Flutter Under The Hood

Михаил Зотьев
Flutter-teamlead, Surf
mbixjkee

Flutter классный!

Как?

Как построить UI?

- Что будем отображать, из каких частей UI будет состоять
- Как эти части будут связаны между собой
- Построение макета и отрисовка

It's All Widgets!

- Является центральным классом в структуре Flutter.
- Неизменяемое описание части пользовательского интерфейса.
- Для изменяемой конфигурации используется специальная сущность State.
- Один и тот же виджет может быть включен в дерево виджетов множество раз, или вовсе не быть включенным.
- Каждый раз, когда виджет включается в дерево виджетов, ему сопоставляется элемент.

Element

- Элемент — представление виджета в определенном месте дерева.
- Каждое использование виджета в построении дерева порождает элемент.
- Виджет, который связан с элементом может со временем изменяться.

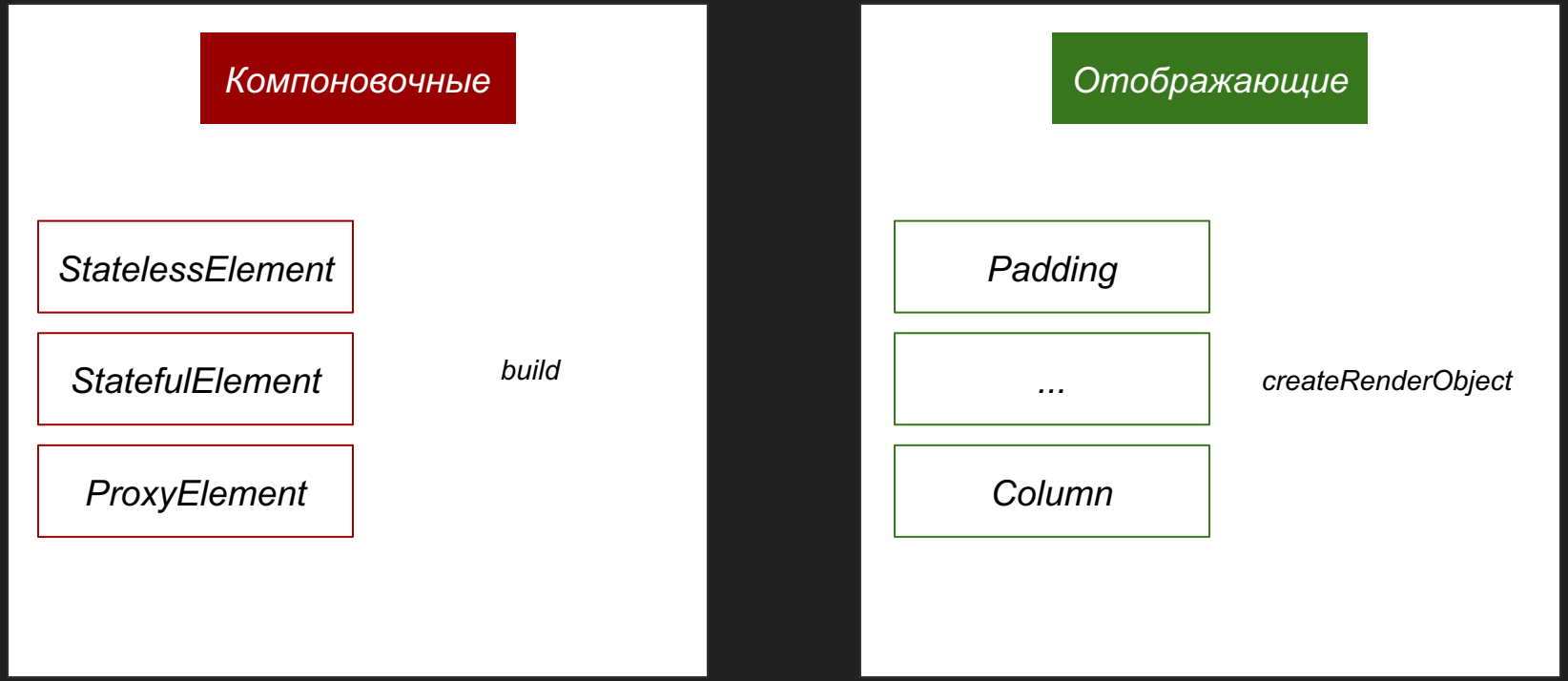
Жизненный цикл Element

Этап	Ключевой метод	Где
Создание	*createElement()*	*Widget*
Встраивание и активация	*mount(Element parent, dynamic newSlot)*	*Element*
Обновление/пересоздание	*static bool canUpdate(Widget oldWidget, Widget newWidget)*	*Widget*
Деактивация	*deactivate()*	*Element*
Переиспользование	*_retakeInactiveElement(GlobalKey key, Widget newWidget)*	*Element*

BuildContext

- управляет положением виджета в дереве виджетов
- BuildContext = Element
- позволяет избежать прямого манипулирования элементом

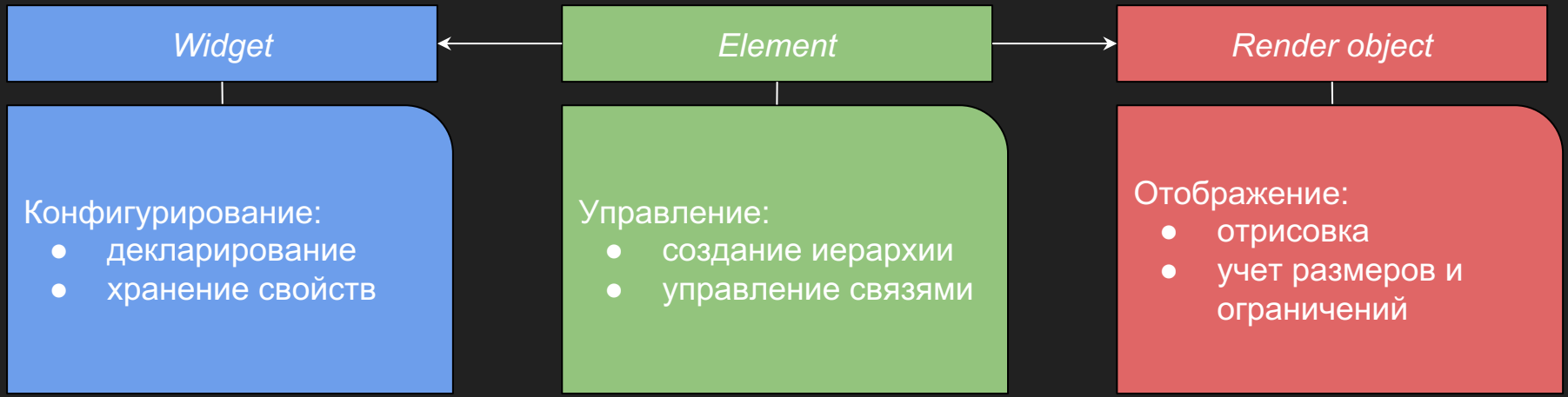
Element


Компоновочные
StatelessElement
StatefulElement
ProxyElement
build
Отображающие
Padding
...
Column
createRenderObject

