Fabric 搭建

如何创建 Fabric?

创建一个 Fabric 组件分为以下步骤：

1. 编写 ArkTS 原生实现代码
2. 编写 JSI 层 C++ 代码
3. 编写 RN 调用 Fabric 组件的代码

本文档以示例工程中的 `MarqueeView` 为例，介绍了 Fabric 自定义组件的实现步骤。

1. 编写 ArkTS 原生实现代码

Descriptor

`Descriptor` 的功能是封装 RN 侧组件代码传递到 ArkUI 组件的参数，`MarqueeView` 对 RN 侧公开了一个 `src` 参数，用于显示跑马灯的滚动内容。原生侧定义 `MarqueeViewDescriptor` 代码如下：

```
export interface MarqueeViewProps extends ViewBaseProps {
  src: string
}

export type MarqueeViewDescriptor = Descriptor<"MarqueeView", MarqueeViewProps>;
```

`Descriptor` 不需要我们手动创建，由 rnoh 自动生成；组件 `tag` 也不需要我们手动设置，rnoh 会为组件自动分配 `tag`。开发者只需要通过 `getDescriptor` 方法获取对应 `tag` 的 `Descriptor`：

```
this.descriptor =
this.ctx.descriptorRegistry.getDescriptor<MarqueeViewDescriptor>(this.tag)
```

当 RN 侧传递过来的属性参数发生变化时，我们需要更新 `Descripotor`：

```
this.unregisterDescriptorChangesListener =
```

```
this.ctx.descriptorRegistry.subscribeToDescriptorChanges(this.tag,
(newDescriptor) => {
    this.descriptor = (newDescriptor as MarqueeViewDescriptor)
})
```

RN 调用原生方法

RN 侧调用 `UIManager.dispatchViewManagerCommand` 向原生发送消息：

```
UIManager.dispatchViewManagerCommand(
  findNodeHandle(nativeRef.current),
  'toggleMarqueeState',
  [],
)
```

原生组件通过 `commandDispatcher.registerCommandCallback` 接收消息并执行对应方法：

```
this.ctx.commandDispatcher.registerCommandCallback(this.tag, (commandName) => {
  if (commandName === "toggleMarqueeState") {
    this.start = !this.start
    console.log("will emitComponentEvent");
  }
})
```

原生组件调用 RN 侧方法

RN 侧添加 `onStop` 方法实现：

```
<MarqueeView
  ...
  onStop={(e) => {
    // 原生组件调用了 RN 侧的 MarqueeView 的 onStop 方法
    const isStop = e.nativeEvent.isStop
    ...
  }}
/>
```

原生侧发送调用 RN 组件事件的消息：

```
this.ctx.rnInstance.emitComponentEvent(
```

```
  this.descriptor.tag,
  "MarqueeView",
  { type: "onStop", isStop: !this.start }
)
```

buildCustomComponent

创建 `RNSurface` 加载 JSBundle 时，传入 `buildCustomComponent` 用于加载原生 Fabric 组件：

```
import { RNAbility, ComponentBuilderContext, RNSurface } from "rnoh";
import { MarqueeView } from '../customView/MarqueeView'

@Builder
public buildCustomComponent(ctx: ComponentBuilderContext) {
  if (ctx.descriptor.type === MarqueeView.NAME) {
    MarqueeView({
      ctx: ctx.rnohContext,
      tag: ctx.descriptor.tag
    })
  }
}
...

RNSurface({
  ...
  buildCustomComponent: this.buildCustomComponent,
})
```

2. 编写 JSI 层 C++ 代码

1. 首先创建属性 `Props` 和 事件 `Emitter` 两部分的 C++ 类，在 `Descriptor` 中进行绑定。
2. 实现 `MarqueeViewEventEmitRequestHandler` 的 `handleEvent` 方法，根据原生消息的事件名，调用 `eventEmitter` 向 RN 侧组件发送事件消息。
3. 实现 `MarqueeViewJSIBinder` 类的属性和事件绑定方法。
4. 实现 `MarqueeViewNapiBinder` 类的属性映射方法。
5. 将以上文件引入到 `SampleTurboModulePackage` 的对应方法实现中进行绑定。

Props

创建 `Props` 的 C++ 文件用于定义 `MarqueeView` 的 `Descriptor` 对应的属性。

`Props.h`：

```
#include <jsi/jsi.h>
#include <react/renderer/components/view/ViewProps.h>
#include <react/renderer/core/PropsParserContext.h>
#include <react/debug/react_native_assert.h>

namespace facebook {
namespace react {
class JSI_EXPORT MarqueeViewProps final : public ViewProps {
  public:
    MarqueeViewProps() = default;
    MarqueeViewProps(const PropsParserContext &context, const MarqueeViewProps
&sourceProps, const RawProps &rawProps);

#pragma mark - Props
    std::string src{""};
};
```

