The Pragmatic Programmers

·新·锐·编·程·语·言·集·萃·

Dart for Hipsters

Fast, Flexible, Structured Code for the Modern Web

Dart语言程序设计

【美】Chris Strom 著

韩国恺 译

人民邮电出版社
POSTS & TELECOM PRESS

The Pragmatic Programmers

·新·锐·编·程·语·言·集·萃·

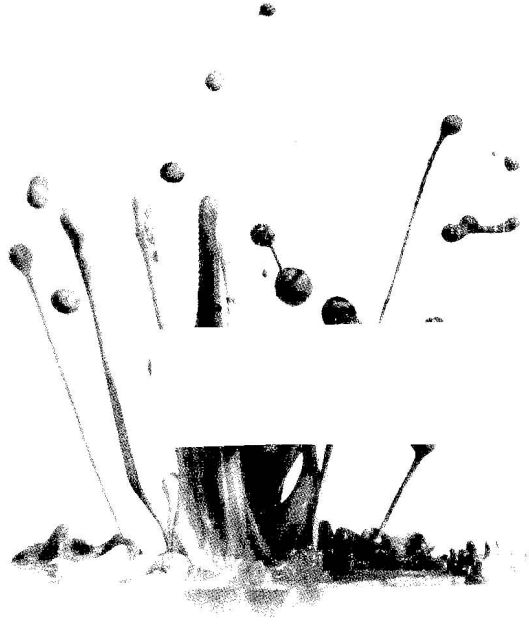
Dart for Hipsters

Fast, Flexible, Structured Code for the Modern Web

Dart语言程序设计

【美】Chris Strom 著

韩国恺 译



人民邮电出版社

北京

图书在版编目（CIP）数据

Dart语言程序设计 / (美) 斯特罗姆 (Strom, C.) 著 ; 韩国恺译. — 北京 : 人民邮电出版社, 2013.1
(新锐编程语言集萃)
ISBN 978-7-115-29694-8

Ⅰ. ①D… Ⅱ. ①斯… ②韩… Ⅲ. ①程序语言—程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2012)第242619号

版权声明

新锐编程语言集萃

Dart 语言程序设计

◆ 著 [美] Chris Strom
译 韩国恺
责任编辑 杨海玲

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000 1/16
印张：8.75
字数：165 千字 2013 年 1 月第 1 版
印数：1–3 000 册 2013 年 1 月河北第 1 次印刷

著作权合同登记号 图字：01-2012-6702 号

ISBN 978-7-115-29694-8

定价：35.00 元

读者服务热线：(010)67132692 印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第 0021 号

目 录

第一部分 入 门

第二部分 有效的编程技术

第三部分 代码组织

第四部分 可维护性

第五部分 Dart 的高级使用

第一部分　入门

Dart 有很多特点，但也许最令人印象深刻的是，它让熟悉 JavaScript 的程序员感到非常熟悉。在前面的几章中，我们从头开始，编写一个 Dart 应用程序。

第 1 章

项目：第一个 Dart 应用程序

大部分编程书籍都以一个“Hello World!”示例开始。我要说，换种方式——我们这里都是编程达人。让我们开始写代码吧！

首先因为 Dart 语言写起来感觉比较熟悉，所以我们在深入之前不应该走得太远。让我们直接跳到更有趣的事情上：一个带 Ajax 的网站。任何一个真正的达人都会收集大量的漫画书（我说的对吗？不是就我一个人这样吧，哈），那么让我们来考虑一个简单的 Dart 应用程序，通过 REST 风格的接口来操作一组漫画书。

在某种程度上，这可能有点儿太激进了。别担心，我们会在后续章节中深入细节。

1.1 后端部分

本章的示例代码可以在 https://github.com/eee-c/dart-comics 的“your_first_dart_app”分支上找到。作为技术爱好者，我们已经在用 Node.js 了，所以后端需要 Node.js 和 npm。说明包含在项目的 README 中。

作为 REST 风格，这个应用程序应该支持下面这些操作：

- GET /comics（返回漫画书的列表）；
- GET /comics/42（返回一本漫画书）；
- PUT /comics/42（更新一本漫画书）；
- POST /comics（在集合中新建一本漫画书）；
- DELETE /comics/42（删除一本漫画书）。

