Google
Developer
Day2009

Google Wave Client: Powered by GWT

Adam Schuck
5 June, 2009

Google
Developer
Day2009

Google Wave client

search

authentication

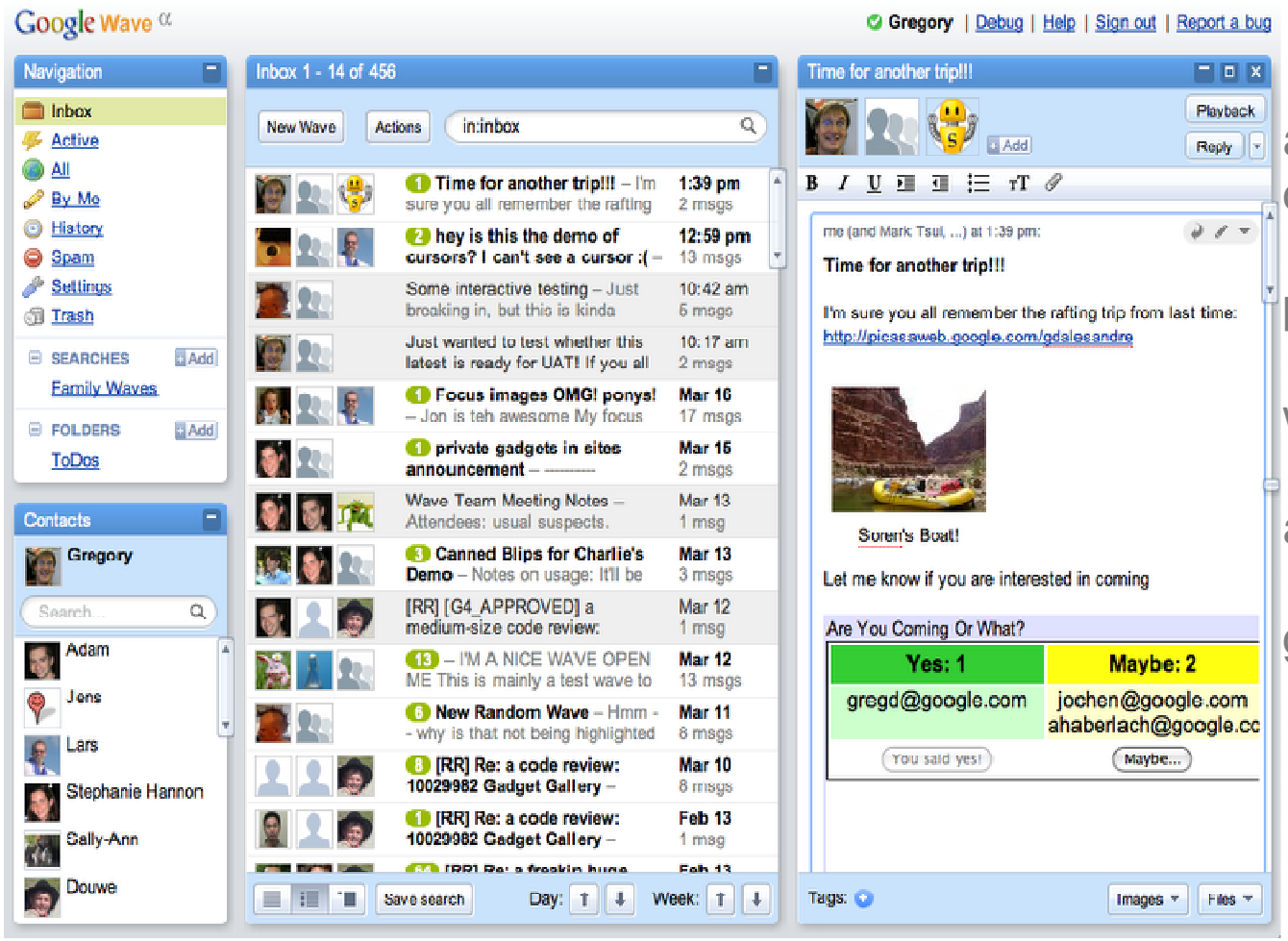
abuse detection

saved searches

folders

contacts

presence



access control

playback

waves

attachments

gadgets

Google Developer Day2009

Outline

- To GWT or not to GWT
- Client architecture
- Changes in GWT
- Improving Gears
- Performance
- Mobile client
- Testability
- UI testing with WebDriver

