Cloud and Web Development

Dr. Dobb's Journal

September 2012

Dart: Build HTML5 Apps Fast

Next

ALSO INSIDE

UBM TechWeb

www.drdobbs.com

Dr. Dobb's Journal

CONTENTS

September 2012

COVER STORY

More on DrDobbs.com

Building QuickBooks: How Intuit Manages 10 Million Lines of Code

Continuous integration and delivery based on automated software configuration management are the keys for developing a major retail product.

http://www.drdobbs.com/tools/240003694

Wall Street and the Mismanagement of Software

How Knight Capital becomes a knight errant when it came to software design and delivery.

http://www.drdobbs.com/architecture-and-design/240005196

Getting Going with Go

The first installment in our five-week tutorial on Google's new native language explains how to set up Go and build programs. It then walks through code examples that highlight some of the language's interesting features.

http://www.drdobbs.com/open-source/240004971

Understanding Core Data on iOS

Using a database in iOS requires careful navigation of the classes in Core Data. Here's a detailed look at how to use Core Data to create and use a simple database.

http://www.drdobbs.com/database/240004648

Parallel Evolution, Not Revolution

This guest editorial by James Reinders is a rebuttal to Andrew Binstock's argument that fine-grained parallelism will never become a standard part of programming skills.

http://www.drdobbs.com/parallel/240005407

Mailbag

Programming at a higher level than code, using variables to hold parallel logic, and checking-in only code that does not duplicate other code… letters this month proposed original solutions.

The One True Coding Style

In response to our editorial suggesting that there might be value in having language designers enforce a single coding style (http://www.drdobbs.com/240004664), we received:

Thank you very much for bringing up the need to control or standardize coding style. I have been coding a long time and I dread maintaining other people's code. It is amazing how illogical and inconsistent software folks can be! It seems that many think that making code needlessly complex and obtuse is a sign of great skill. They have no appreciation of the difficulty they are creating for the compilation or interpretation software. They also have no concern for the poor soul that has to decipher their junk.

— Rich Lovin

Mr. Charles Jacks scolded us for failing to suggest that programs be written in something higher-level than code, specifically abstract syntax trees (ASTs). We responded by asking whether he did not mean model-driven development, in which (typically) UML diagrams are drawn to reflect the software, and then code is generated directly from the diagrams. He responded:

I submit that the AST is a model. And the code is a view on that model that allows the model to be rebuilt. Storing the model and allowing each developer to have their own view solves the problem with picking one canonical style, or language for that matter. If you allow the AST to be mapped into multiple languages you should have a greater base of programmers to assign to any sub-problem. This abates the risk of too "few illuminati." If you allow the AST to be "pretty printed" into each developers style preferences, you solve the problem of lost time training a programmer into the local style, arguments over the style, and beating a programmer into submission when they err.

Note that pretty printers typically create an augmented AST and reverse translate that back into the "correctly formatted" layout. Let the IDE store the augmented AST instead of the input or output and let the programmer choose their preferred layout.

Consider that both C# and Visual Basic can be compiled into an object-oriented intermediate language (IM). The IM can also be reverse compiled into Visual Basic or C#, though that hasn't been an object of much development. The more languages and styles, views on the model as it were, the IDE was able to present, the greater the range of programmers and conceptualizations of solutions one allows. But this would just be phase one.

Normally a compiler takes a "stream of bytes," spiraled into a 2D layout for the programmers' convenience like a string around a cylinder, and rebuilds the model. The rebuilding process is thereby restricted to what can be expressed with a context free LL(k) or LR language. Essentially, we have become very adept at storing views instead of models and pay a price in the results.

But if we store an augmented AST, we can remove many restrictions. The AST can be rethought into an Abstract Semantic Graph. This would allow the full expression of context and the direct mapping of semantics.

— Charles Jacks
Atlantic Undersea Test and Evaluation Center
U.S. Navy

If the VCS runs the code through a formatter, then the VCS knows whether or not a version change has occurred. But let's take the game a little farther down the road. Why not use JIT technology (or compiler intermediate code) to classify identical code fragments and create a new business for IP forensics and super optimizer technology? Then only the comments and docs need saving in a language independent form.

[That] would be a laugh. The VCS says programmer A has checked in the same permuted 50 of 300 code fragments for the last two weeks, and furthermore, all 300 code fragments already existed in the departmental database!

If the goal is reusable code, why not insist only unique code go into the repository? I bet applications written with Android class libraries are getting close to the consistency required for that. Somebody will do it, using a brain-melting amount of CPU power courtesy of the cloud.

Personally, I can't wait to see a 12MB executable reduced to the 200KB it truly is. Of course, I am "kidding on the square," as the expression goes.

— Spencer Cathey

Getting Static Analysis Right

In response to our article on using static code checking, we received:

In the "Deploying Static Analysis" article (http://www.drdobbs.com/testing/240003801), the author cites a false positive rate of 20%. My own experience is that the false positive rate is much higher, typically in the 70–80% range. For instance, if one runs Coverity against a code base that made extensive use of the Factory pattern and the objects being created are Disposable, every single factory method that returns a Disposable will be flagged as a Memory Leak even though the code is exactly as the designer intended.

He also fails to assign a severity rate to the positives. In my experience, about 1/1000 positives are severe, 1/100 are serious, and 1/10 are moderate bugs. The rest, while being actual issues, may not have an effect that is visible to the end user.

It also depends greatly on where in the maintenance cycle the code is when you first run static analysis. In my experience, new code benefits more from static analysis than code that has been in maintenance for five years.

These above factors contribute heavily to the perceived value of static analysis. When people see that most of what they review and even fix doesn't make the product better from the user's perspective, the value of the static analysis process can be questioned. It is not just

politics and quality commitment that affect the decision to adopt these tools. The business case needs to be real. These tools are far easier to adopt, and bring the biggest value to new projects.

A big plus for static analysis tools that he does not mention is that these tools inherently drive a code-review process by requiring the ongoing maintenance of the tool identified "defects." Forcing someone to go back and look at the code a second time, even if it is to clean up false positives, is a good thing and can even result in bugs being found that static analysis did not find. The second reading is done with a purpose rather than simply being a review for reviewing's sake. This side effect of the tool making the code review process a process that requires developer activity beyond simply reading code may be one of the most important aspects of the adoption of these tools.

— Kevin English

Will Parallel Programming Ever Become the Norm?

In response to the editorial discussing whether fine-grained parallelism would ever be adopted (http://www.drdobbs.com/240003926):

RAM in current systems can only hold three types of numeric values: variables, operands, or addresses. What if there were a fourth item comprising logic in each memory location? What if a memory location were able to modify another memory location without the need for CPU code execution? Would not that be the ultimate in hardware parallelism? In 1992, I devised a theoretical system that allowed for this fourth attribute of memory. I referred to it as a DIAL (Data, Instruction, Address, and Logic) array. The normal CPU system would load the logic into the DIAL members to create the interaction with other memory or DIAL locations. The individual DIAL entity would be

loaded to contain a simple Boolean logic component that would work with the memory side as input and/or outputs. The element would also be able to change any bit or bits of any other memory or Dial element with a predictable latency. The DIAL logic, in general, would still function as the logic it was programmed to be, even if the main CPU did nothing. Imagine having 65-thousand electronic ANDs and ORs with the interconnections a fully programmable function. It was very insightful to me at the time through the endeavor of this design…but it was not easily implemented in silicon at that time (PALs were just starting to appear). I put it away, but always wondered "What if?" The real bottleneck was the bus that the DIAL members resided on and operated through. I suspect that is not such a big issue now. Imagine taking the logic of a non-programmed electronic system and loading it into such an array. The predictability in spite of stimulus applied would always be the same in terms of latency.

