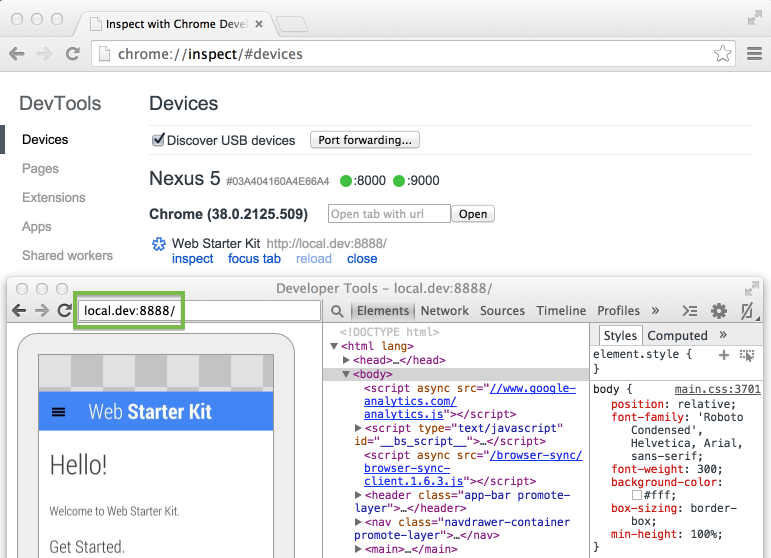
5.  点击“代理”（**Proxy**）菜单选择“手动”（**Manual**）

6.  在“代理服务器主机名”（ **Proxyhostname**）处输入localhost或者127.0.0.1

7.  在“代理服务器端口”（**Proxy port**）处输入端口号，如9000

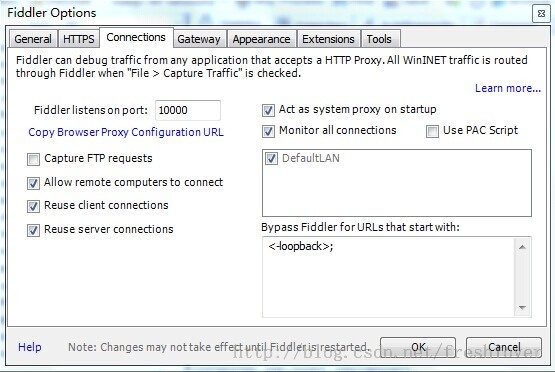
8.  点击保存（Save）。

通过以上参数设置，移动设备上所有请求被转发到主机的代理服务器，代理服务器代表安卓移动设备来发送请求，使得发送到特定域名下的请求得到了合理的解析。现在你可以调试特定域名下的页面了。



