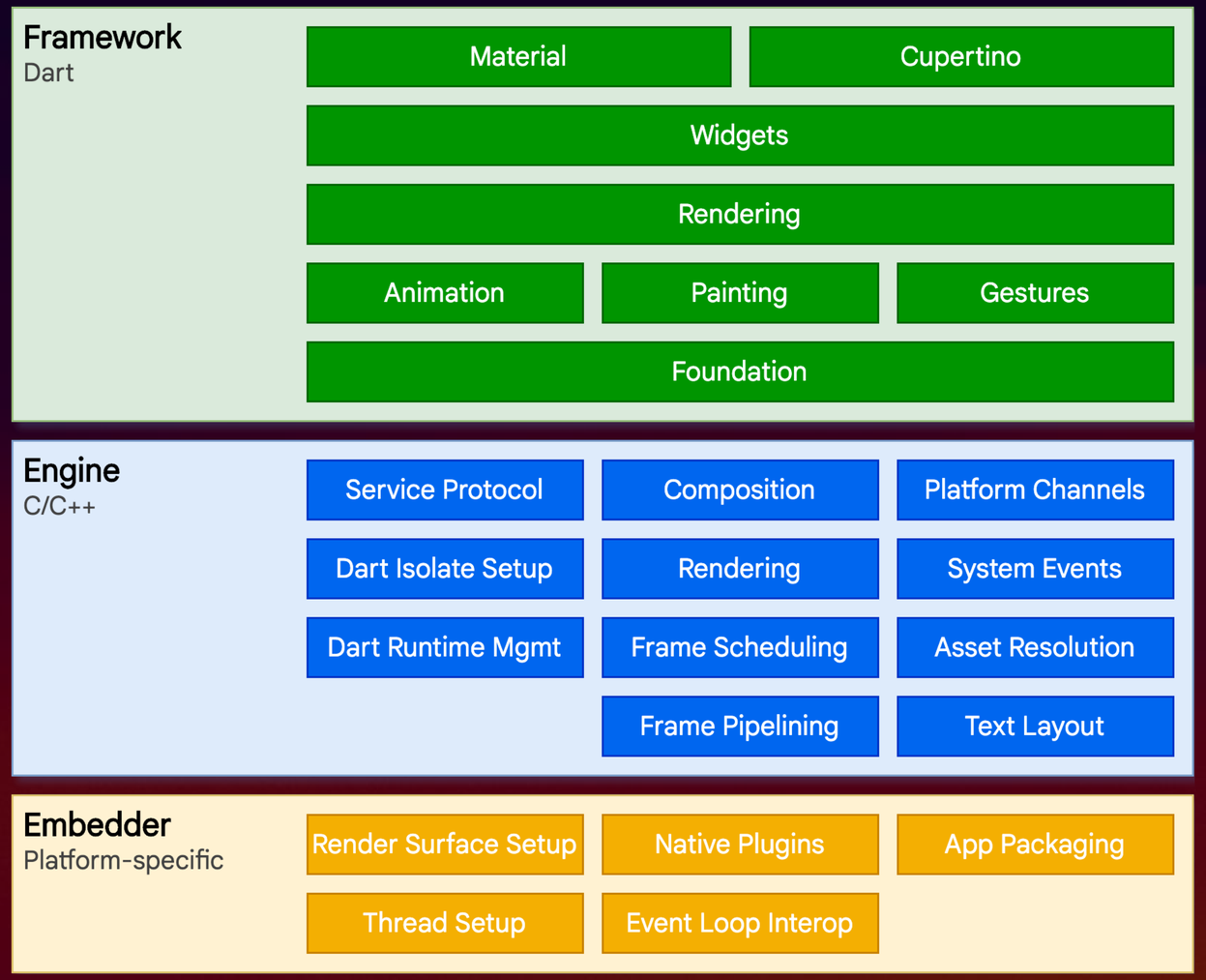
Introduction to Flutter & Dart

Flutter L1, CAL Computer Science Club

Michael Y. Qiu, 2025

What is Flutter?

- Flutter 是 Google 推出并开源的移动应用开发框架
- 使用跨平台引擎 Skia 作为其 2D 渲染引擎
- Flutter App 采用 Dart 语言开发



How to Install Flutter?

1. 下载 Flutter SDK
2. 将 Flutter SDK 解压并放置在任意文件夹 FLUTTER_PATH
3. 添加 PATH:
   ```
   export PATH=[FLUTTER_INSTALL_PATH]/flutter/bin:$PATH
   ```
4. 刷新 PATH:
   ```
   source $HOME/.bash_profile
   ```
5. iOS 开发: 安装 Xcode
   Android 开发: 安装 Android Studio

Intro to Dart

Dart ≈ Java + JS

Intro to Dart/Variables/var

- 强类型语言，任何变量在确定后类型也随之确定

```
Error: A value of type 'String' can't be assigned to a variable of type 'int'.
```

- 在不确定时默认赋值 null，且可后续再指定类型

```
Output:
114
null
Bye, world!
```

```dart
main() {
  var a = 114;
  print(a);

  a = 'Hello, world!';
  print(a);

  var b;
  print(b);

  b = 'Bye, world!';
  print(b);
}
```