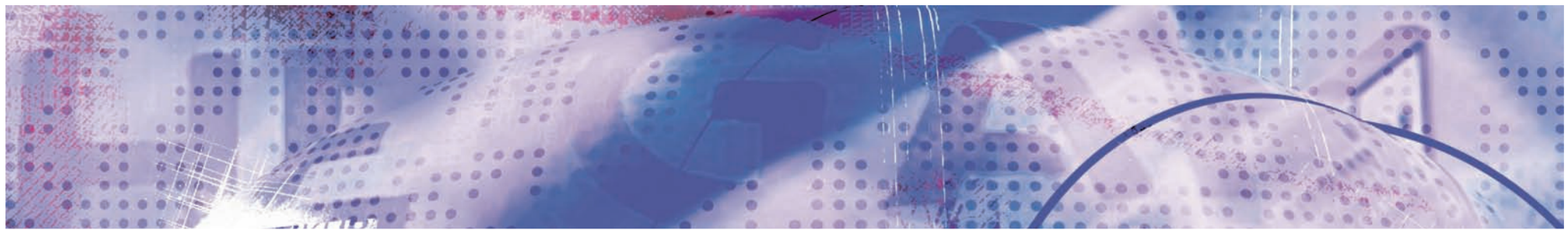
— Mark Cummins
Covance, Inc.

No Wonder I Knew That!

Thanks for your editorial "Coding From Within The Echo Chamber" (http://www.drdobbs.com/240001127) [*which describes how we unconsciously reinforce our own programming and coding beliefs –Ed.*]. It provided me with a number of interesting insights that I had missed on my own. "Epistemic closure" seems to provide a good explanation for an observation I've made over that past few years. Namely, that search engines, such Google, don't seem to provide me with as broad array of information on a search topic as they used to. I hadn't consciously realized that they are now pandering to what they think I want to see, rather than exposing me to "what I need to see" — a truly different idea that could cause me to think off in a different direction. How sad! That means they are slowly degrading the utility of the Internet and Internet search as an intellectual tool.

— Ray Rosich
Raytheon Company

Have a correction or a thoughtful opinion on *Dr. Dobb's* content? Let us know! Write to Andrew Binstock at alb@drdobbs.com. Letters chosen for publication may be edited for clarity and brevity. All letters become property of *Dr. Dobb's*.



Longing for Code Correctness

The longer I write code, the more I yearn for code correctness. Despite the work this extra step presents, commercial ventures, especially Wall Street, would do well to embrace it.

By Andrew Binstock

When I was young and first got into programming seriously, I reveled in working at the lowest levels of code. I loved (loved!) assembly language and could think of no greater happiness than doing clever things with registers and writing tight, fast code that squeezed into the minimum amount of RAM and delivered great performance.

As time passed, the lack of portability and the burden of learning the ins and outs of every new architecture got me interested C. In C, I thought I'd finally found the gateway to paradise: portable assembly language. What could possibly be better?

But after nearly 10 years of working in C on my personal projects, I tired of never being able to bring anything to a close, unless the project was a minor utility. Anything bigger never bore the full elements of my original ambitions. Feature lopping was a common practice. And even then more modestly resized projects, under unsparing hours in the debugger, would eventually be consigned to the groaning shelf of unfinished projects.

I had to move on. I went to Java, which drew me with its siren song of portability, performance, excellent tools, and enormous libraries. Not having to code one more linked list had remarkable appeal. So did the superior testing tools that would allow me to write code and test it quickly. This enabled me to avoid spending lots of time in the debugger. Ever since, Java has been my base language for personal projects, both large and small. (I am not implying that it will be my final destination. I continue to be troubled by the inconvenience of delivering software to non-developer users, who are not at all comfortable downloading or updating Java. A garbage-collected, portable, binary language with good libraries that is not as complex as C++ has great appeal to me. Which is certainly why I feel so interested in Go at the moment.)

As time has passed, though, I've found myself constantly frustrated by the feeling that no matter how much I test my code, I can't be sure with certainty that it's right. The best I can say is that the code is probably right. But when I write code for others to use, I want it to be completely reliable. This concern has led me to embrace tools that enforce the correctness of the code.

Long ago, for example, I adopted Bertrand Meyer's concept of design-by-contract (DBC), which suggests that every function should test for pre-conditions, post-conditions, and invariants. In Java, I do this with the excellent Guava library. So my methods tend to have tests, especially at the beginning where each parameter is carefully checked. I test invariants and post-conditions primarily in unit tests, which is probably not ideal, but it moves some of the validation clutter out of the code.

I'd like to go beyond this. First, I wish the languages provided better support for correctness. For example, I think floats should default to an uninitialized value of NaN (not a number). I wish integers had a NaN equivalent and that all primitive data types and strings had an invalid value to which they would be initialized by the language. A whole class of errors would go away. (In the absence of this, the Java compiler's ability to identify uninitialized values is undeniably helpful.)

But moving beyond these simple levels is where things become a lot more difficult. I have essentially two major options. The first would be to code in some kind of modeling language. Then use a tool, such as Conformiq's products or others, to generate thousands of tests. Then, automatically generate the code from the models and run all the tests. This is a highly effective way of writing solid code, and it is favored widely in Europe. However, I'm a coder at heart, and I'm not sure coding at a diagrammatic level is really my cup of tea.

The second alternative is to use languages that strongly support correctness. There are not many. Ada goes farther than Java in this regard. And Eiffel perhaps farther yet. But probably the one that does the most is Spark, an Ada subset. Developed in the U.K., Spark uses a system of annotations that are embedded in comments at the start of each method. The code can then be verified by assorted tools that check it against the annotations. Free and commercial versions of SPARK tools exist today.

Specifying code functionality in this way is undoubtedly an extra step with lots of additional typing involved. But it has advantages outside of proving correctness: To start with, it more fully documents the code. And, as Edmond Schonberg (formerly a professor of computer science at NYU and now an executive at AdaCore) mentioned to me in a pick-up conversation, by writing out the specifications before the code, most of the same benefits that tests deliver to TDD aficionados accrue to coders — the tests or specifications force you to spell out what you're doing before you begin to code. "Lots of defects get caught by this simple step," according to Schonberg. I believe it.

And I rue that in the U.S., only a few industries (mostly embedded, automotive, and avionics) are interested in high levels of code correctness. It's a topic that is dismissed as a luxury by mainstream programmers because it appears to interfere with the ability to deliver software quickly.

I have some sympathy for this point of view. If I were to indulge my desire for correctness more deeply, I would certainly want to have a scripting language as a secondary tool for testing out ideas and banging out code that doesn't need to be perfect. I don't want correctness to push me back to the days of coding in C when nothing was ever completed. However, as the recent Knight Capital disaster on Wall Street demonstrates, the general resistance to correctness felt by business programmers is a conceit that cannot always be accommodated without considering its occasionally dire consequences.