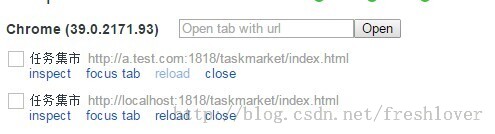
注意：为了不影响移动设备正常上网浏览，断开USB数据线后一定要恢复代理设置。

Fiddler默认使用8888端口，我的本地服务器已经占用8888端口，所以在Fiddler-Tools菜单中将监听端口设为未使用的10000端口，如图：

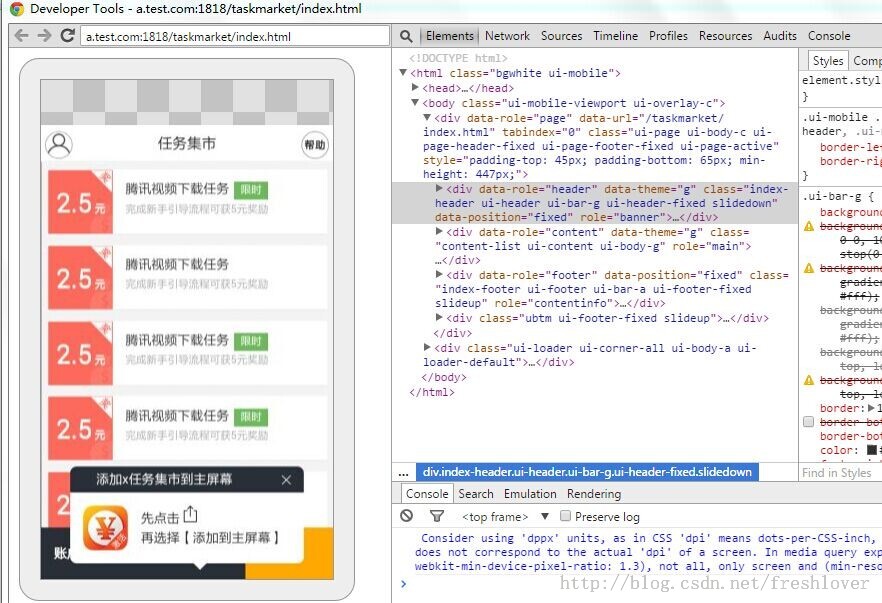


手机端代理设置参数完毕后，打开ChromeDevTools进行调试。点击inspect可以看到页面显示在手机浏览器了，现在开始调试吧。

我的测试页面为<http://a.test.com:1818/taskmarket/index.html>

[[](http://a.test.com:1818/taskmarket/index.html)](http://a.test.com:1818/taskmarket/index.html)

DevTools实例的调试截图如下：



手机端Chrome for Android中的实时截图如下：



注意主机要和移动设备处在同一网段的局域网内。

有问题的童鞋可以留言或查看我博客的其他相关文章。 Chrome远程调试部分就到这里了。移动端的调试还没有结束，下一节我会介绍Weinre远程调试的技巧和详细教程。

# 第三种方法

参考地址：<http://www.codingserf.com/index.php/2015/03/browsersync/>

**贡献者 [杭州] [重庆]Jacky**

# [移动端真机调试终极利器-BrowserSync](http://www.codingserf.com/index.php/2015/03/browsersync/)

[Web](http://www.codingserf.com/index.php/category/1/)    [星期五，3月27日，2015](http://www.codingserf.com/index.php/2015/03/browsersync/)

之前有分享过一篇《[如何在移动设备上调试网页](http://www.codingserf.com/index.php/2014/05/debug-on-devices/)》，感谢 xyyjk 留言推荐[BrowserSync](http://www.browsersync.io/)这个工具，自己也花了点时间研究了一下，还是很好上手的。比起之前介绍的方法更加方便（之前的方法已经过时）。BrowserSync可以实时同步更新CSS、JS文件，此外最关键的是全平台支持，即你可以在手机QQ浏览器、微信浏览器里面调试。今天就介绍一下这个工具的用法（以下针对Mac OSX）。

## 一、安装

其实官网已经有很好的 [Get Started](http://www.browsersync.io/#install)来介绍如何安装。我现在是结合[Gulp](http://gulpjs.com/)来用的，在Gulp下更方便一点。官方也有BrowserSync for Gulp的插件，并有明确文档说明[如何与Gulp整合](http://www.browsersync.io/docs/gulp/)。

#### 1、安装开发依赖

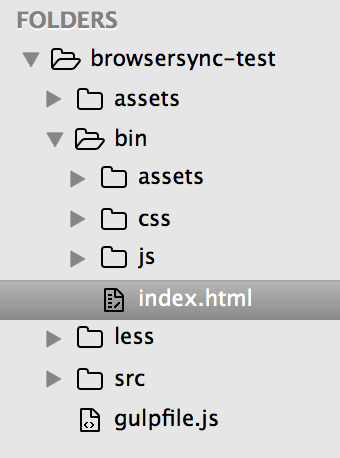
在终端里输入：

$ npm install browser-sync gulp --save-dev

如果安装报错请加sudo：

$ sudo npm install browser-sync gulp --save-dev

#### 2、编辑gulpfile.js

如果项目下没有gulpfile.js文件请新建。先来看一下项目的目录结构，bin文件夹是最终发布的目录，我们把它当作BrowserSync代理服务器启动的根目录：  
  
gulpfile.js文件代码：

**var** gulp = require('gulp');

**var** browserSync = require('browser-sync');

// Static server

gulp.task('browser-sync', **function**() {

browserSync({

server: {

//指定服务器启动根目录

baseDir: "./bin"

