Бесплатная электронная книга

учусь dart

Free unaffiliated eBook created from
Stack Overflow contributors.

#dart

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: dart

It is an unofficial and free dart ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official dart.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с дротиками

замечания

Dart - это открытый, основанный на классе, необязательно типизированный язык программирования для создания веб-приложений - как на клиенте, так и на сервере, созданный Google. Цели дизайна Дарта:

- Создайте структурированный, но гибкий язык для веб-программирования.
- Сделайте Дарт знакомым и естественным для программистов и, следовательно, легко учиться.
- Убедитесь, что Dart обеспечивает высокую производительность во всех современных веб-браузерах и средах, от небольших портативных устройств до выполнения на стороне сервера.

Дарт нацелен на широкий спектр сценариев развития, от проекта с одним человеком без большой структуры до крупномасштабного проекта, требующего формальных типов в коде, чтобы заявить намерение программиста.

Для поддержки этого широкого спектра проектов Dart предоставляет следующие функции и инструменты:

- **Необязательные типы:** это означает, что вы можете запускать кодирование без типов и добавлять их по мере необходимости.
- **Изолирует:** одновременное программирование на сервере и клиенте
- **Легкий доступ DOM:** с помощью селекторов CSS (так же, как это делает jQuery)
- **Инструменты Dart IDE:** плагины Dart существуют для многих обычно используемых IDE, например: WebStorm .
- **Dartium:** сборка веб-браузера Chromium со встроенной виртуальной машиной Дарта

связи

- Главная страница Dart
- Официальные новости и обновления Дартс
- Dartosphere - коллекция недавних сообщений в блоге Dart
- Сообщество Dartisans Dartisans в Google+
- Dart Web Development - Страница групп Google
- Dart Language Misc - Страница групп Google
- DartLang sub-Reddit

Документация

- Тур по языку дартс
- Экскурсия по дарт-библиотекам
- Примеры кода Dart
- Ссылка API Dart

Часто задаваемые вопросы

- Часто задаваемые вопросы

Версии

Версия	Дата выхода
1.22.1	2017-02-22
1.22.0	2017-02-14
1.21.1	2016-01-13
1.21.0	2016-12-07
1.20.1	2016-10-13
1.20.0	2016-10-11
1.19.1	2016-09-07
1.19.0	2016-08-26
1.18.1	2016-08-02
1.18.0	2016-07-27
1.17.1	2016-06-10
1.17.0	2016-06-06
1.16.1	2016-05-23
1.16.0	2016-04-26
1.15.0	2016-03-09
1.14.2	2016-02-09

Версия	Дата выхода
1.14.1	2016-02-03
1.14.0	2016-01-28
1.13.2	2016-01-05
1.13.1	2015-12-17
1.13.0	2015-11-18
1.12.2	2015-10-21
1.12.1	2015-09-08
1.12.0	2015-08-31
1.11.3	2015-08-03
1.11.1	2015-07-02
1.11.0	2015-06-24
1.10.1	2015-05-11
1.10.0	2015-04-24
1.9.3	2015-04-13
1.9.1	2015-03-25
1.8.5	2015-01-13
1.8.3	2014-12-01
1.8.0	2014-11-27
1.7.2	2014-10-14
1.6.0	2014-08-27
1.5.8	2014-07-29
1.5.3	2014-07-03
1.5.2	2014-07-02
1.5.1	2014-06-24
1.4.3	2014-06-16

Версия	Дата выхода
1.4.2	2014-05-27
1.4.0	2014-05-20
1.3.6	2014-04-30
1.3.3	2014-04-16
1.3.0	2014-04-08
1.2.0	2014-02-25
1.1.3	2014-02-06
1.1.1	2014-01-15
1.0.0.10_r30798	2013-12-02
1.0.0.3_r30188	2013-11-12
0.8.10.10_r30107	2013-11-08
0.8.10.6_r30036	2013-11-07
0.8.10.3_r29803	2013-11-04

Examples

Установка или настройка

Dart SDK включает в себя все необходимое для записи и запуска кода Dart: VM, библиотеки, анализатор, диспетчер пакетов, генератор doc, форматирование, отладчик и т. Д. Если вы занимаетесь веб-разработкой, вам также понадобится Dartium.

Автоматическая установка и обновление

- Установка Dart на Windows
- Установка Dart на Mac
- Установка Dart на Linux

Ручная установка

Вы также можете вручную установить любую версию SDK .

Привет, мир!

Создайте новый файл с именем `hello_world.dart` со следующим содержимым:

```
void main() {
  print('Hello, World!');
}
```

В терминале перейдите в каталог, содержащий файл `hello_world.dart` и введите следующее:

```
dart hello_world.dart
```

Нажмите `Enter`, чтобы отобразить `Hello, World!` в окне терминала.

