Google™ 09 I/O

GWT Can Do What?!
A Preview of Google Web Toolkit 2.0

Bruce Johnson
May 27, 2009

Google Web Toolkit Overview

- Write code in the Java™ language using your favorite Java IDE
- Debug as bytecode against a special browser (hosted mode)
- Cross-compile into standalone optimized JavaScript (web mode)
- No browser plugins / no obligatory server-side machinery
- Includes extensive cross-browser libraries
  - User interface (DOM, widgets, ...)
  - Client/server communication (XHR, RPC, JSON, ...)
  - App infrastructure (history, timers, unit testing, i18n, a11y, ...)
  - External services (Gadgets, Gears, Google Maps, ...)
- JavaScript integration
  - JavaScript Native Interface (JSNI)
  - Overlay types
- Fully open source under Apache 2.0

What’s Coming in GWT 2.0?

- In-browser hosted mode (formerly known as “OOPHM”)
- Compiler enhancements
- Developer-guided code splitting
- Resource optimization: ClientBundle
- Faster, easier, more predictable layout
- Also noteworthy...
  - RPC blacklists
  - RpcRequestBuilder
  - Client-side stack traces

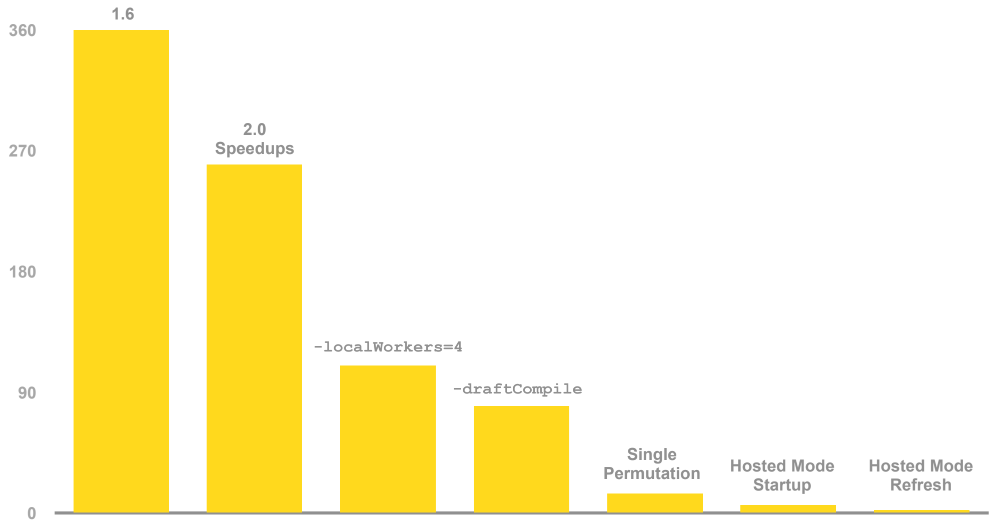
Hosted Mode

Hosted Mode, Reborn

- Hosted mode is *the* key to productive development with GWT
  - Debugging
  - Edit/refresh
  - Compiler isn't meant to be an alternative
- Problem: hosted mode browser is too special
  - On Linux, hosted browser is an ancient Mozilla
  - Hard to debug CSS quirks (e.g. no Firebug)
  - Hard to simulate interactions with other technologies (e.g. Flash)
  - Impossible to debug browsers on non-dev OSes (e.g. IE from Mac)
- Solution: make hosted mode work with "any" browser
  - And make it work across the network
- Let's see how this works...

Compiler Enhancements

Faster Compilation (TODO: real data)



Times compiling Showcase on my 2-core MacBook Pro

-XdisableClassMetadata

- Calling `obj.getClass()` or `clazz.getName()` forces `Class` objects and their names to be generated into JavaScript
- But if you don't really care what Class#getName() returns...
  - **`var javaLangObjectClass = new Class("java.lang.Object");`**
  - can become **`var jarLangObjectClass = new Class();`**

Showcase metadata before

5% - 10%
script reduction

Showcase metadata after

- Size, speed, and obscurity benefits

-XdisableCastChecking

- Nobody actually catches ClassCastException in app code

```
void makeItQuack(Animal animal) {
  try { ((Quacker) animal).quack(); }
  catch (ClassCastException c) { Window.alert("This doesn't quack."); }
}
```

- The above example generates a call like this:

```
dynamicCast(animal, 2).quack();
```

- But with the flag turned on, you get only this:

```
animal.quack();
```

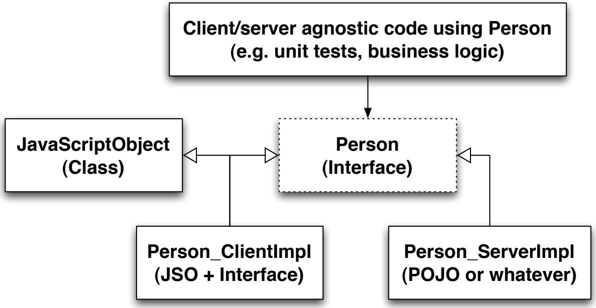
- In a real-world (and very large) Google app...
  - 1% script size reduction
  - 10% speed improvement in performance-sensitive code