}

});

//监听任何文件变化，实时刷新页面

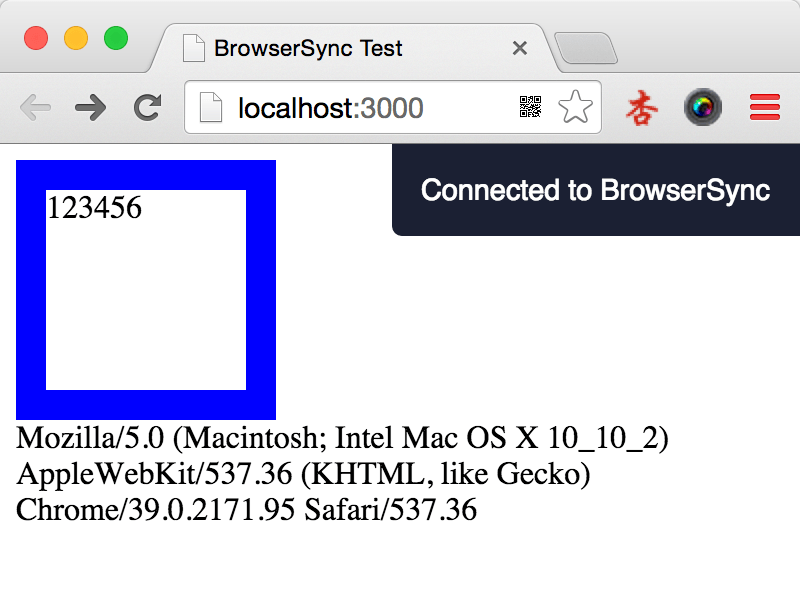
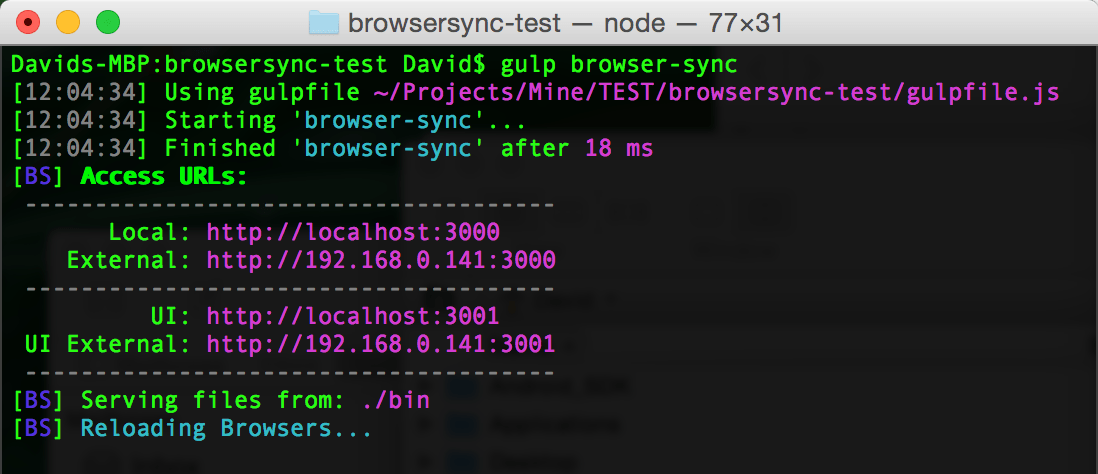
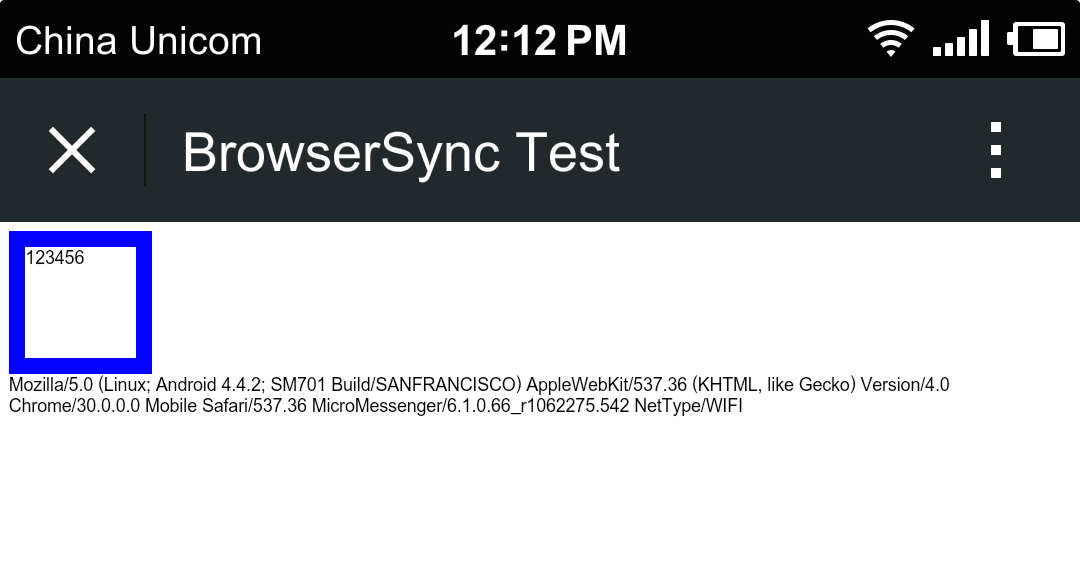
gulp.watch("./bin/\*\*/\*.\*").on('change', browserSync.reload);