Comment

Dart: Build HTML5 Apps Fast

Dart is a language, library, toolset, and virtual machine from Google that greatly facilitates writing fast, interactive HTML5 apps without requiring you to be a JavaScript expert.

By Seth Ladd

Dart helps developers build fast HTML5 apps for the Web. Currently in Technology Preview (with a Beta release planned for this year), this open source project is building a "batteries included" developer platform that integrates a new language, libraries, an editor, a virtual machine, and a compiler (with JavaScript output).

History and Rationale

Before starting the Dart project, engineers Lars Bak and Kasper Lund were responsible for building V8, the original high-speed JavaScript engine now found in Chrome and used as the underlying technology in node.js. Their experience building high-performance virtual machines (including previous work on Java and Smalltalk VMs) led them to conclude that Web application performance and developer productivity could be significantly improved by introducing a new language and runtime for the Web. The result was the Dart project (http://www.dartlang.org).

The Web is advancing at a tremendous pace: For example, the Chrome browser is updated every six weeks to include the latest HTML5 features. The development cycle for Web programming is highly iterative, with the reload button acting as your compiler. However, it should be easier to build larger, more complex applications. It has taken far too long for productive tools to emerge, and there's still much more work to be done. Even though JavaScript engines are getting faster, Web app startup performance can be drastically improved. In addition, it should be easier to understand the program structure and work with larger teams. For large, complex Web apps, we need more structure and tools.

The Dart project was initiated to address these issues and to fulfill the needs of developers from all platforms, not just dedicated Web developers. The engineers on the Dart project want to make it easier to meet the new user demands of full-featured, beautiful, 60-FPS Web apps.

The Dart Language

The Dart language is designed for mass adoption, so it should be easy to use for a wide range of developers. It is an object-oriented language with classes, single inheritance, lexical scope, top-level functions, and a familiar syntax as shown in the following example.

Listing One: A small sample of Dart code.

```
class Point {
  num x, y;
  Point(this.x, this.y);
  num distanceTo(Point other) {
    var dx = x - other.x;
    var dy = y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
  }
}

main() {
  var p = new Point(2, 3);
  var q = new Point(3, 4);
  print('distance from p to q = ${p.distanceTo(q)}');
}
```

The code in this listing includes the following:

- Line 3: A constructor with handy sugar for the common `this.x = x` pattern found in constructors.
- Line 5: Omit the type annotation, because the editor can infer the local variable's type.
- Line 11: Every Dart program starts with `main()`.
- Line 14: Notice the string interpolation for easy string creation.

While keeping things generally recognizable, Dart also introduces a few features that haven't been seen in a mainstream programming language.

Optional Types

Optional static types are perhaps the most interesting feature of Dart. Instead of the "guilty until proven innocent" experience that you get from mainstream statically typed languages, Dart trusts the developer. Think of Dart's static types as annotations or documentation. Tools such as the editor can use the types to give you early warnings and errors. But you have the option of using `var` for dynamically typed variables when types can be inferred or when the developer can't express what they need to with a traditional type annotation. Gilad Bracha has provided more insight into Dart's optional static types at http://is.gd/sj9fE5.

Isolates

Isolates are another new feature of Dart. Experience has shown that shared memory threading is error prone and difficult to debug. To take advantage of multicore machines, isolates are provided to allow multiple and concurrent isolated memory heaps to run inside a single Dart application. Isolates communicate by passing messages over ports. These messages are copied before received, ensuring no state is shared.

Listing Two: How Dart uses isolates.

```
#import('dart:isolate');

 echo() {
   port.receive((msg, SendPort replyTo) {
```

```
    replyTo.send("Echo: $msg");
  });
}

main() {
  var echoService = spawnFunction(echo);
  echoService.call("Hello!").then((answer) => print(answer));
}
```

The code in this listing includes the following:

- Line 1: Import the isolate library.
- Line 10: Create an isolate for the echo function.
- Line 11: Send a message, wait for a reply, print the reply.
- Line 4: Receive messages to be echoed.

Use cases for isolates include safer Web app mash-ups, concurrency, and inter-application communication.

JavaScript Puzzlers Fixed by Dart

Dart also fixes some of JavaScript's more puzzling features. For example, if you are a JavaScript developer, you might know what the following code does. If you are not a JavaScript developer, you'll probably be surprised.

```
// JavaScript
var callbacks = [];
for (var i = 0; i < 2; i++) {
  callbacks.push(function() {
    console.log(i);
  });
}

for (var x = 0; x < callbacks.length; x++) {
  callbacks[x]();
```

This JavaScript code will print 2 and then 2. This confusion is fixed in Dart. The following code prints 0 and then 1, as you would expect:

```
//Dart
main() {
  var callbacks = [];
  for (var i = 0; i < 2; i++) {
    callbacks.add(() => print(i));
  }

  callbacks.forEach((c) => c());
}
```

Here's another example of surprising JavaScript behavior:

```
// JavaScript
> {} + []
0
```

This JavaScript snippet compiles and runs, and returns zero — probably not what you'd expect.

The Dart Editor gives you an early warning that the code won't compile; see Figure 1.

When the code is run, instead of a confusing zero, Dart generates an error indicating where the problem is; see Figure 2.

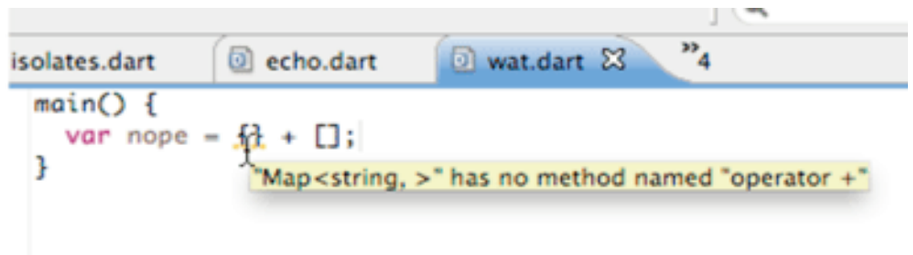


Figure 1: A warning in the Dart Editor that code won't compile.

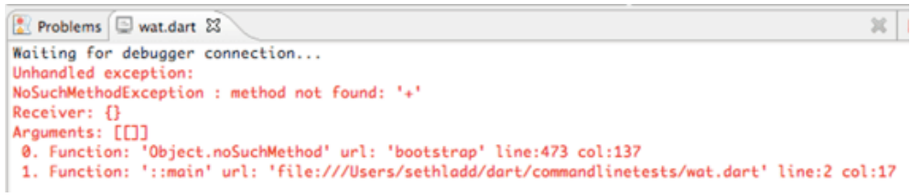


Figure 2: A warning in the Dart editor.

Another surprising JavaScript feature is function binding and the scope of `this`. Consider the following code:

```
// JavaScript
function Awesome(button) {
  button.addEventListener("click", function(e) {
    this.cool();  // 'this' is actually the button, so it fails
  });
}

Awesome.prototype.cool = function() {
  window.alert("ice cold");
}
```

Thanks to Dart's lexical scopes, code is more reasonable and behaves more predictably.

```
// Dart
#import('dart:html');

class Awesome {
  Awesome(Element button) {
    button.on.click.add((e) => cool());// Which cool?
                                       // Awesome's cool,
                                      // which is lexically scoped
  }
  cool() {
    window.alert("ice cold");
  }
}
```

Dart addresses more JavaScript puzzlers; this is just a sample. To be fair, ES.next (the next version of JavaScript) addresses some of these issues, but it's unclear when this new version will be supported by the majority of modern browsers. This situation means developers need to wait for the ES.next spec to be finished, as well as wait for ES.next and ES5 compilers to appear. Instead of waiting, Dart fixes these and offers a compiler for ES5.

