GOOGLE I/O

Surprisingly Rockin'

DOM Programming in GWT 1.5

Bruce Johnson
May 2008

“The big pieces must be independently useful.”

“Widget classes available but not required (i.e. use DOM and/or JSNI)”

“Making GWT Better”

DOM Programming in GWT 1.5

New DOM Classes in GWT 1.5

- A Java client-side DOM API
  - com.google.gwt.dom.client
- Based on W3C standard bindings
- Elegant, convenient, optimal, and cross-browser
- The Java source? Surprisingly rockin'!
- The compiled JS? Surprisingly rockin'!
- One more surprisingly rockin' thing...

Prettier Code, Part 1 – Less yucky old DOM

Old

```
public void tableExample() {
  Element table = DOM.createTable();
  Element tr = DOM.createTR();
  DOM.appendChild(table, tr);
  Element td = DOM.createTD();
  DOM.appendChild(tr, td);
  DOM.setInnerText(td, "The Old Way");
  DOM.appendChild(getBodyElement(), table);
}

public static native Element getBodyElement() /*-{
  return $doc.body;
}-*/;
```

New

```
public void tableExample() {
  TableElement table = doc.createTableElement();
  TableRowElement tr = table.insertRow(0);
  TableCellElement td = tr.insertCell(0);
  td.setInnerText("The New Way");
  doc.getBody().appendChild(table);
}
```

Prettier Code, Part 2 – Less error-prone JSNI

Old

```
public native String elemToString(Element elem) /*-{
  var temp = elem.cloneNode(true);
  var tempDiv = $doc.createElement("DIV");
  tempDiv.appendChild(temp);
  outer = tempDiv.innerHTML;  // bug
  temp.innerHTML = "";
  return outer;
}-*/;
```

New

```
public String elemToString(Element elem) {
  Element temp = elem.cloneNode(true).cast();
  DivElement tempDiv = doc.createDivElement();
  tempDiv.appendChild(temp);
  String outer = tempDiv.getInnerHTML();
  temp.setInnerHTML("");
  return outer;
}
```

Node

```
public class Node extends JavaScriptObject {
  appendChild(Node)
  cloneNode(boolean)
  getChildNodes()
  getFirstChild()
  getLastChild()
  getNextSibling()
  getNodeName()
  getNodeType()
  getNodeValue()
  getOwnerDocument()
  getParentNode()
  getPreviousSibling()
  hasChildNodes()
  insertBefore(Node, Node)
  removeChild(Node)
  replaceChild(Node, Node)
  setNodeValue(String)
}
```

Element

```
public class Element extends Node {
  getAbsoluteLeft(), getAbsoluteTop()
  getAttribute(String)
  getClassName()
  ...
  getElementsByTagName(String)
  getFirstChildElement()
  getId()
  getInnerHTML(), getInnerText()
  ...
  getNextSiblingElement()
  getParentElement()
  getPropertyBoolean(String)
  ...
  getScrollLeft()
  getString()
  getStyle()
  getTagName()
  getTitle()
  ...
  scrollIntoView()
  setAttribute(String, String), setInnerText(String)
}
```

