异常检测代码：

# 设置日志记录

logging.basicConfig(level=logging.WARNING, format='%(asctime)s - %(levelname)s - %(message)s')

# 定义模型架构

class HybridModel(nn.Module):

"""混合模型（LSTM + Transformer，适用于特征数据的版本）"""

def \_\_init\_\_(self, input\_size=5): # 修改 input\_size 为 5

super(HybridModel, self).\_\_init\_\_()

self.lstm = nn.LSTM(input\_size=input\_size, hidden\_size=128, num\_layers=1, batch\_first=True)

encoder\_layer = nn.TransformerEncoderLayer(d\_model=128, nhead=8, dropout=0.4)

self.transformer\_encoder = nn.TransformerEncoder(encoder\_layer, num\_layers=2)

self.fc1 = nn.Linear(128, 64)

self.dropout = nn.Dropout(0.4)

self.fc2 = nn.Linear(64, 2)

def forward(self, x):

# 将输入扩展为 [batch\_size, seq\_len, input\_size]

x = x.unsqueeze(1) # seq\_len=1

lstm\_out, \_ = self.lstm(x)

lstm\_out = lstm\_out.permute(1, 0, 2) # [seq\_len, batch\_size, hidden\_size]

x = self.transformer\_encoder(lstm\_out)

x = x.mean(dim=0) # [batch\_size, hidden\_size]

x = self.dropout(x)

x = torch.relu(self.fc1(x))

out = self.fc2(x)

return out

class RealTimePlot(QWidget):

def \_\_init\_\_(self, parent=None):

super(RealTimePlot, self).\_\_init\_\_(parent)

# 创建一个图表部件

self.plot\_widget = pg.PlotWidget()

layout = QVBoxLayout()

layout.addWidget(self.plot\_widget)

self.setLayout(layout)

# 设置曲线

self.curve1 = self.plot\_widget.plot(pen=pg.mkPen(color='#3498db', width=2), name='传感器1')

self.curve2 = self.plot\_widget.plot(pen=pg.mkPen(color='#e74c3c', width=2), name='传感器2')

# 数据列表

self.xdata = []

self.ydata1 = []

self.ydata2 = []

# 设置图表属性

self.plot\_widget.setTitle('实时监控数据', color='#2c3e50', size='12pt')

self.plot\_widget.setLabel('left', '传感器值')

self.plot\_widget.setLabel('bottom', '时间')

self.plot\_widget.addLegend()

self.plot\_widget.showGrid(x=True, y=True)

@pyqtSlot(str, float, float)

def update\_plot(self, new\_time, new\_val1, new\_val2):

self.xdata.append(new\_time)

self.ydata1.append(new\_val1)

self.ydata2.append(new\_val2)

self.xdata = self.xdata[-100:]

self.ydata1 = self.ydata1[-100:]

self.ydata2 = self.ydata2[-100:]

x\_indexes = list(range(len(self.xdata)))

self.curve1.setData(x\_indexes, self.ydata1)

self.curve2.setData(x\_indexes, self.ydata2)

self.plot\_widget.setXRange(max(0, len(self.xdata) - 100), len(self.xdata))

class WorkerSignals(QObject):

log = pyqtSignal(str)

update\_anomaly = pyqtSignal(int, list)

update\_plot = pyqtSignal(str, float, float)

class DetectionWorker(QThread):

def \_\_init\_\_(self, model, data):

super().\_\_init\_\_()

self.model = model

self.signals = WorkerSignals()

self.running = True

self.data = data

self.index = 0 # 用于遍历数据集

def run(self):

try:

features = self.data['features']

labels = self.data['labels']

num\_samples = features.shape[0]

while self.running:

feature = features[self.index]

input\_tensor = torch.tensor(feature, dtype=torch.float32).unsqueeze(0) # [batch\_size, input\_size]

with torch.no\_grad():

output = self.model(input\_tensor)

probabilities = torch.softmax(output, dim=1).numpy()[0]

prediction = np.argmax(probabilities)

self.signals.update\_anomaly.emit(prediction, probabilities.tolist())

sensor\_val1 = feature[0]

sensor\_val2 = feature[1] if feature.shape[0] > 1 else 0 # 防止索引越界

current\_time = datetime.now().strftime("%H:%M:%S")

self.signals.update\_plot.emit(current\_time, sensor\_val1, sensor\_val2)

if prediction == 0:

log\_msg = f"检测完成: 未发现异常"

elif prediction == 1:

log\_msg = f"检测完成: 发现异常 (传感器 ID: {random.randint(1000,9999)}) - 已发送警报"

self.signals.log.emit(log\_msg)

self.index += 1

if self.index >= num\_samples:

self.index = 0 # 如果需要循环，可以重置索引

self.msleep(100) # 线程休眠100毫秒

self.signals.log.emit("检测任务已停止。")

except Exception as e:

error\_msg = f"检测过程中发生错误: {str(e)}"

self.signals.log.emit(error\_msg)

def stop(self):

self.running = False

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("智能汽车异常检测系统")

self.setGeometry(100, 100, 1600, 1000) # 将窗口尺寸调整为更大

self.setStyleSheet("background-color: #ecf0f1;")

# 设置全局字体

font = QFont("SimHei", 12)

self.setFont(font)

self.model = None

self.detection\_worker = None

self.detection\_running = False

self.data = None # 用于存储加载的数据

# 模型和数据文件路径（请根据实际路径修改）

self.default\_model\_path = r"path\_to\_your\_model\Hybrid\_best.pth"

self.default\_data\_path = r"path\_to\_your\_data\features\_labels.npz"