Запрос Http

Html

```
<img id="cats"></img>
```

дротик

```
import 'dart:html';

/// Stores the image in [blob] in the [ImageElement] of the given [selector].
void setImage(selector, blob) {
  FileReader reader = new FileReader();
  reader.onLoad.listen((fe) {
    ImageElement image = document.querySelector(selector);
    image.src = reader.result;
  });
  reader.readAsDataUrl(blob);
}

main() async {
  var url = "https://upload.wikimedia.org/wikipedia/commons/2/28/Tortoiseshell_she-cat.JPG";

  // Initiates a request and asynchronously waits for the result.
  var request = await HttpRequest.request(url, responseType: 'blob');
  var blob = request.response;
  setImage("#cats", blob);
}
```

пример

см. Пример на https://dartpad.dartlang.org/a0e092983f63a40b0b716989cac6969a

Геттеры и сеттеры

```
void main() {
  var cat = new Cat();

  print("Is cat hungry? ${cat.isHungry}");  // Is cat hungry? true
  print("Is cat cuddly? ${cat.isCuddly}");  // Is cat cuddly? false
  print("Feed cat.");
  cat.isHungry = false;
  print("Is cat hungry? ${cat.isHungry}");  // Is cat hungry? false
  print("Is cat cuddly? ${cat.isCuddly}");  // Is cat cuddly? true
}

class Cat {
  bool _isHungry = true;

  bool get isCuddly => !_isHungry;

  bool get isHungry => _isHungry;
  bool set isHungry(bool hungry) => this._isHungry = hungry;
}
```

Dart class getters и seters позволяют API-интерфейсам инкапсулировать изменения состояния объекта.

См. Пример dartpad : https://dartpad.dartlang.org/c25af60ca18a192b84af6990f3313233

Прочитайте Начало работы с дротиками онлайн: https://riptutorial.com/ru/dart/topic/843/начало-работы-с-дротиками

глава 2: Асинхронное программирование

Examples

Возвращение будущего с помощью Завершающего

```
Future<Results> costlyQuery() {
  var completer = new Completer();

  database.query("SELECT * FROM giant_table", (results) {
    // when complete
    completer.complete(results);
  }, (error) {
    completer.completeException(error);
  });

  // this returns essentially immediately,
  // before query is finished
  return completer.future;
}
```

Async и Await

```
import 'dart:async';

Future main() async {
  var value = await _waitForValue();
  print("Here is the value: $value");
  //since _waitForValue() returns immediately if you un it without await you won't get the
result
  var errorValue = "not finished yet";
  _waitForValue();
  print("Here is the error value: $value");// not finished yet
}

Future<int> _waitForValue() => new Future((){

  var n = 100000000;

  // Do some long process
  for (var i = 1; i <= n; i++) {
    // Print out progress:
    if ([n / 2, n / 4, n / 10, n / 20].contains(i)) {
      print("Not done yet...");
    }

    // Return value when done.
    if (i == n) {
      print("Done.");
      return i;
    }
  }
});
```

См. Пример на Dartpad: https://dartpad.dartlang.org/11d189b51e0f2680793ab3e16e53613c

Преобразование обратных вызовов в фьючерсы

Дарт имеет надежную асинхронную библиотеку с Future , Stream и т. Д. Однако иногда вы можете запускать асинхронный API, который использует *обратные вызовы* вместо *Futures* . Чтобы устранить разрыв между обратными вызовами и фьючерсами, Дарт предлагает класс *Completer* . Вы можете использовать Completer для преобразования обратного вызова в Будущее.

Завершения отлично подходят для подключения API с поддержкой обратного вызова с использованием API на основе будущего. Например, предположим, что ваш драйвер базы данных не использует фьючерсы, но вам нужно вернуть будущее. Попробуйте этот код:

```
// A good use of a Completer.

Future doStuff() {
  Completer completer = new Completer();
  runDatabaseQuery(sql, (results) {
    completer.complete(results);
  });
  return completer.future;
}
```

Если вы используете API, который уже возвращает будущее, вам не нужно использовать Completer.

Прочитайте Асинхронное программирование онлайн: https://riptutorial.com/ru/dart/topic/2520/асинхронное-программирование

глава 3: Библиотеки

замечания

Директивы `import` и `library` могут помочь вам создать модульную и общую базу кода. Каждое приложение Dart представляет собой `library` , даже если она не использует директиву библиотеки. Библиотеки могут быть распределены с использованием пакетов. Информацию о пабе, менеджере пакетов, включенном в SDK, см. В разделе Паб-пакет и Asset Manager .