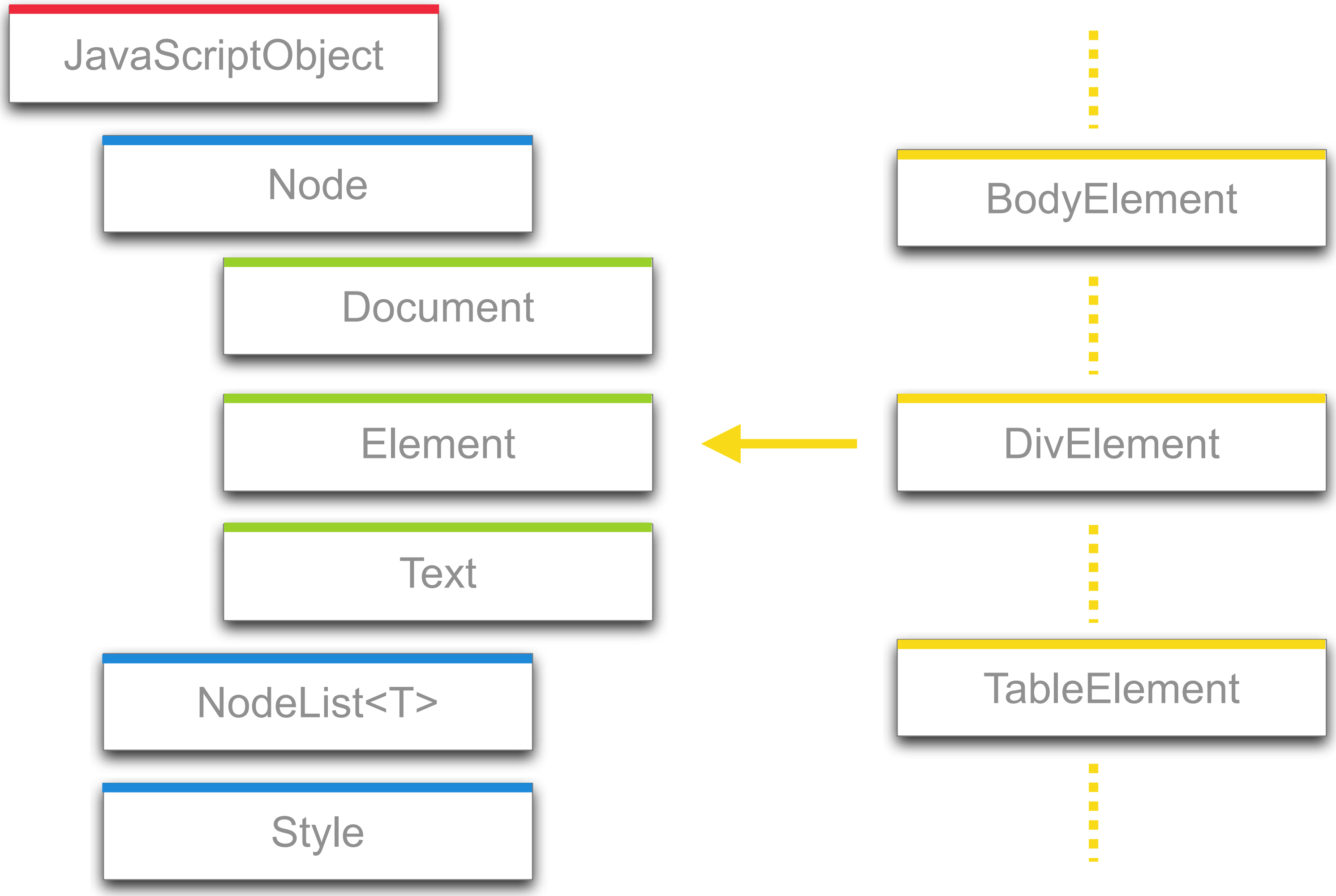
Example Subclass – TextAreaElement

```
public class TextAreaElement extends Element {
  blur(), focus()
  getAccessKey(), setAccessKey(String)
  getCols(), setCols(int)
  getDefaultValue(), setDefaultValue(String)
  getForm()
  getName()
  getReadOnly(), setReadOnly(boolean)
  getRows(), setRows(int)
  getTabIndex(), setTabIndex(int)
  getType()
  getValue(), setValue(String)
  getDisabled(), setDisabled(boolean)
  select()
  setName(String)
}
```

Digression: A word about tools

- TextAreaElement alone has nearly 100 methods
- Aren’t you glad you have...
  - Code completion
  - Refactoring: “Extract method”, “Extract local”, ...
  - Debugging
- New DOM classes are very fine-grained methods
  - Inspired by the pragmatic insights of Eclipse SWT
  - Trivial bindings ⇒ more time in the Java debugger

Hierarchy Overview – Based on W3C



What's all this gonna cost me? (Hint: zero)

Original Java source

```
String s = table.getRows().getItem(0).getCells().getItem
(0).getInnerText();
```

Compiled JavaScript

```
var s = table.rows[0].cells[0].innerText;
```

Hooray for compiler optimizations!

Will the real Element please stand up?

- The tricky part: Element isn't Element
- Compatibility, clarity, and evolution
- "user" Element extends "dom" Element
- UIObject.getElement()
- Upgrade advice
  - Head in the sand is no problem
  - Recommended granularity: one full source file
  - If mixing, pick a dominant type to import and use a fqcn for the other
  - Use elem.cast() or elem.<Whatever>cast()

Non-events

- Memory leaks are still an issue in some modern browsers
- Intentionally excluded: easy event hookup
- Leak-free events require higher-level machinery (Widget)
- Best: Use new DOM classes in the context of writing Widgets
- If you’re a JS/DHTML guru, design your own event handling

Overlay Types

Do-it-yourself awesomeness

- The new DOM classes are actually no big deal
- Everything is built on hot new JavaScriptObject plumbing in GWT 1.5
- Imagine overlaying a Java type directly on top of an existing JS object...
  - HTML DOM elements
  - XML DOM nodes
  - JSON structures
- Benefits of Java typing (i.e. IDE support)
- Zero overhead in terms of size and speed

Modeling <TR> as a Java type

```
public class TableRowElement extends Element {

  public final native String getAlign()
  /*-{ return this.align; }-*/;

  public final native NodeList<TableCellElement> getCells()
  /*-{ return this.cells; }-*/;

  public final native int getRowIndex()
  /*-{ return this.rowIndex; }-*/;

  public final native TableCellElement insertCell(int index)
  /*-{ return this.insertCell(index); }-*/;

  public final native void setAlign(String align)
  /*-{ this.align = align; }-*/;

  // ...more...
}
```