# 初始化计数器

self.total\_predictions = 0

self.normal\_count = 0

self.abnormal\_count = 0

self.initUI()

def initUI(self):

main\_widget = QWidget()

main\_layout = QVBoxLayout()

main\_widget.setLayout(main\_layout)

self.setCentralWidget(main\_widget)

header = self.create\_header()

main\_layout.addWidget(header)

content\_layout = QHBoxLayout()

main\_layout.addLayout(content\_layout)

sidebar = self.create\_sidebar()

content\_layout.addWidget(sidebar)

main\_content = self.create\_main\_content()

content\_layout.addWidget(main\_content)

status\_bar = self.create\_status\_bar()

main\_layout.addWidget(status\_bar)

self.timer = QTimer()

self.timer.timeout.connect(self.update\_system\_status)

self.timer.start(1000)

def create\_header(self):

header = QWidget()

header.setFixedHeight(60)

header.setStyleSheet("""

background: qlineargradient(x1:0%, y1:0%, x2:100%, y2:0%,

stop:0% #2c3e50, stop:100% #34495e);

""")

layout = QHBoxLayout()

header.setLayout(layout)

title = QLabel("智能汽车异常检测系统")

title.setFont(QFont("SimHei", 24))

title.setStyleSheet("color: #ffffff;")

layout.addWidget(title, alignment=Qt.AlignLeft)

return header

def create\_sidebar(self):

sidebar = QWidget()

sidebar.setFixedWidth(200)

sidebar.setStyleSheet("background-color: #34495e;")

layout = QVBoxLayout()

sidebar.setLayout(layout)

control\_label = QLabel("控制面板")

control\_label.setFont(QFont("SimHei", 18))

control\_label.setStyleSheet("color: #ecf0f1;")

control\_label.setAlignment(Qt.AlignCenter)

layout.addWidget(control\_label)

button\_style = """

QPushButton {

background-color: #2c3e50;

color: #ecf0f1;

border: none;

border-radius: 5px;

padding: 10px;

font-size: 14px;

}

QPushButton:hover {

background-color: #34495e;

}

"""

self.load\_model\_button = QPushButton("加载模型")

self.load\_model\_button.setStyleSheet(button\_style)

self.load\_model\_button.clicked.connect(self.load\_model)

layout.addWidget(self.load\_model\_button)

self.start\_button = QPushButton("启动检测")

self.start\_button.setStyleSheet(button\_style)

self.start\_button.clicked.connect(self.start\_detection)

layout.addWidget(self.start\_button)

self.stop\_button = QPushButton("停止检测")

self.stop\_button.setStyleSheet(button\_style)

self.stop\_button.clicked.connect(self.stop\_detection)

self.stop\_button.setEnabled(False)

layout.addWidget(self.stop\_button)

self.report\_button = QPushButton("生成报告")

self.report\_button.setStyleSheet(button\_style)

self.report\_button.clicked.connect(self.generate\_report)

self.report\_button.setEnabled(False)

layout.addWidget(self.report\_button)

self.settings\_button = QPushButton("系统设置")

self.settings\_button.setStyleSheet(button\_style)

self.settings\_button.clicked.connect(self.open\_settings)

layout.addWidget(self.settings\_button)

layout.addStretch()

return sidebar

def create\_main\_content(self):

main\_content = QWidget()

main\_layout = QVBoxLayout()

main\_content.setLayout(main\_layout)

self.plot = RealTimePlot(self)

main\_layout.addWidget(self.plot)

results\_status\_layout = QHBoxLayout()

main\_layout.addLayout(results\_status\_layout)

anomaly\_widget = self.create\_anomaly\_results()

results\_status\_layout.addWidget(anomaly\_widget)

system\_status\_widget = self.create\_system\_status()

results\_status\_layout.addWidget(system\_status\_widget)

logs\_widget = self.create\_logs()

main\_layout.addWidget(logs\_widget)

return main\_content

def create\_anomaly\_results(self):

widget = QWidget()

widget.setStyleSheet("background-color: #ffffff; border-radius: 10px;")

widget.setFixedHeight(250) # 增加高度

layout = QVBoxLayout()

widget.setLayout(layout)

title = QLabel("异常检测结果")

title.setFont(QFont("SimHei", 16))

title.setStyleSheet("color: #2c3e50;")

layout.addWidget(title)

normal\_label = QLabel("正常比例")

normal\_label.setFont(QFont("SimHei", 12))

layout.addWidget(normal\_label)

self.normal\_progress = QProgressBar()

self.normal\_progress.setValue(0)

self.normal\_progress.setStyleSheet("""

QProgressBar {

border: 1px solid #2c3e50;

border-radius: 5px;

text-align: center;

height: 30px; # 调整高度

}

QProgressBar::chunk {

background-color: #3498db;

width: 20px;

}

""")

layout.addWidget(self.normal\_progress)

abnormal\_label = QLabel("异常比例")

abnormal\_label.setFont(QFont("SimHei", 12))

layout.addWidget(abnormal\_label)

self.abnormal\_progress = QProgressBar()

self.abnormal\_progress.setValue(0)

self.abnormal\_progress.setStyleSheet("""

QProgressBar {

border: 1px solid #2c3e50;

border-radius: 5px;

text-align: center;

height: 30px; # 调整高度

}

QProgressBar::chunk {

background-color: #e74c3c;

width: 20px;

