Dart Code Lab

Kyoto GDG

2012/09/29

目 次

1 Introduction

This codelab will help you build and run a simple chat application using Dart. Along the way, you will learn:

- The fundamentals of the Dart programming language
- How to use the Dart HTML libraries
- Bi-directional communication with WebSockets
- The basics of Dart Editor
- How to find answers from the online resources

1.1 Prerequisites

This codelab assumes you are familiar with fundamental web programming. You should know the basics of HTML, CSS, and JavaScript. You should also have experience with a programming language such as Java, C#, or JavaScript.

This codelab assumes you have watched the following video tutorials:

- Google I/O 101: Introduction to Dart with Seth Ladd
- Google I/O 101: Dart Editor with Devon Carew

1.2 Installs

This codelab requires the Dart Editor. You can find builds of the editor for Mac, Windows, and Linux at `http://dartlang.org/editor/`.

1.3 Additional materials

This codelab provides easy to follow, step-by-step instructions. However, you 'll find it quite useful to have the following online resources loaded up and ready to access.

- `http://api.dartlang.org` The Dart API docs list the functions, classes, and methods for all the libraries.
- `http://www.dartlang.org/docs/language-tour/` The Dart Language Tour is your high-level guide through the Dart language.
- `http://www.dartlang.org/docs/library-tour/` The Dart Library Tour covers most of the bundled libraries of the Dart platform.
- `http://synonym.dartlang.org/` Know JavaScript? This is your resource! Translate common idioms, patterns, and snippets from JavaScript to Dart.

2 Step 1: Set up your environment

Dart offers better productivity through powerful and helpful tools. At the center of the toolchain is the Dart Editor, a lightweight text editor that understands how to analyze, run, and debug Dart apps. The editor works with the Dart SDK and Dartium (a build of Chromium with the Dart VM) to give you an integrated experience.

2.1 Objectives

1. Install Dart Editor 2. Send feedback to the editor team 3. Run the sample clock Dart app 4. Learn about Dartium

2.2 Walkthrough

2.2.1 Install the Dart Editor

To get your environment set up, plug in the provided USB. Open the USB drive and find the `editor/` directory inside. Copy over the correct Dart Editor version for your OS/bit combination directory to your machine, and unzip it.

[Image]

Open up the newly unzipped `dart/` directory, and double click the executable:

- DartEditor.app (Mac)
- DartEditor (Linux)
- DartEditor.exe (Windows)

[Image]
When Dart Editor first opens, the Welcome window appears.
[Image]

2.2.2 Use the Send Feedback button

The Dart Editor team really appreciates feedback. The easiest way to let them know your thoughts is to use the Send Feedback button in the upper right of the editors toolbar.

Locate the Send Feedback button and click on it.

[Image]

The Dart Editor Feedback dialog allows you to share bugs and requests directly with the editor team, as well as the larger Dart team. Feel free to send us any and all comments, especially during this codelab. Well turn your suggestions into bug reports and feature requests3 as appropriate.

[Image]

2.2.3 Run the Clock sample

(If the feedback window is still open, close it now.)

Time to run a Dart app!

Click on the Clock sample from the Welcome window, which copies the Clock sample into Dart Editor and sets up a new project for you.

[Image]

Tip: Cant find the Welcome view? You can go to Tools, Welcome Page to display it again.

[Image]

The Files view shows all of the files in the Clock sample, including all Dart files, the HTML file that hosts the app, as well as all of the images and CSS files. clock.dart is a Dart file that defines the "clock" library,

and includes the main() method for the sample. The clock.dart file is automatically opened in the editor.

[Image]

Ensure the clock.dart file is selected and highlighted. Click the Run button in Dart Editor, which launches the application by loading Dartium and pointing it to the clock.html file.

[Image]

Here is the Clock sample app running inside Dartium. Congratulations, you are running your first Dart app!

[Image]

2.2.4 Dartium

Dartium is Chromium with an embedded Dart virtual machine (VM). Even though Dart compiles to JavaScript, you can speed up the "edit, reload" development cycle by running Dart apps directly in the browser.

To verify that Clock is running in Dartium, right-click in the browser and select Inspect Element.

[Image]

The Elements tab should be selected by default, and clock.dart should be listed as the script that is being run.

[Image]

2.2.5 Debug with the Dart Editor

With Dart running directly in Dartium, the editor has debugging support for Dart applications. Set a breakpoint by double clicking in the left hand gutter in Dart Editor on first call to setDigits() inside the updateTime() method in clock.dart, line 66. With the breakpoint set (you will see a little blue dot in the gutter), click the Run button again.

[Image]

Notice how the program stop, and the Debugger view opens in the Dart Editor. To continue the program without leaving the debugger, click the Resume button (the green arrow in the Debugger view). The updateTime() method is called every 1000ms, therefore the breakpoint will be hit again within a second.

[Image]

The Debugger view on the right hand side allows you to see which processes are running and what values are in scope at the breakpoint. Hover over the now field, the value will be displayed in a tooltip.

