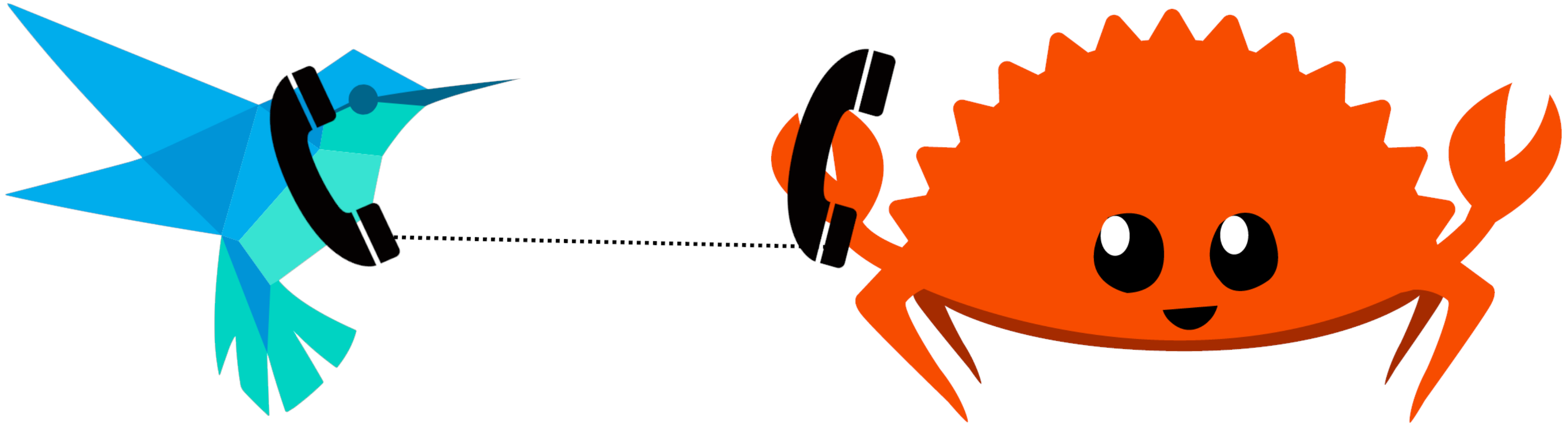
Everything I wish I knew about `flutter_rust_bridge` 6 months ago

Or how to write a phone app with as little non-rust code as possible.



(while still being able to google why that fucking button isn't displaying correctly)

Why is this a strong combo

- `flutter` is a `dart` app framework by google
- `dart` is typesafe like rust but is object orientated (not like rust)
- `dart` has "null safety" features
  - `!` which means `.unwrap()`
  - `??` means `.unwrap_or()`
- `flutter_rust_bridge` is a mature and actively developed library for calling rust code from flutter
- Allows you to hot reload things -- don't have to recompile rust things to make UI changes!
- They speak to each other well -- you can model sum types (enums) and match on them in dart

Arguably the best of both worlds

- Write your application model in rust
- Write your UI in a type safe but garbage collected language
- Deploy on any platform
- Have hot reloading of the UI without compiling rust

Enums can be translated between rust and dart

uses `freezed` dart library to emulate sum types

```rust
pub enum Maybe {
    None,
    Some { value: i32 },
}
```

```dart
Maybe maybe;
final value = switch (maybe) {
  Maybe_None() => 'got nothing',
  Maybe_Some(:final value) => 'got value: $value',
};
// single case à la if-let
if (maybe case Maybe_Some(:final value)) {
  ..
}
```

Protip: Don't use `(..)` parenthetical enum variants - they suck on the dart side

Moar enum translation

In the wild:

```
final Future<KeyId?> finished_key = keygenStream
        .asyncMap((event) => event.whenOrNull(finishedKey: (keyId) => keyId))
        .firstWhere((element) => element != null);
```

is congruent to in rust:

```
let finished_key: Future<Output=Option<KeyId>> = keygen_stream.find_map(|event| match event {
    Event::FinishedKey(key_id) => Some(key_id),
    _ => None
});
```

How to write an app with as little dart as possible

Strategy: Model everything in rust and emit streams of events that flutter will react to.