});

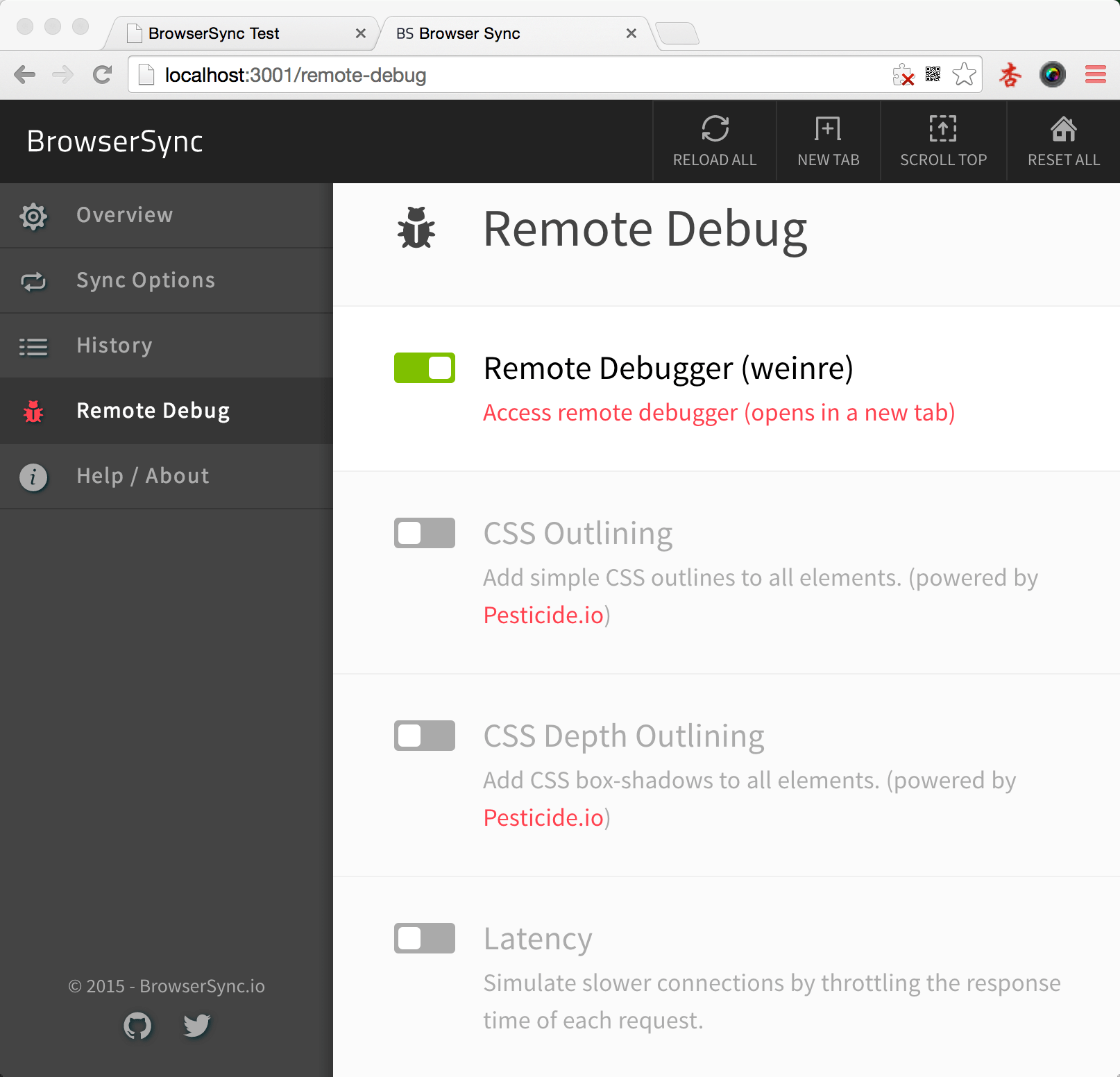
