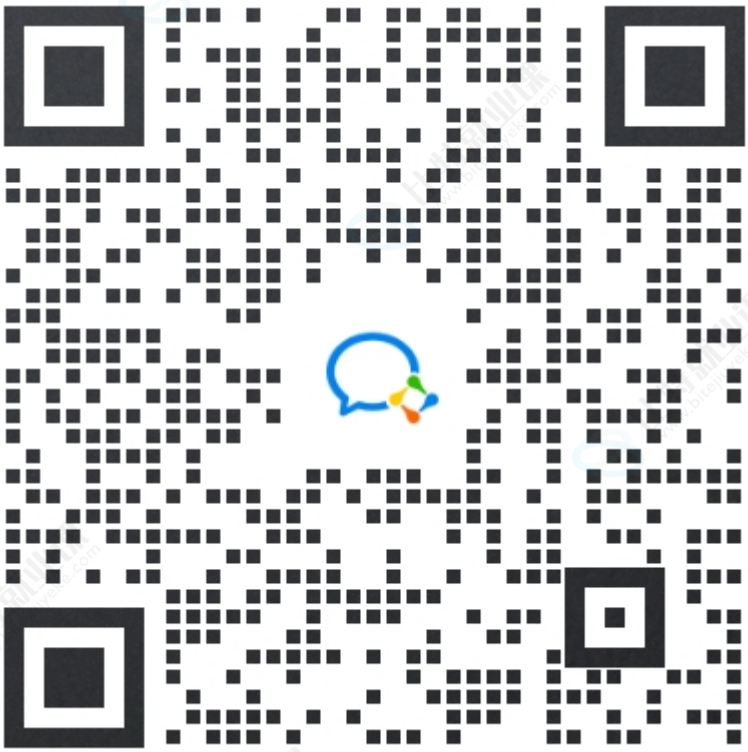
6. 数据管理

版权说明

对比特项目感兴趣，可以联系这个微信。



代码 & 板书链接

> https://gitee.com/bitedu-cpp-team/cpp-microservice-videoplayer/tree/master/client

1. DataCenter类的引入

在前面的实现中，程序中的数据、以及界面操作等全部搅合在一起，不利于代码的维护，为了降低耦合度，引入DataCenter类来专门管理程序中的各种数据，比如：分类和标签、视频信息、用户信息等。

代码块

```
////////////////////////////// DataCenter.h
////////////////////////////////////
#include <QObject>

namespace model {
class DataCenter : public QObject
{
    Q_OBJECT
public:
    // 获取DataCenter对象实例
    static DataCenter* getInstance();

private:
    explicit DataCenter(QObject *parent = nullptr);
    static DataCenter* instance;
};
}

////////////////////////////// DataCenter.cpp
////////////////////////////////////
#include "datacenter.h"

namespace model {
DataCenter* DataCenter::instance = nullptr;

DataCenter *DataCenter::getInstance()
{
    if (instance == nullptr) {
        instance = new DataCenter();
    }
    return instance;
}

DataCenter::DataCenter(QObject *parent)
    : QObject{parent}
{}
}
```

不同结构数据管理时需要不同数据结构，比如用户信息、视频信息等都需要单独的数据结构来组织，引入data.h头文件和data.cpp源文件，专门完成各种数据结构的声明和定义。

2. 分离首页分类和标签

分类和标签数据不仅在首页中使用到，在视频上时也需要选择视频所对应的分类和标签，即分类和标签数据在视频上传页面也需要用到，如果不降分类和标签数据分离出来，该部分数据就需要维护两份。若将来新增分类或标签，数据维护起来波方便。