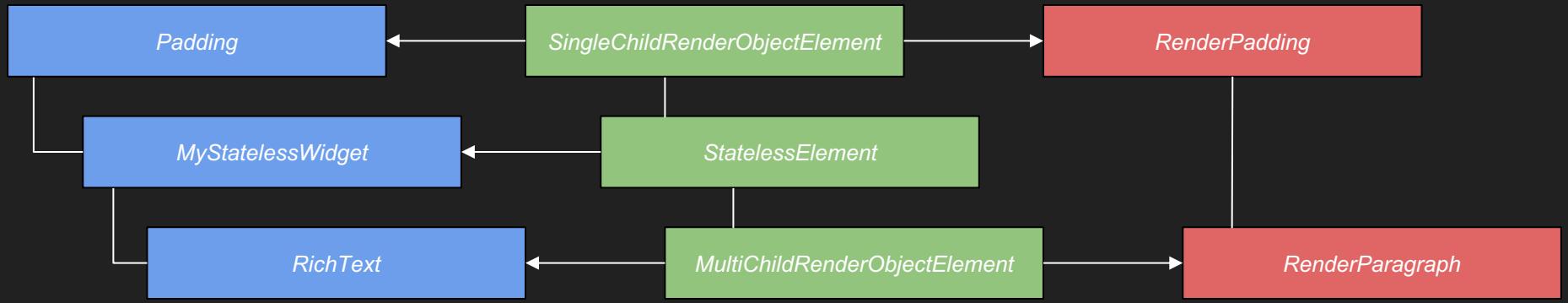
RenderObject

- отвечает за реализацию базовых протоколов отрисовки и расположения
- дочерние объекты могут отсутствовать, или иметься в любом количестве
- система позиционирования: картезианская система, полярные координаты, etc.
- протоколы ограничений: подстройка по ширине или высоте, ограничение размера, задание размеров и расположения родителем, использование данных родительского объекта

Картина мира Flutter



Дерево Flutter



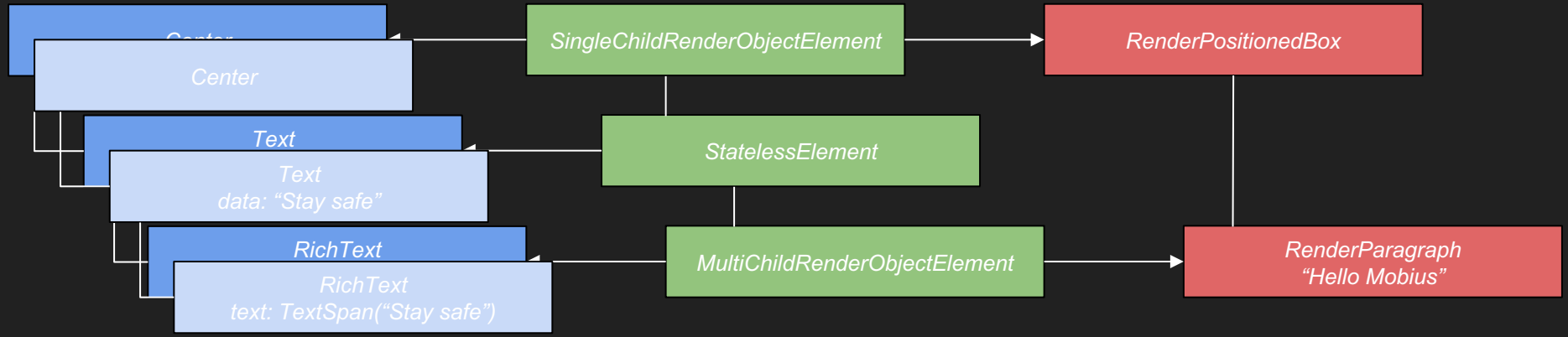
Поиск RenderObject

```
RenderObject get renderObject {
    RenderObject result;
    void visit(Element element) {
        assert(result == null);
        if (element is RenderObjectElement)
            result = element.renderObject;
        else
            element.visitChildren(visit);
    }
    visit(this);
    return result;
}
```

