JavaScript Native Interface

Scott Blum
GWT Engineer

JavaScript Native Interface (JSNI)

- What is JSNI?
- Them be the rules
- JavaScriptObject
- Practical JSNI
- Discussion

Handwritten JavaScript in Java

```
public class Hello implements EntryPoint {

 native void alert(String msg) /*-{
  $wnd.alert(msg);
 }-*/;

 public void onModuleLoad() {
  alert("Hello World!");
 }
}
```

Abusing Java syntax :)

```
native void alert(String msg) /*-{
  $wnd.alert(msg);
}-*/;
```

Abusing Java syntax :)

```
native void alert(String msg) /*-{
  $wnd.alert(msg);
}-*/;
```

Abusing Java syntax :)

```
native void alert(String msg) /*-{
  $wnd.alert(msg);
}-*/;
```

Use Browser APIs Directly

```
public class Hello implements EntryPoint {
 native void inject(String url) /*-{
   var script = $doc.createElement("script");
   script.src = url;
   $doc.body.appendChild(script);
 }-*/;
 public void onModuleLoad() {
   inject("foo.js");
 }
}
```

Actually, that was a contrived example

```
void inject(String url) {
  Element script = DOM.createElement("script");
  DOM.setElementProperty(script, "src", url);
  DOM.appendChild(RootPanel.get().getElement(),
   script);
}
```

Actually, that was a contrived example

```
void inject(String url) {
  Element script = DOM.createElement("script");
  DOM.setElementProperty(script, "src", url);
  DOM.appendChild(RootPanel.get().getElement(),
   script);
}
```

But how does GWT implement this...?

GWT core libraries are built on JSNI

```
native Element createElement(String tag) /*-{
 return $doc.createElement(tag);
}-*/;
native void setElementProperty(Element elem,
    String prop, String value) /*-{
 elem[prop] = value;
}-*/;
native void appendChild(Element parent,
  Element child) /*-{
 parent.appendChild(child);
}-*/;
```

Perhaps in the future...

```
void inject(String url) {
 ScriptElement script = ScriptElement.create();
 script.setSrc(url);
 Body.get().appendChild(script);
}
```