Interfaces on JavaScript Overlay Types (JSOs)

- In GWT 1.6, JSOs are nifty but tied to JavaScript

```
final class Person extends JavaScriptObject {
  protected Person() {}
  public native String getFirstName() /*-{ return this.firstName; }-*/;
  public native String getLastName() /*-{ return this.lastName; }-*/;
}
```

- Server code barfs on JSOs, though, so...



Code Splitting

Big Scripts, Big Problems

- It's easy to ignore compiled script size until it's too big
- Many problems here
  - Initial download can be slooooow
  - Super-linear parse time on some browsers
  - UI hangs during script parsing
  - Script parsing adds latency to initial UI setup
- Why not use multiple compiled modules?
  - Compiled modules are black-boxes unless you explicitly export
  - Splitting functionality across modules with exports...
    - Is laborious
    - Is error-prone
    - Prevents tasty compiler optimizations such as dead-code elimination
    - Is wonderful (but only for the right reasons)

Meet `runAsync`

Drop hints to the GWT compiler where splitting seems reasonable

Split point

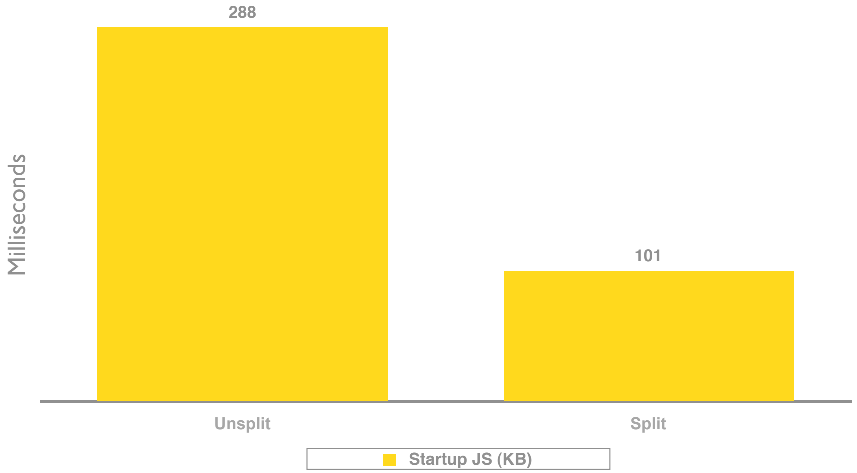
Runs after a possible (probably rare) delay

Runs if required script cannot be downloaded

```
public void onMySettingsLinkClicked() {

  GWT.runAsync(new RunAsyncCallback() {

    public void onSuccess() {
      new MySettingsDialog().show();
    }

    public void onFailure(Throwable ohNoes) {
      // indicate that something went wrong,
      // usually a connectivity or server problem
    }
  });
}
```

Splitting Showcase (TODO: gather data)

- Demo: Showcase
- Split point per link in tree



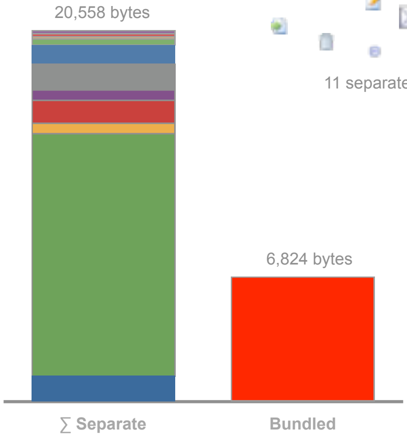
Getting to Know `runAsync`

- Intentionally developer-guided
- Intentionally async
- Intentionally forces you to think about failure paths
- Split point doesn't necessarily split
  - Compiler decides how to cluster code
  - Guaranteed to be correct, ordering-wise...
  - ...but might not split as you had hoped due to cross-refs
- Using modular patterns is key
- Shameless plug: Reading Tea Leaves / Story of Your Compile

Resource Optimization: ClientBundle

ImageBundle Redux

ImageBundle was only the beginning



11 separate images
1 bundled image

- Bundled ("sprited") automatically at compile-time
- Fewer round-trips (1 request vs. 11)
- Many browsers have 2-connection max
- Bundled image can be smaller than the sum of its constituent images
- Great caching story