Обновление дерева

```
body: Center(
    child: Text(“Hello Mobius!”)
),
```

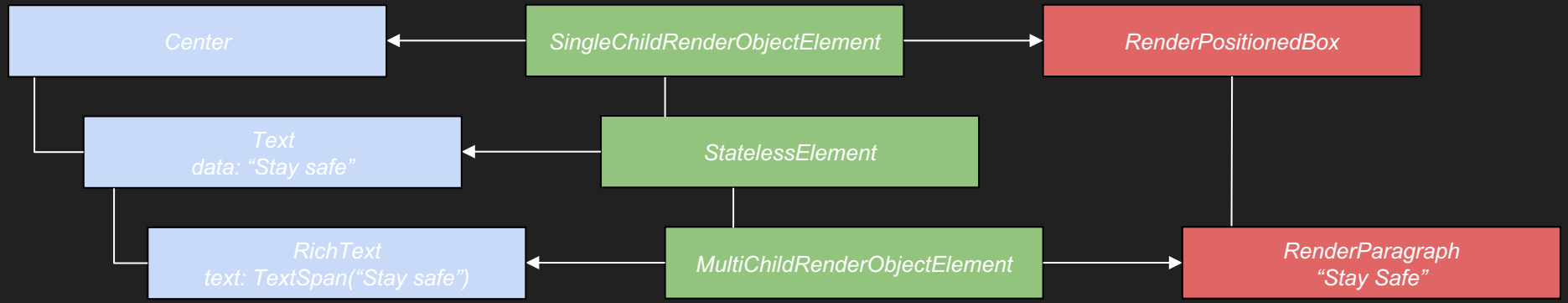
Обновление дерева



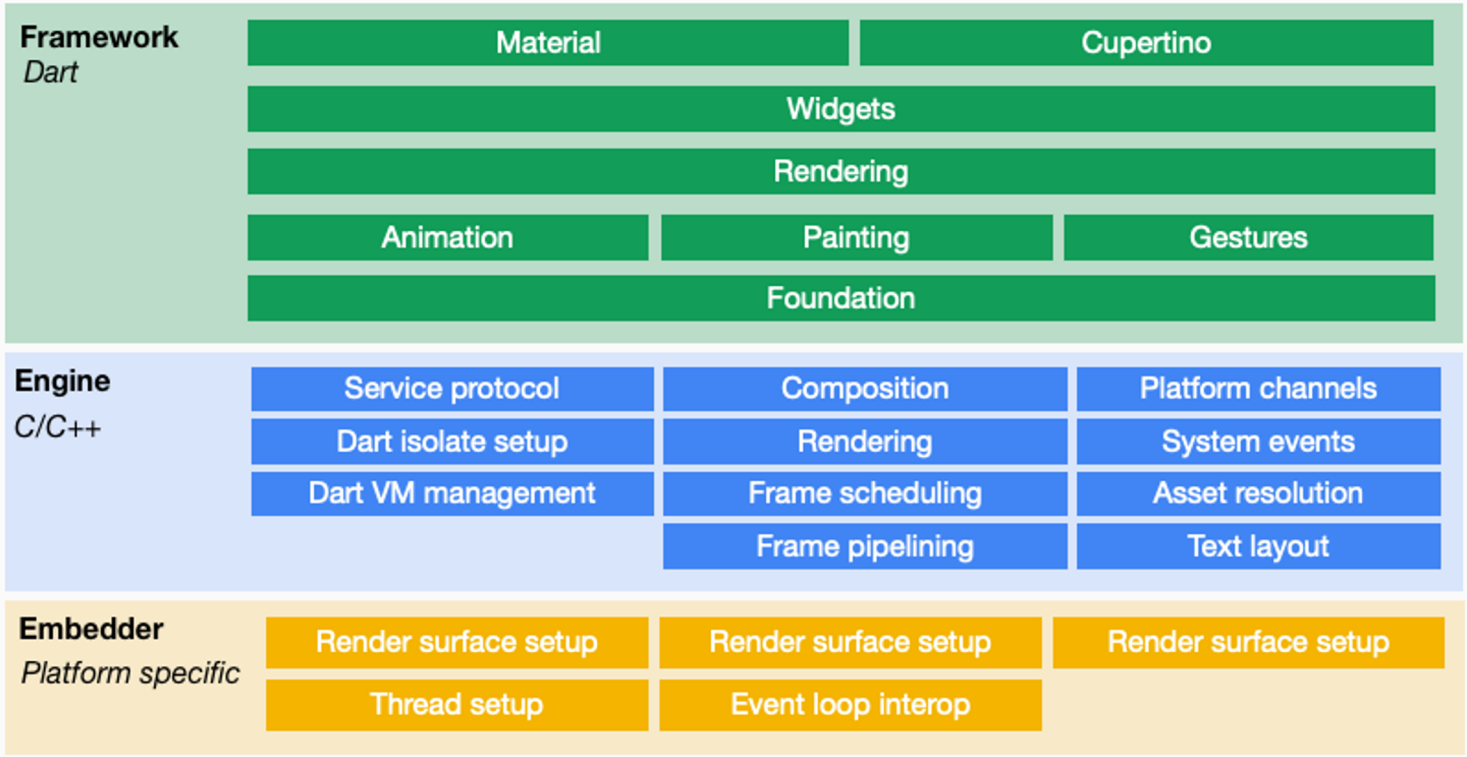
Обновление дерева

```
static bool canUpdate(Widget oldWidget, Widget newWidget) {
    return oldWidget.runtimeType == newWidget.runtimeType && oldWidget.key == newWidget.key;
}
```

Обновление дерева



Flutter system overview



Уровень фреймворка

Всё, с чем мы работаем в момент написания приложения, и все служебные классы, позволяющие взаимодействовать написанному нами с уровнем движка.