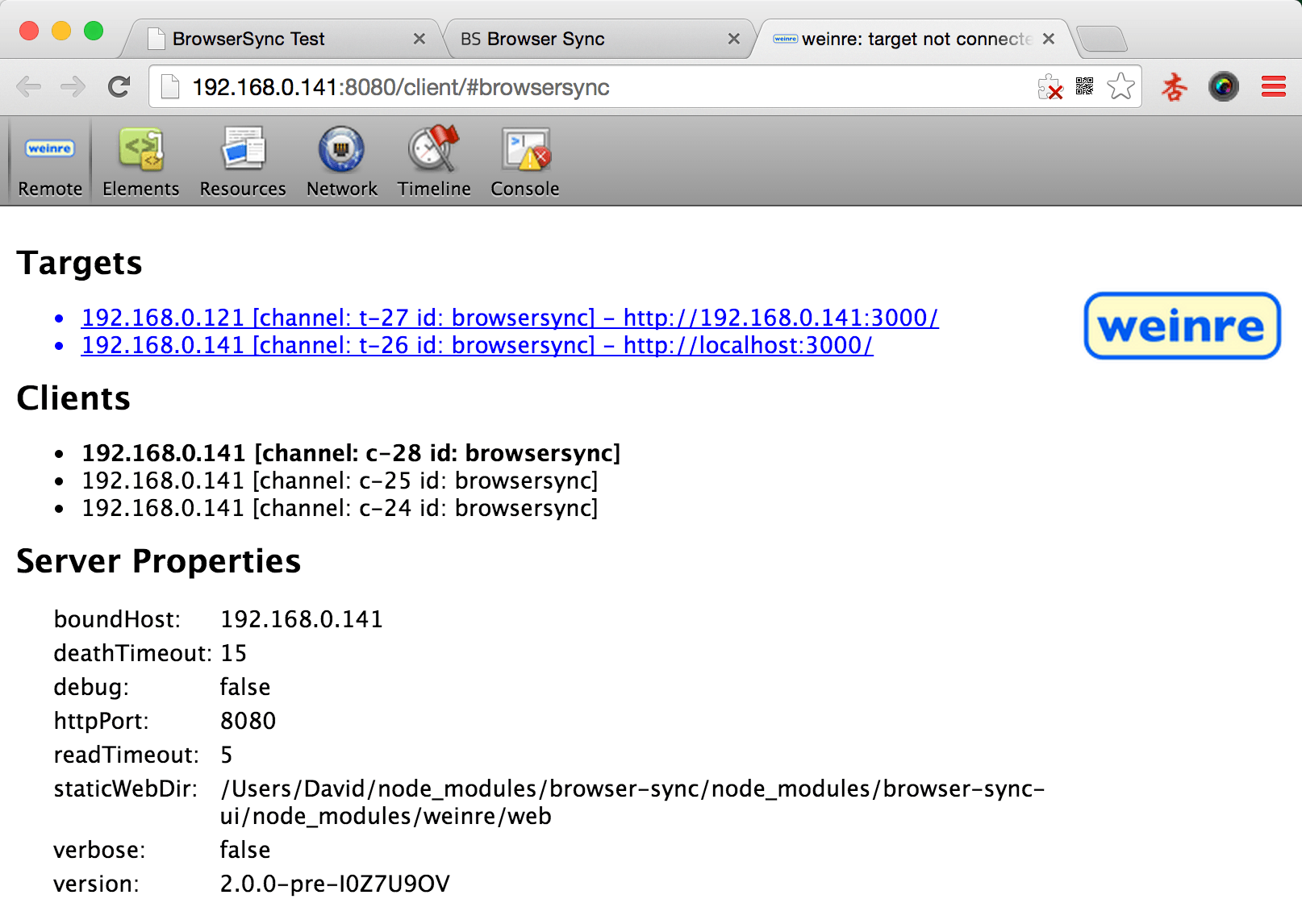