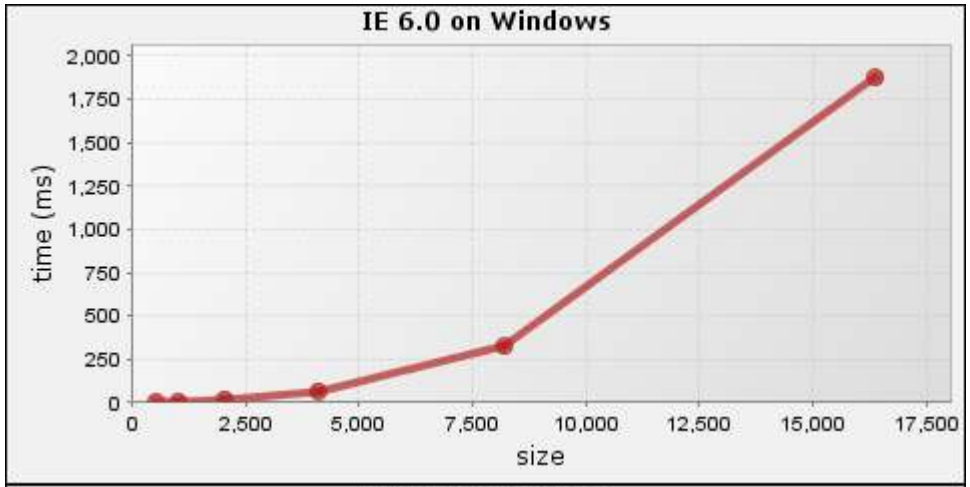
char[] to String, Java

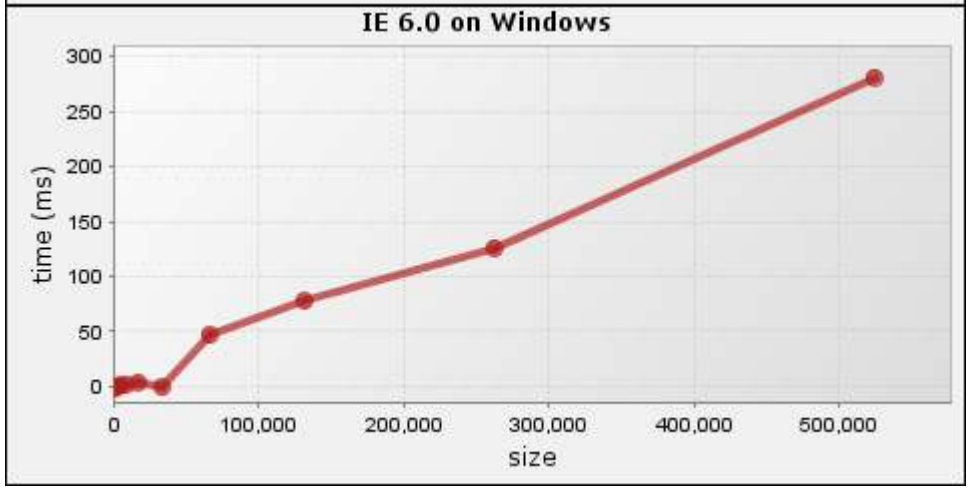
```
String valueOf(char[] in) {
  String result = "";
  for (int i = 0; i < in.length; ++i) {
    result += in[i];
  }
  return result;
}
```

char[] to String, JavaScript

```
native String valueOf(char[] x) /*-{
  // Trick: fromCharCode is a vararg method;
  // Use apply() to pass the input in one shot.
  return String.fromCharCode.apply(null, x);
}-*/;
```

Concat vs. native, char[] to String





← concat

16k char **~2000ms**

Appears **~O(n²)**

← native

500k char **~300ms**

Appears **~O(n)**

JavaScript Native Interface (JSNI)

- What is JSNI?
- Them be the rules
- JavaScriptObject
- Practical JSNI
- Discussion

Passing values

Declared Java Type	JavaScript type
`int`, `byte`, `short`, `double`, `etc...`	Numeric primitive
`String`	String object or primitive
`boolean`	Boolean primitive
`Object`	Opaque object
`JavaScriptObject`	Object

Passing values

```
native int getNumber() /*-{
 return "2";
}-*/;

public void onModuleLoad() {
 int x = getNumber();
 Window.alert(x + x);
}
```