- i.e. BLoC pattern (Business Logic Component)
- Business logic (rust) takes in all events (could be from i/o, user actions etc)
- Business logic (rust) emits stream of states (to dart widgets)
- Includes the full state of the model in the event (not just event details)

How to structure events

If you can, put the whole current state of model in the event:

```rust
#[derive(Clone, Debug)]
pub struct DeviceListUpdate {
    pub changes: Vec<DeviceListChange>,
    pub state: DeviceListState,
}

#[derive(Clone, Debug)]
pub struct DeviceListChange {
    pub kind: DeviceListChangeKind,
    pub index: usize,
    pub device: Device,
}

#[derive(Clone, Debug)]
pub struct DeviceListState {
    pub devices: Vec<Device>,
    pub state_id: usize,
}

#[derive(Clone, Debug)]
pub struct Device {
    pub name: Option<String>,
    pub id: DeviceId,
}

#[frb(mirror(DeviceId))]
pub struct _DeviceId(pub [u8; 33]);
```

Add event stream subscription in `api.rs`

```rust
// api.rs
lazy_static! {
    static ref DEVICE_LIST: Mutex<(DeviceList, Option<StreamSink<DeviceListUpdate>>)> = Default::default();
}


// public api
pub fn sub_device_events(new_stream: StreamSink<DeviceListUpdate>) {
  let mut device_list_and_stream = DEVICE_LIST.lock().unwrap();
  let (_, stream) = &mut *device_list_and_stream;
  if let Some(old_stream) = stream.replace(new_stream) {
    old_stream.close();
  }
}
```

`StreamSink` is a magic word

Tap into them them on the dart side

```dart
import 'ffi.dart' if (dart.library.html) 'ffi_web.dart';

// global stream any widget can tap into at any time
Stream<DeviceListUpdate> deviceListUpdateStream =
    api.subDeviceEvents().asBroadcastStream();
```

Note the `asBroadcastStream()`

Have widgets change when something happens `StreamBuilder`

```dart
@override
Widget build(BuildContext context) {
  final button = StreamBuilder(
      initialData: api.deviceListState(),
      stream: deviceListStateStream,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          final deviceListState = snapshot.data!;
          return DoKeyGenButton(
              deviceListState: deviceListState, onSuccess: onSuccess);
        } else if (snapshot.hasError) {
          return Text('Error: ${snapshot.error}');
        } else {
          return Text('Unreachable: we set initialData');
        }
      });
  ..
}
```

How to set `initialData`

We can't use async api calls in the widget's `build` method so we have to call rust synchronously.

Wrap it in a `SyncReturn`.

```rust
// api.rs
pub fn device_list_state() -> SyncReturn<DeviceListState> {
    SyncReturn(DEVICE_LIST.lock().unwrap().0.state())
}
```

There are other ways to do this but this has served me well.

Problem: can't use `StreamBuilder` all the time

For example, `AnimatedList` doesn't work off a current state.
Rather you have to tell the list which items and being added and removed and where.

```dart
class _DeviceListState extends State<DeviceList> {
  GlobalKey<AnimatedListState> deviceListKey = GlobalKey<AnimatedListState>();
  StreamSubscription? _subscription;
  late DeviceListState currentListState;

  @override
  void initState() {
    super.initState();
    currentListState = api.deviceListState();
    _subscription = deviceListUpdateStream.listen((update) async {
        for (final change in update.changes) {
          switch (change.kind) {
            case DeviceListChangeKind.Added: {
                deviceListKey.currentState!.insertItem(change.index,
                    duration: const Duration(milliseconds: 800));
              }
            case DeviceListChangeKind.Removed: {
                deviceListKey.currentState!.removeItem(change.index,
                    (BuildContext context, Animation<double> animation) {
                      return widget.deviceBuilder(context, change.device, animation);
                    });
              }
          }
      }
      setState(() {
        currentListState = update.state;
      });
    });
  }

  @override
  Widget build(BuildContext context) {
    final list = AnimatedList(
        key: deviceListKey,
        itemBuilder: (context, index, animation) {
          final device = currentListState.devices[index];
          return widget.deviceBuilder(context, device, orientation, animation);
        },
        initialItemCount: currentListState.devices.length,
    );
    return list;
  }

  @override
  void dispose() {
    _subscription?.cancel();
    super.dispose();
  }
}
```