2.1 定义标签和分类类

一个分类对应一组标签，将来需要能很方便获取到所有的分类数据，某个分类下的标签数据，为了方便使用，引入KindAndTag类专门管理分类和标签。

代码块

```
////////////////////////////// data.h ////////////////////////////////////

class KindAndTag{
public:
    KindAndTag();

    // 获取所有分类
    const QList<QString> getAllKinds()const;

    // 获取一个分类下所有标签及id
    const QHash<QString, int> getTagsByKind(QString kind)const;

    // 获取分类的id
    int getKindId(QString kind)const;

    // 获取kind分类下包含的tag的id
    int getTagId(QString kind, QString tag)const;

private:
    // key为分类名称，value为分类Id
    QHash<QString, int> kindIds;

    // 外层QHash的key为分类名称
    // 内层QHash的key为标签名称，value为标签Id
    QHash<QString, QHash<QString, int>> tagIds;
    static int id;
};

////////////////////////////// data.cpp
//////////////////////////////////////
#include "data.h"
```

```cpp

namespace model {
////////////////////////////////////////////////////////////////////////////
/// 标签和分类
////////////////////////////////////////////////////////////////////////////
int KindAndTag::id = 10000;
KindAndTag::KindAndTag()
{
    // 构建分类及其id
    QList<QString> kinds = {"历史", "美食", "游戏", "科技", "运动", "动物", "旅行", "电影"};
    for(auto& kind : kinds){
        kindIds.insert(kind,id++);
    }

    // 构建标签及其id
    QHash<QString, QList<QString>> kindsAndTags = {
  {"历史", {"中国史", "世界史", "历史人物", "艺术", "文化", "奇闻"}},
  {"美食", {"美食测评", "美食制作", "美食攻略", "美食记录", "探店", "水果", "海鲜"}},
  {"游戏", {"游戏攻略", "单机游戏", "电子竞技", "手机游戏", "网络游戏", "游戏赛事", "桌游棋牌"}},
  {"科技", {"数码", "软件应用", "智能家居", "手机", "电脑", "人工智能", "基础设施"}},
  {"运动", {"篮球", "足球", "乒乓球", "羽毛球", "健身", "竞技体育", "运动装备"}},
  {"动物", {"喵星人", "汪星人", "宠物知识", "动物资讯", "野生动物", "动物世界", "萌宠"}},
  {"旅行", {"旅游攻略", "旅行Vlog", "自驾游", "交通", "环球旅行", "露营", "野外生存"}},
  {"电影", {"电影解说", "电影推荐", "电影剪辑", "搞笑", "吐槽", "悬疑", "经典"}}};

    // 构建分类下：各个标签及其对应id
    for(auto& kind : kinds){
        // 构建kind下所有标签及其id
        QList<QString>& tags = kindsAndTags[kind];
        QHash<QString, int> tagIdsOfKind;
        for(auto& tag : tags){
            tagIdsOfKind.insert(tag, id++);
        }

        tagIds.insert(kind, tagIdsOfKind);
    }
}

const QList<QString> KindAndTag::getAllKinds()const{
    return kindIds.keys();
}

const QHash<QString, int> KindAndTag::getTagsByKind(QString kind)const{
```

```
    return tagIds[kind];
}

int KindAndTag::getKindId(QString kind)const{
    return kindIds[kind];
}

int KindAndTag::getTagId(QString kind, QString tag)const{
    return tagIds[kind][tag];
}
}
```

代码块

```
//////////////////////////// data.h ////////////////////////////////////
#include "data.h"
class DataCenter : public QObject
{
    Q_OBJECT
public:
    // 获取DataCenter对象实例
    static DataCenter* getInstance();

    // 获取所有分类
    const KindAndTag* getKindAndTagsClassPtr();
    // ...

private:
    // 分类和标签实例指针
    KindAndTag* kindsAndTags = nullptr;
};
//////////////////////////// data.cpp ////////////////////////////////////
// ...
const KindAndTag* DataCenter::getKindAndTagsClassPtr(){
    if(nullptr == kindsAndTags){
        // 实例化kindsAndTags
        kindsAndTags = new KindAndTag();
    }
    return kindsAndTags;
}
```

2.2 分离首页标签和分类数据

分类和标签数据分离到数据中心管理后，HomePageWidget类中的分类和便签数据就可以删除了，分类和标签数据从DataCenter中进行获取，修改位置：

1. HomePageWidget类中删除tags成员变量
2. initKindsAndTags()方法中获取分类和标签数据时从DataCenter中获取

代码块

```
//////////////////////////////// homepagewidget.cpp
//////////////////////////////
// ...
void HomePageWidget::initKindsAndTags()
{
    // 创建分类按钮
    QPushButton *kindBtn = buildSelectBtn(ui->classifys, "#3ECEFF", "分类");
    ui->classifyHLayout->addWidget(kindBtn);

    // 到数据中心中获取所有分类数据
    auto dataCenter = model::DataCenter::getInstance();
    auto kindAndTagPtr = dataCenter->getKindAndTagsClassPtr();
    auto kinds = kindAndTagPtr->getAllKinds();
    // ...
    // 将按钮添加到分类的布局器中
    ui->classifyHLayout->setSpacing(8);

    // 获取分类下标签，默认显示第0个分类
    auto tags = kindAndTagPtr->getTagsByKind(kinds[0]).keys();
    resetTags(tags);
}
//...
void HomePageWidget::onKindBtnClicked(QPushButton *clickedKindBtn)
{
    // ...
    // 根据当前选中分类，重新添加标签
    auto dataCenter = model::DataCenter::getInstance();
    auto kindAndTagPtr = dataCenter->getKindAndTagsClassPtr();
    resetTags(kindAndTagPtr->getTagsByKind(clickedKindBtn->text()).keys());
}
```