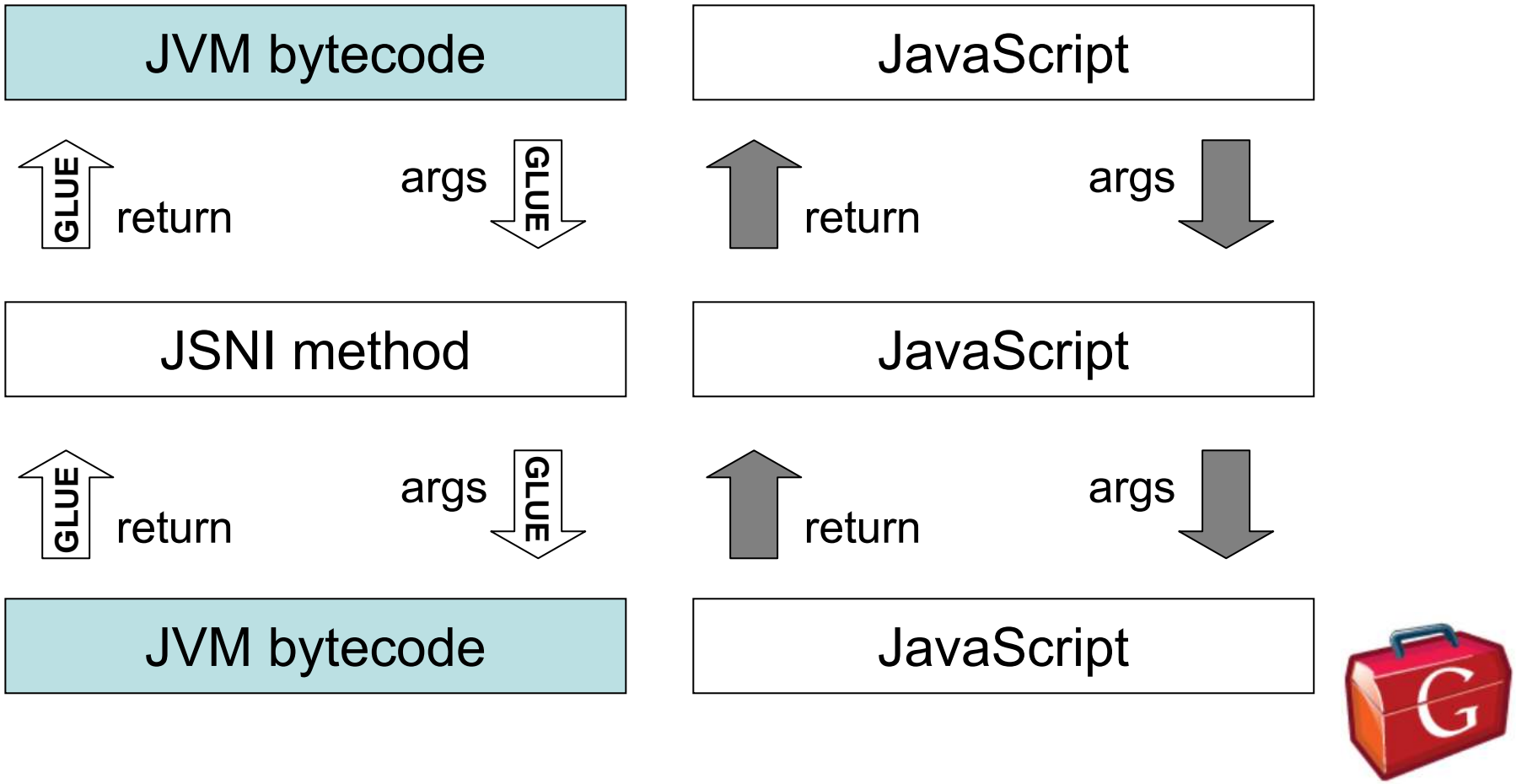
Passing values, surprising results

```
native int getNumber() /*-{
 return "2";
}-*/;

public void onModuleLoad() {
 int x = getNumber();
 Window.alert(x + x); // "22" in web mode
}
```

Passing values, hosted mode checks

```
native int getNumber() /*-{
 return "2";
}-*/;

public void onModuleLoad() {
 int x = getNumber(); // HostedModeException
 Window.alert(x + x);
}
```

Value passing, hosted vs. web mode



Java Objects in JSNI, fields

```
public class ArrayList<E> extends AbstractList<E>{
…
 private JavaScriptObject array;

 private native E getImpl(int index) /*-{
   return this.@java.util.ArrayList::array[index];
 }-*/;
…
}
```