Props.cpp：

```
#include <react/renderer/components/rncore/Props.h>
#include <react/renderer/core/PropsParserContext.h>
#include <react/renderer/core/propsConversions.h>
#include "Props.h"

namespace facebook {
namespace react {
MarqueeViewProps::MarqueeViewProps(
    const PropsParserContext &context,
    const MarqueeViewProps &sourceProps,
    const RawProps &rawProps): ViewProps(context, sourceProps, rawProps),

    src(convertRawProp(context, rawProps, "src", sourceProps.src, {""}))
      {}
} // namespace react
} // namespace facebook
```

MarqueeViewEventEmitter

`MarqueeViewEventEmitter.h` 中添加 `onStop` 方法，并自定义了属性结构体：

```cpp
#include <react/renderer/components/view/ViewEventEmitter.h>
#include <jsi/jsi.h>

namespace facebook {
namespace react {

class JSI_EXPORT MarqueeViewEventEmitter : public ViewEventEmitter {
  public:
    using ViewEventEmitter::ViewEventEmitter;
    struct OnStop {
      bool isStop;
    };

    void onStop(OnStop value) const;
};

} // namespace react
} // namespace facebook
```

`MarqueeViewEventEmitter.cpp` 中实现 `onStop` 事件的发送和参数绑定：

```cpp
#include "MarqueeViewEventEmitter.h"

namespace facebook {
namespace react {

void MarqueeViewEventEmitter::onStop(OnStop event) const {
    dispatchEvent("stop", [event = std::move(event)](jsi::Runtime &runtime) {
        auto payload = jsi::Object(runtime);
        payload.setProperty(runtime, "isStop", event.isStop);
        return payload;
    });
}

} // namespace react
} // namespace facebook
```

MarqueeViewComponentDescriptor.h

将 `MarqueeViewProps`，`MarqueeViewEventEmitter` 绑定到 `MarqueeViewComponentDescriptor` 中：

```
#include <react/renderer/core/ConcreteComponentDescriptor.h>
#include <react/renderer/components/view/ConcreteViewShadowNode.h>
#include <react/renderer/components/view/ViewShadowNode.h>
#include "MarqueeViewEventEmitter.h"
#include "Props.h"

namespace facebook {
namespace react {

extern const char MarqueeViewComponentName[] = "MarqueeView";

  using MarqueeViewShadowNode = ConcreteViewShadowNode<MarqueeViewComponentName,
MarqueeViewProps, MarqueeViewEventEmitter>;
  using MarqueeViewComponentDescriptor =
ConcreteComponentDescriptor<MarqueeViewShadowNode>;

} // namespace react
} // namespace facebook
```

MarqueeViewEventEmitRequestHandler

`handleEvent` 方法中根据事件名调用事件消息发送方法 `eventEmitter->onStop(event)`：

```
class MarqueeViewEventEmitRequestHandler : public EventEmitRequestHandler {
    public:
    void handleEvent(EventEmitRequestHandler::Context const &ctx) override {
        if (ctx.eventName != "MarqueeView") {
            return;
        }
        ArkJS arkJs(ctx.env);
        auto eventEmitter = ctx.shadowViewRegistry-
>getEventEmitter<react::MarqueeViewEventEmitter>(ctx.tag);
        if (eventEmitter == nullptr) {
            return;
        }

        MarqueeViewEventType type = getMarqueeViewEventType(arkJs, ctx.payload);
        switch (type) {
        case MarqueeViewEventType::MARQUEE_VIEW_ON_STOP: {
            bool isStop =
(bool)arkJs.getBoolean(arkJs.getObjectProperty(ctx.payload, "isStop"));
            react::MarqueeViewEventEmitter::OnStop event{isStop};
            eventEmitter->onStop(event);
```

```
            break;
        }
        default:
            break;
        }
    };
};
```

