Far fronte agli eventi… in JavaScript - Parte 3

Con l'arrivo degli smartphone i siti assomigliano sempre più alle app per smartphone e tablet. Già oggi i visitatori da smartphone trovano strano che uno *slideshow* non si possa sfogliare con un dito o che una foto non si ingrandisca con due. Su computer, invece, non è strano ritrovarsi a sfogliare le immagini col mouse, mentre invece queste restano ferme! I mondi *desktop* e *mobile* ormai si intersecano, tanto che sono nati i monitor touchscreen *multitouch* per computer desktop. Se però non ne avete uno, per sviluppare una webapp multitouch vi serve un modo per testarla come si deve.

Il codice riportato in questo articolo è compatibile con tutti i principali browser tranne quelli molto vecchi, come Internet Explorer 8 e precedenti, per i quali esistono semplici fix facilmente reperibili.

Testare su dispositivi mobile

Per sviluppare siti o webapp che fanno uso di eventi multitouch è d'obbligo testare frequentemente su dispositivi mobile. Ci si imbatte subito però in un problema non da poco: i browser mobile non hanno la console. Fortunatamente è possibile usare la console di un browser che gira su un computer desktop per testare ciò che si vede su quello che gira su uno smartphone (o tablet) connettendo i due dispositivi con un cavo usb. Ecco come fare nei casi più frequenti.