Intro to Dart/Variables/dynamic&Object

Object

- 所有对象的根基类，也就是说在 Dart 中所有类型都是Object的子类(包括Function和Null)，所以任何类型的数据都可以赋值给Object声明的对象

dynamic

- 动态类型，编译时关闭类型检查。仅运行时类型检查，所以很可能会导致运行时错误

```
1  main() {
2    dynamic d = 10;
3    Object j = '0.o';
4    d = 'o.0';
5    j = 23;
6
7    var j1 = j + j;
8    print(j1);
9
10   var j2 = (j as int) + (j as int);
11   print(j2);
12 }
```

Intro to Dart/Variables/dynamic&Object

```dart
main() {
  dynamic d = 10;
  Object j = '0.o';
  d = 'o.0';
  j = 23;

  var j1 = j + j;
  print(j1);

  var j2 = (j as int) + (j as int);
  print(j2);
}
```

```
Error: The operator '+' isn't
defined for the type 'Object'.
```

Object

- Object 没有定义 + 操作符
- 需要进行显式类型转化 as

Intro to Dart/Variables/dynamic&Object

```dart
main() {
  dynamic d = 10;
  Object j = 'O.o';
  d = 'o.O';
  j = 23;

  print(d.length)
}
```

```
Unhandled exception:
NoSuchMethodError: Class 'int' has no
instance getter 'length'.
Receiver: 10
Tried calling: length
```

dynamic

- 对于 Integer 没有属性 length
- 编译时跳过检查

Intro to Dart/Variables/final&const

final

- 运行时常量

const

- 编译时常量

```
1  main() {
2    final f = 3.14;
3    const c = 2.72;
4  }
```

Intro to Dart/Variables/final&const

final

- 运行时常量

const

- 编译时常量

```
main() {
  final f = 3.14;
  const c = 2.72;
}
```

```
final ff = ff * 2;
const cc = c + c;
const ccc = ff + 2;
```

Intro to Dart/Variables/final&const

final

- 运行时常量

const

- 编译时常量

```
main() {
  final f = 3.14;
  const c = 2.72;
}
```

1. Valid
2. Valid
3. Invalid

```
final ff = ff * 2;
const cc = c + c;
const ccc = ff + 2;
```

Intro to Dart/Variables/final&const

final

- 运行时常量

const

- 编译时常量

```
final ff = ff * 2;
const cc = c + c;
const ccc = ff + 2;
```

Dart

```
const name = val;
```

⇔

C++

```
#define name val
```

Intro to Dart/Data Structure/List

```dart
main() {
  List phones = ['Apple', 'Samsung'];
  phones.add('Huawei');
  phones.addAll(['Vivo', 'Oppo', 'Xiaomi']);

  phones.remove('Huawei');
  phones.removeLast();
  phones.removeRange(0, 2); // [start, end)
}
```

Intro to Dart/Data Structure/List

```dart
main() {
  List<String> phones = ['Apple', 'Samsung',
    'Huawei', 'Vivo', 'Xiaomi'];

  print(phones.length);
  print(phones.first);
  print(phones.last);
  print(phones.isEmpty);

  phones.forEach((item) {
    print(item);
  });

  print(phones.every((item) {
      return item == 'Apple';
    }));

  print(phones.where((item){
    return item.endsWith('i');
  }).toList());
}
```

Intro to Dart/Data Structure/Map

```dart
main() {
  Map huChenFeng = {
    'iPhone 17': '苹果手机',
    'Xiaomi 17': '安卓手机',
    'Vivo X300': '安卓手机',
    'Mate 60': '安卓手机'
  };

  print(huChenFeng['iPhone 17']);
  print(huChenFeng['iPhone 114']);

  huChenFeng['Mate 60'] = '鸿蒙手机';
  print(huChenFeng['Mate 60']);
}
```

Intro to Dart/Data Structure/Map

```dart
main() {
  Map huChenFeng = {'iPhone 17': '苹果手机', 'Xiaomi 17': '安卓手机',
    'Vivo X300': '安卓手机', 'Mate 60': '鸿蒙手机'};

  huChenFeng.forEach((key, val) {
    print('$key 是$val');
  });

  huChenFeng.addAll({'Find X9': '安卓手机'});
  print(huChenFeng.containsKey('iPhone 17'));
  print(huChenFeng.containsValue('鸿蒙手机'));
  huChenFeng.clear();
  print(huChenFeng); // {}
}
```

Intro to Dart/if&while

```dart
if (condition1) {
  //sentences
} else if (condition2) {
   // sentences
} else {
  // sentences
}
```

```dart
while (condition) {
  // sentences
}

do {
  // sentences
} while (condition);
```

```dart
condition ? val1 : val2

if (condition) {
  return val1;
} else {
  return val2;
}
```

Intro to Dart/Functions

(数据类型) 函数名 定义函数

数据类型留空默认 dynamic

```
type func => val;
```

⇔

```
type func { return val; }
```

```
main() {
  String iAmAFunc() {
    return 'Func';
  }
  funcFunc() {
    return 67;
  }

  bool isEven(int x) => x%2==0;
  print(isEven(2));
}
```