Dart Editor and SDK

The Dart project ships a lightweight editor, which includes support for refactoring, debugging, code completion, code navigation, and more (Figure 3). The editor can run Dart programs in Dartium (Chromium with

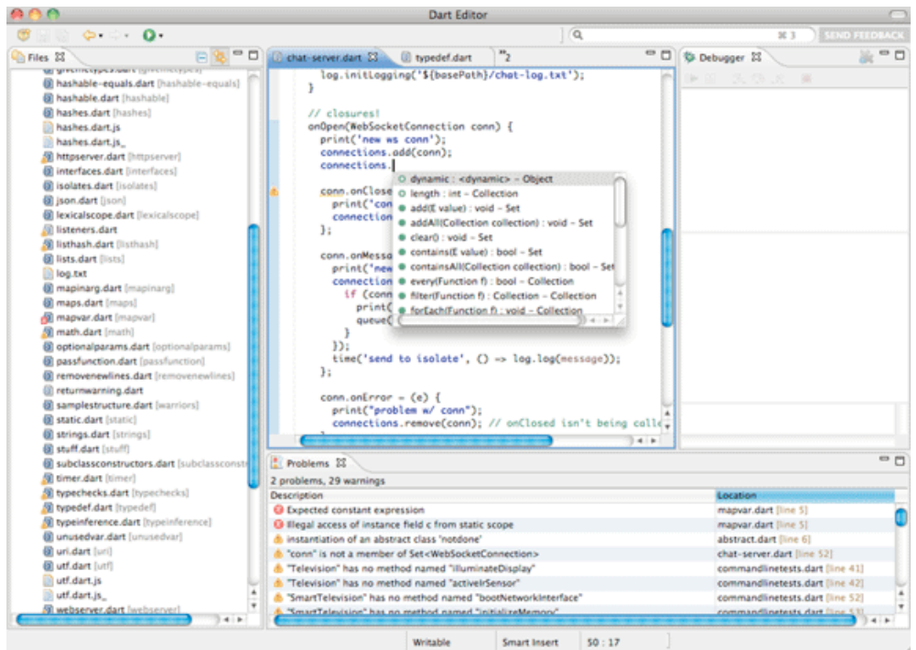


Figure 3: The Dart editor seen full screen.

an embedded Dart VM) and in the standalone Dart VM. Developers can achieve a fast edit-reload cycle using the editor, enabling the quick iterations Web developers appreciate. The editor is built on the Eclipse RCP framework, and it works on Mac, Windows, and Linux. An Eclipse plugin is now available.

The Dart SDK, which is bundled with the Dart Editor, contains three principal tools: the Dart VM (dart), the package manager (pub), and the compiler to JavaScript (dart2js).

The dart2js tool, a Dart to JavaScript compiler, is the key component that enables Dart apps to run anywhere on the Web. The compiler targets ES5 JavaScript, and is itself written in Dart.

Pub is the package manager for Dart. You can use pub to manage third-party dependencies for your Dart app. The pub tool currently supports downloading packages from git repositories, and will eventually support downloads from pub.dartlang.org.

You can run Dart apps on the command line using the Dart VM. Your app can use files, directories, and sockets in standalone Dart VM apps. Also included is an HTTP server and client, and a Web socket server and client. One of the goals of the Dart project is to enable end-to-end Dart applications, running on both the client and the server. You can download all the Dart tools from www.dartlang.org/downloads.html.

Other tool vendors are joining the effort, such as the Dart plugin for IntelliJ (http://plugins.intellij.net/plugin/?id=6351), built by JetBrains.

Learn More

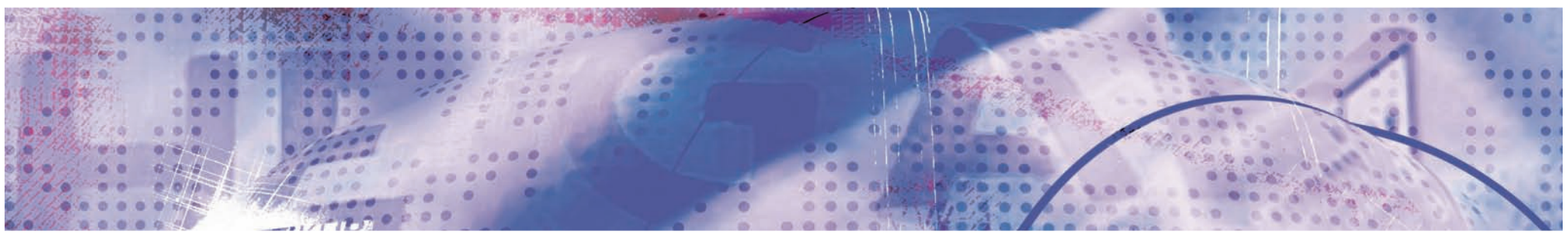
Dart is open source (http://dart.googlecode.com). We welcome discussion and feedback to the Dart mailing list (http://dartlang.org/mailing-list) where engineers and community members chat about the project. Documentation, videos, and downloads are also available at http://dartlang.org.

Some helpful Dart links include:

- The Dart Language Tour is a high level overview of the language: http://www.dartlang.org/docs/language-tour/
- The Dart Library Tour walks you through bundled libraries: http://www.dartlang.org/docs/library-tour/
- If you know JavaScript, you'll appreciate this mapping of JavaScript to Dart: http://synonym.dartlang.org/
- Download the Dart Editor and SDK: http://is.gd/26elOW
- Follow the project at +Dart on Google+: http://is.gd/nYPkzd

— Seth Ladd is a Developer Advocate with the Chrome team. He helps developers use Dart.

Comment



Previous Next

IN THIS ISSUE

Editorial >>
Dart >>
Windows Azure >>
Cloud Ecosystem >>
Letters >>
Links >>
Table of Contents >>

The New Service Levels of Windows Azure

Microsoft's embrace of the IaaS model in addition to its existing PaaS offering means that Windows Azure can now host all forms of operating systems and websites.

By Dino Esposito

Windows Azure is Microsoft's platform for building cloud-based applications. Since its first appearance, Windows Azure has offered a wide range of software services beyond pure computation, including storage, service bus, and access control. The APIs for these services enable companies to create ad hoc applications and host them in some Microsoft data centers using a pricing model that best suits their business needs. According to the classic taxonomy of cloud computing, Windows Azure falls in the category of Platform-as-a-Service (PaaS) cloud solutions.

From a business perspective, PaaS is just one side of the matter. A PaaS service binds you to some operating system and runtime environment. You can exert a limited degree of control over the runtime environment.

But what happens if you want to maintain total control over the software runtime environment (as you would in a traditional on-premise scenario) and just rent the infrastructure (such as equipment and virtual machine)? For a long time, such an Infrastructure-as-a-Service (IaaS) scenario hasn't been an option in Windows Azure, and companies couldn't buy cloud space from Microsoft just to host a LAMP application. The only way to deploy a solution on Windows Azure was to build a new application using the .NET Framework and cloud-specific services, storage solutions, and frameworks.