我们不用太关心除此之外的后端部分的细节。

1.2　Dart 的 HTML 部分

我们的整个应用程序将遵循最近的客户端 MVC 框架的惯例。因此，我们只需一个 Web 页面。

your_first_dart_app/index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Dart Comics</title>
  <link rel="stylesheet" href="/stylesheets/style.css">

  <!-- 强制让 Dartium 启动脚本引擎 -->
  <script language="text/javascript">
    navigator.webkitStartDart();
  </script>

  <!-- 主程序脚本 -->
  <script src="/scripts/comics.dart"
      type="application/dart"></script>
</head>

<body>
  <h1>Dart Comics</h1>
  <p>Welcome to Dart Comics</p>

  <ul id="comics-list"></ul>

  <p id="add-comic">
    Add a sweet comic to the collection.
  </p>
</body>
</html>
```

对于大家而言，这个 Web 页面的大部分应该很熟悉。它包含了简单的 HTML、CSS 的链接和脚本。

1. HTML 头部

唯一要注意的有点儿奇怪的地方是第一个<script>标签。在这个标签中，用 JavaScript 来启动 Dart 脚本引擎。

your_first_dart_app/_index_force_dartium_script_engine.html

```
<!-- 强制让 Dartium 启动脚本引擎 -->
<script language="text/javascript">
  navigator.webkitStartDart();
</script>
```

要点：在写这本书时，在 Dartium 上必须用 navigator.webkitStartDart() 来启用 Dart VM，Dartium 是包含 Dart 语言支持的 Chrome 版本[①]。这个要求可能会在不久的将来去除。

接下来，我们加载了实际代码的内容。这里唯一的变化是<script>标签的 type 属性，用来表示这是 Dart 代码。

your_first_dart_app/_index_src_dart.html

```
<!-- 主程序脚本 -->
<script src="/scripts/comics.dart"
type="application/dart"></script>
```

第 10 章还有更多关于 Dart 加载库和包含代码的内容。现在只要知道它会按照我们期望的方式加载 Dart 即可。

2. HTML 主体

作为 HTML 的主体部分，没什么新东西，不过我们应该注意一下将附加上行为的两个元素的 ID。

your_first_dart_app/_index_body.html

```
<h1>Dart Comics</h1>
<p>Welcome to Dart Comics</p>
<ul id="comics-list"></ul>
<p id="add-comic">
  Add a sweet comic to the collection.
</p>
```

对#comics-list UL 元素，我们将附加上后端数据存储中的漫画书列表。#add-comic 段落标签上将附加上一个表单处理程序。那么，我们就开始吧。

1.3 Dart 的 Ajax 部分

在 scripts/comics.dart 中，我们的 Dart 应用程序以两个 Dart 库的加载和一

① http://www.dartlang.org/dartium/。

个 main() 函数开始。

your_first_dart_app/comics_initial_main.dart

```
import 'dart:html';
import 'dart:json';
main() {
  load_comics();
}
load_comics() {
  // 在这里具体实现
}
```

正如我们将在第 10 章看到的，import 语句有很多功能。现在我们只要简单地认为它导入了 Dart 核心行为之外的其他功能即可。

所有的 Dart 应用程序都是以 main() 作为执行的入口点。在 JavaScript 中，我们只要直接写代码然后就可以运行，但在 Dart 中这样不行。起初这可能看起来类似 C 系列的语言，但这确实是有意义的，难道要让遍布于各处的 JavaScript 源文件和 HTML 文件中的所有代码都立刻执行？[①] 在 Dart 语言中，main() 入口点不只是约定，它是由这一语言强制的最佳实践。

至于 load_comics() 函数，我们一步步来。首先我们需要标识出列表需要附加的 DOM 元素（#comics-list），接着我们需要一个 Ajax 调用来填充这个 DOM 元素。为了实现这两件事，我们最初的 Dart 代码看上去大概像下面这样：

your_first_dart_app/_load_comics.dart