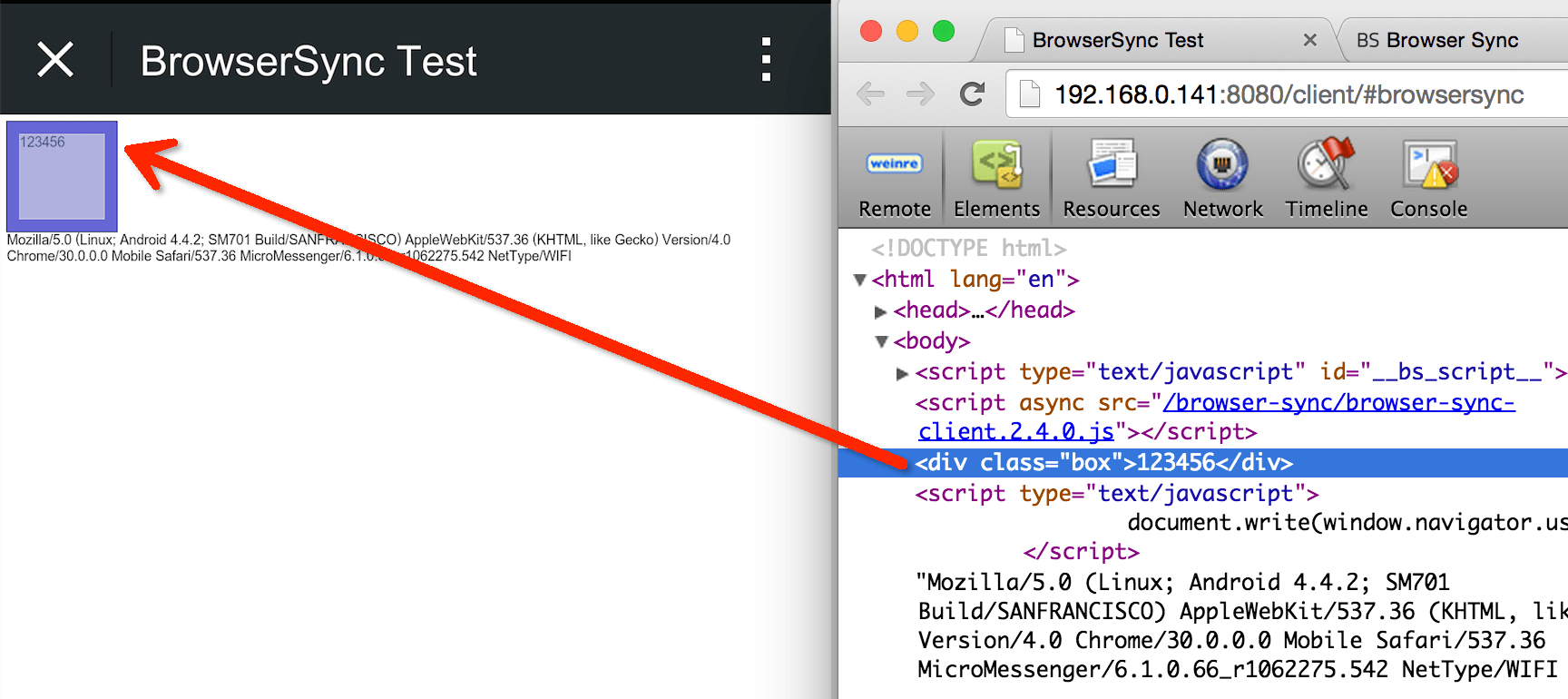
## 二、启动