Summary

State based updating

- use `StreamBuilder`
- emit streams from rust
- The items in the streams are structs with full state
- Get the `initialData` (initial state) from a `SyncReturn` function

Event based updating

- subscribe to event stream in `initState`
- Each event is defined in rust as an `enum`
- use `case` statement in `dart` like you would use `match`

Usually your rust doesn't need to care about which one of these you are doing just emit full state along with the event any time anything changes

Problem: dart can call rust but not the other way around

What if you wanted to use the phone's native HTTP client to do HTTP requests (e.g. respect HTTP proxy settings).

In my case we have to use the phone's USB port to communicate with our devices.

Solution: emit stream of requests to host

```rust
#[derive(Debug)]
pub enum PortRequest {
    Open { request: PortOpen },
    Write { request: PortWrite },
    Read { request: PortRead },
    BytesToRead { request: PortBytesToRead },
}

#[derive(Debug)]
pub struct PortOpen {
    pub id: String,
    pub baud_rate: u32,
    pub ready: RustOpaque<PortOpenSender>,
}

impl PortOpen {
    // called from dart
    pub fn satisfy(&self, err: Option<String>) {
        let result = match err {
            Some(err) => Err(frostsnap_coordinator::PortOpenError::Other(err.into())),
            None => Ok(()),
        };

        let _ = self.ready.0.send(result);
    }
}
```

```rust
#[derive(Debug)]
pub struct PortOpenSender(pub SyncSender<Result<(), PortOpenError>>);

// not in api.rs used internally by our rust lib
fn open_device_port(id: &str, baud_rate: u32) -> Result<SerialPort, PortOpenError> {
    let (tx, rx) = std::sync::mpsc::sync_channel(0);

    crate::api::emit_event(PortRequest::Open {
        request: crate::api::PortOpen {
            id: id.into(),
            baud_rate,
            ready: RustOpaque::new(PortOpenSender(tx)),
        },
    })
    .map_err(|e| PortOpenError::Other(e.into()))?;

    rx.recv().map_err(|e| PortOpenError::Other(Box::new(e)))??;

    let port = FfiSerialPort {
        id: id.to_string(),
        baud_rate,
    };

    Ok(Box::new(port))
}
```

Using rust types in dart

- Equality in dart seems to be referential (sometimes 😞)
- This is even true for `HashSet` and `HashMap` types! 😱
- Haven't found an easy way to do custom equality for rust defined types
- Have to remember to use custom equality functions and types

```rust
// mirror the type in rust
#[frb(mirror(DeviceId))]
pub struct _DeviceId(pub [u8; 33]);
```

```dart

HashSet<DeviceId> deviceIdSet() {
  return HashSet<DeviceId>(
      equals: (a, b) => listEquals(a.field0, b.field0),
      hashCode: (a) => Object.hashAll(a.field0));
}

bool deviceIdEquals(DeviceId lhs, DeviceId rhs) =>
    listEquals(lhs.field0.toList(), rhs.field0.toList());
```

Tips for setup

Install flutter

```
flutter doctor
```

Generally start from a template project.

https://github.com/Desdaemon/flutter_rust_bridge_template

Look at `justfile` you might need to edit it a bit.

Probably won't work first time on all platforms.

Putting it inside a workspace requires extra steps (need to change `rust.cmake`)

Make sure you are using the same version of `flutter_rust_bridge` in all places (`pubspec.yaml`, `Cargo.toml`, `flutter_rust_bridge_codegen` cli).

You will suffer a lot for a day or two.

RTFM

Once you've got the basics working do read the `flutter_rust_bridge` manual

Questions?