# HW4

## 作业内容

二、本次作业的具体内容：

1.掌握 Piccolo 中反射宏和反射标签的用法，关于反射宏和反射标签本文档第三部分会做详细说明。

2.同学们选择自己感兴趣的系统，给它管理的 Component 结构增加或修改一个属性，检查它及 Component 结构的反射宏和反射标签，确保它能够被正确的反射。(反射系统支持的原子类型可点此查看文章 B.1 节内容)

3.在 Piccolo 代码中找到 engine/source/editor/source/editor_ui.cpp，找到 createClassUI ()函数，详细阅读代码，检查通过反射系统创建 UI 面板的逻辑，确保新加的属性能被正确的显示到右侧 Components Details 面板对应的 component 中。（一般来说，基础功能不需要修改相应代码，如果想要实现更复杂的功能，同学们也可以自行对代码进行修改。）

4.基础的参数反射和控件生成完成后，想要继续挑战高难度的同学，可以尝试在对应系统中让新增或修改的属性按照自己预想的方式工作起来！比如接下来的例子中，可以让界面设置的值作为跳跃瞬间初速度控制角色跳跃高度。

例：现在的 Motor 系统中，我们通过 m_jump_height 来控制角色跳跃的高度， 其定义位置在 engine/source/runtime/resource/res_type/components/motor.h 文件的 MotorComponentRes 类中。在 engine/source/runtime/function/framework/component/motor/ motor_component.cpp 中的 MotorCompont::calculatedDesiredVerticalMoveSpeed 方法内,使用该属性计算得 到了跳跃瞬间初速度。假如我们想直接从界面设置跳跃瞬间初速度来控制角色 的跳跃表现呢?

打分部分：

1. 基础：[40 分] 掌握反射宏及反射标签用法，正确完成反射信息标注。

2. 基础：[40 分] 掌握反射访问器的用法，读懂 UI 生成代码，能够在 Detail 面板正确显示新增或修改属性。

3. 提高：[20 分] 使新增或修改属性在其系统内按照预期生效。

# 代码框架说明

## 1. 反射宏

CLASS、STRUCT 反射类型或者结构体的声明

META 属性的反射信息标签

REFLECTION_TYPE 反射访问器的声明、REFLECTION_BODY 反射访问器的友源标签

## 2. 反射标签

Fields

表明类型或结构体中除特殊标记的属性外(参见 Disable)，所有属性均需反射，

该标签需在 CLASS 或 STRUCT 中声明。如图 6 所示，MotorComponentRes 类

型标记了 Fields 标签，因此其所有属性都支持反射。

WhiteListFields

表明类型或结构体中只有标记过的属性才允许反射(参见 Enable)，该标签需在 CLASS 或 STRUCT 中声明。如图 7 所示，MeshComponent 类型标记了 WhiteListFields 标签，因此只有 m_mesh_res 才允许反射，

Disable

表明该属性无需反射，该标签需要在属性前以 META 参数的形式声明,且只有在 CLASS 或 STRUCT 中声明为 Fields 时有效。如图 8 所示，MotorComponentRes 类型标记了 Fields 标签，m_controller_config 标记了 Disable，因此它不会被反 射。

Enable

表明该属性需要反射，该标签需要在属性前以 META 参数的形式声明,且只有在 CLASS 或 STRUCT 中声明为 WhiteListFields 时有效。如图 6。

## 3. 反射功能类

| | |
|---|---|
| FieldAccessor<br>属性访问器类，提供属性名称，属性类型名称以及 set/get 方法。 | const char* getFieldName()：获取属性名称;<br>const char* getFieldTypeName()：获取属性的类型名称;<br>bool getTypeMeta(TypeMeta& filed_type)：获取属性的类型元信息(参见 TypeMeta)，filed_type 为返回的类型元信息，方法本身返回一个布尔值，只有 当属性类型也支持反射时，才会为 true;<br>void* get(void* instance)：instance 为属性所属类型的实例指针，方法返回属性的指针; |