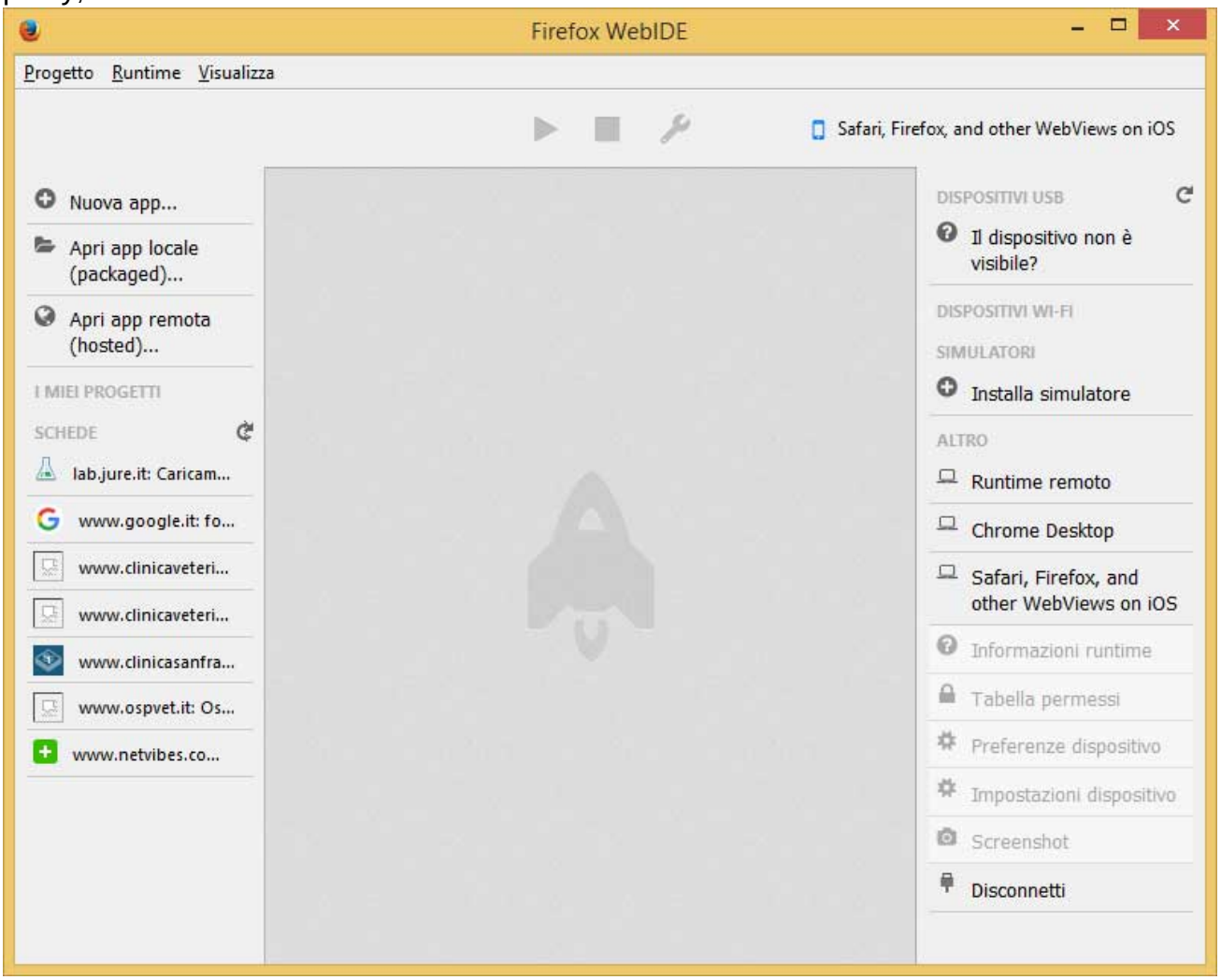
Safari OSX > Safari iOS

1. Su Safari iOS attivare "*Impostazioni > Safari > Avanzate > Inspector web*"
2. Su Safari OSX, assicurarsi che la voce *Sviluppo* sia attivata selezionandola in "*Safari > Impostazioni > Avanzate*"
3. Su Safari OSX andate in "*Sviluppo > [nome dispositivo]*"