```
load_comics() {
  var list_el = document.query('#comics-list');
  ajax_populate_list(list_el);
}
```

除了明显地忽略了 function 关键字之外，这个例子就像是 JavaScript 代码！在第 3 章，我们将介绍更多差异。Dart 语言仍然用我们熟悉和喜欢的分号和大括号——语言设计者有意让人对这门语言感到很熟悉，至少在表面上看起来很熟悉。

> **注意：和 JavaScript 不同，在 Dart 中分号不是可选的。**

除了熟悉，这段代码一看就很容易阅读和理解。没有怪异的、遗留的 DOM 方法。我们用 document.query() 而不是用 document.findByElementId() 来查找元素，并

① 在一个页面中，出现的所有独立 js 文件或嵌入在 HTML 文件中的 JavaScript 都会立即被解析执行。但我们往往需要一个开始执行的入口点，一般是在文档准备好时执行一个函数，如 jQuery 的$(document).ready(handler)。——译者注

且用#comics-list 这种 CSS 选择器的方式，正如我们在 jQuery 中已经习惯的那样。

现在已经找到了需要填充的 UL 元素，让我们来看看如何产生一个 Ajax 请求。就像在 JavaScript 中一样，我们要创建一个新的 XHR 对象[①]，并添加了一个请求加载完成的处理程序，然后打开并发送这个请求。

your_first_dart_app/_ajax_populate_list.dart

```
ajax_populate_list(container) {
  var req = new HttpRequest();

  req.on.load.add((event) {
    var list = JSON.parse(req.responseText);
    container.innerHTML = graphicNovelsTemplate(list);
  });

  // 动作，资源，异步模式
  req.open('get', '/comics', true);
  req.send();
}
```

对于过去做过 Ajax 编程的人，对大部分代码应该立刻就能看懂。我们打开创建的 XHR 对象，指定要获取的资源，然后实际发送请求。

当添加事件处理程序时，我们看到与 JavaScript 的使用方式有所不同。XHR 对象有一个 on 属性，它列出了所有支持的事件处理程序。我们访问其中的一个 load 处理程序类型，这样我们就能用 add()方法添加一个处理程序。在这种情况下，我们解析获得的 JSON 数据为一个散列值的列表。它大概就像这样：

your_first_dart_app/comics.json

```
[
  {"title":"Watchmen",
  "author":"Alan Moore",
  "id":1},
  {"title":"V for Vendetta",
  "author":"Alan Moore",
  "id":2},
  {"title":"Sandman",
  "author":"Neil Gaiman",
  "id":3}
]
```

① 为了减少名称中的冗余信息，在新版的 Dart 中，过去的 XMLHttpRequest 已改为 HttpRequest，相关的名称也是如此，如 XMLHttpRequestException 已改为 HttpRequestException。——译者注

然后，我们要实现这个简单的 Dart 应用程序的最后一部分——一个填充漫画书列表的模板。

your_first_dart_app/_graphic_novels_template.dart

```
graphic_novels_template(list) {
  var html = new StringBuffer();
  list.forEach((graphic_novel) {
    html.add(_graphic_novel_template(graphic_novel));
  });
  return html.toString();
}

_graphic_novel_template(graphic_novel) {
  return """
    <li id="${graphic_novel['id']}">
      ${graphic_novel['title']}
      by
      ${graphic_novel['author']}
    </li>
    """;
}
```

第一个函数简单迭代这个漫画书列表（在心里，我把它们当做漫画小说），构建为一个 HMTL 字符串[①]。

第二个函数展示了两个 Dart 语言特性：多行字符串和字符串变量插值。多行字符串由 3 个引号表示（单引号或双引号）。在字符串中，我们可以用美元符号插入值（甚至一个简单表达式）。对于简单的变量插入，大括号是可以省略的：$name 和${name}是一样的。而对更复杂的插入，如散列查找，大括号就是必需的。

就是这样！我们准备好了一个功能完备的、带 Ajax 的 Web 应用程序。汇总的代码如下：

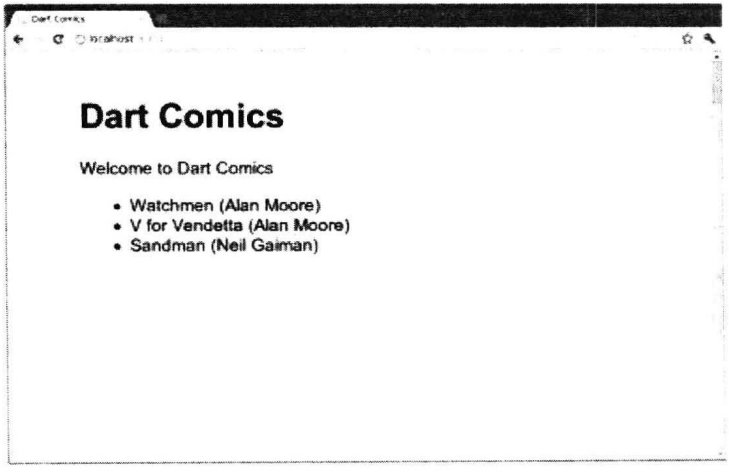
your_first_dart_app/comics.dart