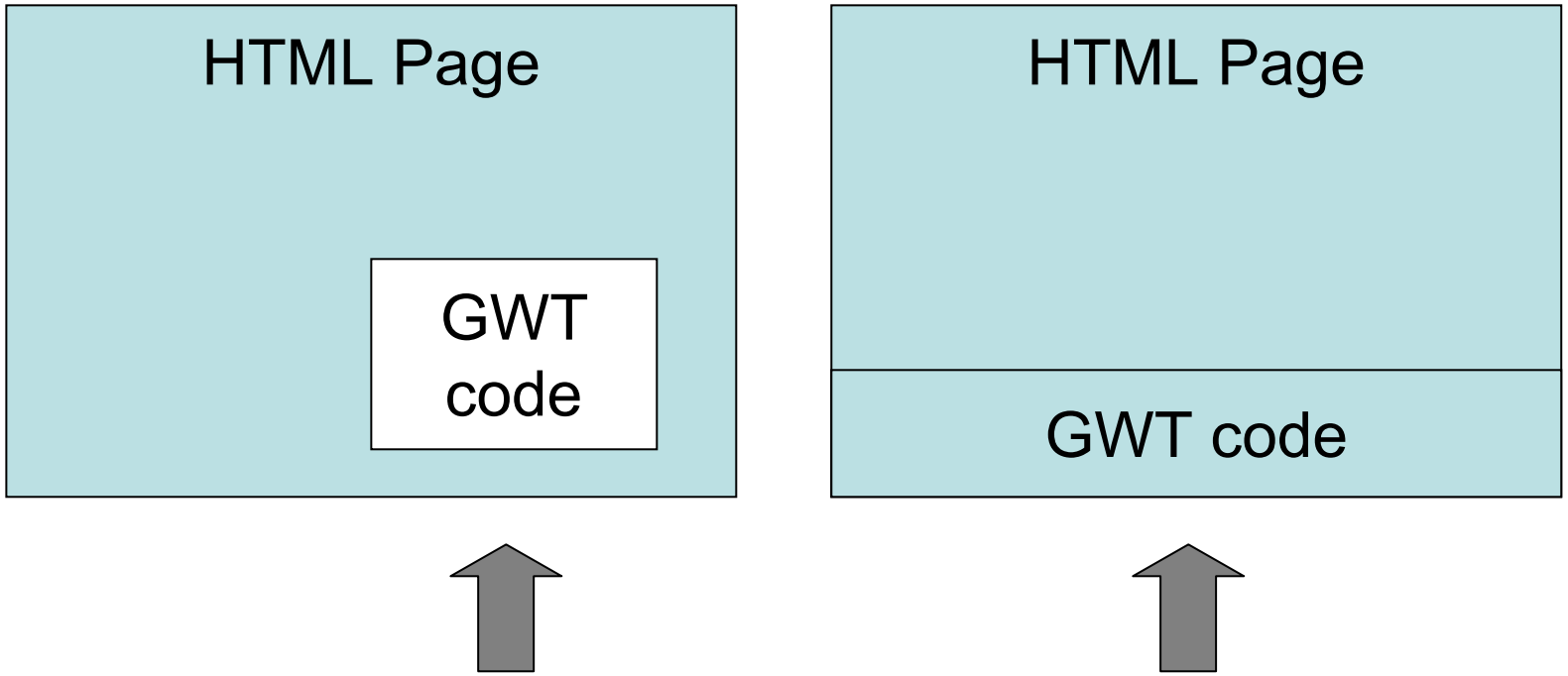
Java Objects in JSNI, methods

```
native void addAllKeys(Set<String> s,
  JavaScriptObject jso) /*-{
 for (var key in jso) {
  s.@java.util.Set::add(Ljava/lang/Object;)(key);
 }
}-*/;
```

Java Objects in JSNI, statics

```
final class XMLHTTPRequest {
 public static final int LOADED = 4;

 static native String send(JSO xhr, …) /*-{
  xhr.onreadystatechange = function() {
   if (xmlHttpRequest.readyState ==
       @package.XMLHTTPRequest::LOADED) {
    // do stuff
   }
 }-*/;
}
```

Accessing the window and document



Normal: runs in an IFRAME

Mashup: runs in an anonymous function body

Accessing the window and document

```
native void alert(String msg) /*-{
 $wnd.alert(msg);
}-*/;

native Element createElement(String tag) /*-{
 return $doc.createElement(tag);
}-*/;
```

JavaScript Exceptions

```
native void throwException() /*-{
 var x; x.foo();
}-*/;

public void onModuleLoad() {
 try {
  throwException();
 } catch (Throwable e) {
  // What is e?
  Window.alert(e.getClass.getName());
 }
}
```

JavaScript Exceptions

```
native void throwException() /*-{
 var x; x.foo();
}-*/;

public void onModuleLoad() {
 try {
  throwException();
 } catch (Throwable e) {
  // com.google.gwt.core.client.JavaScriptException
  Window.alert(e.getClass.getName());
 }
}
```