| | |
|---|---|
| | void set(void* instance, void* value): instance 为属性所属类型的实例指针， value 为要设置的值的指针 |
| TypeMeta<br>类型元信息，提供类型的名称、属性访问器列表等。 | TypeMeta newMetaFromName(std::string type_name)：静态方法，可通过类型名称直接创建 TypeMeta；<br>std::string getTypeName()：返回类型名称；<br>int getFieldsList(FieldAccessor*& out_list)：out_list 为 FieldAccessor 数组头指针，该方法会通过 out_list 返回类型所有反射属性的访问器数组，方法本身返回数组大小；<br>int getBaseClassReflectionInstanceList(ReflectionInstance*& out_list,void* instance)：instance 为类型实例指针，out_list 为 ReflectionInstance 数组头指针，该方法会通过 out_list 返回类型所有基类的 ReflectionInstance 数组，方法本身返回数组大小。 |
| ReflectionInstance:<br>反射实例，包含了类型的元信息及实例指针。 | 属性：<br>m_meta：类型为 TypeMeta，表示类型元信息；<br>m_instance：类型为 void*，表示实例的指针； |

## 4. 反射生成面板过程

### 利用反射动态生成属性面板过程

使用反射系统，将复杂类型拆解为 UI 控件可直接表示的原子类型，生成对应的 UI 控件，形成原子类型控件树，展示复杂类型信息。

engine\source\editor\source\editor_ui.cpp

```
EditorUI::EditorUI()
{
        void EditorUI::showEditorDetailWindow(bool* p_open)
    {
        // ...
        // 给selected_object设置ui上的名字
        const std::string& name = selected_object->getName();
        static char        cname[128];
        memset(cname, 0, 128);
        memcpy(cname, name.c_str(), name.size());

        ImGui::Text("Name");
        ImGui::SameLine();
        ImGui::InputText("##Name", cname, IM_ARRAYSIZE(cname), ImGuiInputTextFla
```

```
15
16          // 为被选择物体创建component ui
17          static ImGuiTableFlags flags = ImGuiTableFlags_Resizable | ImGuiTableFla
18          auto&& selected_object_components = selected_object->getComponents();
19          for (auto component_ptr : selected_object_components)
20          {
21              // m_editor_ui_creator是一个map :unordered_map<std::string, std::func
22              // m_editor_ui_creator["TreeNodePush"] = [this](const std::string& n
23              // 下面这一行调用了["TreeNodePush"]里面存的函数 传入了component name和nu
24              // 感觉["TreeNodePush"]里面存的函数主要起到的作用是判断要不要缩进一格
25              m_editor_ui_creator["TreeNodePush"](("<" + component_ptr.getTypeName
26              // 为每个 component 对象构造反射实例
27              // 反射实例是什么 反射实例，包含了类型的元信息及实例指针。
28              // ReflectionInstance(TypeMeta meta, void* instance) : m_meta(meta),
29              // T* operator->() { return m_instance; }
30              auto object_instance = Reflection::ReflectionInstance(
31                  Piccolo::Reflection::TypeMeta::newMetaFromName(component_ptr.get
32                  component_ptr.operator->());
33              // 对反射实例进行类型拆解 下面会解释这个函数
34              createClassUI(object_instance);
35              // 这个pop又负责干嘛
36              // 应该是这一个组件的tree node结束了 往外一层
37              m_editor_ui_creator["TreeNodePop"](("<" + component_ptr.getTypeName(
38          }
39          ImGui::End();
```

```
1  EditorUI::EditorUI()
2  {
3      void EditorUI::createClassUI(Reflection::ReflectionInstance& instance)
4      {
5          Reflection::ReflectionInstance* reflection_instance;
6          // 先获取该类型 的所有基类反射实例,并递归调用该方法生成基类的 UI
7          int count =
   instance.m_meta.getBaseClassReflectionInstanceList(reflection_instance,
   instance.m_instance);
8          for (int index = 0; index < count; index++)
9          {
10             createClassUI(reflection_instance[index]);
11         }
12         // 调用 creatLeafNodeUI 方法生成该类型属性 UI。下面会解释
13         createLeafNodeUI(instance);
14
15         if (count > 0)
```

```
            delete[] reflection_instance;
        }
    }
```

```
EditorUI::EditorUI()
{
    void EditorUI::createLeafNodeUI(Reflection::ReflectionInstance& instance)
    {
        // 获取到所有属性访问器
        Reflection::FieldAccessor* fields;
        int fields_count = instance.m_meta.getFieldsList(fields);

        // 遍历所有属性访问器，使用属性类型名称查找对应的构建方法，传入属性名称和属性地址，
        for (size_t index = 0; index < fields_count; index++)