}

""")

layout.addWidget(self.abnormal\_progress)

return widget

def create\_system\_status(self):

widget = QWidget()

widget.setStyleSheet("background-color: #ffffff; border-radius: 10px;")

widget.setFixedHeight(250) # 增加高度

layout = QVBoxLayout()

widget.setLayout(layout)

title = QLabel("系统状态")

title.setFont(QFont("SimHei", 16))

title.setStyleSheet("color: #2c3e50;")

layout.addWidget(title)

online\_layout = QHBoxLayout()

self.status\_indicator = QLabel()

self.status\_indicator.setFixedSize(20, 20)

self.status\_indicator.setStyleSheet("background-color: #2ecc71; border-radius: 10px;")

online\_layout.addWidget(self.status\_indicator)

self.status\_text = QLabel("系统正常运行中")

self.status\_text.setFont(QFont("SimHei", 12))

online\_layout.addWidget(self.status\_text)

layout.addLayout(online\_layout)

self.cpu\_label = QLabel("CPU 使用率: 0%")

self.cpu\_label.setFont(QFont("SimHei", 12))

layout.addWidget(self.cpu\_label)

self.cpu\_progress = QProgressBar()

self.cpu\_progress.setValue(0)

self.cpu\_progress.setStyleSheet("""

QProgressBar {

border: 1px solid #2c3e50;

border-radius: 5px;

text-align: center;

height: 30px; # 调整高度

}

QProgressBar::chunk {

background-color: #3498db;

width: 20px;

}

""")

layout.addWidget(self.cpu\_progress)

self.mem\_label = QLabel("内存使用率: 0%")

self.mem\_label.setFont(QFont("SimHei", 12))

layout.addWidget(self.mem\_label)

self.mem\_progress = QProgressBar()

self.mem\_progress.setValue(0)

self.mem\_progress.setStyleSheet("""

QProgressBar {

border: 1px solid #2c3e50;

border-radius: 5px;

text-align: center;

height: 30px; # 调整高度

}

QProgressBar::chunk {

background-color: #3498db;

width: 20px;

}

""")

layout.addWidget(self.mem\_progress)

return widget

def create\_logs(self):

widget = QWidget()

widget.setStyleSheet("background-color: #ffffff; border-radius: 10px;")

# 增加日志窗口高度

widget.setFixedHeight(300)

layout = QVBoxLayout()

widget.setLayout(layout)

title = QLabel("最近检测日志")

title.setFont(QFont("SimHei", 16))

title.setStyleSheet("color: #2c3e50;")

layout.addWidget(title)

self.logs\_text = QTextEdit()

self.logs\_text.setReadOnly(True)

self.logs\_text.setStyleSheet("""

QTextEdit {

background-color: #ecf0f1;

border: none;

font-family: 'SimHei', sans-serif;

font-size: 12px;

}

""")

layout.addWidget(self.logs\_text)

return widget

def create\_status\_bar(self):

status\_bar = QWidget()

status\_bar.setFixedHeight(40)

status\_bar.setStyleSheet("background-color: #34495e;")

layout = QHBoxLayout()

layout.setContentsMargins(20, 0, 20, 0)

status\_bar.setLayout(layout)

self.time\_label = QLabel("当前时间: 2024-10-15 12:00:00")

self.time\_label.setFont(QFont("SimHei", 12))

self.time\_label.setStyleSheet("color: #ecf0f1;")

layout.addWidget(self.time\_label, alignment=Qt.AlignLeft)

online\_status = QLabel()

online\_status.setFixedSize(20, 20)

online\_status.setStyleSheet("background-color: #2ecc71; border-radius: 10px;")

layout.addWidget(online\_status, alignment=Qt.AlignRight)

online\_text = QLabel("在线")

online\_text.setFont(QFont("SimHei", 12))

online\_text.setStyleSheet("color: #ecf0f1;")

layout.addWidget(online\_text, alignment=Qt.AlignRight)

self.status\_timer = QTimer()

self.status\_timer.timeout.connect(self.update\_time)

self.status\_timer.start(1000)

return status\_bar

def update\_time(self):

current\_time\_str = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

current\_time = f"当前时间: {current\_time\_str}"

self.time\_label.setText(current\_time)

def update\_system\_status(self):

cpu = psutil.cpu\_percent()

mem = psutil.virtual\_memory().percent

self.cpu\_progress.setValue(int(cpu))

self.mem\_progress.setValue(int(mem))

self.cpu\_label.setText(f"CPU 使用率: {cpu}%")

self.mem\_label.setText(f"内存使用率: {mem}%")

def load\_model(self):

# 加载模型

model\_file, \_ = QFileDialog.getOpenFileName(self, "加载模型", "", "PTH Files (\*.pth)")

if not model\_file:

model\_file = self.default\_model\_path

try:

input\_size = 5 # 确保 input\_size 与模型匹配

self.model = HybridModel(input\_size)

self.model.load\_state\_dict(torch.load(model\_file, map\_location=torch.device('cpu')))

self.model.eval()

self.log(f"模型 {os.path.basename(model\_file)} 加载成功。")

# 加载数据

data\_file, \_ = QFileDialog.getOpenFileName(self, "加载数据", "", "NPZ Files (\*.npz)")

if not data\_file:

data\_file = self.default\_data\_path

self.data = np.load(data\_file)

self.log(f"数据 {os.path.basename(data\_file)} 加载成功。")

# 检查特征维度

features = self.data['features']

if features.shape[1] != input\_size:

QMessageBox.critical(self, "错误", f"特征维度 ({features.shape[1]}) 与模型输入大小 ({input\_size}) 不匹配。")

self.model = None

self.data = None

return

except Exception as e:

QMessageBox.critical(self, "错误", f"加载模型或数据失败: {str(e)}")

self.model = None

self.data = None

return

def start\_detection(self):

if not self.model or self.data is None:

QMessageBox.warning(self, "警告", "请先加载模型和数据。")

return

if self.detection\_running:

QMessageBox.warning(self, "警告", "检测已经在运行中。")

return

# 重置计数器

self.total\_predictions = 0

self.normal\_count = 0

self.abnormal\_count = 0

self.normal\_progress.setValue(0)

self.abnormal\_progress.setValue(0)

self.detection\_running = True

self.start\_button.setEnabled(False)

self.stop\_button.setEnabled(True)

self.report\_button.setEnabled(False)

self.log("检测开始。")

self.detection\_worker = DetectionWorker(self.model, self.data)

self.detection\_worker.signals.log.connect(self.log)

self.detection\_worker.signals.update\_anomaly.connect(self.update\_anomaly\_results)

self.detection\_worker.signals.update\_plot.connect(self.update\_plot)

self.detection\_worker.start()

def stop\_detection(self):

if self.detection\_worker:

self.detection\_worker.stop()

self.detection\_worker.wait()

self.detection\_worker = None

self.detection\_running = False

self.start\_button.setEnabled(True)

self.stop\_button.setEnabled(False)

self.report\_button.setEnabled(True)

self.log("检测已停止。")

def generate\_report(self):

report\_content = "智能汽车异常检测报告\n"

report\_content += f"生成时间: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n\n"

report\_content += "异常检测结果:\n"

report\_content += f"已检测数据总数: {self.total\_predictions}\n"

report\_content += f"正常数据数: {self.normal\_count}\n"

report\_content += f"异常数据数: {self.abnormal\_count}\n"

report\_content += f"正常比例: {self.normal\_progress.value()}%\n"

report\_content += f"异常比例: {self.abnormal\_progress.value()}%\n\n"

report\_content += "系统状态:\n"

report\_content += f"CPU 使用率: {self.cpu\_progress.value()}%\n"

report\_content += f"内存使用率: {self.mem\_progress.value()}%\n"

report\_path, \_ = QFileDialog.getSaveFileName(self, "保存报告", "", "Text Files (\*.txt)")

if not report\_path:

return

try:

with open(report\_path, 'w', encoding='utf-8') as f:

f.write(report\_content)

self.log(f"报告已生成并保存至: {report\_path}")

QMessageBox.information(self, "成功", "报告生成成功。")

except Exception as e:

QMessageBox.critical(self, "错误", f"生成报告失败: {str(e)}")

def open\_settings(self):

settings\_dialog = QDialog()

settings\_dialog.setWindowTitle("系统设置")

settings\_dialog.setFixedSize(400, 300)

layout = QVBoxLayout()

label = QLabel("系统设置内容在此添加。")

label.setAlignment(Qt.AlignCenter)

layout.addWidget(label)

settings\_dialog.setLayout(layout)

settings\_dialog.exec\_()

@pyqtSlot(str)

def log(self, message):

timestamp = datetime.now().strftime("[%Y-%m-%d %H:%M:%S]")

self.logs\_text.append(f"{timestamp} {message}")

@pyqtSlot(int, list)

def update\_anomaly\_results(self, prediction, probability):

# 更新计数器

self.total\_predictions += 1

if prediction == 0:

self.normal\_count += 1

else:

self.abnormal\_count += 1

# 计算累计比例

normal\_percentage = (self.normal\_count / self.total\_predictions) \* 100

abnormal\_percentage = (self.abnormal\_count / self.total\_predictions) \* 100

# 更新进度条

self.normal\_progress.setValue(int(normal\_percentage))

self.abnormal\_progress.setValue(int(abnormal\_percentage))

@pyqtSlot(str, float, float)

def update\_plot(self, current\_time, val1, val2):

self.plot.update\_plot(current\_time, val1, val2)

def closeEvent(self, event):

if self.detection\_worker and self.detection\_worker.isRunning():

self.detection\_worker.stop()

self.detection\_worker.wait()

event.accept()

def main():

app = QApplication(sys.argv)

# 设置全局字体

font = QFont("SimHei", 12)

app.setFont(font)

window = MainWindow()

window.show()

sys.exit(app.exec\_())

if \_\_name\_\_ == "\_\_main\_\_":

main()

总线数据采集代码：

@Slf4j

@RestController

@TbCoreComponent

@RequestMapping("/api")

@RequiredArgsConstructor

public class GrcTimeseriesRestController extends BaseController {

private static final String DEVICE = "DEVICE";

private static final DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd-HH-mm-ss");

@Autowired

private AccessValidator accessValidator;

@Autowired

private GRCTimeseriesService grcTimeseriesService;

@Autowired

private CanFrameTimeseriesDao canFrameTimeseriesDao;

@Autowired

private FlexRayTimeseriesDao flexRayTimeseriesDao;

@Autowired

private LinFrameTimeseriesDao linFrameTimeseriesDao;

@Autowired

private XcpTimeseriesDao xcpTimeseriesDao;

@Autowired

private VehicleRpcClient vehicleGrpcClient;

@Autowired

private JobService jobService;

@Autowired

private FileService fileService;

@Autowired

private GraphService graphService;

@Autowired

private XcpMeasurementService xcpMeasurementService;

@Autowired

private EthFrameTimeseriesDao ethFrameTimeseriesDao;

@Autowired

private EthService ethService;

private ExecutorService executor;

@PostConstruct

public void initExecutor() {

executor = Executors.newSingleThreadExecutor(ThingsBoardThreadFactory.forName("telemetry-controller"));

NativeLibrary.addSearchPath("MDF4Weiter\_shared", "/usr/lib");

}

@PreDestroy

public void shutdownExecutor() {

if (executor != null) {

executor.shutdownNow();

}

}

@ApiOperation(value = "从边缘计算设备中获取XCP存储的时序数据", tags = "GRCE-Controller")

@PreAuthorize("hasAnyAuthority('SYS\_ADMIN', 'TENANT\_ADMIN', 'CUSTOMER\_USER')")

@RequestMapping(value = "/xcp/timeseries", method = RequestMethod.POST)

@ResponseBody

public DeferredResult<ResponseEntity> getXcpTimeseries(

@ApiParam(value = "一个可以代表Xcp过滤条件的json格式数据") @RequestBody GrcXcpDataFilter conditions,

@ApiParam(value = PAGE\_SIZE\_DESCRIPTION, defaultValue = "10") @RequestParam(required = false, defaultValue = "1000") int pageSize,