Examples

Использование библиотек

Используйте `import` чтобы указать, как пространство имен из одной библиотеки используется в области другой библиотеки.

```
import 'dart:html';
```

Единственным обязательным аргументом для `import` является URI, определяющий библиотеку. Для встроенных библиотек URI имеет специальную схему `dart:` Для других библиотек вы можете использовать путь к файловой системе или схему `package:` `package:` схема указывает библиотеки, предоставляемые диспетчером пакетов, такие как инструмент pub. Например:

```
import 'dart:io';
import 'package:mylib/mylib.dart';
import 'package:utils/utils.dart';
```

Библиотеки и видимость

В отличие от Java, Dart не имеет `public` , `protected` и закрытых ключевых слов. Если идентификатор начинается с символа подчеркивания _ , он является закрытым для его библиотеки.

Если вы, например, имеете класс A в отдельном файле библиотеки (например, `other.dart`), например:

```
library other;

class A {
  int _private = 0;

  testA() {
```

```
    print('int value: $_private');  // 0
    _private = 5;
    print('int value: $_private'); // 5
  }
}
```

а затем импортировать его в основное приложение, например:

```
import 'other.dart';

void main() {
  var b = new B();
  b.testB();
}

class B extends A {
  String _private;

  testB() {
    _private = 'Hello';
    print('String value: $_private'); // Hello
    testA();
    print('String value: $_private'); // Hello
  }
}
```

Вы получаете ожидаемый результат:

```
String value: Hello
int value: 0
int value: 5
String value: Hello
```

Указание префикса библиотеки

Если вы импортируете две библиотеки с конфликтующими идентификаторами, вы можете указать префикс для одной или обеих библиотек. Например, если у библиотеки 1 и библиотеки 2 есть класс Element, тогда у вас может быть такой код:

```
import 'package:lib1/lib1.dart';
import 'package:lib2/lib2.dart' as lib2;
// ...
var element1 = new Element(); // Uses Element from lib1.
var element2 =
    new lib2.Element();       // Uses Element from lib2.
```

Импортирование только части библиотеки

Если вы хотите использовать только часть библиотеки, вы можете выборочно импортировать библиотеку. Например:

```
// Import only foo and bar.
```

```
import 'package:lib1/lib1.dart' show foo, bar;

// Import all names EXCEPT foo.
import 'package:lib2/lib2.dart' hide foo;
```

Ленько загружая библиотеку

Отложенная загрузка (также называемая ленивой загрузкой) позволяет приложению загружать библиотеку по требованию, если и когда это необходимо. Чтобы лениво загрузить библиотеку, вы должны сначала импортировать ее, используя отложенную.

```
import 'package:deferred/hello.dart' deferred as hello;
```

Когда вам нужна библиотека, вызовите loadLibrary (), используя идентификатор библиотеки.

```
greet() async {
  await hello.loadLibrary();
  hello.printGreeting();
}
```

В предыдущем коде ключевое слово `await` приостанавливает выполнение до загрузки библиотеки. Для получения дополнительной информации об `async` и `await` , см. Больше примеров здесь поддержки асинхронной поддержки или посетите часть поддержки асинхронизации языкового тура.

Прочитайте Библиотеки онлайн: https://riptutorial.com/ru/dart/topic/3332/библиотеки

глава 4: Дата и время

Examples

Основное использование DateTime

```
DateTime now = new DateTime.now();
DateTime berlinWallFell = new DateTime(1989, 11, 9);
DateTime moonLanding = DateTime.parse("1969-07-20 20:18:00");  // 8:18pm
```

Вы можете найти более детальную информацию здесь .

Прочитайте Дата и время онлайн: https://riptutorial.com/ru/dart/topic/3322/дата-и-время

глава 5: Исключения

замечания

Код Dart может вызывать и исключать исключения. Исключения - это ошибки, указывающие на то, что произошло что-то неожиданное. Если исключение не было обнаружено, изолированный объект, создавший исключение, приостанавливается, и, как правило, изоляция и ее программа прекращаются.

В отличие от Java, все исключения Дарта - это исключенные исключения. Методы не объявляют, какие исключения они могут выбрасывать, и вам не нужно ловить никаких исключений.

Dart предоставляет типы исключений и ошибок , а также множество предопределенных подтипов. Вы можете, конечно, определить свои собственные исключения. Тем не менее, программы Dart могут вызывать любой ненулевой объект, а не только объекты исключений и ошибок, как исключение.

