"U heeft een verkeerde browser. Deze website werkt alleen met Internet Explorer versie 6.0.2900.3264.xpsp.071130-1427". **Iedereen die regelmatig op het Internet surft zal wel eens een melding zoals bovenstaande (met wellicht een minder specifiek versienummer) gekregen hebben. Vreemd eigenlijk, dat in deze tijd met zoveel goede tools, nog zo vaak dit soort problemen optreden. Blijkbaar is het lastig ervoor te zorgen dat een website er op de meest voorkomende browsers goed uitziet. De onderliggende oorzaak van genoemde meldingen is meestal: *JavaScript.***

Professioneel ontwikkelen met JavaScript?!

ing. Teun Hakvoort
is software developer bij Info Support b.v. in Veenendaal.

Ooit is JavaScript bedacht om enige dynamiek aan te brengen in de toen nogal statische HTML-pagina's. Daarna kwam de tijd dat de statische HTML-pagina's werden vervangen door dynamische serverside pagina's. Met de komst van dynamische serverside pagina's werd het gebruik van JavaScript beschouwd als 'bad practice' en werd het alleen nog daar gebruikt waar het echt niet anders kon. De komst van AJAX heeft JavaScript echter weer ongekend populair gemaakt. Door de komst van AJAX heeft een Java-ontwikkelaar meer dan ooit met JavaScript te maken. Met de komst van het framework Google WebToolkit is het voor een Java-ontwikkelaar aanzienlijk makkelijker geworden om JavaScript te programmeren. Daarom zal in dit artikel vanuit een Java-developer-bril naar de volgende vragen worden gekeken:

- Wat zijn de oorzaken van veel JavaScript-problemen?
- Hoe helpt het Google Web Toolkit-framework genoemde problemen te voorkomen?
- Hoe integreert Google Web Toolkit met bestaande populaire tools?

Problemen

Het gebruik van JavaScript brengt enkele problemen met zich mee. Het *eerste* probleem (althans, wordt vaak als probleem ervaren) is de taal JavaScript zelf. JavaScript is (in tegenstelling tot Java) niet *type-safe*. Dat wil zeggen, een Integer-object kan bijvoorbeeld worden behandeld als een String. De volgende regels – die in Java ondenkbaar zijn – werken prima in JavaScript:

```
var myVar = 5;
myVar = "Nu is myVar een String";
```

Codevoorbeeld 1: Eerst een int, dan een String: ondenkbaar in Java.

Een *tweede* probleem is: JavaScript wordt niet gecompileerd maar wordt runtime (als de webpagina wordt opgevraagd) geïnterpreteerd door de browser. Dat betekent dat bijvoorbeeld syntaxfouten, maar ook semantische fouten (niet gedeclareerde variabelen, etc.) pas runtime gezien worden.

Een *derde* probleem dat zich voordoet: de manier waarop JavaScript wordt geïnterpreteerd, is per browser verschillend. Zo werkt het volgende codevoorbeeld prima onder Internet Explorer 6 maar geeft onder Firefox 3 de melding 'window.frm is undefined'.

```
<form name="frm">
  <input type="text" name="txtNaam" value="Info
Support"/>
 <input type="button"
  onclick="javascript:alert(window.frm.txtNaam.
value)"
  value="klik!"/>
</form>
```

Codevoorbeeld 2: JavaScript voorbeeld, alleen werkend in IE.

Oplossing

Mogelijk rijst de vraag: "Waarom wordt er dan niet goed gecontroleerd of bovenstaande problemen zich voordoen?". De oorzaak van dit probleem zit grotendeels in de tool support. Een Java ontwikkelaar heeft beschikking over tools, waarmee het mogelijk is om te debuggen, refactoren en tevens helpt de tool hem met zogenaamde code-completion en quickfix. Er zijn tools waarmee dit ook binnen JavaScript mogelijk is. De ondersteuning gaat echter veelal minder ver en het is veel lastiger fouten op te sporen.
Om de genoemde problemen te voorkomen zijn er verschillende initiatieven gestart die het gebruik van JavaScript zoveel mogelijk verbergen. Dit verbergen kan op verschillende manieren worden gedaan:

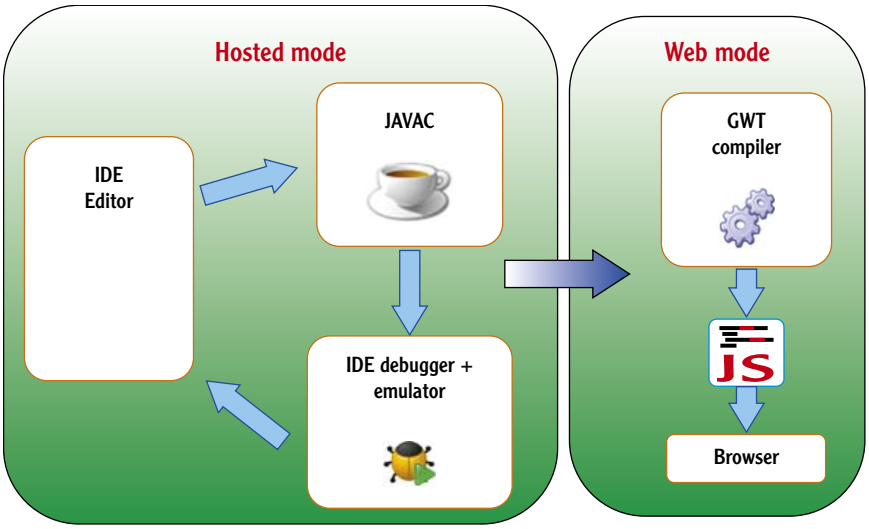
- Runtime JavaScript genereren op basis van Java.class files (dit is wat Echo3 doet);
- JavaScript encapsuleren in tags, zowel JSP tags als JSF tags (bijvoorbeeld JMaki, Jboss RichFaces).
- Een andere mogelijkheid is domweg Java sourcecode omzetten naar JavaScript code. Anders gezegd: een **Java to JavaScript compiler**. Een framework dat een 'Java to JavaScript' compiler geïmplementeerd heeft, is **Google Web Toolkit** (hierna afgekort als *GWT*).

GWT basics

GWT is een relatief nieuw framework dat binnen de Java community snel aan populariteit wint. Een feature (!) sinds versie 1.4 is dat het – in tegenstelling tot veel andere Google-producten – niet meer in bèta status verkeert.

Figuur 1: Google Web Toolkit, één van de weinige Google producten die niet in bèta status verkeert.

Het basisprincipe van GWT is: 'Ontwikkel in Java met behoud van toolsupport, taalsupport, objectoriëntatie en genereer op basis van Javacode browseronafhankelijke JavaScript'.
Omdat Javacode wordt omgezet naar JavaScriptcode, mag alleen datgene in Java geprogrammeerd worden wat in JavaScript mogelijk is en wat door de compiler is om te zetten. Het starten van bijvoorbeeld een thread zal dan ook een compilatiefout opleveren. In de zojuist verschenen versie GWT 1.5 RC1 is het mogelijk om de meeste klassen uit de `java.lang` en `java.util` package te gebruiken. Ook Java 5 features als: annotations, generics en for-each-loops kunnen probleemloos worden gebruikt.



Figuur 2: Hosted mode versus Web mode.

Zoals reeds beschreven wordt een applicatie geschreven in Java en uiteindelijk gecompileerd naar JavaScript. Maar, hoe kan er dan in Java gedebugd worden? Daarvoor moet toch eerst worden gecompileerd? Het antwoord hierop is dat een GWT applicatie in twee modi kan draaien:

- **Hosted** mode;
- **Web** mode.

In de **hosted** mode wordt de geschreven Java code gecompileerd naar Java-bytecode. Deze mode wordt gebruikt tijdens debuggen. Een door GWT meegeleverde browser-emulator zorgt ervoor dat de bytecode er in de emulator uitziet als een webpagina. Wanneer de ontwikkelaar de applicatie uiteindelijk wil publiceren of wil tonen in een echte browser, compileert hij de applicatie naar JavaScript waarna hij de applicatie kan draaien in de **web** mode. In figuur 2 wordt het onderscheid schematisch weergegeven.