打开终端输入：

$ gulp browser-sync

此时浏览器会自动打开http://localhost:3000的页面，我们会发现这个页面正是指向./bin目录下的index.html，这与我们在gulpfile.js中的设置相符。我们在index.html中有一个蓝色边框的box，并输出浏览器的UA：  
  
BrowserSync启动后终端界面会有两个端口提示，分别是:3000的项目页面，和:3001的BrowserSync的UI界面，并且每个端口都有供本地（localhost）和外部（局域网IP）访问的URL：  
  
因为我们把BrowserSync的reload加到了gulp的watch里，所以只要./bin下面的文件发生变化所有访问local URL或External URL的客户端（client）都会自动刷新。  
接下来我们把手机和电脑要连接在同一个无线局域网里，我们可以直接通过微信扫码访问External的URL，如本例中的http://192.168.0.141:3000。此时手机显示如下：  


## 三、调试

我们在PC的浏览器中打开http://localhost:3001，这个端口为3001的URL是BrowserSync的操作界面：  
  
界面操作简洁易懂。我们着重关注的是Remote Debug这一项，默认它的所有选项都是关闭的。我们开启Remote Debugger (weinre)这一项，然后点击出现的红色字：Access remote debugger (opens in a new tab)，就会打开weinre的控制台界面：  
  
这个界面上会列出当前连接在BrowserSync上的客户端，我们可以选择一个目标（target）来调试，因为我们的微信浏览器访问的是外部链接，即http://192.168.0.141:3000，所以我们点击图中Targets下的第一个link。点击后link会变为绿色，表示当前已经可以对这个客户端进行调试了。此时我们点到顶端的Elements选项卡上，就会看到微信浏览器里网页的HTML结构。鼠标移动到.box的div上的时候，微信浏览器里的相应的元素就会高亮起来：  
  
此时能做的事情就可想而知了。  
需要注意的是，要先打开自己的项目页面，再打开Remote Debugger，这样才能列出当前已经连接的客户端。否则的话，即使项目页面和BrowserSync是连接状态，Remote Debugger也会出现捕捉不到的情况。因此，可能需要关闭再重新开启Remote Debugger进行调试。