Examples

Пользовательское исключение

```
class CustomException implements Exception {
  String cause;
  CustomException(this.cause);
}

void main() {
  try {
    throwException();
  } on CustomException {
    print("custom exception is been obtained");
  }
}

throwException() {
  throw new CustomException('This is my first custom exception');
}
```

Прочитайте Исключения онлайн: https://riptutorial.com/ru/dart/topic/3334/исключения

глава 6: Классы

Examples

Создание класса

Классы могут быть созданы следующим образом:

```
class InputField {
  int maxLength;
  String name;
}
```

Класс может быть создан с использованием `new` ключевого слова, после которого значения полей по умолчанию будут пустыми.

```
var field = new InputField();
```

Значения поля могут быть доступны:

```
// this will trigger the setter
field.name = "fieldname";

// this will trigger the getter
print(field.name);
```

члены

У класса могут быть члены.

Переменные экземпляра могут быть объявлены с аннотациями / без типов и, возможно, инициализированы. Неинициализированные члены имеют значение `null` , если конструктор не устанавливает другое значение.

```
class Foo {
  var member1;
  int member2;
  String member3 = "Hello world!";
}
```

Переменные класса объявляются с использованием ключевого слова `static` .

```
class Bar {
  static var member4;
  static String member5;
  static int member6 = 42;
}
```

Если метод не принимает аргументов, он быстро, возвращает значение и не имеет видимых побочных эффектов, тогда можно использовать метод геттера:

```
class Foo {
  String get bar {
    var result;
    // ...
    return result;
  }
}
```

Getters никогда не принимают аргументы, поэтому круглые скобки для (пустых) списков параметров опущены как для объявления геттеров, так и выше, и для их вызова:

```
main() {
  var foo = new Foo();
  print(foo.bar); // prints "bar"
}
```

Существуют также методы setter, которые должны принимать ровно один аргумент:

```
class Foo {
  String _bar;

  String get bar => _bar;

  void set bar(String value) {
    _bar = value;
  }
}
```

Синтаксис для вызова сеттера такой же, как назначение переменной:

```
main() {
  var foo = new Foo();
  foo.bar = "this is calling a setter method";
}
```

Конструкторы

Конструктор класса должен иметь то же имя, что и его класс.

Давайте создадим конструктор для класса Person:

```
class Person {
  String name;
  String gender;
  int age;

  Person(this.name, this.gender, this.age);
}
```

Приведенный выше пример - это более простой и эффективный способ определения конструктора, чем следующий способ, который также возможен:

```
class Person {
  String name;
  String gender;
  int age;

  Person(String name, String gender, int age) {
    this.name = name;
    this.gender = gender;
    this.age = age;
  }
}
```

Теперь вы можете создать экземпляр Person следующим образом:

```
var alice = new Person('Alice', 'female', 21);
```

Прочитайте Классы онлайн: https://riptutorial.com/ru/dart/topic/1511/классы

глава 7: Коллекции

Examples

Создание нового списка

Списки могут быть созданы несколькими способами.

Рекомендуемый способ - использовать литерал `List` :

```
var vegetables = ['broccoli', 'cabbage'];
```

Может также использоваться конструктор `List` :

```
var fruits = new List();
```

Если вы предпочитаете более сильный ввод текста, вы также можете указать параметр типа одним из следующих способов:

```
var fruits = <String>['apples', 'oranges'];
var fruits = new List<String>();
```

Для создания небольшого расширяемого списка, либо пустого, либо содержащего некоторые известные начальные значения, предпочтительной является литеральная форма. Существуют специальные конструкторы для других видов списков:

```
var fixedLengthList1 = new List(8);
var fixedLengthList2 = new List.filled(8, "initial text");
var computedValues = new List.generate(8, (n) => "x" * n);
var fromIterable = new List<String>.from(computedValues.getRange(2, 5));
```

См. Также Руководство по стилю Эффективного Дарта о коллекциях .

Создание нового набора

Наборы могут быть созданы с помощью конструктора:

```
var ingredients = new Set();
ingredients.addAll(['gold', 'titanium', 'xenon']);
```

Создание новой карты

Карты могут быть созданы несколькими способами.

Используя конструктор, вы можете создать новую карту следующим образом:

```
var searchTerms = new Map();
```

Типы ключа и значения также могут быть определены с помощью дженериков:

```
var nobleGases = new Map<int, String>();
var nobleGases = <int, String>{};
```

В противном случае карты могут быть созданы с использованием символа карты:

```
var map = {
    "key1": "value1",
    "key2": "value2"
};
```

Сопоставьте каждый элемент в коллекции.

Все объекты коллекции содержат метод `map` который принимает `Function` как аргумент, который должен принимать один аргумент. Это возвращает `Iterable` поддерживаемый коллекцией. Когда `Iterable` итерируется, каждый шаг вызывает функцию с новым элементом коллекции, а результат вызова становится следующим элементом итерации.