Wave UI Requirements

- fast!
- stunning!
- think beyond the browser
- optimistic

<Demo>

To GWT or not to GWT

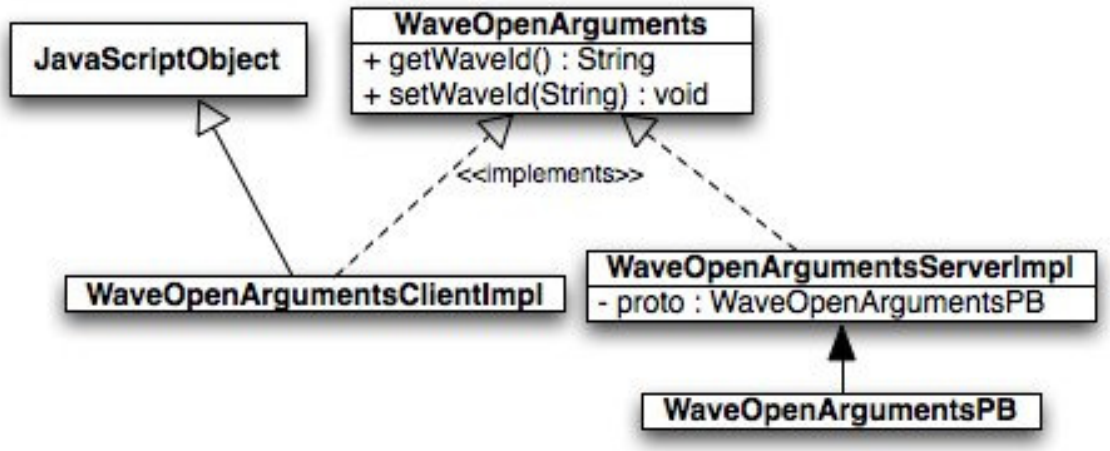
- What is GWT?
  - Java (compiled to JS)
    - use your favourite IDE (Eclipse, IntelliJ)
    - can share code between client + server
  - Deferred binding
  - JavaScript Native Interface (JSNI)

To GWT or not to GWT

- Prototype demoed late 2007
- Then: The No. 1 GWT Skeptic: me
- What changed my mind?
- Myth: "Can't build a real app!"
- Mindset: e.g. scrolly, panels

Client Architecture

- Bi-directional communication channel
- Protocol compiler
  - Generates interfaces, client + server implementations



- Concurrency Control stack
- Learn more: Wave APIs; Under the Hood

Supported Browsers

- FF3
- Safari
- Chrome

- In development:
  - IE7
  - Android
  - iPhone

Evolution of GWT

Google
Developer
Day2009

What was GWT missing, late 2007?

- GWT "areas for improvement" late 2007:
  - UI code cumbersome
  - Cross-browser CSS
  - JSON handling heavy-handed
  - Debugging environment != browser
  - Monolithic compile -> everything downloaded at start
  - Mapping from Java <-> JS unclear
  - Inefficiencies in compiler

I have to write how much code?

Issue: creating widgets is time-consuming and heavy handed

```
promptPanel = new DivPanel();
VerticalPanel panel = new VerticalPanel();
HTML heading = new HTML("Identification");
Label lblPrompt = new Label("Please identify me by:");
final RadioButton r1 = new RadioButton("identity",
    "my Google Wave account (" + uName + ")");
Image imgUser = new Image("images/" + uImage);
final RadioButton r2 = new RadioButton("identity",
    "the following name: ");
Image imgBlog = new Image("images/" + blog.getImage());
final TextBox t = new TextBox();
HorizontalPanel hPanel = new HorizontalPanel();
Button btnOk = new Button("OK");
Button btnCancel = new Button("Cancel");

...
```

I have to write how much code?

Solution: UiBinder (formerly declarative UI)

- Templates allow our UI designer to modify the UI directly!

```
<ui:UiBinder xmlns:ui='urn:ui.com.google.gwt.uibinder'>
  <div>
    Hello, <span ui:field='nameSpan'/>.
  </div>
</ui:UiBinder>
```

See: http://code.google.com/p/google-web-toolkit-incubator/wiki/UiBinder

But most cross-browser bugs are CSS!

Issue: GWT abstracts cross-browser JS quirks, but not CSS

Solution: StyleInjector + CssResource

- Provides:
  - Validation
  - Minification + Image Spriting
  - Allows modularization of CSS: download only when needed
  - Different CSS for different browsers (compile-time):

```
@if user.agent safari {
  \-webkit-border-radius: 5px;
}
```

See: http://code.google.com/p/google-web-toolkit/wiki/CssResource

Inefficient JSON handling

Issue: JSON handling inefficient, requires extra objects

Solution: JavaScriptObject (JSO)

- Subclass JavaScriptObject to create an "overlay type"
  - avoid using JSONObject: use JSO / StringBuffer

```
private native void setPayload(String val) /*-{
  this.payload = val;
}-*/;

private native String getPayload() /*-{
  return this.payload;
}-*/;
```

See: http://code.google.com/p/google-web-toolkit/wiki/OverlayTypes

Debugging in Eclipse rocks! - but...