        {
            auto field = fields[index];
            // 对于容器类型，如 array，需要遍历 array，对于每个元素进 行类型判断，原子类
            if (field.isArrayType())
            {
                Reflection::ArrayAccessor array_accessor;
                if (Reflection::TypeMeta::newArrayAccessorFromName(field.getFiel
                {
                    void* field_instance = field.get(instance.m_instance);
                    int   array_count    = array_accessor.getSize(field_instance
                    m_editor_ui_creator["TreeNodePush"](
                        std::string(field.getFieldName()) + "[" + std::to_string
                    auto item_type_meta_item =
                        Reflection::TypeMeta::newMetaFromName(array_accessor.get
                    auto item_ui_creator_iterator = m_editor_ui_creator.find(ite
                    for (int index = 0; index < array_count; index++)
                    {
                        if (item_ui_creator_iterator == m_editor_ui_creator.end(
                        {
                            m_editor_ui_creator["TreeNodePush"]("[" + std::to_st
                            auto object_instance = Reflection::ReflectionInstanc
                                Piccolo::Reflection::TypeMeta::newMetaFromName(i
                                array_accessor.get(index, field_instance));
                            createClassUI(object_instance);
                            m_editor_ui_creator["TreeNodePop"]("[" + std::to_str
                        }
                        else
                        {
                            if (item_ui_creator_iterator == m_editor_ui_creator.
```

```
40                              {
41                                  continue;
42                              }
43                              m_editor_ui_creator[item_type_meta_item.getTypeName(
44                                  "[" + std::to_string(index) + "]", array_accesso
45                          }
46                      }
47                      m_editor_ui_creator["TreeNodePop"](field.getFieldName(), nul
48                  }
49              }
50              // 除了原子类型外，对于复杂类型，需要继续调用 createClassUI 方法进行拆解。
51              auto ui_creator_iterator = m_editor_ui_creator.find(field.getFieldTy
52              if (ui_creator_iterator == m_editor_ui_creator.end())
53              {
54                  Reflection::TypeMeta field_meta =
55                      Reflection::TypeMeta::newMetaFromName(field.getFieldTypeName
56                  if (field.getTypeMeta(field_meta))
57                  {
58                      auto child_instance =
59                          Reflection::ReflectionInstance(field_meta, field.get(ins
60                      m_editor_ui_creator["TreeNodePush"](field_meta.getTypeName()
61                      createClassUI(child_instance);
62                      m_editor_ui_creator["TreeNodePop"](field_meta.getTypeName(),
63                  }
64                  else
65                  {
66                      if (ui_creator_iterator == m_editor_ui_creator.end())
67                      {
68                          continue;
69                      }
70                      m_editor_ui_creator[field.getFieldTypeName()](field.getField
71                                                                    field.g
72                  }
73              }
74              // 对于原子类型，使用UI构建方法表m_editor_ui_creator
75              else
76              {
77                  m_editor_ui_creator[field.getFieldTypeName()](field.getFieldName
78                                                                field.get(i
79              }
80          }
81          delete[] fields;
82      }
83  }
```