Вы можете снова перевернуть `Iterable` в коллекцию с помощью методов `Iterable.toSet()` или `Iterable.toList()` или с помощью конструктора коллекции, который принимает итерируемый, например `Queue.from` или `List.from` .

Пример:

```
main() {
  var cats = [
    'Abyssinian',
    'Scottish Fold',
    'Domestic Shorthair'
  ];

  print(cats); // [Abyssinian, Scottish Fold, Domestic Shorthair]

  var catsInReverse =
  cats.map((String cat) {
    return new String.fromCharCodes(cat.codeUnits.reversed);
  })
  .toList(); // [nainissybA, dloF hsittocS, riahtrohS citsemoD]

  print(catsInReverse);
}
```

См. Пример dartpad: https://dartpad.dartlang.org/a18367ff767f172b34ff03c7008a6fa1

Отфильтровать список

Дарт позволяет легко фильтровать список, используя `where` .

```
var fruits = ['apples', 'oranges', 'bananas'];
fruits.where((f) => f.startsWith('a')).toList(); //apples
```

Конечно, вы можете использовать некоторые операторы AND или OR в своем предложении where.

Прочитайте Коллекции онлайн: https://riptutorial.com/ru/dart/topic/859/коллекции

глава 8: Комментарии

Синтаксис

- // Однострочный комментарий
- / * Многострочный / Встроенный комментарий * /
- /// Комментарий Дартдока

замечания

Хорошая практика - добавлять комментарии к вашему коду, чтобы объяснить, почему что-то сделано или объяснить, что что-то делает. Это поможет любому будущему читателю вашего кода более легко понять ваш код.

Связанная тема (ы) не на StackOverflow:

- Эффективный дартс: документация

Examples

Комментарий к концу строки

Прокомментировано все справа от `//` в одной строке.

```
int i = 0; // Commented out text
```

Многострочный комментарий

Все между комментариями `/*` и `*/` .

```
void main() {
  for (int i = 0; i < 5; i++) {
    /* This is commented, and
    will not affect code */
    print('hello ${i + 1}');
  }
}
```

Документация с использованием Dartdoc

Использование комментария doc вместо обычного комментария позволяет dartdoc находить его и генерировать документацию для него.

```
/// The number of characters in this chunk when unsplit.
```

```
int get length => ...
```

Вам разрешается использовать большинство уценки форматирования в ваш документ комментарии и dartdoc будет обрабатывать его соответствующим образом, используя пакет уценки .

```
/// This is a paragraph of regular text.
///
/// This sentence has *two* _emphasized_ words (i.e. italics) and **two**
/// __strong__ ones (bold).
///
/// A blank line creates another separate paragraph. It has some `inline code`
/// delimited using backticks.
///
/// * Unordered lists.
/// * Look like ASCII bullet lists.
/// * You can also use `-` or `+`.
///
/// Links can be:
///
/// * http://www.just-a-bare-url.com
/// * [with the URL inline](http://google.com)
/// * [or separated out][ref link]
///
/// [ref link]: http://google.com
///
/// # A Header
///
/// ## A subheader
```

Прочитайте Комментарии онлайн: https://riptutorial.com/ru/dart/topic/2436/комментарии

глава 9: Паб

замечания

Когда вы устанавливаете Dart SDK, одним из инструментов, которые вы получаете, является паб. Инструмент pub предоставляет команды для самых разных целей. Одна команда устанавливает пакеты, другая запускает HTTP-сервер для тестирования, другой подготавливает ваше приложение к развертыванию, а другой публикует ваш пакет на pub.dartlang.org . Вы можете получить доступ к командам pub либо через IDE, например WebStorm, либо в командной строке.

Обзор этих команд см. В разделе Команды Pub .

Examples

создание паба

Используйте pub build, когда вы будете готовы развернуть свое веб-приложение. Когда вы запускаете сборку pub, она генерирует активы для текущего пакета и всех его зависимостей, помещая их в новый каталог с именем build.

Чтобы использовать `pub build` , просто запустите ее в корневом каталоге вашего пакета. Например:

```
$ cd ~/dart/helloworld
$ pub build
Building helloworld......
Built 5 files!
```

паб служить

Эта команда запускает сервер разработки или dev-сервер для вашего веб-приложения Dart. Сервер dev - это HTTP-сервер на локальном хосте, который обслуживает активы вашего веб-приложения.