Exception Sandwich

```
native void sandwich() /*-{
 try {
  @package.Hello::throwMyException()();
 } catch (e) {
  throw e; // may not hold identity in hosted mode
 }
}-*/;
public void onModuleLoad() {
 try {
  sandwich();
 } catch (MyException e) {
  Window.alert(e.toString()); // identity retained
 }
}
```

undefined vs. null

```
native String getString() /*-{
}-*/;

public void onModuleLoad() {
 String s = getString();
 if (s != null) {
  Window.alert(s);
 }
}
```

undefined vs. null

```
native String getString() /*-{
 // implicitly returns undefined
}-*/;

public void onModuleLoad() {
 String s = getString();
 if (s != null) {
  Window.alert(s); // web mode: alerts "undefined"!
 }
}
```

undefined vs. null

```
function getString() {
 // implicitly returns undefined
}

function onModuleLoad() {
 var s = getString();
 if (s !== null) {
   // undefined !== null; execution reaches here
   $wnd.alert(s);
 }
}
```

undefined vs. null

```
undefined == null => true
undefined != null => false
undefined === null => false
undefined !== null => true
```

undefined vs. null

```
undefined == null => true
undefined != null => false
undefined === null => false
undefined !== null => true
```

Then why not generate “==” tests?

undefined vs. null

```
undefined == null => true
undefined != null => false
undefined === null => false
undefined !== null => true
```

Then why not generate “==” tests?

```
var o = new Object();
o == "[object Object]" => true(!)
o === "[object Object]" => false
o == new String("[object Object]") => false(!!)
o === new String("[object Object]") => false
```

JavaScript identity, pick two

- Triple equals identity testing
- String primitive and object interchangeable
- undefined and null interchangeable

Never return undefined

```
native String getElementProperty(Element elem,
  String prop) /*-{
 var ret = elem[prop];
 return (ret == null) ? null : ret;
}-*/;
```

Never return undefined

```
native String getString() /*-{
}-*/;

public void onModuleLoad() {
 String s = getString(); // HostedModeException
 if (s != null) {
  Window.alert(s);
 }
}
```

JavaScript Native Interface (JSNI)

- What is JSNI?
- Them be the rules
- JavaScriptObject
- Practical JSNI
- Discussion

JavaScriptObject

```
public class Button {
 static native Element createButton() /*-{
  return $doc.createElement("button");
 }-*/;

 private Element element = createButton();
…
}
```