UI 生成过程中，需要对 UI 支持的原子类型进行数据和 UI 控件的绑定，因此 需要建立原子类型的 UI 构建方法表，即 5 中用到的 m_editor_ui_creator。 构建方法如图 14 所示，m_editor_ui_creator 是一个 map >,对于每一种 UI 支持的 原子类类型，都需要注册其属性名称及处理方法。

```
EditorUI::EditorUI()
{
    const auto& asset_folder            = g_runtime_global_context.m_config_
    m_editor_ui_creator["TreeNodePush"] = [this](const std::string& name, vo
        static ImGuiTableFlags flags      = ImGuiTableFlags_Resizable | ImGu
        bool                   node_state = false;
        g_node_depth++;
        if (g_node_depth > 0)
        {
            if (g_editor_node_state_array[g_node_depth - 1].second)
            {
                node_state = ImGui::TreeNodeEx(name.c_str(), ImGuiTreeNodeFl
            }
            else
            {
                g_editor_node_state_array.emplace_back(std::pair(name.c_str(
                return;
            }
        }
        else
        {
            node_state = ImGui::TreeNodeEx(name.c_str(), ImGuiTreeNodeFlags_
        }
        g_editor_node_state_array.emplace_back(std::pair(name.c_str(), node_
    };
    m_editor_ui_creator["TreeNodePop"] = [this](const std::string& name, voi
        if (g_editor_node_state_array[g_node_depth].second)
        {
            ImGui::TreePop();
        }
        g_editor_node_state_array.pop_back();
        g_node_depth--;
    };
    // m_editor_ui_creator 是一个 map >,对于每一种 UI 支持的 原子类类型，都需要注
    m_editor_ui_creator["Transform"] = [this](const std::string& name, void*
        if (g_editor_node_state_array[g_node_depth].second)
        {
            Transform* trans_ptr = static_cast<Transform*>(value_ptr);

            Vector3 degrees_val;

            degrees_val.x = trans_ptr->m_rotation.getPitch(false).valueDegre
```

```
43                  degrees_val.y = trans_ptr->m_rotation.getRoll(false).valueDegree
44                  degrees_val.z = trans_ptr->m_rotation.getYaw(false).valueDegrees
45
46                  DrawVecControl("Position", trans_ptr->m_position);
47                  DrawVecControl("Rotation", degrees_val);
48                  DrawVecControl("Scale", trans_ptr->m_scale);
49
50                  trans_ptr->m_rotation.w = Math::cos(Math::degreesToRadians(degre
51                                            Math::cos(Math::degreesToRadians(d
52                                            Math::cos(Math::degreesToRadians(d
53                                          Math::sin(Math::degreesToRadians(degre
54                                            Math::sin(Math::degreesToRadians(d
55                                            Math::sin(Math::degreesToRadians(d
56                  trans_ptr->m_rotation.x = Math::sin(Math::degreesToRadians(degre
57                                            Math::cos(Math::degreesToRadians(d
58                                            Math::cos(Math::degreesToRadians(d
59                                          Math::cos(Math::degreesToRadians(degre
60                                            Math::sin(Math::degreesToRadians(d
61                                            Math::sin(Math::degreesToRadians(d
62                  trans_ptr->m_rotation.y = Math::cos(Math::degreesToRadians(degre
63                                            Math::sin(Math::degreesToRadians(d
64                                            Math::cos(Math::degreesToRadians(d
65                                          Math::sin(Math::degreesToRadians(degre
66                                            Math::cos(Math::degreesToRadians(d
67                                            Math::sin(Math::degreesToRadians(d
68                  trans_ptr->m_rotation.z = Math::cos(Math::degreesToRadians(degre
69                                            Math::cos(Math::degreesToRadians(d
70                                            Math::sin(Math::degreesToRadians(d
71                                          Math::sin(Math::degreesToRadians(degre
72                                            Math::sin(Math::degreesToRadians(d
73                                            Math::cos(Math::degreesToRadians(d
74                  trans_ptr->m_rotation.normalise();
75
76                  g_editor_global_context.m_scene_manager->drawSelectedEntityAxis(
77              }
78          };
79          // m_editor_ui_creator 是一个 map >,对于每一种 UI 支持的 原子类类型，都需要注
80          m_editor_ui_creator["int"] = [this](const std::string& name, void* value
81              if (g_node_depth == -1)
82              {
83                  std::string label = "##" + name;
84                  ImGui::Text("%s", name.c_str());
85                  ImGui::SameLine();
86                  ImGui::InputInt(label.c_str(), static_cast<int*>(value_ptr));
87              }
88              else
89              {
```

```
90                  if (g_editor_node_state_array[g_node_depth].second)