Een standaard GWT applicatie bestaat uit verschillende onderdelen:

- Clientside Java-code (wordt gecompileerd naar JavaScript);
- Serverside Java-code (wordt gecompileerd naar Java-bytecode);
- Resources;
- XML-configuratiebestand.

Bij het onderdeel 'Clientside Java-code' moet gedacht worden aan alle code die uiteindelijk in de browser moet draaien. Dit betekent dat alle events – mouse events, key events, etc. – uiteindelijk clientside in JavaScript worden afgehandeld.
Zoals in bovenstaande applicatieonderdelen te zien is, wordt tijdens een compilatie niet alle code naar JavaScript gecompileerd. Code die aan de serverkant moet draaien (bijvoorbeeld GWT-services, zie verderop), wordt naar

Java-bytecode gecompileerd. Bij het onderdeel *Resources* valt te denken aan stylesheets, plaatjes, etc. In het XML-configuratiebestand worden verschillende onderdelen geconfigureerd: afhankelijkheden naar andere GW-modules, GWT-services, enz.

Voordelen GWT

Doordat er nu niet meer in JavaScript geprogrammeerd hoeft te worden, kan er op dezelfde manier worden ontwikkeld als bij een gewone desktop applicatie. Dat wil zeggen, de ondersteuning vanuit de IDE (denk aan refactoring, debug support, etc.) is gewoon beschikbaar. Tevens zijn de programmeerconcepten zoals die aanwezig zijn bij Swing, terug te vinden bij een GWT-applicatie. Zo is in onderstaand voorbeeld dezelfde structuur te ontdekken zoals deze ook bij een Swing applicatie zou zijn. Dit wil overigens niet zeggen dat een GWT ontwikkelaar niets meer van Web behoeft te weten.

```
final Label lblMyLabel = new Label();
final TextBox txtName = new TextBox();
lblMyLabel.setText("Enter your name and click on the
button");
```

```
Button btnSayHello = new Button("Click");
btnSayHello.addClickListener(new ClickListener() {
   public void onClick(Widget sender) {

   String name = txtName.getText();

   if (name == null || name.length() == 0) {

   Window.alert("Enter your name please!");

   } else {

   Window.alert("Hello " + name);

   }
   }
});
FlowPanel flowLayout = new FlowPanel();
flowLayout.add(lblMyLabel);
flowLayout.add(txtName);
flowLayout.add(btnSayHello);
RootPanel.get().add(flowLayout);
```

Codevoorbeeld 3: Is het nou Swing of GWT?

Naast het programmeervoordeel biedt GWT nog tal van andere voordelen. Doordat de gebouwde applicatie 'leeft' in de browser, kan de sessiedata worden verplaatst van de server naar de cliënt. GWT biedt een rijke suite aan widgets, biedt tevens ondersteuning voor Internationalization (I18N), imagebundles, integratiemogelijkheden

met andere JavaScript frameworks en eenvoudige client-server communicatie. De JavaScript die de GWT-compiler genereert, is platform onafhankelijk. Daarmee hoeft de ontwikkelaar geen moeite meer te doen om de applicatie in de meest gebruikte browsers te laten werken. Gesteld kan worden dat GWT een volwaardig platform is om Rich Internet Applications mee te bouwen.

JavaScript integratie

In sommige gevallen zal JavaScript aanwezig blijven. Bijvoorbeeld wanneer er al bestaande JavaScript-code aanwezig is, of wanneer er al gebruikt wordt gemaakt van bestaande JavaScript-frameworks (denk aan een framework als Dojo). Om in dat geval toch gebruik te kunnen maken van GWT is de JavaScript Native Interface (JSNI) geïntroduceerd. Daarbij kunnen aanroepen naar JavaScript methoden als native methods in Java gedeclareerd worden. Dit ziet er dan als volgt uit:

```
public native void callMyFavoriteFramework();/*{
   $wnd.alert(ëHello JavaScriptí);
}*/
```