Уровень движка

Содержит классы и библиотеки, позволяющие работать уровню фреймворка. В том числе виртуальная машина Dart, Skia и тд.

Уровень платформы

Специфичные механизмы, относящиеся к конкретной платформе запуска

Уровень фреймворка

Foundation

Функции, определённые в этой библиотеке, представляют собой служебные классы и функции самого низкого уровня, используемые всеми другими уровнями фреймворка Flutter.

BindingBase

Это базовый класс для различных сервисов связи, которые представлены в виде миксинов. Каждый такой миксин инициализируется и гарантирует единственность своего экземпляра во время жизни приложения.

Binding

- ServicesBinding

  перенаправление сообщений от текущей платформы в обработчик данных сообщений

- PaintingBinding

  связь с библиотекой отрисовки

- RenderBinding

  связь между деревом рендеринга и движком Flutter

- WidgetBinding

  связь между деревом виджетов и движком Flutter

Binding

- SchedulerBinding

  планировщик очередных задач: приходящих колбеков, непрерывных колбеков, посткадровых колбеков, задач не связанных с рендерингом, которые должны быть выполнены между кадрами

- SemanticsBinding

  связь слоя семантики с движком Flutter

- GestureBinding

  работа с подсистемой жестов

WidgetsFlutterBinding

```
void runApp(Widget app) {
  WidgetsFlutterBinding.ensureInitialized()
    ..scheduleAttachRootWidget(app)
    ..scheduleWarmUpFrame();
}
```

class WidgetsFlutterBinding extends BindingBase with GestureBinding, SchedulerBinding, ServicesBinding, PaintingBinding, SemanticsBinding, RendererBinding, WidgetsBinding

ScheduleAttachRootWidget

```
void attachRootWidget(Widget rootWidget) {
    _renderViewElement = RenderObjectToWidgetAdapter<RenderBox>(
      container: renderView,
      debugShortDescription: '[root]',
      child: rootWidget,
    ).attachToRenderTree(buildOwner, renderViewElement);
}
```

RenderObjectToWidgetAdapter является корневым виджетом дерева виджетов и используется как мост, связывающий деревья между собой.