Recently, however, Microsoft extended Windows Azure by adding a few loudly requested services on top of the platform — virtual machines and websites. In doing so, Microsoft made its platform a lot more flexible and gave cloud architects the ability to build solutions that combine infrastructure and platform services. As a result, you can now bring an existing LAMP or Node.js web solution to the Windows Azure cloud and even build hybrid solutions that are partly on-premise and partly cloud-based and that use a variety of server technologies.

Windows Azure Virtual Machines

Windows Azure Virtual Machines (WAVM) allow you to create a virtual machine on a cloud host and move an existing virtual hard disk (VHD) there. You can pick up an existing disk image from a Microsoft-provided gallery or deliver your own customized image. You have full admin rights and can remotely manage your virtual machines at will. Virtual machines can be equipped with a variety of operating systems, including Windows Server and a few Linux builds, based on versions of OpenSUSE, Ubuntu, and a few others.

Without beating around the bush, the real breakthrough here is the opportunity to deploy a Linux-based image on the Microsoft cloud. I have personally helped a few customers assess how to integrate cloud storage and services in their applications and, in a few cases, we had to quickly and ruthlessly rule out Windows Azure because the client's application was half done (or just finished) and there was no time and/or budget to adjust or rewrite it for the PaaS requirements of Windows Azure. In my opinion, opening the Windows Azure cloud to Linux and to user-defined virtual machines represents a smart move that adds more credibility to the platform as a whole. With WAVM, you can now consider Windows Azure a full-fledged cloud platform and evaluate it with a focus on the benefits it can bring to your business rather than first checking whether it can even serve your own business needs.

Windows Azure Websites

In some circumstances, you may not need to create and deploy your own virtual machine in the cloud. Sometimes, all you need is the ability to quickly create a website based on the Microsoft web server stack

and quickly deploy it to the cloud, where it just works and is easy to manage and scale. Windows Azure Web Sites (WAWS) makes it easy to create such websites from Visual Studio 2012, then deploy them automatically to the Windows Azure cloud in a variety of ways. In particular, you can deploy from Git, TFS, or even via FTP. Likewise, you can access log files for your website via FTP or through a provided command-line tool.

"The real breakthrough here is the opportunity to deploy a Linux-based image on the Microsoft cloud"

WAWS doesn't end there, though. More and more Web applications result from a mix of server-side frameworks such as Node.js, PHP or perhaps WordPress, Umbraco, or DotNetNuke. You need to set up all these frameworks when you deploy. If you are going to deploy to the cloud, either you create a virtual HDD (or VHD) or you use the WAWS services. WAWS, in fact, comes with ready-made templates in an extensible gallery that work for popular CMS tools, a variety of server frameworks, and databases including MySQL.

However you build your website, regardless of the technologies and server applications you employ, you can likely deploy it to the Windows Azure cloud in a greatly simplified way and in the context of guided procedures.

Conclusion

There are many ways to approach the cloud. One is building from the ground up (or significantly restructuring) an application to live and operate in the cloud. In this regard, Windows Azure works well because it tightly integrates with the rest of the Microsoft stack. In many other cases, though, you just need a high-scale environment, elastic enough to fit your frequently changing needs. This IaaS approach to the cloud has been the first form of cloud computing to appear. And it wasn't addressed by Microsoft for years. Instead, customers with a classic ASP.NET or Linux-based application they wanted to "cloud-ify" had to look to other vendors such as Amazon.

The Windows Azure Virtual Machines addition is more of a necessary step to stay in the market than a way to offer more development power to customers. The bottom line is that Microsoft is catching up and recovering from a vision of the cloud that has been too partial for too long.

In other areas, Microsoft is moving bigger pieces of flagship products to the cloud such as TFS and Office. And the SkyDrive file system is boldly gaining ground and development support. It seems that a clear company-wide strategy is taking shape at Microsoft: Offering everything through the Windows Azure cloud except those few things that just need an "old-fashioned" desktop application running locally. This is what's really new with Windows Azure.

— Dino Esposito specializes in Microsoft technologies and contributes frequently to **Dr. Dobb's.**

Comment

From the Vault

Getting Started with The Cloud: The Ecosystem

Scalability, storage, security — this article serves as a primer on programming for the public cloud.

— *DDJ*

By Allen Holub

A friend recently reported a conversation he had with one of those wide-eyed, gee-golly developers who's half Techie and half Moonie. When asked what he was working on, the speaker came back with, "cloud cloud cloud cloud cloud cloud," and my friend said, "but, what if ...," to which the speaker replied, "cloud cloud cloud cloud cloud," to which my friend said, "but that won't work because...," to which the developer responded, "cloud cloud cloud cloud cloud" — and so it went. Many use "cloud" as a synonym for "good." Cloud architectures indeed have a lot going for them, but they're not a panacea, and you need to know what you're doing to jump to the cloud successfully.

This article discusses general cloud-related issues and looks specifically at the Amazon and Google cloud architectures. Subsequent articles will be more practical, delving deeply into code that comprises cloud-based applications, but let's start with some background.

What Is The Cloud?

First, what exactly does "cloud computing" mean? The term "cloud" dates to the early days of the Internet; back before domains existed (yes, there was such a time). An email address was essentially a route specified in what was called "bang notation." To send an email, you needed to know the name of every machine between you and the recipient. Here's a particularly nasty example that I pulled out of an old newsgroup post:

```
dog.ee.lbl.gov!ucbvax!cis.ohio-state.edu!zaphod.mps.ohio-
state.edu!qt.cs.utexas.edu!cs.utexas.edu!utgpu!utzoo!sq!msb
```

The "dog" and "msb" are the sending and receiving machines. The rest is the route from one machine to the other. The routes didn't have to be fully specified — most email systems knew about a handful of major hubs, so a minimum-length address just specified a route from that hub to you — but there was zero flexibility.

Things changed with the introduction of domains. Instead of an explicit route, you sent an email to a gateway machine, and the email's recipient got the mail from a different gateway machine. The servers through which the mail passed on the way from one gateway to the other were anonymous, and the network topology was unknown. The word "cloud" was coined to describe that amorphous network. You didn't know what went on inside the cloud (and believe me, you didn't want to). As long as the mail ended up at the right place, everything was copacetic.

So, here's my rather strict definition of "cloud": A network of computers in an unknown topology, arranged in such a way that you don't need to know anything about this network except how to talk to a machine at the edge.

A "cloud application" is then an application deployed to the cloud itself, not to a specific machine. The application could be running on one or more machines that may or may not be physically collocated. Its data store could also be distributed, and may not be on the same machine as the application.

In my own mind, I see a difference between cloud applications and Web 2.0 or AJAX applications, which we used to call "thin client" applications. Here, the UI is a standalone program (typically written in JavaScript and running in the browser), and it talks to a server (which is primarily a data repository) using HTTP.

Web 2.0 applications are typically implemented using a traditional client-server architecture (one server hosted on an ISP talking to multiple browser-based clients). However, there's no reason why you can't have Web 2.0 cloud applications. In fact, that's most likely the way that all applications will work five years from now. It's useful, however, to separate the concepts in your head. Most current Web 2.0 applications are not cloud based.

What Difference Does It Make?