Using TableRowElement

```
public void printRow(String rowId) {
  Element el = doc.getElementById(rowId);
  TableRowElement tr = TableRowElement.as(el);
  NodeList<TableCellElement> cells = tr.getCells();
  for (int i = 0, n = cells.getLength(); i < n; ++i) {
    System.out.println(cells.getItem(i).getInnerText());
  }
}
```

```
public class TableRowElement extends Element {
  public static TableRowElement as(Element elem) {
    // Sanity check
    assert elem.getTagName().equalsIgnoreCase("tr");
    // “Trust me” downcast
    return (TableRowElement) elem;
  }
  // ...more...
}
```

Modeling JSON using overlay types

```
native Friend getFriendFromTheWild() /*-{
  return {
    "firstName" : "Joel",
    "lastName" : "Webber",
    "hobbies" :
      ["coding","thinking about coding","talking about coding"]
  };
}-*/;
```

```
String friendExample() {
  Friend f = getFriendFromTheWild();
  String s = f.firstName() + " " + f.lastName() + " likes";
  JsArray<String> hs = f.hobbies();
  for (int i = 0, n = hs.size(); i < n; ++i) {
    s += (i > 0 ? ", " : "") + hs.get(i);
  }
  return s;
}
```

The anatomy of an overlay type

```
public class Friend extends JavaScriptObject {

  protected Friend() {}

  public final native String getFirstName() /*-{
    return this.firstName;
  }-*/;

  public final native String getLastName() /*-{
    return this.lastName;
  }-*/;

  public final native JsArray<String> getHobbies() /*-{
    return this.hobbies;
  }-*/;
}
```

Works for generics, too

```
public class JsArray<E> extends JavaScriptObject {

  protected JsArray() {}

  public final native E get(int i) /*-{
    return this[i];
  }-*/;

  public final native int size() /*-{
    return this.length;
  }-*/;
}
```

But is it fast?

```
function $friendExample(){
  var f, hs, i, n, s;
  f = {
    firstName: 'Joel',
    lastName: 'Webber',
    hobbies:
    ['coding', 'thinking about coding', 'talking about coding']
  };

  s = f.firstName + ' ' + f.lastName + ' likes ';

  hs = f.hobbies;
  for (i = 0 , n = hs.length; i < n; ++i) {
    s += (i > 0?', ':'') + hs[i];
  }

  return s;
}
```

To put it another way...

```
function oc(){var a,b,c,d,e;a={firstName:t,lastName:u,hobbies:
[v,w,x]};e=a.firstName+y+a.lastName+z;b=a.hobbies;for
(c=0,d=b.length;c<d;++c){e+=(c>0?A:rb)+b[c]}return e}
```

- All obfuscatable names obfuscated
  - Evidence for why it's better to do more in Java source than JS
  - Obfuscation alphabet recycled across disjoint scopes
- String literals interned to reduce heap pressure
- All function calls inlined
- All non-essential punctuation and spacing removed
- Would you write it like this by hand in JavaScript?

Using Overlay Types, Part 1 – Rules

- Must extend JavaScriptObject, possibly indirectly
- Must have an empty, zero-param protected constructor
- Cannot use "new" to create JSO subclasses
- No declared instance fields
- All methods must be somehow final to avoid polymorphism, so that this

```
final native void foo(int x, int y) /*-{ ... }-*/;
```

can magically become this

```
final static native void foo(MyJSO this_, int x, int y)
/*-{ ... }-*/;
```

Using Overlay Types, Part 2 – Crazy casts

- With JSNI, things can most definitely blow up
- When it comes to overlay types, GWT trusts the programmer
- Zero-overhead ⇒ no GWT type metadata at runtime
- If x is a JavaScriptObject or any "subclass" thereof...
  - (x instanceof AnyOverlayType) *is always true*
  - ((AnyOverlayType)x) *always succeeds*
  - Or use `JavaScriptObject.cast()` for “cross-casting”
- Crazy talk, eh?
  - This behavior is the norm in JavaScript :-)

Putting it all together / Demo

Summary

When you upgrade to GWT 1.5

- Not urgent! Don't do this two days before you plan to ship!
- Strongly prefer the new DOM classes
  - Clearer, more correct code
  - Leverage your Java IDE more than ever
  - Don't fret about peephole syntactical performance
- Do fret about actual browser performance
  - There is no silver bullet (maybe in GWT 2.0?)
- Use overlay types liberally
  - Never, ever fear integration with JavaScript again
  - Never let anyone tell you GWT is slower than handwritten JS
- Reinvest all the time you save into GWTC...

Huge Thanks!

- Scott Blum
- Bob Vawter
- Joel Webber
- Lex Spoon
- Matthew Mastracci
- Ray Cromwell
- Everybody else in the truly rockin' GWT open source community!

Shop Talk

GOOGLE I O