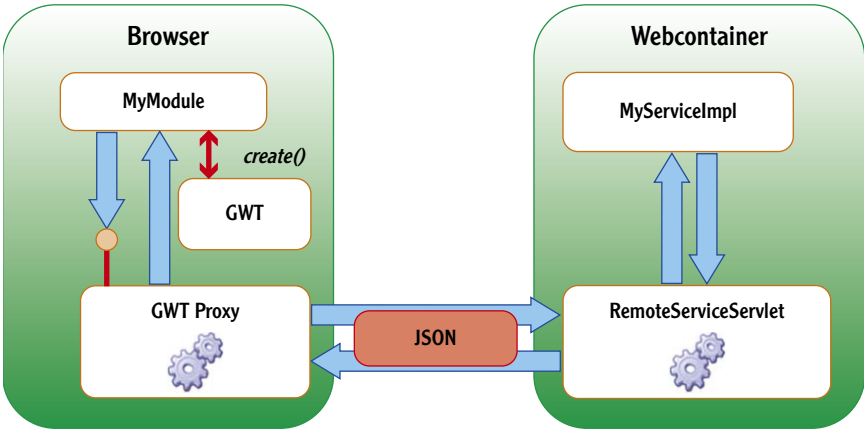
Codevoorbeeld 4: JavaScript Native Interface (JSNI)

De compiler zal het codegedeelte tussen de `/*{` en `}*/` tags letterlijk overnemen als JavaScript.

Servercommunicatie met GWT

Om vanuit de JavaScript-applicatie te communiceren met de server heeft GWT zogenaamde GWT-services geïntroduceerd. Dit zijn GWT specifieke servlets die draaien in een web container. Zo'n GWT-service is vanuit de browser zowel synchroon als asynchroon te benaderen. De conversie van Java-objecten naar iets wat begrepen wordt door JavaScript, wordt door het GWT-framework opgelost. In onderstaand figuur is het gebruik van een GWT-service schematisch weergegeven.
De stappen die doorlopen worden, zijn de volgende:

- Vanuit een GW-module wordt GWT gevraagd een proxy te creëren voor een specifieke GWT-service;
- Er wordt een (a)synchrone call gedaan op de verkregen proxy;
- De proxy zet de call om naar JSON en verstuurt deze naar de GWT-service;
- Daar komt deze call binnen bij de *com.google.gwt.user.server.rpc.RemoteServiceServlet*. Het binnengekomen bericht wordt getransformeerd naar Java-objecten/primitieven;
- Uit het bericht leest de RemoteServlet af welke methode op de MyServiceImpl-klasse moet worden aangeroepen;
- De betreffende methode wordt aangeroepen en geeft het resultaat terug;



Figuur 3: Schematisch overzicht van een RPC call naar een GWT-service.

- De RemoteServlet transformeert het resultaat naar JSON en geeft dit door aan de GWT proxy;
- De GWT proxy transformeert het naar een JavaScript-resultaat en geeft dit aan de aanroepende GW-module terug.

Integratie met JSF en Java EE

GWT maakt AJAX development dus aanzienlijk eenvoudiger. Betekent dit nu dat GWT voor alle webapplicaties de beste oplossing is? Het antwoord daarop is: nee! GWT moet dáár gebruikt worden waar het goed is: namelijk componenten waarbij de standaard frameworks afhaken en waar de ontwikkelaar zelf JavaScript moet schrijven.

De integratie met Java EE is relatief eenvoudig. Omdat een GWT-service leeft in een webcontainer, kunnen achterliggende Java EE-beans gemakkelijk worden aangeroepen. De GWT-services zijn dus niet bedoeld om de Java EE-sessionbeans te vervangen. De plaats in de architectuur van GWT-services kan daarmee vergeleken worden met de plaats van 'managed-beans' in JSF.
Integratie met JSF is iets complexer: standaard is dit namelijk niet mogelijk. Gelukkig biedt het framework Ajax4JSF (onderdeel van JBoss SEAM) uitkomst. Met Ajax4JSF kan op eenvoudige wijze aan een webpagina AJAX ondersteuning worden toegevoegd. Eén van de mogelijkheden van Ajax4JSF is het integreren van JSF met GWT.