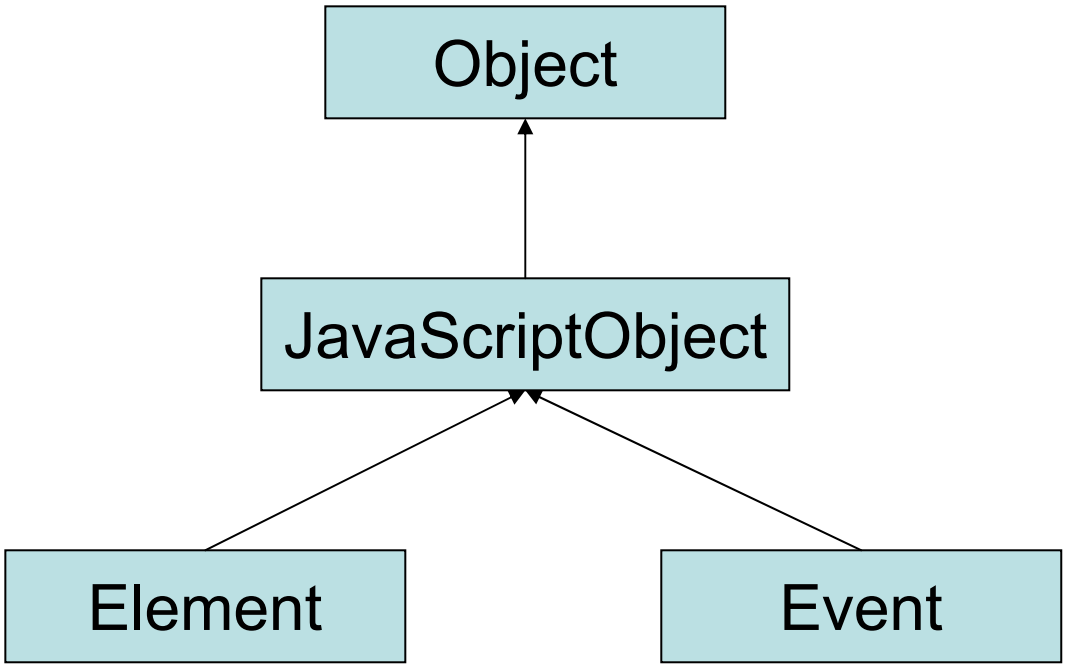
JavaScriptObject

com.google.gwt.core.client.JavaScriptObject

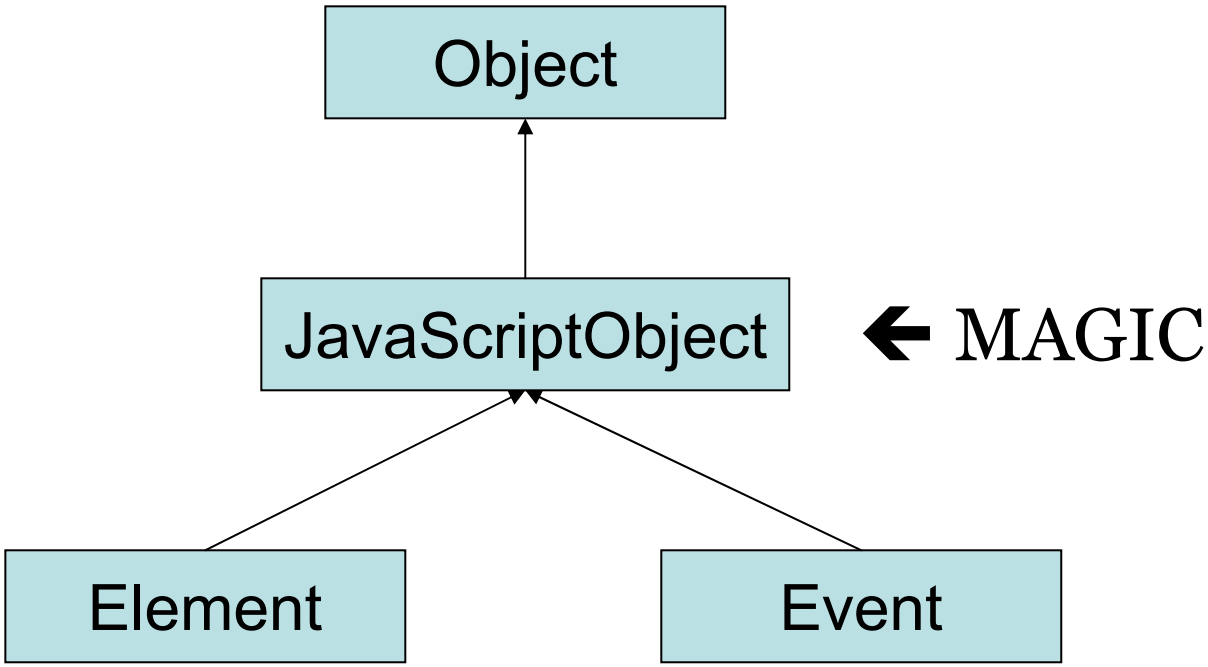
An opaque handle to a native JavaScript object. A **JavaScriptObject** cannot be created directly. **JavaScriptObject** should be declared as the return type of a JSNI method that returns native (non-Java) objects. A **JavaScriptObject** passed back into JSNI from Java becomes the original object, and can be accessed in JavaScript as expected.