@ApiParam(value = "起始值为0的顺序值", defaultValue = "0") @RequestParam(required = false, defaultValue = "0") int page,

@ApiParam(value = SORT\_ORDER\_DESCRIPTION, allowableValues = SORT\_ORDER\_ALLOWABLE\_VALUES)

@RequestParam(name = "sortOrder", defaultValue = "DESC") String orderBy,

@ApiParam(value = SORT\_PROPERTY\_DESCRIPTION, allowableValues = GRC\_TS\_DATA\_SORT\_PROPERTY\_ALLOWABLE\_VALUES)

@RequestParam(required = false, defaultValue = "") String sortProperty) throws ThingsboardException {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

List<String> channels = conditions.getChannelIds() != null ? Arrays.asList(conditions.getChannelIds()) : new ArrayList<>();

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

return accessValidator.validateEntityAndCallback(getCurrentUser(), Operation.READ\_TELEMETRY, DEVICE, deviceIdStr,

(result, tenantId, deviceId) -> {

ReadXcpQuery query = new BaseReadXcpQuery(deviceId.getId(), startTs, endTs, orderBy, page, channels, pageSize);

Futures.addCallback(grcTimeseriesService.findXcp(tenantId, (DeviceId) deviceId, query), getGrcTsListCallback(result), MoreExecutors.directExecutor());

});

}

@ApiOperation(value = "根据页码和当前页条目数查出业务相关数据", tags = "GRCE-Controller")

@PreAuthorize("hasAnyAuthority('SYS\_ADMIN', 'TENANT\_ADMIN', 'CUSTOMER\_USER')")

@RequestMapping(value = "/canframe/timeseries", method = RequestMethod.POST)

@ResponseBody

public DeferredResult<ResponseEntity> getCanFrameTimeseries(

@ApiParam(value = "一个代表CAN Frame过滤条件的json数据") @RequestBody EolCanFrameDataFilter conditions,

@ApiParam(value = PAGE\_NUMBER\_DESCRIPTION) @RequestParam(required = false, defaultValue = "0") int page,

@ApiParam(value = PAGE\_SIZE\_DESCRIPTION) @RequestParam(required = false, defaultValue = "1000") int pageSize,

@ApiParam(value = SORT\_ORDER\_DESCRIPTION, allowableValues = SORT\_ORDER\_ALLOWABLE\_VALUES)

@RequestParam(name = "sortOrder", defaultValue = "DESC") String orderBy,

@ApiParam(value = SORT\_PROPERTY\_DESCRIPTION, allowableValues = GRC\_TS\_DATA\_SORT\_PROPERTY\_ALLOWABLE\_VALUES)

@RequestParam(required = false, defaultValue = "") String sortProperty) throws ThingsboardException {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

String[] channelIds = Optional.ofNullable(conditions.getChannelIds()).orElse(new String[]{});

List<String> channelIndexIds = Arrays.stream(channelIds).map(ModelConstants::convertFromCanNameToCanIndex).collect(Collectors.toList());

String channelId = channelIndexIds.isEmpty() ? "" : channelIndexIds.get(0);

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

String vin = Optional.ofNullable(conditions.getVin()).orElse("");

if (conditions.isEcr()) {

return accessValidator.validateEntityAndCallback(getCurrentUser(), Operation.READ\_TELEMETRY, DEVICE, deviceIdStr,

(result, tenantId, deviceId) -> {

ReadCanFrameQuery query = new BaseReadCanQuery(deviceId.getId(), startTs, endTs, orderBy, page, pageSize, channelIndexIds, channelId);

Futures.addCallback(grcTimeseriesService.findEcrCanFrame(tenantId, (DeviceId) deviceId, query), getGrcTsListCallback(result), MoreExecutors.directExecutor());

});

} else {

return accessValidator.validateEntityAndCallback(getCurrentUser(), Operation.READ\_TELEMETRY, DEVICE, deviceIdStr,

(result, tenantId, deviceId) -> {

ReadCanFrameQuery query = new BaseReadCanQuery(deviceId.getId(), startTs, endTs, orderBy, page, pageSize, channelIndexIds, channelId, vin);

Futures.addCallback(grcTimeseriesService.findCanFrame(tenantId, (DeviceId) deviceId, query), getGrcTsListCallback(result), MoreExecutors.directExecutor());

});

}

}

@ApiOperation(value = "根据页码和当前页条目数查出业务相关数据", tags = "GRCE-Controller")

@PreAuthorize("hasAnyAuthority('SYS\_ADMIN', 'TENANT\_ADMIN', 'CUSTOMER\_USER')")

@RequestMapping(value = "/linframe/timeseries", method = RequestMethod.POST)

@ResponseBody

public DeferredResult<ResponseEntity> getLinFrameTimeseries(

@ApiParam(value = "一个代表LIN Frame过滤条件的json数据") @RequestBody GrcCanFrameDataFilter conditions,

@ApiParam(value = PAGE\_NUMBER\_DESCRIPTION) @RequestParam(required = false, defaultValue = "0") int page,

@ApiParam(value = PAGE\_SIZE\_DESCRIPTION) @RequestParam(required = false, defaultValue = "1000") int pageSize,

@ApiParam(value = SORT\_ORDER\_DESCRIPTION, allowableValues = SORT\_ORDER\_ALLOWABLE\_VALUES)

@RequestParam(name = "sortOrder", defaultValue = "DESC") String orderBy,

@ApiParam(value = SORT\_PROPERTY\_DESCRIPTION, allowableValues = GRC\_TS\_DATA\_SORT\_PROPERTY\_ALLOWABLE\_VALUES)

@RequestParam(required = false, defaultValue = "") String sortProperty) throws ThingsboardException {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