So, why would you want a cloud application instead of a simple client/server arrangement? Consider the following ping times:

```
> ping www.google.cn
PING www.google.cn (203.208.37.99): 56 data bytes
64 bytes from 203.208.37.99: icmp_seq=0 ttl=239 time=273.340 ms
64 bytes from 203.208.37.99: icmp_seq=1 ttl=239 time=478.394 ms
64 bytes from 203.208.37.99: icmp_seq=2 ttl=239 time=421.920 ms
64 bytes from 203.208.37.99: icmp_seq=3 ttl=239 time=343.003 ms
64 bytes from 203.208.37.99: icmp_seq=4 ttl=239 time=263.843 ms
64 bytes from 203.208.37.99: icmp_seq=5 ttl=239 time=482.231 ms
...
```

The round-trip time between my desk in Berkeley, California, and one of Google's servers in Hong Kong ranges from a bit over a quarter to almost half a second, and we're sending only 56 bytes. There's essentially no server overhead, but we're hostage to both distance and the speed of the routers through which the data is passing. (A tracert reports only 17 hops, so the latency is probably all distance.) The picture is different when the server is close by. Here are the results from Berkeley to San Jose (12 hops):

```
> ping google.com
PING google.com (74.125.224.52): 56 data bytes
64 bytes from 74.125.224.52: icmp_seq=0 ttl=54 time=19.815 ms
64 bytes from 74.125.224.52: icmp_seq=1 ttl=54 time=20.466 ms
64 bytes from 74.125.224.52: icmp_seq=2 ttl=54 time=35.547 ms
...
```

A cloud application (or at least the instance of the application that we're talking to), would ideally be running on the machine with the best access time. That's the main advantage — the cloud can effectively reconfigure itself to take care of pesky details like network latency.

However, there's actually no way to guarantee that this reconfiguration will actually happen, which brings us to the dark underbelly of a cloud app: We need to program for the worst case.

Imagine a cloud app that's doing some kind of word completion. Every time you type a character, it's sent off to a server, which finds words prefixed with whatever you've typed. The server sends back a list of possible matches, and your program displays these. Most of the cloud books, in fact, demonstrate this sort of thing in exactly that way — the local application talks to the server with literally every keystroke. Given the look-up times, etc., your user isn't going to be particularly happy with your worst-case response time. You can, however, rethink your strategy. When the first few characters are typed, the server could send you a large, perhaps exhaustive, list of every word that could possibly start with those characters. Thereafter, the application can use that list to update its display rather than going back to the server with every key press. By eliminating the redundant network queries, we make the application much more responsive.

On the plus side, the cloud is amorphous and can indeed reconfigure itself based on observed load. If Google notices that there is a lot of traffic between Berkeley and Hong Kong, it may well replicate the Hong Kong server somewhere in California, and the latency would suddenly improve. The same applies to your cloud application: It will, ideally, be running on several geographically distributed servers, with the topology scaling to accommodate actual requests. In other words, the size of the network (and your cloud-services provider) matters. For cloud services to be effective, the provider has to be large. If you deploy to the Google or Amazon infrastructure, you're effectively leveraging the flexibility inherent in a very large network. By my rather strict definition, an application running on a single server, whether it's an ISP or so-called cloud host, isn't a cloud application at all because it looses the scalability and flexible topology of a true cloud infrastructure.

Scalability and Cloud-Service Architecture

A significant advantage to a cloud infrastructure is automatic scalability, but here's one place where the basic architecture matters. Amazon's Elastic Compute Cloud (EC2), like most cloud providers, rents you a "virtual machine" to host your application. Your VM may or may not share a physical machine with other apps, and it has an unknown number of physical processors attached to it. At its heart, though, your EC2 VM is just a Linux (or Windows) box, and you can configure it however you want. You typically pay only for the time that the VM is actually busy doing something, which is great for a software startup that's effectively getting rack space for free. As the load increases, so do your expenses (but hopefully, so does your revenue). You use your VM pretty much the same way you'd use a shell login to a shared server at an ISP, deploying with FTP, etc.

The inherent flexibility of a hosted-VM approach is particularly important the day that your application gets reviewed in *The New York*

Times, and suddenly you have 1000 hits/second. Your ISP-hosted shared server would just crash at this point. An EC2 VM will scale, however, running on a dedicated machine if necessary, with cores added as necessary. Amazon will automatically increase the "umph" of your VM — giving you more machine cycles on the physical machine, for example, or assigning more cores to your application. Of course, you'll pay for this extra umph.

"A significant advantage to a cloud infrastructure is automatic scalability"

The main downside of this approach is that there is an effective upper limit on the scalability. Adding cores can get you only so far, and there's a diminishing return on the number of cores. Eventually, you're using everything that the machine can give you. What if that's still not enough to handle the volume? In theory, your app can be placed on several machines at this juncture, with Amazon handling the load balancing (you can run several EC2 VMs in separate physical locations that you specify), but that scaling doesn't happen automatically, and the app has to be written with scaling in mind.

That is, if you're really planning on scaling, you have to do exactly the same amount of programming work that you'd do if you were running the application on multiple machines in your own data center. This is a nontrivial amount of work. So, Amazon and its brethren give you a lot of flexibility in configuration. You can put anything you want on your virtual Linux box, write your app in any language, augment it with custom processes; go crazy! The downside is that you have to worry about administration and scaling, and that can add to the complexity (and cost) of the application very quickly.

Fortunately, there is another approach — the one used by the Google "App Engine." Google doesn't rent you a VM at all — you have no control of the operating system and can't install arbitrary applications on "your" machine. Instead, you rent time on a virtual application server (think Tomcat). You write your application in an approved language (Python or Java) and you deploy your application directly to Google's app server, not to the operating system. For example, if you're using Java, your application is a standard Java "web app" packaged into a WAR file and deployed to Google exactly the same way that you'd deploy to a Tomcat instance, by uploading the WAR. Google handles the Tomcat part. (It's not actually using Tomcat, but I usually test locally using Tomcat and haven't found any problems. Google's own development tools use Jetty.) Part of deploying the app is telling Google what URL to use to access it. You can use both a Google-provided URL (something.appspot.com), or a subdomain of your own domain.

I personally prefer the Google approach for several reasons. First, I hate doing system-administration work. Since Amazon just gives me a box with an OS on it, I'm forced into that role if I use EC2. Google, on the other hand, does all the SA work for me. All I need to worry about is my application. Second, Google's virtual application server can, itself, scale and take my application along with it. For example, the app server could, at least in theory, run on multiple machines simultaneously in the same way that you can cluster Tomcat instances. Scaling, then, is

in no way limited by the number of cores or speed of a single machine. As a consequence, my application can be vastly simpler, since I don't have to deal with the scalability issues in the source code. Finally, Google provides a rich set of development tools (mostly Eclipse extensions) that ease development considerably, though many of those tools will work with EC2 as well. For example, the Google Web Toolkit (GWT) provides you with a way to build a browser-agnostic AJAX front end in Java. GWT includes a Java-to-JavaScript compiler that translates your code into platform-independent JavaScript when it's time to deploy — but when you're developing, it's all Java. That means that you

"The Google Web Toolkit (GWT) provides you with a way to build a browser-agnostic AJAX front end in Java"

can use the Eclipse debugger on both the client and server side, trace execution from client to server, etc., all within a single development environment. GWT applications will even run fine under Tomcat on an EC2 instance. I can develop much faster with GWT than I ever could when I was writing JavaScript by hand.