91                  {
92                      std::string full_label = "##" + getLeafUINodeParentLabel() +
93                      ImGui::Text("%s", (name + ":").c_str());
94                      ImGui::InputInt(full_label.c_str(), static_cast<int*>(value_
95                  }
96              }
97          };
98          m_editor_ui_creator["float"] = [this](const std::string& name, void* val
99              if (g_node_depth == -1)
100             {
101                 std::string label = "##" + name;
102                 ImGui::Text("%s", name.c_str());
103                 ImGui::SameLine();
104                 ImGui::InputFloat(label.c_str(), static_cast<float*>(value_ptr))
105             }
106             else
107             {
108                 if (g_editor_node_state_array[g_node_depth].second)
109                 {
110                     std::string full_label = "##" + getLeafUINodeParentLabel() +
111                     ImGui::Text("%s", (name + ":").c_str());
112                     ImGui::InputFloat(full_label.c_str(), static_cast<float*>(va
113                 }
114             }
115         };
116         m_editor_ui_creator["Vector3"] = [this](const std::string& name, void* v
117             Vector3* vec_ptr = static_cast<Vector3*>(value_ptr);
118             float    val[3]  = {vec_ptr->x, vec_ptr->y, vec_ptr->z};
119             if (g_node_depth == -1)
120             {
121                 std::string label = "##" + name;
122                 ImGui::Text("%s", name.c_str());
123                 ImGui::SameLine();
124                 ImGui::DragFloat3(label.c_str(), val);
125             }
126             else
127             {
128                 if (g_editor_node_state_array[g_node_depth].second)
129                 {
130                     std::string full_label = "##" + getLeafUINodeParentLabel() +
131                     ImGui::Text("%s", (name + ":").c_str());
132                     ImGui::DragFloat3(full_label.c_str(), val);
133                 }
134             }
135             vec_ptr->x = val[0];
136             vec_ptr->y = val[1];
```

```
137             vec_ptr->z = val[2];
138         };
139         m_editor_ui_creator["Quaternion"] = [this](const std::string& name, void
140             Quaternion* qua_ptr = static_cast<Quaternion*>(value_ptr);
141             float       val[4]  = {qua_ptr->x, qua_ptr->y, qua_ptr->z, qua_ptr->
142             if (g_node_depth == -1)
143             {
144                 std::string label = "##" + name;
145                 ImGui::Text("%s", name.c_str());
146                 ImGui::SameLine();
147                 ImGui::DragFloat4(label.c_str(), val);
148             }
149             else
150             {
151                 if (g_editor_node_state_array[g_node_depth].second)
152                 {
153                     std::string full_label = "##" + getLeafUINodeParentLabel() +
154                     ImGui::Text("%s", (name + ":").c_str());
155                     ImGui::DragFloat4(full_label.c_str(), val);
156                 }
157             }
158             qua_ptr->x = val[0];
159             qua_ptr->y = val[1];
160             qua_ptr->z = val[2];
161             qua_ptr->w = val[3];
162         };
163         m_editor_ui_creator["std::string"] = [this, &asset_folder](const std::st
164             if (g_node_depth == -1)
165             {
166                 std::string label = "##" + name;
167                 ImGui::Text("%s", name.c_str());
168                 ImGui::SameLine();
169                 ImGui::Text("%s", (*static_cast<std::string*>(value_ptr)).c_str(
170             }
171             else
172             {
173                 if (g_editor_node_state_array[g_node_depth].second)
174                 {
175                     std::string full_label = "##" + getLeafUINodeParentLabel() +
176                     ImGui::Text("%s", (name + ":").c_str());
177                     std::string value_str = *static_cast<std::string*>(value_ptr
178                     if (value_str.find_first_of('/') != std::string::npos)
179                     {
180                         std::filesystem::path value_path(value_str);
181                         if (value_path.is_absolute())
182                         {
183                             value_path = Path::getRelativePath(asset_folder, val
```

```
184                         }
185                         value_str = value_path.generic_string();
186                         if (value_str.size() >= 2 && value_str[0] == '.' && valu
187                         {
188                             value_str.clear();
189                         }
190                     }
191                     ImGui::Text("%s", value_str.c_str());
192                 }
193             }
194         };
195     }
```