String[] channelIds = Optional.ofNullable(conditions.getChannelIds()).orElse(new String[]{});

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

return accessValidator.validateEntityAndCallback(getCurrentUser(), Operation.READ\_TELEMETRY, DEVICE, deviceIdStr,

(result, tenantId, deviceId) -> {

ReadLinFrameQuery query = new BaseReadLinQuery(deviceId.getId(), startTs, endTs, orderBy, page, pageSize, Arrays.asList(channelIds));

Futures.addCallback(grcTimeseriesService.findLinFrame(tenantId, (DeviceId) deviceId, query), getGrcTsListCallback(result), MoreExecutors.directExecutor());

});

}

@ApiOperation(value = "根据页码和当前页条目数查出业务相关数据", tags = "GRCE-Controller")

@PreAuthorize("hasAnyAuthority('SYS\_ADMIN', 'TENANT\_ADMIN', 'CUSTOMER\_USER')")

@RequestMapping(value = "/flexrayframe/timeseries", method = RequestMethod.POST)

@ResponseBody

public DeferredResult<ResponseEntity> getFlexRayFrameTimeseries(

@ApiParam(value = "一个代表FlexRay Frame过滤条件的json数据") @RequestBody GrcCanFrameDataFilter conditions,

@ApiParam(value = PAGE\_NUMBER\_DESCRIPTION) @RequestParam(required = false, defaultValue = "0") int page,

@ApiParam(value = PAGE\_SIZE\_DESCRIPTION) @RequestParam(required = false, defaultValue = "1000") int pageSize,

@ApiParam(value = SORT\_ORDER\_DESCRIPTION, allowableValues = SORT\_ORDER\_ALLOWABLE\_VALUES)

@RequestParam(name = "sortOrder", defaultValue = "DESC") String orderBy,

@ApiParam(value = SORT\_PROPERTY\_DESCRIPTION, allowableValues = GRC\_TS\_DATA\_SORT\_PROPERTY\_ALLOWABLE\_VALUES)

@RequestParam(required = false, defaultValue = "") String sortProperty) throws ThingsboardException {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

String[] channelIds = Optional.ofNullable(conditions.getChannelIds()).orElse(new String[]{});

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

return accessValidator.validateEntityAndCallback(getCurrentUser(), Operation.READ\_TELEMETRY, DEVICE, deviceIdStr,

(result, tenantId, deviceId) -> {

ReadFlexRayFrameQuery query = new BaseReadFlexRayQuery(deviceId.getId(), startTs, endTs, orderBy, page, pageSize, Arrays.asList(channelIds));

Futures.addCallback(grcTimeseriesService.findFlexRayFrame(tenantId, (DeviceId) deviceId, query), getGrcTsListCallback(result), MoreExecutors.directExecutor());

});

}

@ApiOperation(value = "Get Grc Uds stored in timeseries db from edge compute device", produces = MediaType.APPLICATION\_JSON\_VALUE)

@PreAuthorize("hasAnyAuthority('SYS\_ADMIN', 'TENANT\_ADMIN', 'CUSTOMER\_USER')")

@RequestMapping(value = "/uds/timeseries", method = RequestMethod.POST)

@ResponseBody

public DeferredResult<ResponseEntity> getUdsTimeseries(

@ApiParam(value = "A JSON value representing conditions to filter can frame") @RequestBody GrcUdsDataFilter conditions,

@ApiParam(value = PAGE\_SIZE\_DESCRIPTION, defaultValue = "10") @RequestParam(required = false, defaultValue = "10") int pageSize,

@ApiParam(value = "Sequence number of page starting from 0", defaultValue = "0") @RequestParam(required = false, defaultValue = "0") int page,

@ApiParam(value = SORT\_ORDER\_DESCRIPTION, allowableValues = SORT\_ORDER\_ALLOWABLE\_VALUES)

@RequestParam(name = "sortOrder", defaultValue = "DESC") String orderBy,

@ApiParam(value = SORT\_PROPERTY\_DESCRIPTION, allowableValues = GRC\_TS\_DATA\_SORT\_PROPERTY\_ALLOWABLE\_VALUES)

@RequestParam(required = false, defaultValue = "") String sortProperty) throws ThingsboardException {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

List<String> originChannelIds = Arrays.asList(Optional.ofNullable(conditions.getChannelIds()).orElse(new String[]{}));

UdsType type = Optional.ofNullable(conditions.getType()).orElse(UdsType.ALL);

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

return accessValidator.validateEntityAndCallback(getCurrentUser(), Operation.READ\_TELEMETRY, DEVICE, deviceIdStr,

(result, tenantId, deviceId) -> {

ReadUdsQuery query = new BaseReadUdsQuery(deviceId.getId(), startTs, endTs, orderBy, page, pageSize, originChannelIds.toArray(String[]::new), type);

Futures.addCallback(grcTimeseriesService.findUds(tenantId, (DeviceId) deviceId, query), getGrcTsListCallback(result), MoreExecutors.directExecutor());

});

}

@ApiOperation(value = "导出mf4文件", tags = "GRCE-Controller")

@RequestMapping(value = "/v1/canframe/export", method = RequestMethod.POST)

@ResponseBody

public DeferredResult<ResponseEntity> getFile(@ApiParam(value = "一个可以代表导出的 Can Frame的json格式数据") @RequestBody GrcCanFrameDataFilter conditions,

HttpServletResponse response) throws ThingsboardException {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

String[] channelIds = Optional.ofNullable(conditions.getChannelIds()).orElse(new String[]{});

String channelId = channelIds.length > 0 ? channelIds[0] : "";

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

final String path = "/tmp/";

final Long ts = System.currentTimeMillis();

final String filename = ts + "\_MDF.mf4";

return accessValidator.validateEntityAndCallback(getCurrentUser(), Operation.READ\_TELEMETRY, DEVICE, deviceIdStr,