Запустите dev-сервер из каталога, содержащего файл `pubspec.yaml` вашего веб-приложения:

```
$ cd ~/dart/helloworld
$ pub serve
Serving helloworld on http://localhost:8080
```

Прочитайте Паб онлайн: https://riptutorial.com/ru/dart/topic/3335/паб

глава 10: Перечисления

Examples

Основное использование

```
enum Fruit {
  apple, banana
}

main() {
  var a = Fruit.apple;
  switch (a) {
    case Fruit.apple:
      print('it is an apple');
      break;
  }

  // get all the values of the enums
  for (List<Fruit> value in Fruit.values) {
    print(value);
  }

  // get the second value
  print(Fruit.values[1]);
}
```

Прочитайте Перечисления онлайн: https://riptutorial.com/ru/dart/topic/5107/перечисления

глава 11: Поток управления

Examples

Если еще

Дарт имеет, если еще:

```
if (year >= 2001) {
  print('21st century');
} else if (year >= 1901) {
  print('20th century');
} else {
  print('We Must Go Back!');
}
```

Dart также имеет трехкомпонентной `if` оператор:

```
var foo = true;
print(foo ? 'Foo' : 'Bar'); // Displays "Foo".
```

Пока цикл

Хотя циклы и делают, пока в Dart разрешены петли:

```
while(peopleAreClapping()) {
  playSongs();
}
```

а также:

```
do {
  processRequest();
} while(stillRunning());
```

Циклы могут быть завершены с использованием разрыва:

```
while (true) {
  if (shutDownRequested()) break;
  processIncomingRequests();
}
```

Вы можете пропустить итерации в цикле, используя continue:

```
for (var i = 0; i < bigNumber; i++) {
  if (i.isEven){
    continue;
  }
```

```
  doSomething();
}
```

Для цикла

Допускаются два типа циклов:

```
for (int month = 1; month <= 12; month++) {
  print(month);
}
```

а также:

```
for (var object in flybyObjects) {
  print(object);
}
```

Цикл `for-in` удобен при простое повторение по коллекции `Iterable` . Существует также метод `forEach` который вы можете вызвать объекты `Iterable` которые ведут себя как `for-in` :

```
flybyObjects.forEach((object) => print(object));
```

или, более кратко:

```
flybyObjects.forEach(print);
```

Корпус выключателя

Dart имеет случай переключения, который можно использовать вместо длинных операторов if-else:

```
var command = 'OPEN';

switch (command) {
  case 'CLOSED':
    executeClosed();
    break;
  case 'OPEN':
    executeOpen();
    break;
  case 'APPROVED':
    executeApproved();
    break;
  case 'UNSURE':
    // missing break statement means this case will fall through
    // to the next statement, in this case the default case
  default:
    executeUnknown();
}
```

Вы можете сравнить только константы integer, string или compile-time. Сравнимые объекты должны быть экземплярами одного и того же класса (а не любого из его подтипов), и класс не должен переопределять ==.

Одним из удивительных аспектов переключения в Дарте является то, что непустые аргументы case должны заканчиваться разрывом или, реже, продолжать, бросать или возвращаться. То есть непустые аргументы case не могут провалиться. Вы должны явно закрыть предложение непустого случая, обычно с перерывом. Вы получите статическое предупреждение, если вы опустите break, continue, throw или return, и код будет ошибкой в этом месте во время выполнения.

```
var command = 'OPEN';
switch (command) {
  case 'OPEN':
    executeOpen();
    // ERROR: Missing break causes an exception to be thrown!!

  case 'CLOSED': // Empty case falls through
  case 'LOCKED':
    executeClosed();
    break;
}
```

Если вы хотите провалиться в непустом `case` , вы можете использовать `continue` и label:

```
      var command = 'OPEN';
      switch (command) {
        case 'OPEN':
          executeOpen();
          continue locked;
locked: case 'LOCKED':
          executeClosed();
          break;
      }
```

Прочитайте Поток управления онлайн: https://riptutorial.com/ru/dart/topic/923/поток-управления

глава 12: Преобразование данных

Examples

JSON

```
import 'dart:convert';

void main() {
    var jsonString = """
      {
        "cats": {
                "abysinnian": {
                    "origin": "Burma",
                    "behavior": "playful"
                }
            }
    }
  """;

  var obj = JSON.decode(jsonString);

  print(obj['cats']['abysinnian']['behavior']); // playful
}
```

См. Пример на dartpad: https://dartpad.dartlang.org/7d5958cf10e611b36326f27b062108fe

Прочитайте Преобразование данных онлайн: https://riptutorial.com/ru/dart/topic/2778/преобразование-данных

глава 13: Регулярные выражения

Синтаксис

- var regExp = RegExp (r '^ (. *) $', multiLine: true, caseSensitive: false);

параметры

параметр	подробности
`String source`	Регулярное выражение как `String`
`{bool multiline}`	Является ли это многострочным регулярным выражением. (соответствует `^` и `$` в начале и в конце каждой строки отдельно не всей строки)
`{bool caseSensitive}`	Если выражение чувствительно к регистру

замечания

Регулярные выражения Дарта имеют тот же синтаксис и семантику, что и регулярные выражения JavaScript. См. Http://ecma-international.org/ecma-262/5.1/#sec-15.10 для спецификации регулярных выражений JavaScript.