Issue: each browser behaves slightly differently to hosted mode

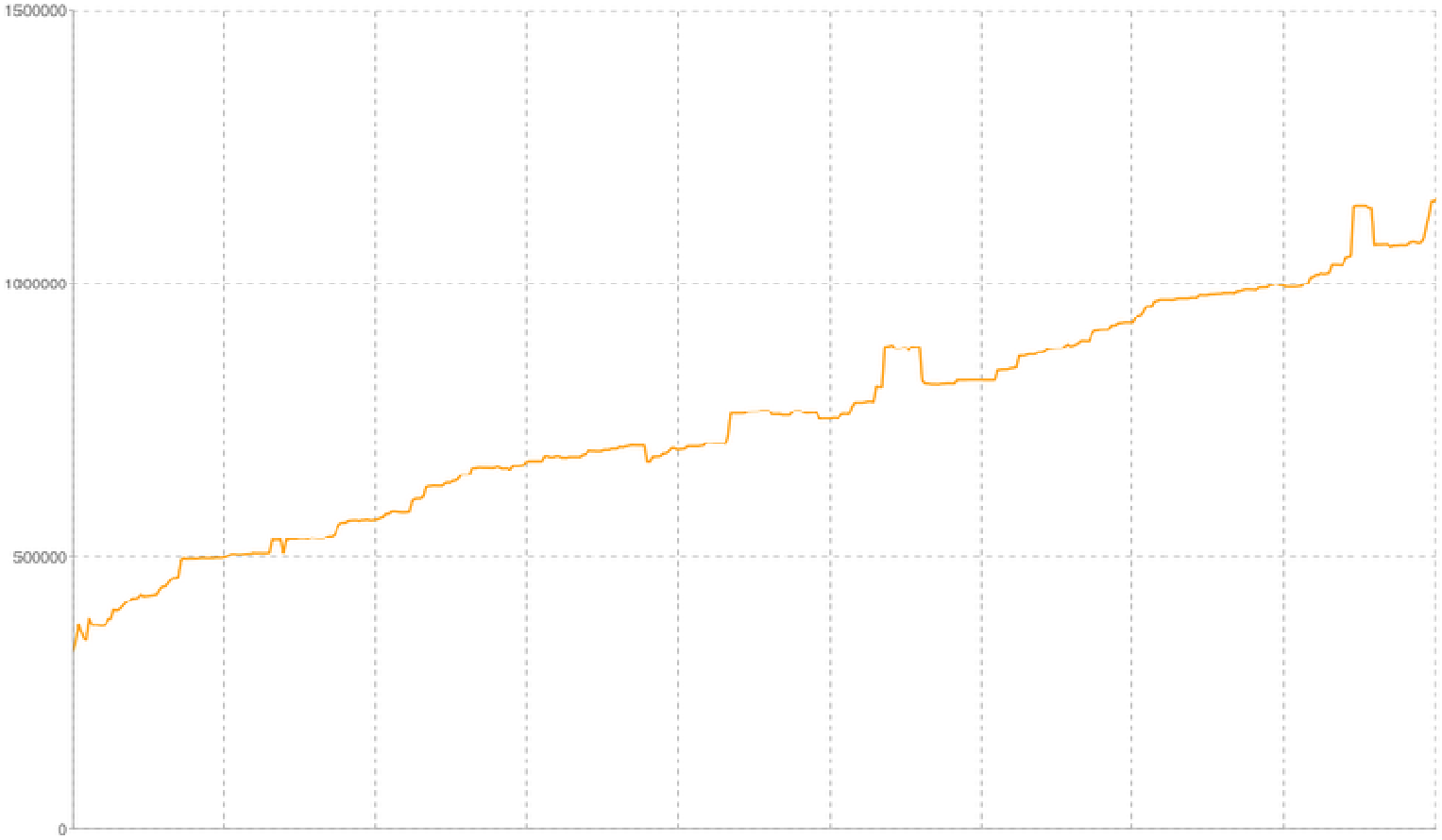
Solution: Out-of-process Hosted Mode (OOPHM)

- Browser plugin to debug in Eclipse, but run in real browser!
- Firebug only for FF:
  - OOPHM allows Java debugging in FF, Safari, IE (so far)
- See: http://code.google.com/p/google-web-toolkit/wiki/DesignOOPHM

<Time for a demo!>

Distribute as a CD-ROM?