[Image]

To terminate the debugger, click on the red square in the Debugger view. This will also stop the application.

[Image]

2.3 Advanced

Load and launch the other samples. In the Clock sample, try changing the ball velocity or the gravity. Start with lines 12 and 117 in balls.dart. Set more breakpoints and inspect the values by opening up the variables.

[Image]

3 Step 2: Import and run the chat app

This codelab walks you through a custom chat application. You will now load this chat app into the editor and learn how to run both client and server Dart apps.

3.1 Objectives

1. Load existing code into Dart Editor
2. Run the finished Dart Chat app
   (a) Run a command-line app
   (b) Run a Dartium app
3. View running processes
4. Read and clear console output
5. Close a running process

3.2 Walkthrough

3.2.1 Load Dart Chat into Dart Editor

If you are using a provided USB drive, copy over the dartchat/ directory from the USB thumb drive to your computer. If you do not have a provided USB drive, you can download the source code.

Load the sample project for this codelab into the editor. Select File ¿ Open Folder... in the editor. Find the dartchat/ directory that you copied from the USB or downloaded, select it, and click Open.

[Image]

You will see a new dartchat project in the editor.

[Image]

3.2.2 Launch the completed version of the Dart Chat sample

The sample chat app has both a client and a server component. Run the server first. In the Files view on the left hand side of Dart Editor, navigate into the dartchat directory, dartchat ¿ finished ¿ chat-server.dart. Right click chat-server.dart and select Run.

[Image]

Verify the server is running by checking the chat-server.dart console output window at the bottom of your editor. You should see a message: "listening for connections on 1337".

[Image]

Next, run the client. Navigate into the client directory, and run the chat-client.dart file in the same way.

[Image]

Notice how Dartium opens the chat app. Verify the chat client is connected to the server by looking for a "[system]: Connected" message in the chat window.

[Image]

Open up another tab to have a proper chat. Right click the Dart Chat tab and select Duplicate.

[Image]

Drag the new tab into its own window. Experiment by sending messages between the two tabs. For more fun, add more clients.

[Image]

3.2.3 Debugger view and console output

Switch back to Dart Editor and select the Tools ¿ Debugger in the top level menu. This lists the two processes that you started, the server and the client.

[Image]

On the bottom of Dart Editor are two views, chat-server.dart and Dartium launch. Each view has the output from the respective process.

[Image]

To clear a console output, click on the gray X icon in upper right of the console output view. To kill the process, click on the red box in the upper right of the console output view. After clicking on the red box, you will notice that the Debugger is updated to show that the process was killed. Stop both processes now, first for the Dartium launch, and then for the chat- server.dart.

[Image]

3.3 Advanced

If you finish early, explore more of the chat sample app code. Specifically, investigate the chat server code. The server logs to a file via an isolate, you can learn more about isolates in the Dart Library Tour.

4 Step 3: Navigate the chat app, and learn basic Dart language features

The Dart language is familiar to a wide range of developers. It is a class-based object oriented language with single inheritance, curly braces, and semicolons. The syntax is instantly recognizable and the concepts are comfortable.

As you tour the chat code, you will learn the basics of the Dart language. You can use Dart Editors code navigation features, such as search,

outlines, and jump to definition to come up to speed more quickly with unfamiliar code.

4.1 Objectives

1. Search for code using Dart Editor
2. Jump to definition of class or method
3. Use the Outline view to see file structure
4. Learn about classes, superclass, generics, functions, methods, variables, libraries, and more
5. Learn about optional static types
6. Understand basic layout of sample app
7. Understand warnings in Dart Editor
8. Understand errors in Dart Editor

4.2 Before You Begin - Code

Begin your coding journey in the start-here directory of dartchat. If you need to catch up, or if you need to start over, you can copy step03 into start-here for this portion of the codelab.

4.3 Walkthrough

In order to demonstrate Dart language and editor features, you will create a class to represent the chat window on the HTML page. This process will teach you more about the Dart Editor and the Dart language.

4.3.1 Project layout

Before you begin coding, it's helpful to understand the layout of the project. Open the finished directory, inside of the dartchat project in Dart Editor. Now, open the client directory to get a full layout of the finished app.

[Image]

You will spend your time inside the client directory, mostly working on chat-client.dart. The project includes the code for both the client (chat-client.dart) and the server (chat-server.dart).

4.3.2 Search for code

Use the Search feature of the editor, in the upper right of the tool, to find "Very first edit" (a helpful comment that the authors included for you :), so you can start editing.

[Image]

The search results are displayed in a dedicated view at the bottom of the editor.

[Image]

Double-click on the chat-client.dart from start-here to open the file and start writing some Dart!

Tip: Use start-here to make all of your edits. Be sure not to edit step03, as this is intended to give you a fallback if you need to restart your tasks.