Это означает, что любой ресурс JavaScript, который вы найдете о регулярных выражениях онлайн, относится к дротику.

Examples

Создание и использование регулярного выражения

```
var regExp = new RegExp(r"(\w+)");
var str = "Parse my string";
Iterable<Match> matches = regExp.allMatches(str);
```

Рекомендуется использовать «сырые строки» (префикс с `r`) при написании регулярных выражений, чтобы вы могли использовать неизолированные обратные косые черты в своем выражении.

Прочитайте Регулярные выражения онлайн: https://riptutorial.com/ru/dart/topic/3624/регулярные-выражения

глава 14: Совместимость Dart-JavaScript

Вступление

Совместимость Dart-JavaScript позволяет запускать код JavaScript из наших программ Dart.

Взаимодействие достигается за счет использования библиотеки `js` для создания Dart-заглушек. Эти заглушки описывают интерфейс, который мы хотели бы иметь с базовым кодом JavaScript. Во время выполнения вызова Dart-заглушка вызывается код JavaScript.

Examples

Вызов глобальной функции

Предположим, мы хотели бы вызвать функцию JavaScript `JSON.stringify` которая получает объект, кодирует его в строку JSON и возвращает ее.

Все, что нам нужно сделать, это написать подпись функции, пометить ее как внешнюю и аннотировать ее с `@JS` аннотации `@JS` :

```
 @JS("JSON.stringify")
external String stringify(obj);
```

Аннотация `@JS` будет использоваться здесь, чтобы отметить классы Дарта, которые мы хотели бы использовать в JavaScript.

Объединение классов JavaScript / пространств имен

Предположим, мы хотели бы `google.maps` API Google Maps JavaScript API `google.maps` :

```
@JS('google.maps')
library maps;

import "package:js/js.dart";

@JS()
class Map {
  external Map(Location location);
  external Location getLocation();
}
```

Теперь у нас есть класс карты Dart, который соответствует классу JavaScript `google.maps.Map` .

Запуск `new Map(someLocation)` в Dart вызовет `new google.maps.Map(location)` в JavaScript.

Обратите внимание, что вам не нужно называть свой класс Dart таким же, как класс JavaScript:

```
@JS("LatLng")
class Location {
  external Location(num lat, num lng);
}
```

Класс Dart `Location` соответствует классу `google.maps.LatLng` .

Использование несогласованных имен обескураживается, поскольку они могут создавать путаницу.

Передача объектных литералов

В JavaScript распространена практика передачи объектных литералов в функции:

```
// JavaScript
printOptions({responsive: true});
Unfortunately we cannot pass Dart Map objects to JavaScript in these cases.
```

Нам нужно создать объект Dart, который представляет объектный литерал и содержит все его поля:

```
// Dart
@JS()
@anonymous
class Options {
  external bool get responsive;

  external factory Options({bool responsive});
}
```

Обратите внимание, что класс Dart `Options` не соответствует какому-либо классу JavaScript. Таким образом, мы должны отметить это `@anonymous` аннотацией.

Теперь мы можем создать заглушку для исходной функции printOptions и вызвать ее с помощью нового объекта Options:

```
// Dart
@JS()
external printOptions(Options options);

printOptions(new Options(responsive: true));
```

Прочитайте Совместимость Dart-JavaScript онлайн: https://riptutorial.com/ru/dart/topic/9240/совместимость-dart-javascript

глава 15: Список фильтров

Вступление

Фильтры Dart перечисляются через методы `List.where` и `List.retainWhere` . Функция `where` принимает один аргумент: булева функция, которая применяется к каждому элементу списка. Если функция оценивает значение `true` элемент списка сохраняется; если функция вычисляет значение `false` , элемент удаляется.

Вызов `theList.retainWhere(foo)` практически эквивалентен установкеList `theList = theList.where(foo)` .

Examples

Фильтрация списка целых чисел

```
[-1, 0, 2, 4, 7, 9].where((x) => x > 2) --> [4, 7, 9]
```

Прочитайте Список фильтров онлайн: https://riptutorial.com/ru/dart/topic/10948/список-фильтров

глава 16: Струны

Examples

Конкатенация и интерполяция

Вы можете использовать оператор plus (`+`) для объединения строк:

```
'Dart ' + 'is ' + 'fun!'; // 'Dart is fun!'
```

Вы также можете использовать смежные строковые литералы для конкатенации:

```
'Dart ' 'is ' 'fun!';    // 'Dart is fun!'
```