2.3 更新上传视频页的标签和分类显示

上传视频页面上需要用到的视频和标签数据，页直接从DataCenter中获取。

分类在构造函数中直接添加到QComoBox中，当分类改变时，在标签位置添加该分类下的标签。

代码块

```
////////////////////////// uploadvideopage.h ////////////////////////////////
```

```
class UploadVideoPage : public QWidget
{
    // ...
    // 更改视频封面图按钮槽函数
    void onChangeBtnClicked();
    // QComoBox中分类选择改变槽函数
    void onUpdateTags(const QString &kind);
    // ...
};
////////////////////////// uploadvideopage.cpp ////////////////////////////////
#include "model/datacenter.h"
#include "util.h"

UploadVideoPage::UploadVideoPage(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::UploadVideoPage)
{
    ui->setupUi(this);
    // 获取所有分类，并更新到界面
    auto dataCenter = model::DataCenter::getInstance();
    auto kindAndTag = dataCenter->getKindAndTagsClassPtr();
    ui->kinds->addItems(kindAndTag->getAllKinds());
    ui->kinds->setCurrentIndex(-1);  // 默认不选中
    // ...
    // 分类选择改变
    connect(ui->kinds, &QComboBox::currentTextChanged, this,
    &UploadVideoPage::onUpdateTags);
}
// ...
void UploadVideoPage::onUpdateTags(const QString &kind)
{
    LOG()<<"分类更新: "<<kind;
}

```

当用户选择对应的分类时，该分类下的标签应该以按钮的形式显示在界面上。因此需要添加addTagsByKind(...)函数，实现标签按钮的动态创建。

代码块

```
////////////////////////// uploadvideopage.h ////////////////////////////////
// ...
class UploadVideoPage : public QWidget
{
    //...
    // QComoBox中分类选择改变槽函数
```

```
7       void onUpdateTags(const QString &kind);
8
9       // 将kind下标签以按钮形式展示在界面上
10      void addTagsByKind(const QString& kind);
11      // ...
12  };
13  ////////////////////////// uploadvideopage.cpp ////////////////////////////////
14  // ...
15  void UploadVideoPage::onUpdateTags(const QString &kind)
16  {
17      addTagsByKind(kind);
18  }
19
20  void UploadVideoPage::addTagsByKind(const QString &kind)
21  {
22      // 1.添加之前先清空之前的标签
23      QList<QPushButton*> tagBtnList = ui->tagWidget->findChildren<QPushButton*>
    ();
24      for(auto tagBtn : tagBtnList){
25          // 删除标签按钮，提交按钮不能删除--
26          ui->tagLayout->removeWidget(tagBtn);
27          delete tagBtn;
28      }
29
30      // 将添加的弹簧删除掉
31      // 弹簧本来是最后一个控件，当layoyut中的按钮删除之后，弹簧就变成第0个控件了
32      QLayoutItem* spacerItem = ui->tagLayout->itemAt(ui->tagLayout->count()-1);
33      ui->tagLayout->removeItem(spacerItem);
34
35      // 2. 根据kind获取标签--注意：如果key不存在直接返回
36      if(kind.isEmpty()){
37          return;
38      }
39
40      auto kindAndTagPtr = model::DataCenter::getInstance()-
    >getKindAndTagsClassPtr();
41      auto kinds = model::DataCenter::getInstance()->getKindAndTagsClassPtr()-
    >getAllKinds();
42      auto tags = kindAndTagPtr->getTagsByKind(kind);
43
44      // 3. 创建该标签对应的按钮
45      for(auto& tag : tags.keys()){
46          QPushButton* tagBtn = new QPushButton(ui->tagContent);
47          tagBtn->setFixedSize(98, 49);
48          tagBtn->setText(tag);
49          tagBtn->setCheckable(true);    // 设置按钮的状态为选中和未选中两种状态
50          // QPushButton:checked    当按钮选中时
```

```
51          // QPushButton:unchecked 当按钮未选中时
52          tagBtn->setStyleSheet("QPushButton{"
53                                "border : 1px solid #3ECEFE;"
54                                "border-radius : 4px;"
55                                "color : #3ECEFE;"
56                                "font-family : 微软雅黑;"
57                                "font-size : 16px;"
58                                "background-color : #FFFFFF;}"
59                                "QPushButton:checked{"
60                                "background-color : #3ECEFE;"
61                                "color : #FFFFFF;}"
62                                "QPushButton:unchecked{"
63                                "background-color : #FFFFFF;"
64                                "color : #3ECEFE;}");
65
66          ui->tagLayout->addWidget(tagBtn);
67      }
68
69      // 在tagLayout最后放一个空白间距，将按钮挤到左侧
70    ui->tagLayout->insertSpacing(tags.size(), ui->tagContent->width() -
    (98+20)*tags.size());
71      ui->tagLayout->setSpacing(20);
72  }
```

程序中还有其他数据需要DataCenter管理，等后续获取到这些数据时再进行处理。