On the downside (to paraphrase Henry Ford): Your app can come in any color, provided that it's black. Your choice of implementation language is Java (my own predilections preclude writing an enterprise application in Python). You have to structure your application as a Java web application, built around servlets, and you have to access your data using JDO or JPA (there's no JDBC support).

Google's working on adding SQL (due to be released within the next few months at the time of writing), but it's not there yet, and is available only to "App Engine For Business" customers. Unfortunately, Google's pricing model for the "For Business" customers effectively makes SQL inaccessible to a standard web application meant as a public Software-as-a-Service (SaaS) app. Google charges $8 per year per user for a "For Business" application, which makes sense if you're implementing your HR application on Google instead of running it in your own data center. But a per-user fee is nonsensical if you're writing a SaaS app to expose to the entire Internet. The standard (not "For Business") App Engine charges are based on CPU and data usage, not the per-user model. Google has made similarly stupid (a technical term we analysts use) choices on other fronts as well. For example, a standard App Engine application can use SSL only if you deploy the page to a Google URL (*MyDomain.appspot.com*), which could be disconcerting to one of your users if they look at the address bar. Similarly, though you can host subdomains on the Google App Engine, you cannot host your main domain on Google. You have to get an account with a standard ISP, and then redirect access to a Google-hosted subdomain. (For what it's worth, *www.foo.com* is a subdomain, so it can be hosted on Google. It's the *foo.com*, without the *www*, that's the problem.)

Amazon EC2, on the other hand, gives you several database choices: You can run an RDMS on your VM, you can use Amazon's Relational Database Service (RDS), or you can use Amazon's SimpleDB service if you're doing something very simple. You can easily host your domain on an EC2 instance, and you can easily access that domain using SSL (because you're just accessing your own instance of Apache, running on the VM).

So, to sum up the differences before moving on to other issues: Google provides a better programming environment, with easy deployment, and very good scalability; but, Google's services are

marred by an inability to easily host your top-level domain, inability to use SSL with your domain's URL, and lack of SQL support. The last two can be resolved if you're an "App Engine For Business" user, but the pricing model for that service effectively makes it useful only for large companies who want to move in-house applications from their own data centers to Google, something that I have a hard time believing will happen. Amazon has none of those particular problems, but system administration is difficult with EC2, and scalability is not fully automated. It's the scalability issue that's the show stopper for me, so I'm using the Google App Engine in spite of its limitations.

Storage

Before finishing up, there are a few other issues that you need to consider that are just part and parcel of cloud applications, regardless of the provider.

First of all, you need to rethink your relationship with your data. Data may or may not be replicated by the service, and it may or may not be stored on the same physical machine (or on the machine that's running the service that accesses the data). Securing the data can be a significant problem. You can encrypt sensitive information like credit-card numbers, but encrypting everything is not a practical solution because you can't issue queries on encrypted data. More to the point, if you're doing the encryption, then there's always a point at which the data is unencrypted and your encryption key is in plain sight. If your application is running on someone else's server, you're potentially vulnerable. Of course, a dedicated VM is less vulnerable than a shared server at an ISP that allows shell access, but you'll never be as secure as you would be in your own data center.

Also, bear in mind that your data is certainly replicated on many servers. This organization can be an important performance enhancer. Consider a video-streaming application that stores its data in the cloud. The actual application may be in only one place, but once we start streaming, we hope that we're connected to the version of the data with the fastest transmission time. However, more servers in more data centers means more vulnerability. And every cloud-service-provider employee who has administrative access to a physical cloud server also has access to your data, so your vulnerability is replicated along with the data.

There's also the issue of back up. A cloud provider may or may not actually back up your data. Google, for example, does tape back ups of gmail, but as far as I know, does not back up its general storage system. Feel free to correct me if I'm wrong, but practically speaking, cloud applications have to assume that there's no underlying backup mechanism. The data is replicated on many servers, and the servers themselves use RAID drives (if they have disks in them), so it would take a global catastrophe to loose your data altogether, but it's not possible to go back in time as you could do using a back up tape. You can write a web application that transfers data from Google to your own local repository, but that's actually a surprisingly difficult (and painfully slow) operation to perform using JDO/JPA, which is the only access method that Google provides.

Security

The biggest problem with cloud applications is actually application security. Security is, of course, a huge problem with most software. Pro-

grammers are simply unaware of how to write secure applications, and management is typically unwilling to spend the paltry sum required for the training that would eliminate 90% of the problem. It's symptomatic that most of the really big breaches I've read about in the past few years have been done using SQL Injection, which is not only a venerable, well-understood exploit, but is literally a trivial matter to defeat. This is a case where 10 minutes of training could eliminate millions of dollars of vulnerability, but the training still doesn't happen.

One of the problems is perception. Application security has nothing at all to do with things like firewalls and SSL. The IT department is simply not involved. Most exploits attack an application through a bug of some sort, usually a minor one. And most hackers get at that bug simply by using the program in predictable ways, just like all your other users do. There are no secret back doors, and the vulnerabilities are typically in plain sight. For example, the classic example of a SQL-injection exploit that you find in the books shows you how to get a dump of someone's entire database by exploiting a very-simple bug in a website's password-recovery page (access to which doesn't typically require a password, I would hope). It doesn't matter whether you access that page using HTTPS; if you can access the page at all, you can do the damage.

So, the biggest security problem with a cloud-based Web 2.0 application is the size of the attack surface — the number of places where a hacker can potentially use a bug to break into the system. Most Web 2.0 applications use remote procedure calls, or an equivalent mechanism like REST, heavily; and every one of those calls — in fact, every argument to every one of those calls — represents a potential vulnerability. It's easy to deal with these problems if you know about them (e.g., check that all arguments are valid and reasonable on both the client and server side; if you're using Google's Web Toolkit, you can literally use the same Java code on both sides to do the checking).

The real solution to this problem is simple: training.

The Entire Ecosystem

There are a few loose ends to cover. First, Google provides a reasonably rich set of support services for your web application. Get complete details at http://code.google.com/appengine/docs/java/apis.html, but here's a list of functionalities to be aware of. I'll demonstrate a few of these in future articles.

- **Blobstore**: Lets you store very large objects that can't be handled by the standard JDO mechanism, and serve them directly to your users if you like. It's handy for things like big images.
- **Capabilities**: A management API that lets you dynamically detect whether other Google services are operational. You can use it to disable features of your own app when a Google service on which it depends goes down for maintenance.
- **Channel**: A mechanism for pushing information down to a browser-based client, so that client can update itself without polling.
- **Images**: A library for doing simple image manipulation. Supports transformations like resizing, rotation, darkening and lightening the image, etc.
- **Mail**: Lets you both send and receive email from your application. You send using standard JavaMail APIs. You receive by writing a servlet that waits for email to arrive. (Google receives the email, then posts it to your servlet.) This facility is useful, but Google limits the number of emails that you can send in a day to 500, so you can't use this service for mailing lists or bulk mail.
- **Memcache**: A mechanism for caching chunks of data in "mem-

ory." This is a wrapper around Java's JCache APIs. Caching through this API is better than rolling your own cache because Memcache can scale properly if the application is running on multiple machines.