```
import 'dart:html';
import 'dart:json';
main() {
  load_comics();
}
load_comics() {
```

① Dart 语言现在已不支持字符串拼接（+）操作，所以需要改用 StringBuffer。简单（非循环）的字符串拼接，如 str1+str2，可以用字符串内的变量插值实现，如“$a $b”。或者使用相邻字符串的方式，在 Dart 语言中相邻的字符串会自动拼接，包括多行情况。——译者注

```
  var list_el = document.query('#comics-list');
  ajax_populate_list(list_el);
}
ajax_populate_list(container) {
  var req = new HttpRequest();
  req.on.load.add((event) {
    var list = JSON.parse(req.responseText);
    container.innerHTML = graphicNovelsTemplate(list);
  });
  // 动作，资源，异步模式
  req.open('get', '/comics', true);
  req.send();
}
graphic_novels_template(list) {
  var html = new StringBuffer();
  list.forEach((graphic_novel) {
    html.add(_graphic_novel_template(graphic_novel));
  });
  return html.toString();
}
_graphic_novel_template(graphic_novel) {
  return """
    <li id="${graphic_novel['id']}">
      ${graphic_novel['title']}
      by
      ${graphic_novel['author']}
    </li>
    """;
}
```

页面加载后看上去像这样：



这便是我们探索 Dart 语言的良好开端。的确，我们这里掩饰了许多成就 Dart 语言

的地方。但是这样做，让我们有了一个使用 Ajax 的 Web 应用程序的良好开始。最棒的是，我们写的代码看起来似乎与 JavaScript 没有什么不同。一些语法比我们使用 JavaScript 更整洁一点（没人会抱怨整洁的代码），并且这些字符串特性也很不错。但是总的来说，可以肯定的是，使用相对更短的 Dart 语言是有生产力的。

1.4　这个应用程序还无法运行

在写作本书时，这个应用还不能在（几乎）任何地方实际运行。

任何浏览器（甚至 Chrome）都还不支持 Dart 语言。为了原生地运行这个 Web 应用程序，我们需要安装 Dartium——一个嵌入 Dart VM 的 Chrome 分支版本。Dartium 可以从 Dart 语言网站获得[①]。

即使在 Chrome 支持 Dart 语言之后，我们仍然面临着只能被市场上个别浏览器支持的情况。这有点儿糟糕。

好在 Dart 能够编译为 JavaScript，这意味着你可以在利用 Dart 能力的同时对所有平台可用。为了更容易实现这一点，我们添加一个小的 JavaScript 库，用来探测浏览器是否支持 Dart 语言，如果不支持就加载编译成 JavaScript 的等价物。

your_first_dart_app/_index_src_js_fallback.html

