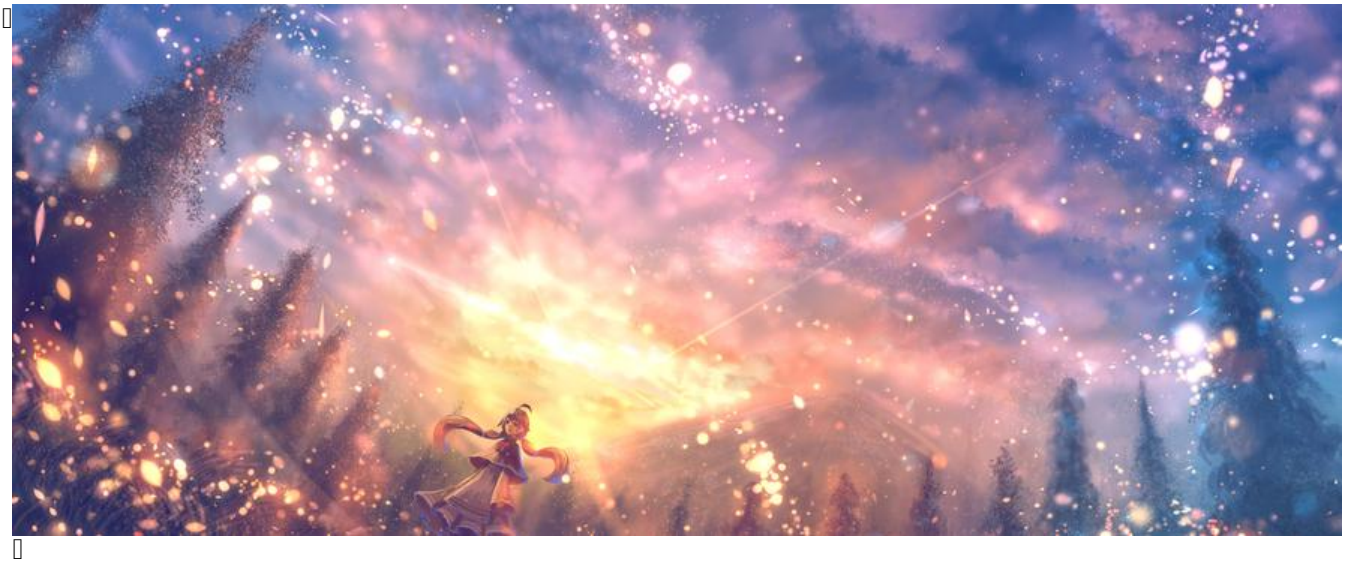
Day 13 Yew (二)： 组件 CSS 与通信

作者：KuMa
原文链接：https://ld246.com/article/1681150718734
来源网站：链滴



books

Yew 的 CSS

创建时间：2023-04-10 14:16 星期一

Yew本身不提供css 的处理我们需要一个 stylist 来进行处理

```
[package]
name = "project"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.h
ml


[dependencies]
...
stylist = { version = "0.10.0", features = ["yew"]}
...
```

然后我们就可以编写代码了

```
use stylist::{yew::styled_component, style};

#[styled_component(StyleComponent)]  // 这里使用 `style_component` 宏来创建组件
pub fn style_component() -> Html {
    let style = style!(
        r#"
            color: red;
            background-color: black;
        "#
    ).unwrap();
    html!{
        <div>
            <p class={style}>{"你好"}</p>
        </div>
    }
}
```

我们也可以定义一些临时(行内)的样式

```
use stylist::{yew::styled_component, style};

#[styled_component(StyleComponent)]  // 这里使用 `style_component` 宏来创建组件
pub fn style_component() -> Html {

    html!{
        <div>
            <p class={css!("color:red; font-size: 75px;")}>{"你好"}</p>
        </div>
    }
}
```

CSS 文件分离

我们创建一个 index.css 这是文件结构


```
use gloo::console::*;
use stylist::{yew::styled_component, style, Style};

#[styled_component(FileStyleComponent)]
pub fn file_style_component() -> Html{
    const FILE_NAME : &str = include_str!("./style/index.css");
    log! (format!("加载css -> {}", FILE_NAME));
    let style = Style::new(FILE_NAME).unwrap();
    html!{
        <div class={style}>
            <p>{"你好3"}</p>
        </div>
    }
}
```

组件通信 （父传子）

组件通信 一般使用 结构 通过 ==参数传入某个组件 ==, 这个结构一般实现了 Properties, PartialEq 这两个特性:

```
use yew::prelude::*;

const A :&str = "q23";
#[derive(Properties, PartialEq)]
pub struct TitleHeaderStruct{
    pub value: Option<String>
}

#[function_component(TitleHeader)]
pub  fn title_header(prop: &TitleHeaderStruct) -> Html{

    html!{
        <div>
            {"这里是我的代码: "}{match (&prop.value) {
                Some(Val) => Val,
```

```
                None => "什么都没有力"
            }}
        </div>
    }
}

#[function_component(Title)]
pub fn title() -> Html{
    html! {
        <TitleHeader value="python"/>
        <TitleHeader/>
    }
}
```

枚举与Class

```
use stylist::{yew::styled_component, Style};
use yew::prelude::*;

const A: &str = "q23";

const STYLE: &str = include_str!("./style/color.css");

#[derive(Properties, PartialEq)]
pub struct TitleHeaderStruct {
    pub value: Option<String>,
    pub color: Option<ColorEnum>,
}

#[derive(PartialEq)]
pub enum ColorEnum {
    Err,
    Okay,
    Nomal,
}

impl TitleHeaderStruct {
    pub fn color_to_string(&self) -> String {
        match &(self.color) {
            Some(value) => match value {
                ColorEnum::Err => String::from("err"),
                ColorEnum::Okay => String::from("okay"),
                ColorEnum::Nomal => String::from("nomal"),
            },
            None => "nomal".to_string()
        }
    }
}

#[styled_component(TitleHeader)]
pub fn title_header(prop: &TitleHeaderStruct) -> Html {
    html! {
        <div class={Style::new(STYLE).expect("错误的css")}>
```

```
            // 如果定义了一些选择器的样式的话 这些样式只对他的子元素生效
            <div class={prop.color_to_string()}>
                {"重新开始了"}
            </div>
        </div>

    }
}

#[function_component(Title)]
pub fn title() -> Html {
    html! {
        <div>
            <TitleHeader value="python" color={ColorEnum::Okay}/>
            <TitleHeader/>
        </div>
    }
}
```

组件通信 （子传父）

我们一般使用回调函数给父组件传值

```
use stylist::{yew::styled_component, Style};
use yew::prelude::*;
use gloo::console::*;

#[derive(Properties, PartialEq)]
pub struct TitleHeaderStruct {
    pub value: Option<String>,
    pub color: Option<ColorEnum>,
    pub on_load: Option<Callback<String>>
}


#[styled_component(TitleHeader)]
pub fn title_header(prop: &TitleHeaderStruct) -> Html {
    // 子

    match &(prop.on_load) {
        Some(handler) => {handler.emit("加载开始".to_string());},
        None => {}
    }

    html! {
        <div>

        </div>
    }
}

#[function_component(Title)]
pub fn title() -> Html {
```

```
    // 父

    let children = Callback::from(
        |message: String|{
            log! (message);
        }
    );

    html! {
        <div>
            <TitleHeader on_load={children}/>
            <TitleHeader/>
        </div>
    }
}
```