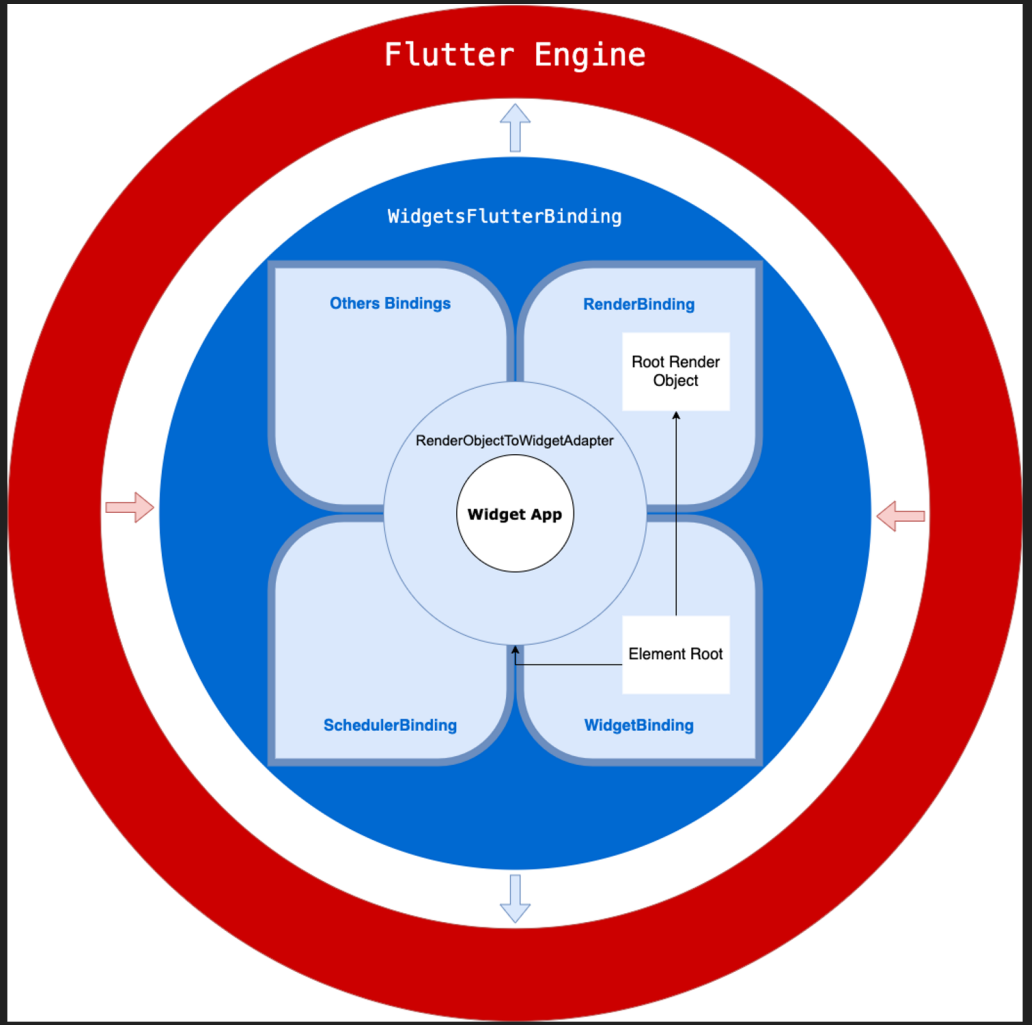
```
RenderView get renderView => _pipelineOwner.rootNode;
```

attachToRenderTree

```
RenderObjectToWidgetElement<T> attachToRenderTree(BuildOwner owner, [ RenderObjectToWidgetElement<T> element]) {
    if (element == null) {
      owner.lockState(() {
        element = createElement();
        assert(element != null);
        element.assignOwner(owner);
      });
      owner.buildScope(element, () {
        element.mount(null, null);
      });
      // This is most likely the first time the framework is ready to produce
      // a frame. Ensure that we are asked for one.
      SchedulerBinding.instance.ensureVisualUpdate();
    } else {
      element._newWidget = this;
      element.markNeedsBuild();
    }
    return element;
  }
```

ScheduleWarmUpFrame

используется для того, чтобы запланировать запуск кадра как можно скорее, не ожидая системного сигнала «Vsync»


Flutter Engine
WidgetsFlutterBinding
Others Bindings
RenderBinding
Root Render Object
RenderObjectToWidgetAdapter
Widget App
Element Root
SchedulerBinding
WidgetBinding