```
<!-- 启用回退到 Javascript -->
<script src="/scripts/conditional-dart.js"></script>
```

第 5 章中将讨论这个帮助文件的细节。目前，只要知道我们的 Dart 代码不是锁定在一个浏览器厂商那里就足够了。我们肯定不会回到 VBScript 那样。

1.5　下一步做什么

不可否认，这是对 Dart 语言的快速入门。能够如此快速地搭建和运行真是太棒了。仿佛我们此刻就获得了生产力。

尽管如此，我们才刚刚开始学习 Dart 语言，别搞错了，我们的 Dart 代码还可以改进。因此，让我们用下面的几章来熟悉 Dart 语言中的一些重要概念。之后，在第 6 章中，我们将准备好把这个 Dart 应用转变成 MVC 的方式。

① http://www.dartlang.org/dartium/。

第 2 章

基本类型

本书中一个反复出现的主题是，Dart 语言就是要让人觉得很熟悉。如果做到了这一点，那么对这一语言的基本组成部分的论述应该是相对简短的，并且确实是这样。尽管如此，对一些核心类型的介绍还是有帮助的。自然，其中也会有几处“陷阱”。

2.1 数字类型

整数和小数都是数字类型[①]，这意味着它们支持很多相同的方法和运算符。Dart 语言中的数字类型与许多其他语言中的数字类型非常像。

```
2 + 2;        // 4
2.2 + 2;      // 4.2
2 + 2.2;      // 4.2
2.2 + 2.2;    // 4.4
```

可以看出，在二者混合运算时 Dart 语言的数字类型做了“正确的事”。

2.2 字符串类型

字符串是不可变的，换种直观的说法就是，字符串的操作会产生新的字符串而不是修改现有的字符串。字符串（像数字一样）是可散列的，这意味着唯一的对象有唯

① 在 Dart 中，int 和 double 都继承自 num。——译者注

一的一个散列值来区分彼此[①]。如果我们将一个字符串变量赋值给另一个变量，它们将有一样的散列值，因为它们是同一个对象。

```
var str1 = "foo",
str2 = str1;

str1.hashCode; // 425588957
str2.hashCode; // 425588957
```

但是，如果我们修改了第一个字符串变量，那么它将获得一个全新的对象，而另一个字符串对象还是指向原来的字符串。

```
str1 = "bar";

str1.hashCode; // 617287945
str2.hashCode; // 425588957
```

Dart 语言使字符串的处理更容易。例如，可以用三重引号括起来的方式新建多行字符串。

```
"""Line #1
Line #2
Line #3""";
```

Dart 语言也支持相邻字符串拼接。

```
'foo' ' ' 'bar'; // 'foo bar'
```

这种相邻字符串的便利用法甚至扩展到多行字符串上。

```
'foo'
' '
'bar'; // 'foo bar'
```

最后一个 Dart 字符串的便利用法是字符串内的变量插值。Dart 语言使用$表示要插值的变量。

```
var name = "Bob";

"Howdy, $name"; // "Howdy, Bob"
```

① 严格地说，散列值并不要求一定唯一，但应该很好地分布。如果两个对象相等（equals），那么二者的散列值也应该相等，但反之不一定成立。也就是说，不同内容的字符串的散列值也有可能相同，只是概率很小。——译者注

如果在变量表达式结束和字符串开始的地方存在混淆，可以将大括号和$一起使用。

```
var comic_book = new ComicBook("Sandman");

"The very excellent ${comic_book.title}!";
// "The very excellent Sandman"
```

2.3 布尔类型

在 Dart 语言中，布尔类型（bool）的值只允许是 true 和 false。在 Dart 语言中“真值”和它自身的概念一样简单：如果不是 true，那就是 false。考虑下面的代码：

```
var name, greeting;

greeting = name ? "Howdy $name" : "Howdy";
// "Howdy"

/*** Name 仍然不是 true ***/
name = "Bob";
greeting = name ? "Howdy $name" : "Howdy";
// "Howdy"

greeting = (name != null) ? "Howdy $name" : "Howdy";
// "Howdy Bob"
```

如果你用过许多其他语言，那么不会奇怪在布尔上下文中 null、""和 0 的值为 false。你可能也习惯于"Bob"和 42 的值为 false。

“真值”的语义运行在“类型检查”模式下会稍有不同（见 2.7 节），但是最好不要依靠这种差异。如果我们总是用最后那种写法，那么我们就肯定不会犯错了。

在第 7 章中，我们将探讨操作符定义，这允许我们自定义特定类的 equals/== 方法的含义。这给 Dart 语言关于布尔类型带来了一定的灵活性。

2.4 HashMap（也称为 Hash 或关联数组）

在 Dart 语言中，键值对的数据结构是由 HashMap 对象实现的。下面的代码定义