2.同学们选择自己感兴趣的系统，给它管理的 Component 结构增加或修改一个属性，检查它及 Component 结构的反射宏和反射标签，确保它能够被正确的反射。(反射系统支持的原子类型可点此查看文章 B.1 节内容)

3.在 Piccolo 代码中找到 engine/source/editor/source/editor_ui.cpp，找到 createClassUI ()函数，详细阅读代码，检查通过反射系统创建 UI 面板的逻辑，确保新加的属性能被正确的显示到右侧 Components Details 面板对应的 component 中。（一般来说，基础功能不需要修改相应代码，如果想要实现更复杂的功能，同学们也可以自行对代码进行修改。）

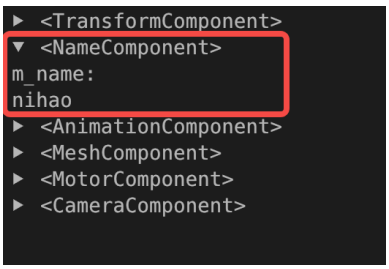
4.基础的参数反射和控件生成完成后，想要继续挑战高难度的同学，可以尝试在对应系统中让新增或修改的属性按照自己预想的方式工作起来！比如接下来的例子中，可以让界面设置的值作为跳跃瞬间初速度控制角色跳跃高度。

# 基础功能报告

## 实验结果

我给player增加了一个NameComponent

并用反射让它显示在了角色的属性面板中：



## 具体过程

### 定义NameComponent类

首先，要定义NameComponent类，才能给角色安装这个类。

所以我仿照TransformComponent类给NameComponent类编写了name_component.cpp和name_component.h文件。

定义这个类的时候需要使用到上面提到过的反射宏。

```
#pragma once

#include "runtime/function/framework/component/component.h"
#include "runtime/function/framework/object/object.h"

namespace Piccolo
{
    REFLECTION_TYPE(NameComponent)
    CLASS(NameComponent : public Component, WhiteListFields)
    {
        REFLECTION_BODY(NameComponent)

    public:
        NameComponent() = default;

        void postLoadResource(std::weak_ptr<GObject> parent_object) override;

        std::string    getName() const { return m_name; }
        void tick(float delta_time) override;

    protected:
        META(Enable)
        std::string m_name;
    };
} // namespace Piccolo

```

### 给player增加NameComponent组件

接下来的目标是：给player增加NameComponent组件。

我本来以为要增加新的组件需要在player类中增加新的成员。后来发现不是，我尝试过给character.h中的Character类添加m_name属性，但是我发现给它添加这个属性并不会起到什么效果。

然后我又查找了一下，发现这个编辑器的场景是从engine\asset\objects里面定义的各种json文件load出来的，和character相关的组件是从定义在engine\asset\objects\character\player\player.object.json里面的。

所以我修改了player.object.json，增加了如下内容：

```
{
    "components": [
        ...
        {
            "$context": {"name":"nihao"},
            "$typeName": "NameComponent"
        },
        ...
    ]
}
```