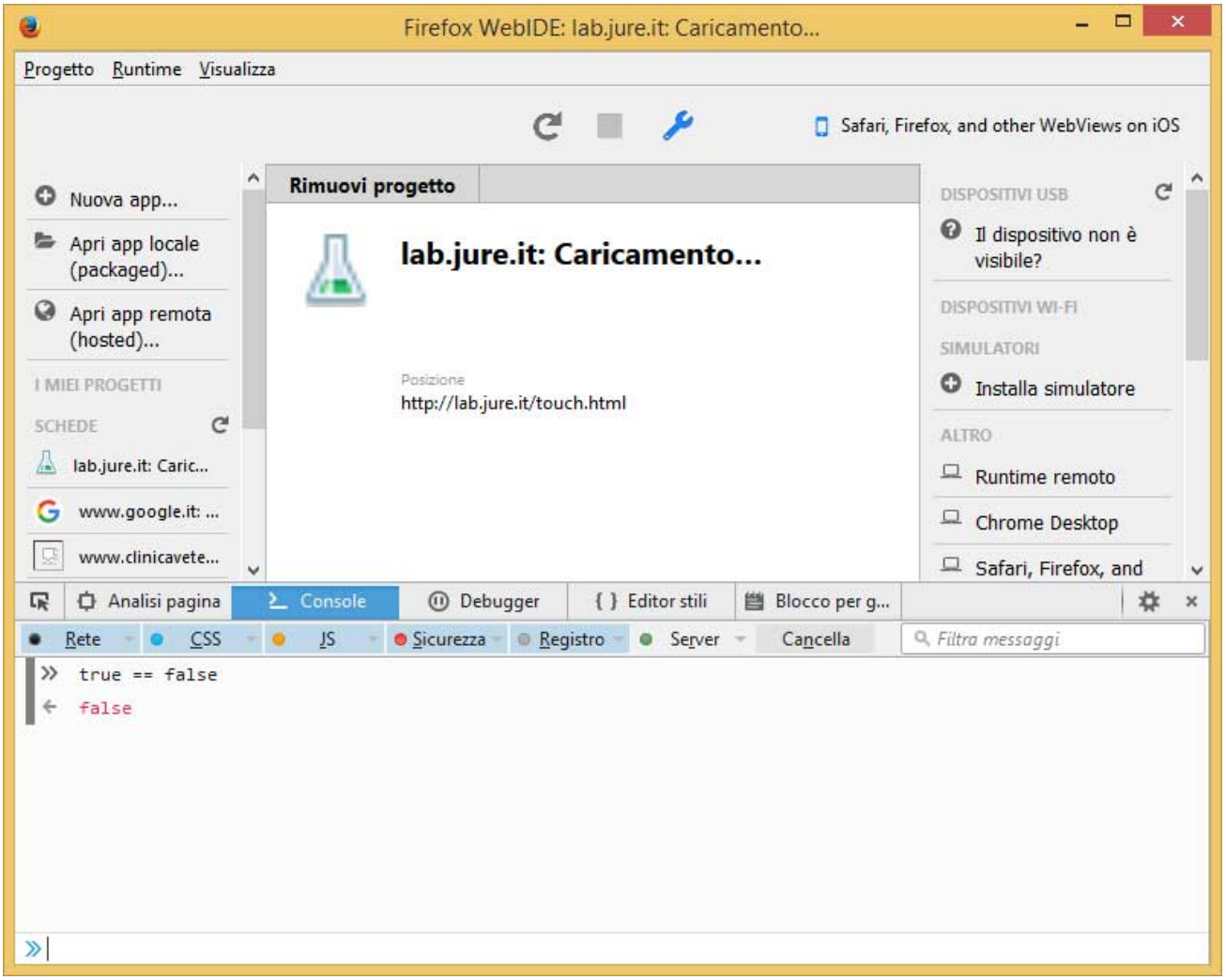
Firefox Windows > Safari iOS

1. Installare l'ultima versione di iTunes
2. Su iOS, attivare "*Impostazioni > Safari > Avanzate > Inspector web*"
3. Su Windows scaricare ios-webkit-debug-proxy-win32 e avviare *ios_webkit_debug_proxy.exe* accettando di aggiungere un'eccezione al firewall: si aprirà una finestra di DOS che va lasciata aperta mentre si lavora
4. Su Firefox Windows aprire *WebIDE* premendo "SHIFT + F8" oppure selezionando "*Sviluppo > WebIDE*" dal menù.
5. Nella colonna di destra, cliccare su "*Safari, Firefox, and other WebViews on iOS*". Questa funzione non è ancora perfetta e a volte per riuscire a connettermi devo riprovare più volte, aspettare diversi minuti, scollegare e ricollegare il cavo usb, chiudere e riavviare il debug

proxy, ma alla fine ce la si fa.

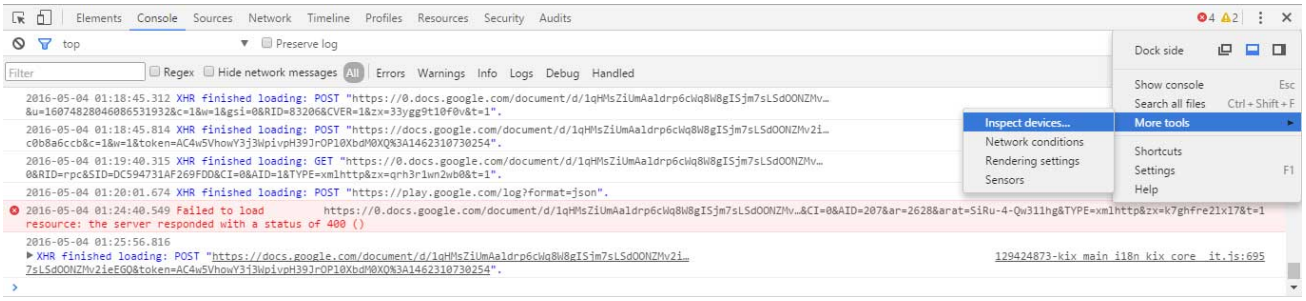


6. Nella colonna di sinistra vengono mostrate le tab di Safari iOS (soltanto quelle che avete portato in primo piano da quando vi siete connessi): cliccare sulla tab desiderata

