Práctica extra

1. **Lea el texto a continuación y realice las siguiente actividades.**
2. **Divida la siguiente oración en bloques nominales y verbales, identificando sus nucleos:**
   Even if your site or app doesn't have this level of traffic, it's important to test web apps on multiple platforms, and to develop a strategy for prioritising which devices and browsers to test on.
3. **Traduzca la oración anterior al español.**
4. **Busque en el texto ejemplos de los siguientes elementos. Identifique página, párrafo y línea para facilitar la corrección.**
   a. **Bloque verbal:**
      i. Distintos verbos modales y auxiliares.
      ii. Verbos copulativos.
      iii. Adjetivos con función predicativa.
      iv. Adverbios modificando a verbos.
      v. 2 verbos no conjugados **to + infinitivo** con diferentes traducciones y traduzca el fragmento en que se encuentran.
      vi. 2 verbos no conjugados **-ing** con diferentes traducciones y traduzca el fragmento en que se encuentran.
   b. **Bloque nominal:**
      i. Sustantivos simples y derivados.
      ii. Sustantivos con flexión de número.
      iii. Sustantivos modificadores.
      iv. Sustantivos en caso posesivo.
      v. Adjetivos con función atributiva.
      vi. Adjetivos determinantes.
      vii. Adverbios modificando a adjetivos

Mobile web debugging

Summary

Debugging web applications on multiple platforms requires careful prioritisation, different from developing for desktop alone. This article explains techniques and tools available, with links to external resources.

Understanding hardware

Debugging for mobile can be more complex and demanding than for desktop, simply because mobile devices often have a complex variety of functions and inputs:

- Telephone

- Camera
- GPS
- Accelerometer
- Gyroscope
- Proximity detector
- Ambient light sensor
- Multi-touch, gestures and keyboards

CPU, memory and storage are far more constrained and variable on mobile devices than for desktops and laptops, and debugging needs to be done with the whole mobile stack in mind:

- network
- hardware
- browser software
- app software

Context

Two factors are characteristic of mobile web usage: variability and instability. Both can lead to a buggy user experience.

When dealing with bugs, it's also important to consider user context: environment, mood, posture and time of day. Different contexts have different performance requirements: watching video while lying on a sofa compared to quickly checking a map while on a crowded bus, for example. The variety of usage contexts on mobile is likely to expose more bugs than for a traditional desktop setup in which a person sits at a table using a keyboard and mouse, with a large display and a computer 'plumbed in' for power and network connectivity.

To get a sense of what people are actually consuming on the mobile web, take a look at HTTP Archive Mobile. This provides analysis of browsing data for Alexa's top one million sites.

Devices

Facebook data show that over 7000 different devices access their site every day. Even if your site or app doesn't have this level of traffic, it's important to test web apps on multiple platforms, and to develop a strategy for prioritising which devices and browsers to test on.

If you don't have access to many devices, there may be an open device lab near you, where you can use or even borrow devices for free: www.opendevicelab.com.

Mobile simulation in the browser

Before testing on mobile devices, it's often best to do as much design and development as possible on a development computer. With that in mind, a number of tools have been built to mimic the mobile environment in the browser.

Debugging responsive design

Responsive Web Design (RWD for short) is an approach that emphasises design which works well across a variety of platforms, in particular by using media queries, fluid grids (layouts) and fluid images.

For responsive design to work well, it is especially important to test sites and applications at different viewport sizes. Remy Sharp’s responsivepx makes it possible to adjust viewport width and height values, in order to find points at which responsive layouts 'break’. Developer Brad Frost recently produced the ish resizer, which emphasises that ‘breakpoints’ – size ranges for displaying particular layouts and content – should be based on content rather than fixed device dimensions.

For responsive typography, designers have found that body text should generally have between 45 and 75 characters per line, A very simple trick for testing layouts, suggested by Trent Walton, is to add an asterisk to _lorem_ text after 45 and 75 characters. If both asterisks appear on the same line at a particular viewport width, the font size needs to be changed.

A huge range of other tools is available for testing and debugging responsive design components including images, fonts and layouts.

Simulators and emulators

Simulators aim only to mimic the behaviour of a device. Emulators are designed to replicate the way a device actually works, in terms of hardware and software. For example, the Android emulator runs a full Android system stack down to the kernel level, whereas iOS provides a simulator. (There’s a long discussion on Stack Overflow on the difference.)

Simulators and emulators are available for many platforms, and browsers can be installed on most of these: mobilexweb.com/emulators. (It’s not included on this list, but a simulator is also available for the Kindle Fire.)

- Android emulator
- iOS simulator

The Opera Mobile simulator can run multiple Opera Mobile instances, each with its own settings (such as resolution and pixel density) and can be used in collaboration with the Opera Dragonfly development tools.

Remote debugging

Several tools enable developers to run a debugger user interface on one device in order to debug a web page running on another: for example, from your laptop debug a web page displayed on your phone.

Most tools for remote debugging require some setup; in particular, browser tools require port forwarding to be initiated from the command line.

WebKit remote debugging

The WebKit Web Inspector is implemented as an HTML + CSS + JavaScript web application that can run outside of the browser environment, and has been incorporated in other debugging tools. Chrome DevTools developer Pavel Feldman’s article about remote debugging provides a quick tutorial (albeit oriented to desktop Chromium) and discusses the Remote Debugging Protocol. Safari’s Web inspector uses the WebKit remote debugging protocol.

Safari on Mac OS X can be used for remote debugging of HTML, CSS and JavaScript running in Safari on iOS 6 (or later) or on a simulator. There's a short screencast at screenr.com/GYZ8.

iWebInspector

This tool, built by Maximiliano Firtman, can be useful for debugging apps running on Safari for iOS pre version 6.0 – can also be used with PhoneGap apps: iwebinspector.com.

Debugging performance issues

Performance bugs are a significant barrier to producing a successful site or web app.

For more information about improving mobile web performance, see Google's mobile web best practice documentation.

Memory

When considering performance problems, some bugs go unnoticed on desktop computers but become problematic on devices with more constrained memory and caching. Remember that users – even on mobile – are leaving web apps open for longer than they used to. In particular, beware of memory leaks when using event listeners or creating DOM elements.

The Chrome DevTools have several features for analysing memory usage. These tools also provide a timeline and Frame Mode for understanding the time taken to display each 'frame' in the browser – loading, scripting, rendering and painting – and how to maintain an acceptable 'frame rate'.

Performance checking

In order to avoid performance bugs, it is crucial to incorporate performance checking in your workflow. There are several tools for this.

Page Speed

developers.google.com/speed/pagespeed
Google tool to 'help developers optimize their web pages by applying web performance best practices'. A similar tool is available from the Audit panel in the Chrome DevTools.

Blaze

akamai.com/blaze
Akamai service for 'front end optimisation'.

W3C also offers a tool to check web pages for 'mobile friendliness': validator.w3.org/mobile.

Browser and device capability

It's important to ensure that your target platforms support the HTML, JavaScript and CSS features you use, and to plan sensible fallback strategies. Several sites provide detailed, up-to-date information about this.

Reporting browser bugs

It's extremely important for the developer community that you report browser bugs as soon as you encounter them, however frustrating or trivial they may seem.

And finally…

Whatever you build, try to test it on a variety of mobile devices in a variety of contexts. You don't need a complex test suite and usability lab for that: just get friends and colleagues to try out what you've built. The mobile web is changing fast, unpredictably – what matters now may not matter in six months – so make sure to incorporate testing early in your workflow.