Issue: download size >1 MB (pre-gzip) and counting...



Distribute as a CD-ROM? No!

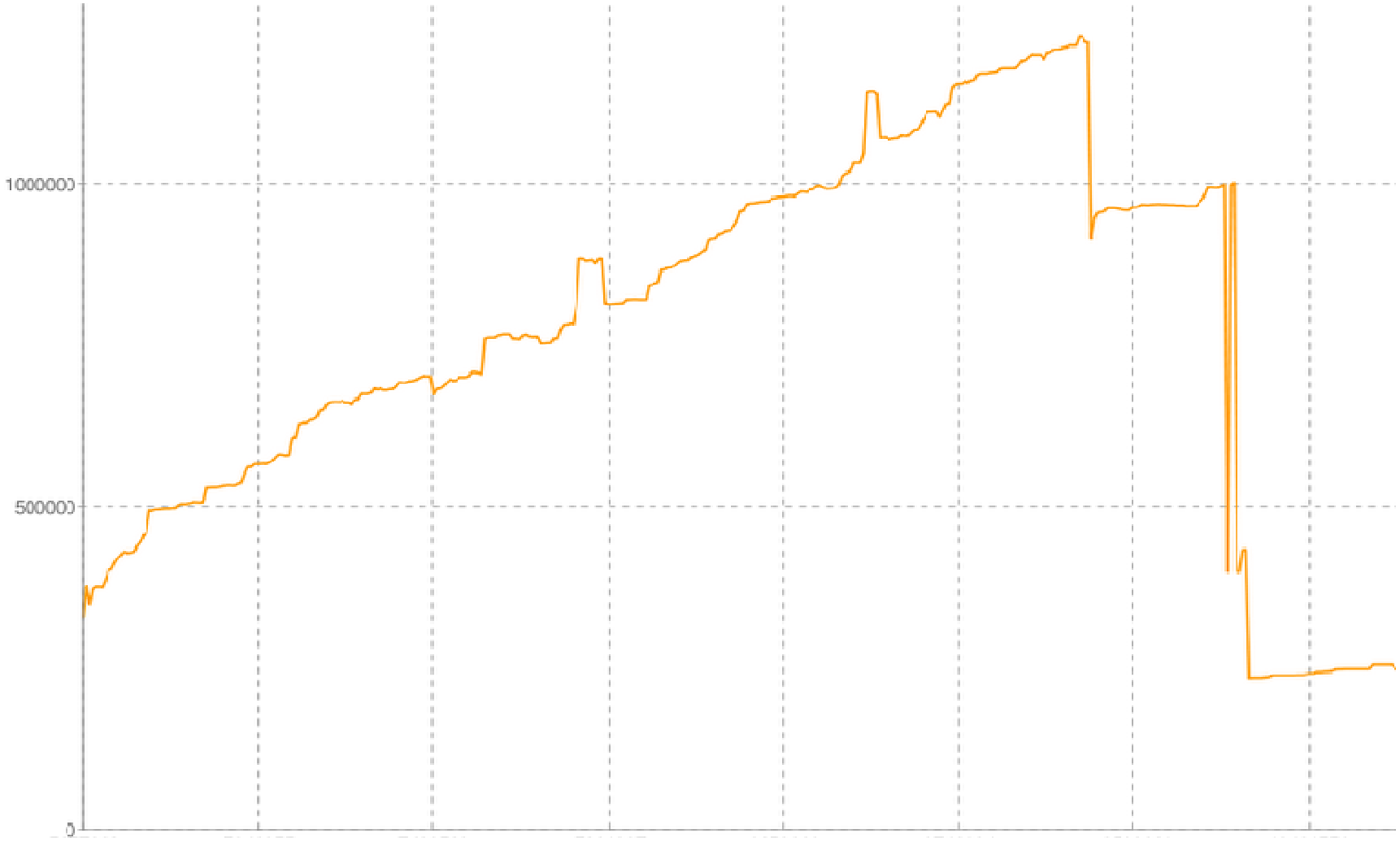
Solution: runAsync (dynamic loading of code)

- GWT.runAsync() signals a "cut point" to the GWT compiler:
  - Download what you need, when you need it
  - Resources (CSS, images, msgs) come with the code that uses it
  - Automatically handled by GWT compiler!

```
public void onNewWaveClicked() {
  GWT.runAsync(new RunAsyncCallback() {
    public void onSuccess() {
      WaveCreator.createNewWave();
    }
  });
}
```

See: http://code.google.com/p/google-web-toolkit/wiki/CodeSplitting

Down and to the right



Where's all the JS coming from?

