Async Widgets

AsyncSnapshot

Immutable representation of the most recent interaction with an asynchronous computation.

Неизменяемое представление последнего взаимодействия с асинхронным вычислением.

Constructors

AsyncSnapshot.nothing()

Creates an AsyncSnapshot in ConnectionState.none with null data and error.

AsyncSnapshot.waiting()

Creates an AsyncSnapshot in ConnectionState.waiting with null data and error.

AsyncSnapshot.withData(ConnectionState state, T data)

Creates an AsyncSnapshot in the specified state and with the specified data.

AsyncSnapshot.withError(ConnectionState state, Object error, [StackTrace stackTrace = StackTrace.empty])

Creates an AsyncSnapshot in the specified state with the specified error and a stackTrace. [...]

Properties

connectionState → ConnectionState

Current state of connection to the asynchronous computation.

data → T?

The latest data received by the asynchronous computation. [...]

error → Object?

The latest error object received by the asynchronous computation. [...]

hasData → bool

Returns whether this snapshot contains a non-null data value. [...]

hasError → bool

Returns whether this snapshot contains a non-null error value. [...]

hashCode → int

The hash code for this object. [...]

requireData → T

Returns latest data received, failing if there is no data. [...]

stackTrace → StackTrace?

The latest stack trace object received by the asynchronous computation. [...]

ConnectionState

none, maybe with some initial data.

waiting, indicating that the asynchronous operation has begun, typically with the data being null.

active, with data being non-null, and possible changing over time.

done, with data being non-null.

FutureBuilder

Constructors

FutureBuilder({Key? key, Future<T>? future, T? initialData, required AsyncWidgetBuilder<T> builder})

Creates a widget that builds itself based on the latest snapshot of interaction with a Future. [...]

Properties

builder → AsyncWidgetBuilder<T>

The build strategy currently used by this builder. [...]

future → Future<T>?

The asynchronous computation to which this builder is currently connected, possibly null. [...]

initialData → T?

The data that will be used to create the snapshots provided until a non-null future has completed. [...]

Examaple

```
//some wrap
child: FutureBuilder<String>(
    future: _calculation, // same calculation
    builder: (BuildContext context, AsyncSnapshot<String> snapshot) {
      List<Widget> children;
      if (snapshot.hasData) {
        children = <Widget>[
          Icon(
            Icons.check_circle_outline,
            color: Colors.green,
            size: 60,
          ),
          Padding(
            padding: const EdgeInsets.only(top: 16),
            child: Text('Result: ${snapshot.data}'),
          )
        ];
```

hasErroe snapshot

```
} else if (snapshot.hasError) {
        children = <Widget>[
          Icon(
            Icons.error_outline,
            color: Colors.red,
            size: 60,
          ),
          Padding(
            padding: const EdgeInsets.only(top: 16),
            child: Text('Error: ${snapshot.error}'),
          )
        ];
      }
```

Another use-case

```
} else {
        children = <Widget>[
          SizedBox(
            child: CircularProgressIndicator(),
            width: 60,
            height: 60,
          ),
          const Padding(
            padding: EdgeInsets.only(top: 16),
            child: Text('Awaiting result...'),
          ) ]; }
      return Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          children: children,
        ),);},
```

StreamBuilder

Constructors

StreamBuilder({Key? key, T? initialData, Stream<T>? stream, required AsyncWidgetBuilder<T> builder})

Creates a new StreamBuilder that builds itself based on the latest snapshot of interaction with the specified stream and whose build strategy is given by builder. [...]

Properties

builder → AsyncWidgetBuilder<T>

The build strategy currently used by this builder. [...]

initialData → T?

The data that will be used to create the initial snapshot. [...]

Example

```
Stream<int> _bids = (() async* {
  await Future<void>.delayed(Duration(seconds: 1));
  yield 1;
  await Future<void>.delayed(Duration(seconds: 1));
})();
```

```
//Some wrap
child: StreamBuilder<int>(
        stream: _bids,
        builder: (BuildContext context, AsyncSnapshot<int> snapshot) {
          List<Widget> children;
          if (snapshot.hasError) {
            children = <Widget>[
              Icon(
                Icons.error_outline,
                color: Colors.red,
                size: 60,
              ),
              Padding(
                padding: const EdgeInsets.only(top: 16),
                child: Text('Error: ${snapshot.error}'),
              ),
              Padding(
                padding: const EdgeInsets.only(top: 8),
                child: Text('Stack trace: ${snapshot.stackTrace}'),
              ),
            ];
```

ConnectionState.None

```
..//
 } else {
          switch (snapshot.connectionState) {
            case ConnectionState.none:
              children = <Widget>[
                Icon(
                  Icons.info,
                  color: Colors.blue,
                  size: 60,
                ),
                const Padding(
                  padding: EdgeInsets.only(top: 16),
                  child: Text('Select a lot'),
                )
              ];
              break;
```

ConnectionState.waiting

```
case ConnectionState.waiting:
        children = <Widget>[
          SizedBox(
            child: const CircularProgressIndicator(),
            width: 60,
            height: 60,
          ),
          const Padding(
            padding: EdgeInsets.only(top: 16),
            child: Text('Awaiting bids...'),
          )
        ];
        break;
```

ConnectionState.active

```
case ConnectionState.active:
            children = <Widget>[
              Icon(
                Icons.check_circle_outline,
                color: Colors.green,
                size: 60,
              ),
              Padding(
                padding: const EdgeInsets.only(top: 16),
                child: Text('\$${snapshot.data}'),
              )
            ];
            break;
```

ConnectionState.done

```
case ConnectionState.done:
          children = <Widget>[
            Icon(
              Icons.info,
              color: Colors.blue,
              size: 60,
            ),
            Padding(
              padding: const EdgeInsets.only(top: 16),
              child: Text('\$${snapshot.data} (closed)'),
            )
          ];
          break;
      }
    }
```

Return result

```
return Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.center,
        children: children,
      );
    },
```