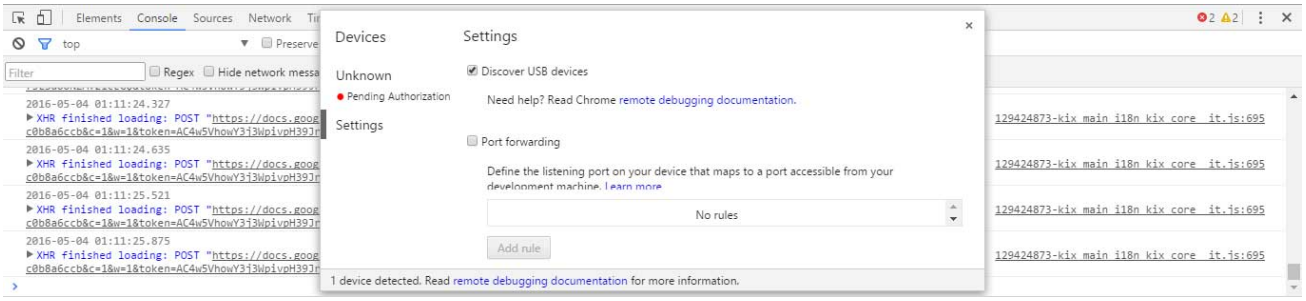


Chrome Windows/OSX > Chrome Android

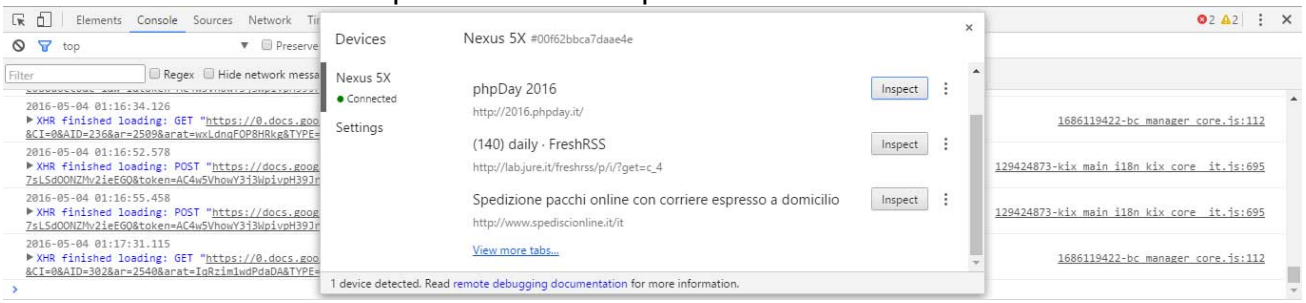
1. Su Android, verificare che esista la voce “*Impostazioni > Opzioni sviluppatore*”. Se non esiste, abilitarla andando su “*Impostazioni > Info sul dispositivo*” e cliccando ripetutamente su “*Numero build*” fino a veder apparire il messaggio “Ora sei uno sviluppatore!”
2. Abilitare “*Impostazioni > Opzioni sviluppatore > Debug USB*”
3. Su Chrome Windows/OSX, aprire la console premendo [F12] oppure selezionando “*Menù > Altri strumenti > Strumenti per sviluppatori*”
4. Individuare l’icona con tre pallini in verticale (vicino alla “X” per chiudere la console) e cliccare su di essa: si aprirà un menù dal quale bisogna selezionare “*More tools > Inspect devices...*”



5. Si aprirà una finestra che sulla colonna destra "*Device*" mostrerà "*Unknown - Pending Authorization*"



6. Su Android accettare la richiesta di debug
7. Su Chrome per Windows, cliccare il nome del dispositivo nella colonna destra "*Device*": nella colonna di destra compariranno le tab aperte su Chrome Android



8. Cliccare sul tasto "*Inspect*" relativo alla tab desiderata

Altri casi

1. Per i più smanettoni è possibile utilizzare weinre

Finalmente possiamo iniziare!