Intro to Dart/mixin

mixin

- 用于 “组合” 多个类

with

- “组合” 成不同的类

extends

- 继承

```
class Gezi {
  say() {
    print('say');
  }
}
mixin Yali {
  yali() {
    print('压!力! ');
  }
}
class Gezi with Yali{}

class Person {
  eat() {
    print('eat');
  }
}
class Man extends Person with Yali{}

main() {
  var a = Gezi();
  a.yali();
}
```

异步非阻塞语言

异步 Asynchronous

异步Asynchronous

含义:
程序在发起一个耗时操作后，不会等待它完成，而是立即继续执行后续代码。当操作完成后再通知程序结果已就绪。

目的:
避免主线程被长时间挂起，提高程序的并发能力和响应速度。

你点了一杯咖啡（发起请求）

继续刷手机（执行其他代码）

叫你去拿咖啡（处理结果）

非阻塞Non-Blocking

非阻塞Non-Blocking

含义:
调用一个函数时，如果该操作不能立即完成（比如数据还没从网络到达），函数会立刻返回，而不是卡住当前线程等待结果。

目的:
“非阻塞”强调的是不占用线程资源等待，让线程可以去做别的事。

你点了一杯咖啡（发起请求）

继续刷手机（执行其他代码）

叫你去拿咖啡（处理结果）

Intro to Dart/Async/Future

```dart
Future<String> orderCoffee() {
  return Future.delayed(
    Duration(seconds: 3), () => "拿铁 ☕ "
  );
}

void main() {
  var ticket = orderCoffee();
  print("继续刷手机...");
  ticket.then((coffee) => print("拿到 $coffee! "));
}
```

Future<typename>

- 取餐号
- 不代表咖啡本身，而是未来会拿到咖啡的承诺

Intro to Dart/Async/Future

```dart
Future<String> orderCoffee() {
  return Future.delayed(
    Duration(seconds: 3), () => "拿铁 ☕ "
  );
}

void main() {
  var ticket = orderCoffee();
  print("继续刷手机...");
  ticket.then((coffee) => print("拿到 $coffee! "));
}
```

Future<typename>

- 取餐号
- 不代表咖啡本身，而是未来会拿到咖啡的承诺

点咖啡

零帧起手刷手机

3秒后拿咖啡

Intro to Dart/Async/Future

```dart
Future<String> orderCoffee() {
  return Future.delayed(Duration(seconds: 2), () {
    throw Exception("咖啡机坏了！");
    // return "拿铁 ☕";
  });
}

Future<String> orderSandwich() {
  return Future.delayed(Duration(seconds: 3), () {
    return "三明治 🥪";
  });
}

void main() {
  Future.wait([
    orderCoffee(),
    orderSandwich(),
  ])
      .then((items) => print("享用 ${items.join(' + ')}！"))
      .catchError((error) => print("😭 $error → 改吃泡面吧"))
      .whenComplete(() => print("🧹 托盘已回收"));
}
```

Future.wait()

任意错误都将 .catchError

全部执行完后才 .then

Callback Hell

```
validateCard().then((valid) {
  if (valid) orderCoffee().then((coffee) {
    addSugar(coffee).then((sweet) {
      postToSocial(sweet);
    });
  });
});
```

Intro to Dart/Async/Callback Hell

```dart
Future<void> enjoyCoffee() async {
  try {
    await validateCard();      // 等会员验证
    var coffee = await orderCoffee(); // 等咖啡
    var sweet = await addSugar(coffee); // 等加糖
    await postToSocial(sweet); // 等发朋友圈
    print("完美下午茶 ☕✨ ");
  } catch (e) {
    print("出错了：$e");
  }
}
```

async/await 让异步代码看起来像同步，逻辑线性清晰。

店员（事件循环）在后台协调所有任务。

Intro to Dart/Async/Stream

```dart
Stream<String> refillService() async* {
  for (int i = 1; i <= 3; i++) {
    await Future.delayed(Duration(seconds: 2));
    yield "第 $i 杯咖啡 ☕ ";
  }
}

void main() {
  refillService().listen(
    (cup) => print("拿到 $cup"),
    onDone: () => print("续杯结束，谢谢！ ")
  );
}
```

- Stream 处理多次异步事件（如实时消息、传感器数据）
- 用 .listen() 订阅事件流
- onDone: 处理结束
- 注意 async* 的 * 代表返回一系列值而非单个值

Asynchronous

异步

点单后等，不堵柜台

Future, Stream

非阻塞

店员接单后立刻服务下一位

事件循环 + 回调

错误处理

咖啡机坏了及时通知

catchError, try/catch

并发

同时处理多张订单

Future.wait

可读性

用自然语言描述流程

async/await

Appendix & Resources

- Flutter SDK 清华大学镜像: https://mirrors.tuna.tsinghua.edu.cn/flutter/flutter_infra_release/releases/stable
- 若使用终端是 zsh，请在 ~/.zshrc 中添加 source ~/.bash_profile
- 在 VSCode 下，长按 Control + Option 可显示自动推断的数据类型