注意的是增加了这些内容之后，需要删掉原来build文件夹的内容，重新CMake一遍，否则会报错。

增加了之后就可以在属性界面上看到想要的结果了。

## 场景是如何被从json文件中load出来的 TODO：没看懂

engine\source\runtime\function\framework\object\object.cpp

```
    bool GObject::load(const ObjectInstanceRes& object_instance_res)
    {
        // clear old components
        m_components.clear();

        setName(object_instance_res.m_name);

        // load object instanced components
        m_components = object_instance_res.m_instanced_components;
        for (auto component : m_components)
        {
            if (component)
            {
                component->postLoadResource(weak_from_this());
            }
        }

        // load object definition components
        m_definition_url = object_instance_res.m_definition;

```

```
        ObjectDefinitionRes definition_res;

        const bool is_loaded_success = g_runtime_global_context.m_asset_manager-
        if (!is_loaded_success)
            return false;

        for (auto loaded_component : definition_res.m_components)
        {
            const std::string type_name = loaded_component.getTypeName();
            // don't create component if it has been instanced
            if (hasComponent(type_name))
                continue;

            loaded_component->postLoadResource(weak_from_this());

            m_components.push_back(loaded_component);
        }

        return true;
    }
```

# 附加功能报告

附加题的要求是：让新增或修改的属性按照自己预想的方式工作起来，可以让界面设置的值作为跳跃瞬间初速度控制角色跳跃高度。

## 具体过程

### 界面设置的值叫什么

在engine\source\runtime\resource\res_type\components\motor.cpp中有MotorComponentRes的定义。

在engine\source\runtime\function\framework\component\motor\motor_component.h中有MotorComponent的定义。

MotorComponent包含了MotorComponentRes，MotorComponentRes里面有：

```
        float m_move_speed { 0.f};
        float m_jump_height {0.f};
        float m_max_move_speed_ratio { 0.f};
        float m_max_sprint_speed_ratio { 0.f};
        float m_move_acceleration {0.f};
        float m_sprint_acceleration { 0.f};
```

在MotorComponent中有各种函数的定义

```
MotorComponent::calculatedDesiredHorizontalMoveSpeed
```

## 界面设置的值如何反应回原始定义的数据

一定是要反映回MotorComponent里面的

```
MotorComponentRes m_motor_res;
```

但是具体怎么反应呢?

应该是editor ui改变了之后会回调?

我看一下editor ui是被创建成了什么样好了

在engine\source\runtime\resource\res_type\components\motor.h里面给MotorComponentRes定义了一个ControllerConfig:

```
CLASS(MotorComponentRes, Fields)
{
  ...
public:
    Reflection::ReflectionPtr<ControllerConfig> m_controller_config;
}
```

这个ControllerConfig是干嘛的呢 好像没啥用

```
        {
            "$context": {
                "motor_res": {
                    "controller_config": {
                        "$context": {
                            "capsule_shape": {
                                "half_height": 0.69999998807907104,
                                "radius": 0.30000001192092896
                            }
                        },
                        "$typeName": "PhysicsControllerConfig"
                    },
                    "jump_height": 1,
```

```
                    "max_move_speed_ratio": 1,
                    "max_sprint_speed_ratio": 2,
                    "move_acceleration": 2,
                    "move_speed": 2,
                    "sprint_acceleration": 2
                }
            },
            "$typeName": "MotorComponent"
        },
```

总之ui控制应该是用类似于这个实现的：

```
ImGui::DragFloat("##Y", &values.y, 0.1f, 0.0f, 0.0f, "%.2f");
ImGui::InputFloat(label.c_str(), static_cast<float*>(value_ptr));
```

## 如何控制角色跳跃速度

在MotorComponentRes里面增加一个属性：`m_jump_speed`

然后在`MotorComponent::calculatedDesiredVerticalMoveSpeed`里面将：

```
m_vertical_move_speed = Math::sqrt(m_motor_res.m_jump_height * 2 * gravity);
```

改成：

```
m_vertical_move_speed = m_motor_res.m_jump_speed;
```

即可