(result, tenantId, deviceId) -> {

CanMdfQuery query = new CanMdfQuery(deviceId.getId(), Collections.singletonList(channelId), startTs, endTs);

Futures.addCallback(grcTimeseriesService.findExportCanFrame(tenantId, deviceId.getId(), query), new FutureCallback<>() {

@Override

public void onSuccess(List<CanFrameProtoMapping> canFrameProtoList) {

if (canFrameProtoList.size() > 0) {

CanFrameMdf[] canFrameMdfs = (CanFrameMdf[]) new CanFrameMdf().toArray(canFrameProtoList.size());

for (int i = 0; i < canFrameProtoList.size(); i++) {

CanFrameProtoMapping canFrameProto = canFrameProtoList.get(i);

canFrameMdfs[i].channel = (byte) canFrameProto.getChannel();

canFrameMdfs[i].data = canFrameProto.getDataBytes();

canFrameMdfs[i].msgId = canFrameProto.getID();

canFrameMdfs[i].msgLength = (byte) canFrameProto.getDLC();

canFrameMdfs[i].timestamp = canFrameProto.getTimeStamp();

canFrameMdfs[i].isCanFD = canFrameProto.getIsCanFD() ? (byte) 1 : (byte) 0;

canFrameMdfs[i].isTx = canFrameProto.getIsTx() ? (byte) 1 : (byte) 0;

canFrameMdfs[i].isBrs = canFrameProto.getIsBrs() ? (byte) 1 : (byte) 0;

canFrameMdfs[i].isErr = canFrameProto.getIsErr() ? (byte) 1 : (byte) 0;

canFrameMdfs[i].isRtr = canFrameProto.getIsRtr() ? (byte) 1 : (byte) 0;

canFrameMdfs[i].isEff = canFrameProto.getIsEff() ? (byte) 1 : (byte) 0;

}

try {

// call mdf JNA interface

MdfHelper.INSTANCE.HostMDF4writer(canFrameMdfs, canFrameMdfs.length, path, filename);

// get your file as InputStream

InputStream is = null;

is = new FileInputStream(path + filename);

// copy it to response's OutputStream

org.apache.commons.io.IOUtils.copy(is, response.getOutputStream());

response.flushBuffer();

result.setResult(new ResponseEntity<>(null, HttpStatus.OK));

} catch (Exception ex) {

log.error("Error writing file to output stream.", ex);

throw new RuntimeException("IOError writing file to output stream");

}

}

}

@Override

public void onFailure(Throwable e) {

log.error("Failed to export mdf data", e);

AccessValidator.handleError(e, result, HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}, MoreExecutors.directExecutor());

});

}

@ApiOperation(value = "下载mf4文件", tags = "GRCE-Controller")

@RequestMapping(value = "/canframe/download/{timestamp}", method = RequestMethod.GET)

@ResponseBody

public void downloadFile(@ApiParam(value = "时间戳") @PathVariable("timestamp") String timestamp,

HttpServletResponse response) throws ThingsboardException {

long ts = Long.parseLong(timestamp);

final String path = "/tmp/";

final String filename = ts + "\_MDF.mf4";

try {

// get your file as InputStream

InputStream is = null;

is = new FileInputStream(path + filename);

// copy it to response's OutputStream

org.apache.commons.io.IOUtils.copy(is, response.getOutputStream());

// TODO remove the server mf4 file with write protection

response.flushBuffer();

} catch (IOException ex) {

log.info("Error writing file to output stream.", ex);

throw new RuntimeException("IOError writing file to output stream");

}

}

@ApiOperation(value = "graph单独的数据查询", tags = "GRCE-Controller")

@PreAuthorize("hasAnyAuthority('SYS\_ADMIN', 'TENANT\_ADMIN', 'CUSTOMER\_USER')")

@RequestMapping(value = "/graph/timeseries", method = RequestMethod.POST)

@ResponseBody

public DeferredResult<ResponseEntity> getGraphTimeseries(

@ApiParam(value = "一个可以代表 can frame 过滤条件的json格式数据") @RequestBody GraphDataFilter conditions) throws ThingsboardException {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

Map<String, List<GraphSubCmdData>> signalLists = Optional.ofNullable(conditions.getSignalLists()).orElse(new HashMap<>());

long maxPushDataSize = Optional.of(conditions.getMaxPushDataSize()).orElse(0L);

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

return accessValidator.validateEntityAndCallback(getCurrentUser(), Operation.READ\_TELEMETRY, DEVICE, deviceIdStr,

(result, tenantId, deviceId) -> {

ReadCanFrameQuery query = new BaseReadCanQuery(deviceId.getId(), startTs, endTs, "DESC", 1, "");

Futures.addCallback(grcTimeseriesService.findGraphData(tenantId, deviceId.getId(), query, signalLists, maxPushDataSize), getGraphListCallback(result), MoreExecutors.directExecutor());

});

}

@ApiOperation(value = "下载mf4文件", tags = "GRCE-Controller")

@RequestMapping(value = "/canframe/export", method = RequestMethod.POST)

@ResponseBody

public JobId uploadFile(@ApiParam(value = "一个可以代表导出的 Can Frame的json格式数据")

@RequestBody EolCanFrameDataFilter conditions,

@ApiParam(value = "导出的文件类型", defaultValue = "MF4") @RequestParam(required = false, defaultValue = "MF4") ExportFileType fileType,

@RequestParam(required = false, defaultValue = "Normal") String dataType) throws ThingsboardException {

try {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

boolean isEol = deviceIdStr.isEmpty();

UUID deviceId = isEol ? UUID.randomUUID() : UUID.fromString(deviceIdStr);

final TenantId tenantId = getTenantId();