Этапы построения кадра

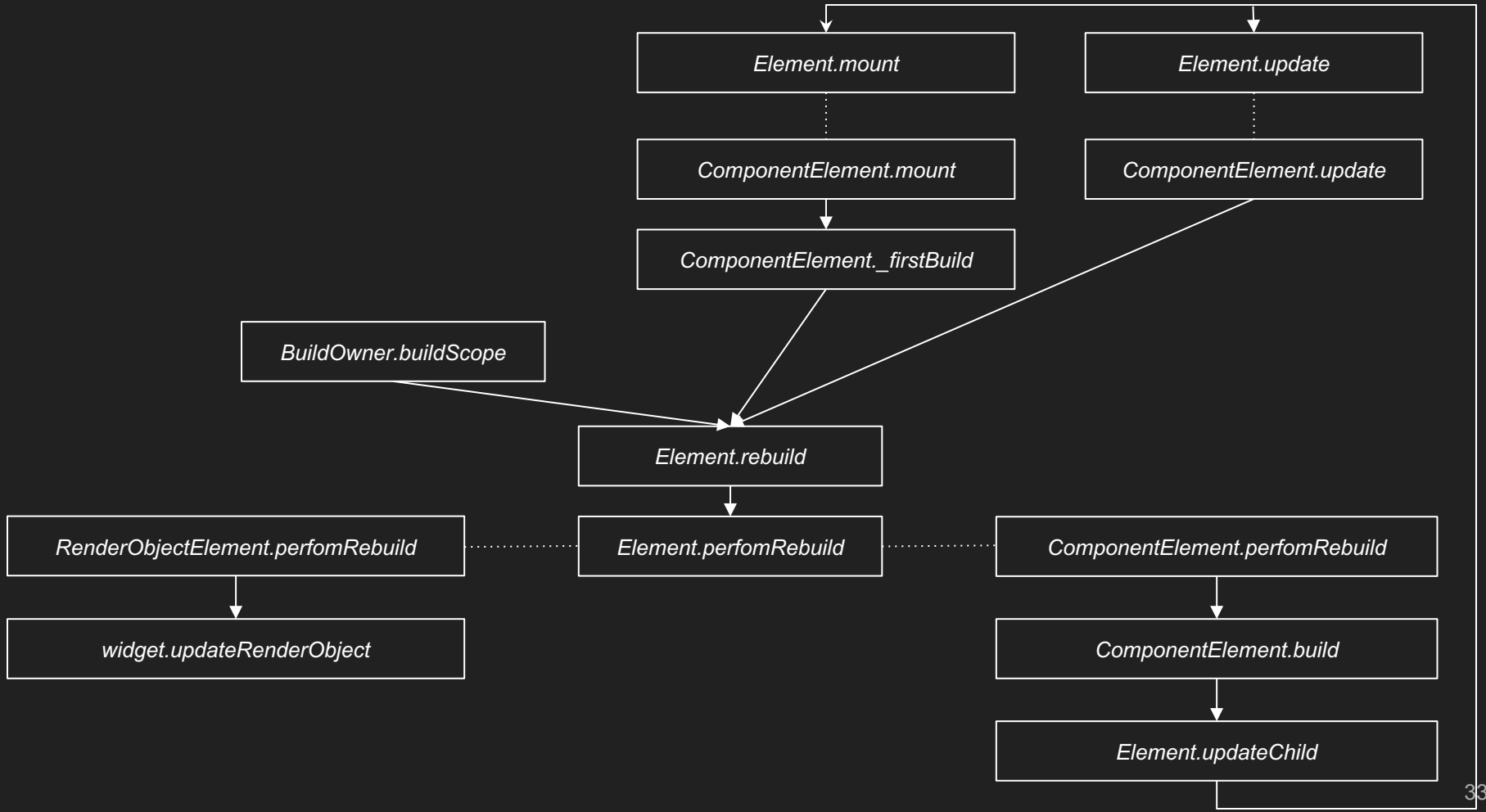
- Фаза анимации
- Фаза микротасков
- Фаза сборки
- Фаза построения макета
- Фаза композиции частей
- Фаза отрисовки
- Фаза композиции
- Фаза семантики
- Фаза завершения сборки
- Фаза завершения подготовки кадра

BuildOwner

- хранит списки неактивных элементов и элементов, нуждающихся в обновлении
- *scheduleBuildFor* дает возможность пометить элемент как нуждающийся в обновлении.
- *lockState* обеспечивает защиту от обновления уничтоженных элементов.
- *buildScope* осуществляет пересборку дерева.
- *finalizeTree* завершает построение дерева.
- *reassemble*, который обеспечивает работу механизма HotReload.

WidgetsBinding.drawFrame

```
if (renderViewElement != null)
  buildOwner.buildScope(renderViewElement);
super.drawFrame();
buildOwner.finalizeTree();
```


Element.mount
Element.update
ComponentElement.mount
ComponentElement.update
ComponentElement._firstBuild
BuildOwner.buildScope
Element.rebuild
RenderObjectElement.perfomRebuild
Element.perfomRebuild
ComponentElement.perfomRebuild
widget.updateRenderObject
ComponentElement.build
Element.updateChild

Вызов scheduleBuildFor

- Элемент активируется из деактивированного состояния при переиспользовании.
- При присоединении дерева отрисовки в момент старта приложения.
- При вызове setState у состояния
- При использовании HotReload
- Когда изменились зависимости элемента (использование InheritedElement).