MarqueeViewJSIBinder

JSIBinder 是 RN 侧的属性和方法在 JSI 层的实现，主要调用了 `object.setProperty(rt, "src", "string")` 和 `events.setProperty(rt, "topStop", createDirectEvent(rt, "onStop"))` 这两个方法，`events.setProperty` 中注意 `topStop` 和 `onStop` 的命名规则：

```
#pragma once
#include "RNOHCorePackage/ComponentBinders/ViewComponentJSIBinder.h"

namespace rnoh {
class MarqueeViewJSIBinder : public ViewComponentJSIBinder {
    facebook::jsi::Object createNativeProps(facebook::jsi::Runtime &rt) override
{
        auto object = ViewComponentJSIBinder::createNativeProps(rt);
        object.setProperty(rt, "src", "string");
        return object;
    }

    facebook::jsi::Object createDirectEventTypes(facebook::jsi::Runtime &rt)
override {
        facebook::jsi::Object events(rt);
        events.setProperty(rt, "topStop", createDirectEvent(rt, "onStop"));
        return events;
    }
};
} // namespace rnoh
```

NapiBinder

实现 C++ 代码和原生组件代码之间的属性映射，其中 `.addProperty("src", props->src)` 为 `MarqueeViewDescriptor` 的 `props` 增加了 `src` 字段；如果未添加该代码，`MarqueeView` 就需要从 `rawProps` 中获取 `src`：

```
#include "RNOHCorePackage/ComponentBinders/ViewComponentNapiBinder.h"
#include "Props.h"

namespace rnoh {
class MarqueeViewNapiBinder : public ViewComponentNapiBinder {
public:
    napi_value createProps(napi_env env, facebook::react::ShadowView const
shadowView) override {
        napi_value napiViewProps = ViewComponentNapiBinder::createProps(env,
shadowView);
        if (auto props = std::dynamic_pointer_cast<const
facebook::react::MarqueeViewProps>(shadowView.props)) {
            return ArkJS(env)
                .getObjectBuilder(napiViewProps)
                .addProperty("src", props->src)
                .build();
        }
        return napiViewProps;
    };
};
} // namespace rnoh
```

SampleTurboModulePackage

在 SampleTurboModulePackage.h 中添加自定义组件相关的方法声明：

```
#include "RNOH/Package.h"

namespace rnoh {
class SampleTurboModulePackage : public Package {
    public:
        std::vector<facebook::react::ComponentDescriptorProvider>
createComponentDescriptorProviders() override;
        ComponentNapiBinderByString createComponentNapiBinderByName() override;
        ComponentJSIBinderByString createComponentJSIBinderByName() override;
        EventEmitRequestHandlers createEventEmitRequestHandlers() override;
    };
} // namespace rnoh
```

使用 `MarqueeViewComponentDescriptor`、`MarqueeViewEventEmitRequestHandler`、`MarqueeViewNapiBinder`、`MarqueeViewJSIBinder` 在 `SampleTurboModulePackage.cpp` 中完成对应方法实

现：

```
std::vector<react::ComponentDescriptorProvider>
SampleTurboModulePackage::createComponentDescriptorProviders() {
    return {
react::concreteComponentDescriptorProvider<react::MarqueeViewComponentDescriptor
>(),
    };
}

EventEmitRequestHandlers
SampleTurboModulePackage::createEventEmitRequestHandlers() {
    return {std::make_shared<MarqueeViewEventEmitRequestHandler>()};
}

ComponentNapiBinderByString
SampleTurboModulePackage::createComponentNapiBinderByName() {
    return {{"MarqueeView", std::make_shared<MarqueeViewNapiBinder>()}};
};

ComponentJSIBinderByString
SampleTurboModulePackage::createComponentJSIBinderByName() {
    return {{"MarqueeView", std::make_shared<MarqueeViewJSIBinder>()}};
};
```

3. 编写 RN 调用 Fabric 组件的代码

使用 `codegenNativeComponent` 创建 `MarqueeView` 组件，其中 `MarqueeViewProps` 里声明了 `src` 属性和 `onStop` 事件：

```
type OnStopEventData = Readonly<{
  isStop: boolean
}>;

interface MarqueeViewProps extends ViewProps {
  src: string,
  onStop?: DirectEventHandler<OnStopEventData>;
}

const MarqueeView = codegenNativeComponent<MarqueeViewProps>(
  'MarqueeView'
) as HostComponent<MarqueeViewProps>;
```

和其他标准组件的创建方式一样，在组件容器内添加 `MarqueeView` 标签：

```
<MarqueeView
  src="双十一大促，消费是社会再生产过程中的一个重要环节，也是最终环节。它是指利用社会产品来满足人们各种需要的过程。"
  style={{height: 180, width: '100%', backgroundColor: 'hsl(210, 80%, 50%)'}}
  onStop={(e) => {
    SampleTurboModule.rnLog("native调用了RN的 onStop, isStop = "+e.nativeEvent.isStop)
    setMarqueeStop(e.nativeEvent.isStop)
  }}
/>
```