Multitouch

Si parla di *multitouch* quando abbiamo contemporaneamente più eventi touch, quindi quando abbiamo più dita che toccano lo schermo.

Gli eventi touch

Gli eventi in questione sono quattro: *touchstart, touchmove, touchend e touchcancel*. Come logica, i primi tre sono paragonabili a *mousedown, mousemove* e *mouseup*. Il quarto, invece, è particolare, poiché *touchcancel* viene scatenato in casi come il passaggio ad un'altra app o il raggiungimento del limite massimo di tocchi supportati dal dispositivo.

Facendo un bel *console.log()* di uno di questi eventi ci accorgiamo che ci sono varie proprietà interessanti, ma quelle su cui ci concentreremo sono tre:

- *touches*
  Una lista di "Touch" per ogni punto di contatto che sta toccando la superficie
- *targetTouches*
  Una lista di "Touch" per ogni punto di contatto che sta toccando la superficie ed è iniziato sull'elemento che è il target dell'evento corrente
- *changedTouches*
  Una lista di "Touch" per ogni punto di contatto che ha contribuito all'evento. Per l'evento *touchstart* è la lista dei punti che sono stati aggiunti. Per l'evento *touchmove* è la lista dei punti che sono stati spostati. Per gli eventi *touchend* e *touchcancel* è la lista dei punti che sono stati rimossi.

Ecco uno dei "Touch" che compongono queste liste:

Potete notare 3 gruppi di coordinate, che sono relative all'angolo in alto a sinistra dello schermo del dispositivo (screen), della pagina html (page) e del *viewport* del browser (client).

Il viewport in pratica è quell'area interna alla finestra del browser dove viene renderizzata la parte visibile del documento HTML che abbiamo aperto; quando si parla di coordinate relative al "*viewport*" non viene contato quanto si è scrollato, a differenza di quanto accade invece con le coordinate relative alla "*page*".

Identifier è esattamente quello che sembra, cioè un ID numerico e univoco. Purtroppo, probabilmente a causa di un bug, Chrome per Android lo valorizza sempre a 0.

Force è il livello di pressione, e va da 0.0 a 1.0.

Esempio

Ora che sappiamo di cosa disponiamo, possiamo provare a realizzare una demo: toccando lo schermo faremo comparire dei cerchi che seguiranno i movimenti delle dita.

Purtroppo, a causa del bug "identifier" di cui sopra, l'esempio non funzionerà su Chrome per Android.

Innanzitutto, stiamo per usare *touchmove*, cioè un evento che verrà scatenato decine di volte al secondo: serve quindi un'ottimizzazione simile a quelle spiegate nella parte 2 di questo articolo.

La tecnica che useremo viene spesso chiamata **Game loop** perché viene adottata nei videogame, ma viene usata molto anche in applicazioni multimediali e interattive. In pratica, non eseguiremo gli eventi ad ogni input, bensì faremo partire un ciclo infinito e salveremo gli input in un oggetto, dimodoché gli input vengano processati al prossimo passaggio del ciclo.

Una cosa interessante che ho notato è che fare affidamento su *changedTouches*, pratica che dovrebbe ottimizzare le performance dello script, in realtà si è rivelato poco performante: usando sempre *touches* e calcolandomi le differenze "a mano" invece ho ottenuto un buon risultato.

Potete vedere l'esempio commentato qui.

Puoi scaricare il codice completo da qui.

Gesture

Sapendo tutto questo, è possibile calcolare qualsiasi tipo di gesture. Esistono, però, varie librerie già pronte che le gestiscono e per evitare di reinventare la ruota vi consiglio caldamente di usare Hammer.js, che gestisce perfettamente sia le *gesture* più comuni che quelle più complesse con poche righe di codice, e rimuove in automatico il famigerato delay di 300ms.

Coming soon

Nel prossimo articolo parleremo dei *Custom Events*: cosa sono, a cosa servono e quando possono essere utili. Perché gli eventi... non sono mai abbastanza!