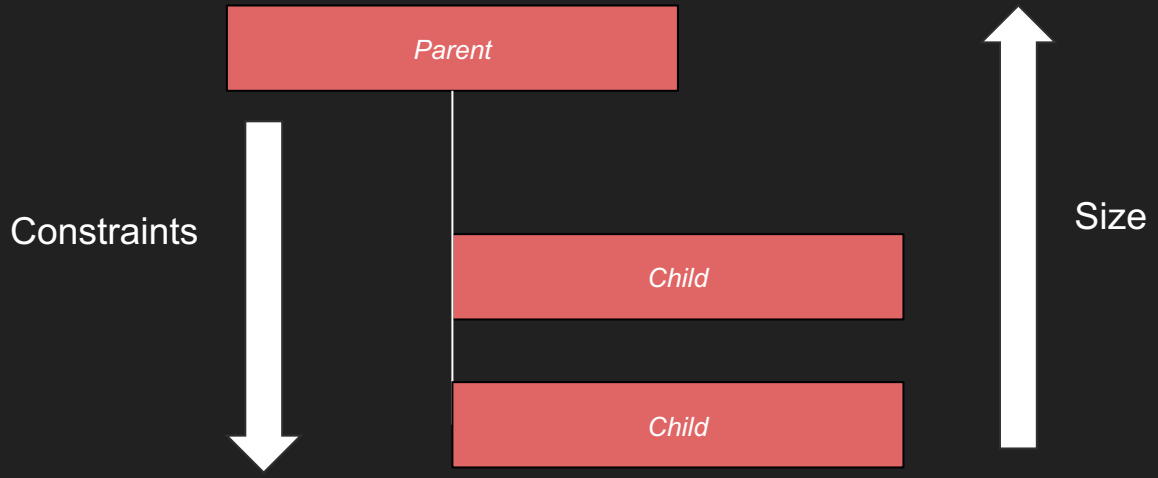
PipelineOwner

```
@protected
  void drawFrame() {
    assert(renderView != null);
    pipelineOwner.flushLayout();
    pipelineOwner.flushCompositingBits();
    pipelineOwner.flushPaint();
    if (sendFramesToEngine) {
      renderView.compositeFrame(); // this sends the bits to the GPU
      pipelineOwner.flushSemantics(); // this also sends the semantics to the OS.
      _firstFrameSent = true;
    }
  }
```

compositeFrame

```
final ui.SceneBuilder builder = ui.SceneBuilder();
final ui.Scene scene = layer.buildScene(builder);
if (automaticSystemUiAdjustment)
    _updateSystemChrome();
_window.render(scene);
```

Расчет макета



Оптимизация layout

При прежних ограничениях от родителя, дочерний объект может не перестраивать свой макет до тех пор, пока сам не посчитает это необходимым.

Оптимизация layout

Каждый раз, когда родитель вызывает у дочернего объекта метод layout, родитель указывает, использует ли он информацию о размере, возвращаемую дочерним объектом с помощью параметра *parentUsesSize*. Если он не использует информацию о размере ребенка,то ему не приходится повторно вычислять свой размер, даже при изменении размеров ребенка.

Оптимизация layout

Жесткие ограничения - это те ограничения, которым может удовлетворять только один допустимый размер. В случае задания жестких ограничений, родительский элемент не должен повторно вычислять свой размер, при перерасчете дочернего элемента,даже в случае использования родителем размеров ребенка в своем макете, потому что дочерний элемент не может изменить размер без новых ограничений от своего родителя.

Оптимизация layout

RenderObject может объявить, что использует для вычисления своих размеров только ограничения предоставленные родителем - *sizedByParent*. Это значит, что родительскому объекту этого объекта рендеринга не нужно выполнять перерасчет, при повторном вычислении у самого объекта, даже если ограничения не жесткие и даже если макет родительского элемента зависит от размера дочернего элемента, потому что дочерний элемент не может измениться своих размеров без новых ограничений от его родителя.

Surf Flutter Team

https://t.me/surf_flutter_team

NEWS Flutter Dev Подкаст

https://t.me/flutterdevpodcast_news

Oh, my Flutter

https://t.me/ohmyflutter

Спасибо за внимание!