4.3.3 Add a top-level variable

Ensure client/chat-client.dart is open and scrolled to the top. Add a top-level variable for an object to represent the the chat window. Use chatWindow as the variable name and ChatWindow as the type annotation.

```
// client/chat-client.dart
...
UsernameInput usernameInput;
// Step 3: Very first edit
```

```
// Step 3: Add variable for chat window.
ChatWindow chatWindow;
class ChatConnection {
...
```

Tip: You can use top-level functions and variables instead of wrapping everything in a class.

Save the file. Notice that after you add the new variable with a type annotation, the editor gives you a warning (with a yellow underline) indicating that it doesnt know what ChatWindow is. That makes sense, as we havent defined it yet.

[Image]

Advanced Topic: Dart is an optionally typed language, which means you can use type annotations when you want them, and leave them off when you dont. If the runtime cant find the type you are referring to, it will ignore that static type annotation and treat the variable as Dynamic (the stand-in type when no other type annotation is provided). A missing type mentioned in a type annotation does not prevent the program from compiling and running (because, again, Dart is optionally typed). It is only a warning that the editor cant find ChatWindow.

4.3.4 Instantiate an object

Scroll down to main() to create a new instance of ChatWindow. Much like other languages, use the new keyword to construct a new object.

```
// client/chat-client.dart
main() {
// Step 4: Identify elements by ID.
TextAreaElement chatElem = null;
InputElement usernameElem = null;
InputElement messageElem = null;
// Step 3: Instantiate ChatWindow.
chatWindow = new ChatWindow(chatElem);
usernameInput = new UsernameInput(usernameElem);
...
```

Notice how the editor is reporting an error now, with a red underline. Specifically, the error is "no such type: "ChatWindow"".

[Image]

An error will stop a program from compiling and running, so this is a problem that must be fixed before continuing. Dart strives to minimize the number of situations that result in a true error, but Dart does need to know what kind of object you want to construct.

4.3.5 Define a class

To silence the error, add the ChatWindow class. Find the Define the ChatWindow class comment and add the following class:

```
// client/chat-client.dart
...
// Step 3: Define the ChatWindow class.
class ChatWindow extends View<TextAreaElement> {
}
...
```

The editor will give you an error about a missing constructor. Don't worry, we'll fill this in next. The ChatWindow class extends the View class (already defined for you in the application). The View class uses generics to further specify the type of the HTML element the view encapsulates. The above code says ChatWindow is a specialized View of a TextAreaElement.

Advanced Topic: Dart support parameterized types, also known as generics. As with Darts optional types, you dont need use generics. However, they are a powerful tool to help add more expressiveness to your code. Darts generics8 are probably simpler than those found in other mainstream languages.

Add a constructor to ChatWindow that delegates to the constructor from View:

```
// client/chat-client.dart
...
```

```
// Step 3: Define the ChatWindow class.
class ChatWindow extends View<TextAreaElement> {
ChatWindow(TextAreaElement elem) : super(elem);
...
```

What follows the : is the initializer list9 for a class, used to initialize final variables and the super constructor. Next, add two public methods and one private method to ChatWindow. These methods will display messages to the ¡textarea¿ element.

```
// client/chat-client.dart
...
class ChatWindow extends View<TextAreaElement> {
ChatWindow(TextAreaElement elem) : super(elem);
displayMessage(String msg, String from) {
_display("$from: $msg\n");
}
displayNotice(String notice) {
_display("[system]: $notice\n");
}
_display(String str) {
elem.text = "${elem.text}$str";
}
}
...
```

n the above code, the _display() method is library private. Names that are prefixed with an underscore are private to the library they are defined within. The ChatWindow class is defined in a file that is marked as #library('chat-client'), therefore _display() is only visible to code also in the 'chat-client' library. The other two methods are public. The text property of elem accessing the contents of the ¡textarea¿ tag. One of Darts handy features is string interpolation, used in all three methods above. You can compose strings like "$from: $msg
n" by directly referencing variables with a $ prefix. Note that elem, used inside of _display(), is found in the View superclass. ChatWindow extends View, thus gaining access to View's instance variables.

4.3.6 View the code outline

With the ChatWindow class now added, you can use the editor to easily browse the outline of the chat-client.dart file. Select the chat-client.dart file and select the menu, Tools, Outline.

[Image]

The Outline view will now appear, with a tree view of the classes, methods, and variables found in chat-client.dart.

[Image]

4.4 Advanced

Learn more about the Dart language, such as strings and classes. Right-click a class name and select Open Declaration. Try this with classes defined in chat-client.dart as well as classes from Dart.

[Image]

Right-click a method and select Open Callers. Youll see a list of all the locations that call this method.

[Image]

謝辞

この文書の翻訳にあたり，翻訳にご協力を下さりました皆様に感謝いたします.

- Introduction XY さん
- Step1 XY さん
- Step2 YZ さん

また，このようなすばらしい CodeLab のテキスト及び教材をご提供下さり，また，日本語への翻訳をご快諾下さりました Google Inc. の...

最後に...