De integratie van GWT en JSF wordt gedaan door de GW-module te wrappen in een JSF component. Concreet betekent dit dat er een nieuwe klasse aangemaakt moet worden die overerft van `javax.faces.component.UIComponentBase` en de interfaces `org.ajax4jsf.gwt.jsf.GwtComponent` en `org.ajax4jsf.gwt.jsf.GwtSource` implementeert. Na het aanmaken van deze klasse en wat configuratiewerk kan een GW-module worden gebruikt binnen een JSF pagina (zie codevoorbeeld 5).

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
...
    xmlns:gwt="https://ajax4jsf.dev.java.net/gwt"
     xmlns:myWidget="https://ajax4jsf.dev.java.net/
gwt/MyGwtWidget " ... >
  <ui:define name="content">
   <myGwtWidget:component id="gwtComponent">
     <gwt:gwtListener serviceBean="#{gwtHandler}"/>
   </myGwtWidget:component>
  </ui:define>
</ui:composition>
```

Codevoorbeeld 5: GW-module ingebed in een JSF pagina.

Door deze integratiemogelijkheid kan een GW-module goed als herbruikbaar component gemaakt worden. Seam gecombineerd met Ajax4JSF en GWT levert dus een applicatie waarbij

- standaard webpagina's gemaakt worden met JSF en Seam;
- eenvoudige dynamische delen gemaakt worden met JSF en Seam/Ajax4JSF;
- complexe componenten waarbij Ajax4JSF onvoldoende ondersteuning biedt, gebaseerd zijn op GWT.

Professional software development en GWT

Standaard levert GWT ondersteuning voor unittesten mee. Met de GWT-testcase-klasse is het mogelijk unittesten voor een GW-module te schrijven. Het voordeel daarvan is dat de unittest zowel in de hosted mode als in de web mode uitgevoerd kan worden. Anders gezegd: op zowel de Java-bytecode als de gegenereerde JavaScript kunnen unittests worden losgelaten. Dit is een belangrijk verschil met andere frameworks waarbij veelal de JavaScript die uiteindelijk gegenereerd gaat worden, niet binnen het bereik van eventuele unittests ligt.
Om de integratie van GWT met een automatische buildomgeving mogelijk te maken, is er een aparte Maven-plugin ontwikkeld: **maven-google-webtoolkit2-plugin**. Deze plugin zorgt ervoor dat tijdens een build de Java-code van het betreffende GWT-component door de GWT-compiler wordt gecompileerd naar JavaScript. Integratie met Ant is ook mogelijk.

Voor de IDE zijn een tweetal Eclipse-plugins beschikbaar. De eerste heet Cypal Studio (voorheen Googlipse) en biedt de mogelijkheid om op eenvoudige wijze nieuwe GWT-services en GW-modules te creëren en voegt tevens een runconfiguratie toe om GWT-applicaties te debuggen/runnen. De tweede plugin betreft een commerciële plugin: **Instantiations GWT designer**. Deze plugin biedt een goede schermdesigner. Ook voor NetBeans is inmiddels een plugin beschikbaar, die vergelijkbaar is met Cypal Studio: **gwt-4nb** . Liefhebbers van IntelliJIDEA behoeven geen plugins te downloaden omdat deze standaard GWT ondersteuning bevat.

Conclusie

In dit artikel is een aantal belangrijke oorzaken van veel voorkomende JavaScript problemen aan de orde gekomen. Naast de taal, vormen ook de toolsupport en de verschillende browsers een potentiële haard van problemen in JavaScript programmeren.
Google heeft met het Google Web Toolkitframework (GWT) een goede poging gedaan om de JavaScript-pijn zoveel mogelijk te verminderen. GWT is één van de weinige frameworks die al tijdens de compilatiefase Java naar JavaScript omzet. GWT biedt de mogelijkheid de kracht van Java en het Java-platform te gebruiken voor JavaScript-development. Omdat het programmeren in Java gebeurt, kunnen codekwaliteitstools ten behoeve van unittests, testcoverage en codestijl zonder meer worden toegepast op de GWT-Java-code. Naast de standaard IDE toolsupport is inmiddels ook een rijke set aan GWT-tools beschikbaar gekomen: Maven plugins, Eclipse plugins, designers en een NetBeansplugin.