Meet ClientBundle

ClientBundle generalizes ImageBundle to arbitrary resource types

```
interface MyBundle extends ClientBundle {
  public static final MyBundle INSTANCE = GWT.create(MyBundle.class);

  @Source("smiley.gif")
  ImageResource smileyImage();

  @Source("frowny.png")
  ImageResource frownyImage();

  @Source("app_config.xml")
  TextResource appConfig();

  @Source("wordlist.txt")
  ExternalTextResource wordlist();

  @Source("manual.pdf")
  DataResource ownersManual();

  @Source("super-fancy.css")
  CssResource superFancy();
}
```

A Simple Example: TextResource

```
interface MyBundle extends ClientBundle {
  public static final MyBundle INSTANCE = GWT.create(MyBundle.class);
  @Source("app_config.xml") TextResource appConfig();
}
```

Figure 1 – Declaration

```
<app-config animation-speed="1500" failover-strategy="give-up" ... />
```

Figure 2 – Text Resource (app_config.xml) found on your classpath at compile-time

```
void configureMyApp() {
  MyBundle bundle = MyBundle.INSTANCE;
  TextResource txtres = bundle.appConfig();
  String xml = txtres.getText();
  Document doc = XMLParser.parse(xml);
  // ...configure application using XML DOM...
}
```

Figure 3 – Point of Use

- Guaranteed to succeed because the text resource is compiled in
- Use the file format that is most appropriate; separate data from code if desired
- No HTTP request required

ClientBundle's Killer Feature: CssResource

Compiles CSS with an enhanced syntax

- Define and use constants in CSS

```
@def hardToMissThickness 8px;
@def scaryColor #F00;
.error-border {
  border: hardToMissThickness solid scaryColor;
}
```

- Conditional rules for user agent, locale, or...anything

```
@if user.agent safari {
  .error-border {
    -webkit-border-radius: 4px;
  } @elif user.agent gecko {
    -moz-border-radius: 4px;
  }
}
```

- More...Demo!

Predictable Layout

Predictable Layout

- The philosophy of layout in GWT: Don't do it yourself
  - Have cold sweats when considering measuring anything
  - Use implicit layout for speed instead
  - See Kelly Norton's "Measuring in Milliseconds" for details
- Downside: unintuitive and inconsistent layout behavior
  - Width 100%? Often fits like a glove
  - Height 100%? Often smells like a glove
- Demo of yuckiness: Look at Mail in FF 3.5
- Standards mode provides new leverage
  - Constraint-based layout that actually does what you say
  - You'll never have to hook the window resize event again
  - An updated set of Panels in GWT 2.0

Demo: New and Improved DockPanel

```
public void onModuleLoad() {
  final DockLayoutPanel p = new DockLayoutPanel(Unit.PX);

  p.add(createHtml("north"), Direction.NORTH, 48);
  p.addSplitter();
  p.add(createHtml("south"), Direction.SOUTH, 48);
  p.addSplitter();
  p.add(createHtml("east"), Direction.EAST, 60);
  p.addSplitter();
  p.add(createVerticalStack(), Direction.WEST, 160);
  p.addSplitter();
  p.add(createLoremIpsum(), Direction.CENTER, 0);

  Window.setMargin("0px");
  Window.enableScrolling(false);

  RootPanel.get().add(new RootLayoutPanel(p));
}
```

- Doesn't run JavaScript during resize
- Constraint-based layout similar to Cocoa on OS X
- Animation to switch between constraints is baked-in

Also Noteworthy

Also Noteworthy

- RPC blacklist: Tell the RPC subsystem to skip types that you know aren't ever sent across the wire

```
<extend-configuration-property name="rpc.blacklist"
    value="com.example.myapp.client.WidgetList"/>
<extend-configuration-property name="rpc.blacklist"
    value="com.example.myapp.client.TimerList"/>
...
```

- RpcRequestBuilder: Customize XHRs for all RPCs in a service

```
ServiceDefTarget sdt = (ServiceDefTarget)myService;
sdt.setRpcRequestBuilder(myBuilderWithCustomHttpHeaders);
...
// All calls will use the same XHR settings
// (e.g. custom HTTP request headers)
myService.doSomethingOnTheServer(a, b, c);
```

- Client-side stack traces on some browsers
  - In other words, `Throwable#getStackTrace()` actually does something sometimes
  - Let's talk details in the GWT developer forum

Summary

Recap of What's Coming in GWT 2.0

Feature	Productivity for you	Performance for your users
• In-browser hosted mode	Debug in real browsers	
• Faster compilation	Less thumb-twiddling	
• Script size reductions and speed improvements	Simple flags enable size/speed gains	Apps start faster; run faster
• Code splitting	High-leverage, low-risk way to spread download time	Apps start faster; stay interactive
• ClientBundle (w/ CssResource!)	Project organization != deployment organization	Fewer HTTP round-trips
• Layout you can count on	Less time fighting with CSS and layout	Faster, smoother layout and resizing

Google™ 09 I/O