- **Multitenency**: Effectively adds namespaces to the storage system so that you can partition your data easily.
- **OAuth**: Provides a mechanism to grant third-party access to Google services. For example, a customer of yours could use this mechanism to allow your application to access his Google Docs files for persistent storage.
- **Task Queues**: Allows your application to execute background tasks that are not necessarily triggered by a user action.
- **URL Fetch**: A wrapper around java.net.URL and related classes that lets you access other web content using URLs. Handy for doing things like sending bulk email from a non-Google server.
- **Users**: Allows you to use Google's login mechanism for your application. That is, one of your users can log in to your application using Google's login page. I have mixed feelings about this service because it's one of the few services that doesn't just implement a standard Java library. If you use it as your sole log-in mechanism, you're effectively giving your user list to Google, and I'd rather know who my users are, thank you.
- **XMPP**: Support for XMPP-compatible IM services (like Google Talk).

APIs

The final thing to think about are the services that can coexist with your web application. Google, under the moniker "GData," provides API access to literally all of its web applications — from Calendar to YouTube, making it easy to do things like integrate a Google Docs page into your application or update an appointment on a Google Calendar. You can find the complete list of APIs at http://code.google.com/apis/gdata/docs/directory.html. Most of these are simple REST-based APIs. You typically encode a request in the URL and HTTP GET or POST, and receive a result in JSON. However, Google provides both Java and Python libraries that wrap the REST APIs, and it also provides an Eclipse plug-in that makes it easier to write to the APIs. I'll talk about how to use these APIs in future articles.

Related Articles

Getting Started With the Cloud: Logging On With Google OAuth:
http://www.drdobbs.com/web-development/229625374

Getting Started with Google Apps and OAuth:
http://www.drdobbs.com/web-development/229401853

Getting Started with the Cloud: Amazon Web Services:
http://www.drdobbs.com/web-development/231601598

— Allen Holub provides technical training, OO design and Agile-process consulting, and web-application/SaaS development services. He is the author of **Holub on Patterns: Learning Design Patterns by Looking at Code**, ***C+ C++*: Programming With Objects in C and C++**, *and numerous articles for* **SD Times**, **JavaWorld**, *and IBM Developer Works. Contact him via http://www.holub.com/contact.*

Comment

This Month on DrDobbs.com

Items of special interest posted on www.drdobbs.com over the past month that you may have missed

GO TUTORIAL: OBJECT ORIENTATION AND GO'S SPECIAL DATA TYPES

In this second installment of our five-week course on the Go language, we explore Go's unusual approach to object orientation, its special built-in object types, the syntax for multiple return values, and how Go handles exceptions.

http://www.drdobbs.com/240005402

CONTRACTS CHANGE THE DESIGN OF METRO APPS

Bringing the "simpl"" back to Windows application development

http://www.drdobbs.com/windows/240003942

EXERCISE WHILE YOU CODE

Interview with Andrew Reese, who works in secure program coding, and is the grandson of the inventor of the Reese's Peanut Butter Cup

http://www.drdobbs.com/tools/240004886

JOLT AWARDS: UTILITIES

The Jolt judges combed through more than 40 products to find the very best developer utilities. We now reveal the Jolt Award winner and the runners-up.

http://www.drdobbs.com/joltawards/240004295

IMPROVING FUTURES AND CALLBACKS IN C++ TO AVOID SYNCHING BY WAITING

In C++, futures are a great way of decomposing a program into concurrent parts, but a poor way of composing those parts into a responsive and scalable program. Microsoft's Parallel Pattern Library (PPL) provides a solution using tasks.

http://www.drdobbs.com/parallel/240004255

THE NEED FOR ONE TRUE CODING STYLE

More unproductive coding time is wasted on coding style disagreement than any other issue. Language designers are finally starting to take notice and enforce a preferred style.

http://www.drdobbs.com/architecture-and-design/240004664

C++ PRIMER 5TH EDITION, PART 4: WHAT MAKES A GOOD EXAMPLE?

A useful example of a language feature should give the reader an idea of how — and why — one might use that feature in practice. It doesn't have to be complete, but it does have to be detailed enough to let readers understand how they might complete it. In the case of a copy constructor, there are some additional requirements.

http://www.drdobbs.com/cpp/240005166

IN THIS ISSUE

Dr. Dobb's

Andrew Binstock Editor in Chief, Dr. Dobb's
alb@drdobbs.com

Deirdre Blake Managing Editor, Dr. Dobb's
dblake@techweb.com

J. D. Hildebrand Associate Editor, Dr. Dobb's
jdhildebrand@drdobbs.com

Amy Stephens Copyeditor, Dr. Dobb's
astephens@techweb.com

Sean Coady Webmaster, Dr. Dobb's
scoady@techweb.com

Jon Erickson Editor in Chief Emeritus , Dr. Dobb's

CONTRIBUTING EDITORS

Scott Ambler
Mike Riley
Herb Sutter

DR DOBB'S
UBM TECHWEB
303 Second Street,
Suite 900, South Tower
San Francisco, CA 94107
1-415-947-6000

Dr.Dobb's Business Contacts

INFORMATIONWEEK BUSINESS TECHNOLOGY NETWORK

EVP of Group Sales, InformationWeek Business Technology Network, Martha Schwartz
(212) 600-3015, mschwartz@techweb.com

Sales Assistant, Salvatore Silletti
(212) 600-3327, ssilletti@techweb.com

SALES CONTACTS—WEST

Western U.S. (Pacific and Mountain states) and Western Canada (British Columbia, Alberta)

Sales Director, Michele Hurabiell
(415) 378-3540, mhurabiell@techweb.com

Strategic Accounts

Account Director, Sandra Kupiec
(415) 947-6922, skupiec@techweb.com

Account Manager, Vesna Beso
(415) 947-6104, vbeso@techweb.com

Account Executive, Matthew Cohen-Meyer
(415) 947-6214, mmeyer@techweb.com

MARKETING

VP, Marketing, Winnie Ng-Schuchman
(631) 406-6507, wng@techweb.com

Marketing Director, Angela Lee-Moll
(516) 562-5803, aleemoll@techweb.com

Marketing Manager, Monique Kakegawa
(949) 223-3609, mluttrell@techweb.com

Director, Client Marketing, Michelle Somers
(516) 562-7928, msomers@techweb.com

SALES CONTACTS—EAST

Midwest, South, Northeast U.S. and Eastern Canada (Saskatchewan, Ontario, Quebec, New Brunswick)

District Manager, Steven Sorhaindo
(212) 600-3092, ssorhaindo@techweb.com

Strategic Accounts

District Manager, Mary Hyland
(516) 562-5120, mhyland@techweb.com

Account Manager, Tara Bradeen
(212) 600-3387, tbradeen@techweb.com

Account Manager, Jennifer Gambino
(516) 562-5651, jgambino@techweb.com

Account Manager, Elyse Cowen
(212) 600-3051, ecowen@techweb.com

Sales Assistant, Kathleen Jurina
(212) 600-3170, kjurina@techweb.com

AUDIENCE DEVELOPMENT

Director, Karen McAleer
(516) 562-7833, kmcaleer@techweb.com

BUSINESS OFFICE

General Manager, Marian Dujmovits

United Business Media LLC
600 Community Drive
Manhasset, N.Y. 11030 (516) 562-5000