Kortom, om JavaScript hanteerbaar te maken en om de JavaScript-kwaliteit beter te kunnen garanderen, is het inzetten van GWT een stap in de goede richting naar *Professional Software Development in JavaScript-land!* «

Voor degene die met GWT aan de slag wil, is het raadzaam om eerst de tutorials op de projectsite van GWT te lezen. Daarnaast is het aan te bevelen te beginnen met versie 1.5 RC1.Ten opzichte van versie 1.4 heeft zijn namelijk een aantal ingrijpende wijzigingen (bijv. Java 5 support) doorgevoerd.

Referenties:

- Google Web Toolkit
  http://code.google.com/webtoolkit/
- Echo3
  http://echo.nextapp.com/site/
- Jmaki
  https://ajax.dev.java.net/
- boss RichFaces
  http://www.jboss.org/jbossrichfaces/
- Jboss Seam
  http://www.jboss.com/products/seam
- Cypal Studio (Googlipse)
  http://www.cypal.in/studio
- Gwt4nb
  https://gwt4nb.dev.java.net
- maven-googlewebtoolkit2
  http://code.google.com/p/gwt-mav

Patches Patches Patches Patches Patches Patches Patches Pa

Artikelen over onderwerpen als software-ontwikkeling, Java, UML, eXteme Programming en nog veel meer vindt u in het Online Archief van Array Publications. Vaktijdschriften als Software Release, Java Magazine, Database Magazine en ons Oracle vakblad Optimize hebben hun artikelenarchief online gezet. Dankzij de heldere zoekstructuur vindt u snel wat u zoekt op www.release.nl.

Eerste versie van Java Gems project

Java Gems zijn simpele code snippets die van het ene naar het andere project kunnen worden gekopieerd, zaken die niet in java.util en subpackages zijn te vinden. Op dit moment beslaat het ondermeer een general purpose filtering interface en een CLI library implementatie.
Op de Serverside ontspon zich naar aanleiding van de aankondiging ervan een scherpe discussie, waarvan de teneur was dat het niet echt iets toevoegde aan bijv. Jakarta Commons. Eén iemand vroeg zich zelfs af: een deel van de dingen zal zeker zin hebben, maar het lijkt een trend om opensource-projecten te beginnen alleen maar om opensource-projecten te beginnen zonder werkelijk problemen ermee op te lossen. Op zich een valide vraag, waar we in één van de volgende nummers van Java Magazine verder op in zullen gaan.

IntelliJ IDEA 8 Milestone

Versie acht is nog niet af, maar JetBrains vond al wel dat er een mijlpaal bereikt was met deze versie. Nieuw is onder meer:
- JBoss Seam integratie
- Javascript/Flex Debugger
- Spring 2.5
- FreeMarker, Velocity, GWT 1.5
- Goede ondersteuning voor REST WebServices
- Struts 2
- Multi-dialect SQL Console
- Nieuwe automatische refactorings
- Verbeterde coding assistance
- Veel nieuwe code inspections

JDOInstruments 3.0

Het kent nu – naast andere verbeteringen en bugfixes, nieuwe "Automatic Crash Recovery" en "Schema Evolution" features. JDOInstruments is een embedded OO database geschreven in Java, het is ook een implementatie van Sun's Java Data Objects (JDO) specificatie voor de transparante persistence van Java objects.
Om die reden heeft het geen JDBC-driver of relationele database nodig. Het gebruikt zijn eigen object store en kan dus storage en retrieval van persistent data met weinig werk voor de ontwikkelaar uitvoeren.
Het is geïntegreerd met de Netbeans IDE (via een plugin). Het is geschikt om pure OO-systemen mee te bouwen. Het is gratis en kent de GNU LGPL licentie.

SmartInspect 3.0 logging tool vrijgegeven

SmartInspect 3.0 is een geavanceerd .NET, Java en Delphi logging tool voor debugging en monitoringvan onder meer high-performance productiesystemen.
Deze versie introduceert onder meer een nieuwe log server applicatie (SmartInspect Router), een high-performance named pipe protocol voor local live logging, asynchronous logging, de mogelijkheid log files te versleutelen en een nieuw en flexibeler log rotatie mechanisme.