Issue: need to know what Java causes the most JS

Solution: Story-of-your-Compile (SOYC) reports

- What is it?: Java package to JS breakdown report
- Helped us identify:
  - messages too large
  - compiled class names
  - what's in the initial download
- e.g. before and after messages optimisation project

JSOs cannot implement interfaces

Issue: we need interfaces for our messages, because:

- client + server both have common libraries
- they should be implementation-agnostic

Solution: GWT's SingleJsoImpl

- In order to inline, JSOs cannot have polymorphic dispatch
- SingleJsoImpl: allow at most one JSO class to implement any interface

GWT changes summarised

- Declarative UI / UiBinder
- StyleInjector + CssResource + ClientBundle
- JavaScriptObject
- OOPHM
- runAsync
- Story-of-your-Compile (SOYC)
- SingleJsoImpl
- -XdisableClassMetadata (saved us ~90KB)

Improving the user experience

Google
Developer
Day2009

Improving Gears

- Client-side Thumbnailing
  - send thumbnails before image upload
  - uses WorkerPool to avoid blocking UI
- Desktop Drag + Drop
- Resumable uploading

Performance

- Startup:
  - runAsync
  - fast start
  - inline images + CSS
  - smaller download
  - stats collection
  - server-side script selection

- Loaded client:
  - optimistic UI
  - prefetching
  - flyweight pattern
  - rendering tricks (prefer DOM over GWT's Widget)

Mobile Client

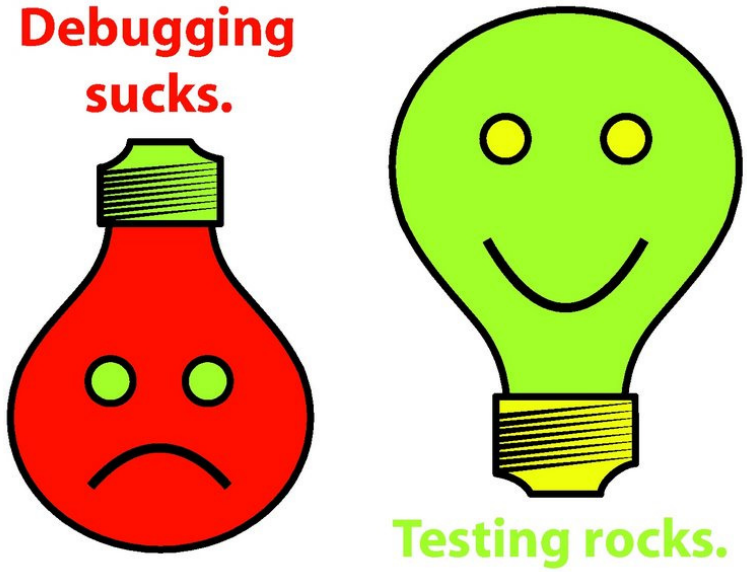
- GWT deferred binding saves the day!
- v1 AJAX only
- iPhone browser always running
  - browser starts up faster than native apps
- uses mobile-specific communication channel
- HTML5 / Gears caching: AppCache manifest GWT linker

<Time for another demo!>

Testing

Testability

- Model View Presenter
- Prefer JUnit tests over GWTTestCase
- Browser automation: WebDriver



WebDriver

- What is it?
  - developer-focused tool for browser automation
- Why do we use it?
  - native keyboard and mouse events, rather than synthesised via JS
- Challenges:
  - adopted early by Wave
  - incomplete
- Google Wave's commitment
- What's new?
  - iPhoneDriver
  - RemoteWebdriver on a grid
- <Demo!>

WebDriver Tips

- Avoid xpath: slow (JS on IE), brittle
  - rather: ids, names, and sub-dom navigation
- Intent of tests should be clear: use literate programming
- Each UI class has a WebDriver helper class

```
// Type in some stuff
BlipPanel blip = wavePanel.getFocusedBlip();
Editor editor = blip.getEditor();
editor.type("Do you know your abc?")
    .enter()
    .type("And your alpha beta gamma?")
    .back()
    .type("...?");
editor.check("<p _t='title'>Do you know your abc?</p>" +
             "<p>And your alpha beta gamma...?|</p>");
// This will cause a contacts popup
blip.clickSubmit();
assertEquals("Do you know your abc?",
wavePanel.getTitle());
```

Summary

- To GWT or not to GWT?
- Client architecture
- Changes in GWT
- Improving Gears
- Performance
- Mobile client
- Testability
- UI testing with WebDriver

Thanks! Questions?

- Learn more:
  - http://wave.google.com
  - http://code.google.com/intl/zh-CN/webtoolkit/
  - http://code.google.com/p/webdriver/