SUBCLASSING IS NOT SUPPORTED EXCEPT FOR EXISTING SUBCLASSES

JavaScriptObject



JavaScriptObject



JavaScriptObject and virtual dispatch

```
public final class Element extends JavaScriptObject {
 public String toString() {
  return DOM.toString(this); // innerHTML
 }
}
public void onModuleLoad() {
 Element e = DOM.createButton();
 e.toString();
 alert(e instanceof Element);
}
```

JavaScriptObject and virtual dispatch

```
public final class Element extends JavaScriptObject {
 public String toString() {
  return DOM.toString(this); // innerHTML
 }
}
public void onModuleLoad() {
 Element e = DOM.createButton();
 Object o = rnd() ? this : e;
 o.toString();
 alert(o instanceof Element);
}
```

JavaScriptObject decoration

```
public final class Element extends JavaScriptObject {
 public String toString() {
  return DOM.toString(this); // innerHTML
 }
}
public void onModuleLoad() {
 Element e = wrap(DOM.createButton(), Element.class);
 Object o = rnd() ? this : e;
 o.toString();
 alert(o instanceof Element);
}
```

JavaScriptObject and implicit upcast

```
public final class Element extends JavaScriptObject {
 public String toString() {
  return DOM.toString(this); // innerHTML
 }
}
public void onModuleLoad() {
 Element e = DOM.createButton();
 Object o = rnd() ? this : wrap(e, Element.class);
 o.toString();
 alert(o instanceof Element);
}
```

JavaScriptObject

- Can be zero-overhead
- The rules are still changing
- Native instance methods supported in 1.5
- Use final instance methods
- Avoid implicit upcasts

JavaScript Native Interface (JSNI)

- What is JSNI?
- Them be the rules
- JavaScriptObject
- Practical JSNI
- Discussion

Where does GWT use it?

- JRE emulation (String, ArrayList, HashMap)
- DOM manipulation
- History
- XMLHttpRequest
- Timer / DeferredCommand
- JSON library

RPC field serialization

```
public class Person implements IsSerializable {
 private String description;
 private String name;
…
}
public class Person_FieldSerializer {
 String getDescription(Person instance) {
  return instance.description;  // Compile error
 }
…
}
```

The Violator Pattern

```
public class Person implements IsSerializable {
 private String description;
 private String name;
…
}
public class Person_FieldSerializer {
 native String getDescription(Person instance) /*-{
  return instance.@package.Person::description;
 }-*/;
…
}
```

JSON Parsing

```
{ name: "doglover",
  url: "http://flickr.com/people/doglover/",
  images: [
  { title: "Rover",
    url: "http://flickr.com/photos/doglover/12345/",
    width: 300,
    height: 200 },
  … ]
}
```

JSON Parsing

```
class Album extends JavaScriptObject {
 native String getName() /*-{ return this.name; }-*/;
 native String getUrl() /*-{ return this.url; }-*/;

 native int getImageCount() /*-{
  return this.images.length;
 }-*/;

 native Image getImage(int index) /*-{
  return this.images[index];
 }-*/;
}
```

JSON Parsing

```
class Image extends JavaScriptObject {
 native String getTitle() /*-{ return this.title; }-*/;
 native String getUrl() /*-{ return this.url; }-*/;
 native int getWidth() /*-{ return this.width; }-*/;
 native int getHeight() /*-{ return this.height; }-*/;
}
```

...and beyond!

```
class Image extends JavaScriptObject {
 native void show(Element parent, int w, int h) /*-{
  this.show(parent, w, h);
 }-*/;
}
```

Beyond beyond

```
class Image extends JavaScriptObject {
 interface ImageEvents {
  void onFadeIn(Image sender);
 }
 native void fadeIn(ImageEvents evts) /*-{
  var img = this;
  this.fadeIn(function() {
    evts.@pkg.ImageEvents::onFadeIn(Lpkg/Image;)(img);
  });
 }-*/;
}
```

Appropriate use of JSNI

- Prefer to use Java
- Write smaller “leaf” methods and keep complex logic in Java
- Use JS->Java references sparingly
- In GWT 1.5, simple JSNI methods are inlined
- JavaScriptObject is still evolving

Resources

- GWT documentation
  http://code.google.com/webtoolkit/documentation
- GWT source code
  http://code.google.com/p/google-web-toolkit/
- JavaScript InterOp Library (JSIO)
  http://code.google.com/p/gwt-api-interop/

Discussion?