Вы можете использовать `${}` для интерполяции значения выражений Дарта в строках. При вычислении идентификаторов фигурные скобки могут быть опущены:

```
var text = 'dartlang';
'$text has ${text.length} letters'; // 'dartlang has 8 letters'
```

Допустимые строки

Строка может быть как одиночной, так и многострочной. Строки одной строки записываются с использованием совпадающих одиночных или двойных кавычек, а многострочные строки записываются с использованием тройных кавычек. Все действующие строки Дарта:

```
'Single quotes';
"Double quotes";
'Double quotes in "single" quotes';
"Single quotes in 'double' quotes";

'''A
multiline
string''';

"""
Another
multiline
string""";
```

Корпус из деталей

Программировать создание String лучше всего с помощью StringBuffer . StringBuffer не генерирует новый объект String до тех пор, пока не вызывается `toString()` .

```
var sb = new StringBuffer();

sb.write("Use a StringBuffer");
sb.writeAll(["for ", "efficient ", "string ", "creation "]);
sb.write("if you are ")
sb.write("building lots of strings");

// or you can use method cascades:

sb
  ..write("Use a StringBuffer")
  ..writeAll(["for ", "efficient ", "string ", "creation "])
  ..write("if you are ")
  ..write("building lots of strings");

var fullString = sb.toString();

print(fullString);
// Use a StringBufferfor efficient string creation if you are building lots of strings

sb.clear(); // all gone!
```

Прочитайте Струны онлайн: https://riptutorial.com/ru/dart/topic/5003/струны

глава 17: функции

замечания

Дарт - это истинный объектно-ориентированный язык, поэтому даже функции - это объекты и имеют тип «Функция». Это означает, что функции могут быть назначены переменным или переданы в качестве аргументов другим функциям. Вы также можете вызвать экземпляр класса Dart, как если бы это была функция.

Examples

Функции с именованными параметрами

При определении функции используйте {param1, param2, ...}, чтобы указать именованные параметры:

```
void enableFlags({bool bold, bool hidden}) {
  // ...
}
```

При вызове функции вы можете указать именованные параметры, используя paramName: value

```
enableFlags(bold: true, hidden: false);
```

Область применения

Функции Дарта также могут быть объявлены анонимно или вложенными. Например, чтобы создать вложенную функцию, просто откройте новый функциональный блок в существующем функциональном блоке

```
void outerFunction() {

    bool innerFunction() {
        /// Does stuff
    }
}
```

Функция `innerFunction` теперь может использоваться внутри и только внутри, `outerFunction` . Никакие другие другие функции не имеют к нему доступа.

Функции в Dart также могут быть объявлены анонимно, что обычно используется в качестве аргументов функции. Общим примером является метод `sort` объекта `List` . Этот метод принимает необязательный аргумент со следующей сигнатурой:

```
int compare(E a, E b)
```

В документации указано, что функция должна возвращать `0` если `a` и `b` равны. Он возвращает `-1` если `a < b` и `1` если `a > b` .

Зная это, мы можем отсортировать список целых чисел, используя анонимную функцию.

```
List<int> numbers = [4,1,3,5,7];

numbers.sort((int a, int b) {
   if(a == b) {
      return 0;
   } else if (a < b) {
      return -1;
   } else {
      return 1;
   }
});
```

Анонимная функция также может быть привязана к таким идентификаторам:

```
Function intSorter = (int a, int b) {
   if(a == b) {
      return 0;
   } else if (a < b) {
      return -1;
   } else {
      return 1;
   }
}
```

и используется как обычная переменная.

```
numbers.sort(intSorter);
```

Прочитайте функции онлайн: https://riptutorial.com/ru/dart/topic/2965/функции

кредиты

S. No	Главы	Contributors
1	Начало работы с дротиками	4444, Challe, Community, Damon, Florian Loitsch, Gomiero, Kleak, losnake, martin, Raph, Timothy C. Quinn
2	Асинхронное программирование	Challe, Damon, Ray Hulha, Zied Hamdi
3	Библиотеки	Challe, Ganymede
4	Дата и время	Challe
5	Исключения	Challe
6	Классы	Ganymede, Hoylen, Jan Vladimir Mostert, Raph
7	Коллекции	Alexi Coard, Damon, Jan Vladimir Mostert, Kleak, lrn, Pacane, Raph
8	Комментарии	Challe
9	Паб	Challe
10	Перечисления	Challe
11	Поток управления	Ganymede, Jan Vladimir Mostert, Pacane, Raph
12	Преобразование данных	Damon
13	Регулярные выражения	enyo
14	Совместимость Dart-JavaScript	Meshulam Silk
15	Список фильтров	jxmorris12
16	Струны	Challe
17	функции	Jan Vladimir Mostert, Kim Rostgaard Christensen