String[] channelIds = Optional.ofNullable(conditions.getChannelIds()).orElse(new String[]{});

List<String> channelIndexIds = Arrays.stream(channelIds).map(ModelConstants::convertFromCanNameToCanIndex).collect(Collectors.toList());

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

String vin = Optional.ofNullable(conditions.getVin()).orElse("");

String ts = startTs > 0 ? LocalDateTime.ofInstant(Instant.ofEpochMilli(startTs / 1000), ZoneId.systemDefault()).format(formatter) : LocalDateTime.now().format(formatter);

StringBuilder nameBuilder = new StringBuilder();

if (!vin.isEmpty()) {

nameBuilder.append(vin).append("\_");

}

final Job job = new Job();

final String filename = ts + "." + fileType.getFileSuffix();

job.setEntityId(UUID.fromString(deviceId.toString()));

job.setDownloadName(filename);

JobId jobId = jobService.createJob(tenantId, job);

CanMdfQuery query = new CanMdfQuery(deviceId, channelIndexIds, startTs, endTs, vin);

Futures.addCallback(canFrameTimeseriesDao.findPkg(tenantId, deviceId, query, filename, fileType, "Ecr".equals(dataType), isEol), new FutureCallback<>() {

@Override

public void onSuccess(List<String> data) {

if (data != null && data.size() > 0) {

boolean isTmp = data.get(0).startsWith("/tmp");

Collections.sort(data, (s1, s2) -> {

// 提取 "\_" 前面的部分并转为 Long

Long t1, t2;

if (isTmp) {

t1 = Long.parseLong(s1.split("/tmp/")[1].split("\_")[0]);

t2 = Long.parseLong(s2.split("/tmp/")[1].split("\_")[0]);

} else {

t1 = Long.parseLong(s1.split("\_")[0]);

t2 = Long.parseLong(s2.split("\_")[0]);

}

return t1.compareTo(t2);

});

long st = startTs > 0 ? startTs : Long.parseLong(isTmp ? data.get(0).split("/tmp/")[1].split("\_")[0] : data.get(0).split("\_")[0]);

nameBuilder.append(LocalDateTime.ofInstant(Instant.ofEpochMilli(st / 1000), ZoneId.systemDefault()).format(formatter)).append("\_");

long et = endTs > 0 ? endTs : Long.parseLong(isTmp ? data.get(data.size() - 1).split("/tmp/")[1].split("\_")[1] : data.get(data.size() - 1).split("\_")[1]);

nameBuilder.append(LocalDateTime.ofInstant(Instant.ofEpochMilli(et / 1000), ZoneId.systemDefault()).format(formatter));

nameBuilder.append(".").append(fileType.getFileSuffix());

String presignedUrl = "";

if (fileType == ExportFileType.MF4) {

presignedUrl = fileService.compose(data, nameBuilder.toString());

} else {

// merge blf

FrameConverter converter = new FrameConverter();

if (fileType == ExportFileType.ASC) {

if (converter.mergeAscFiles(nameBuilder.toString(), data.toArray(String[]::new)) == 0) {

log.info("merge asc file success");

} else {

log.error("merge asc file failed");

}

} else {

if (converter.mergeBlfFiles(nameBuilder.toString(), data.toArray(String[]::new)) == 0) {

log.info("merge blf file success");

} else {

log.error("merge blf file failed");

}

}

presignedUrl = fileService.uploadLocalFile(nameBuilder.toString(), nameBuilder.toString());

}

data.add(nameBuilder.toString());

fileService.deleteLocalFiles(data);

jobService.finishJob(tenantId, jobId, JobStatus.SUCCEEDED, presignedUrl, nameBuilder.toString());

} else {

jobService.finishJob(tenantId, jobId, JobStatus.FAILED, "");

}

}

@Override

public void onFailure(Throwable e) {

log.error("Failed to export mdf data", e);

jobService.finishJob(tenantId, jobId, JobStatus.FAILED, "");

}

}, MoreExecutors.directExecutor());

return jobId;

} catch (Exception ex) {

log.info("Error writing file to output stream.", ex);

throw new RuntimeException("IOError writing file to output stream");

}

}

@ApiOperation(value = "下载flexray mf4文件", tags = "GRCE-Controller")

@RequestMapping(value = "/flexray/export", method = RequestMethod.POST)

@ResponseBody

public JobId uploadFlexrayFrameFile(@ApiParam(value = "一个可以代表导出的 Flexray Frame的json格式数据")

@RequestBody GrcCanFrameDataFilter conditions,

@ApiParam(value = "导出的文件类型", defaultValue = "MF4") @RequestParam(required = false, defaultValue = "MF4") ExportFileType fileType,

@RequestParam(required = false, defaultValue = "Normal") String dataType) throws ThingsboardException {

try {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

UUID deviceId = UUID.fromString(deviceIdStr);

final TenantId tenantId = getTenantId();

String[] channelIds = Optional.ofNullable(conditions.getChannelIds()).orElse(new String[]{});

List<String> channelIndexIds = Arrays.stream(channelIds).map(ModelConstants::convertFromCanNameToCanIndex).collect(Collectors.toList());

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

String ts = startTs > 0 ? LocalDateTime.ofInstant(Instant.ofEpochMilli(startTs / 1000), ZoneId.systemDefault()).format(formatter) : LocalDateTime.now().format(formatter);

final Job job = new Job();

final String filename = ts + "." + fileType.getFileSuffix();

job.setEntityId(UUID.fromString(deviceIdStr));

job.setDownloadName(filename);

JobId jobId = jobService.createJob(tenantId, job);

FlexRayMdfQuery query = new FlexRayMdfQuery(deviceId, startTs, endTs);

Futures.addCallback(flexRayTimeseriesDao.findPkg(tenantId, deviceId, query, filename, fileType, "Ecr".equals(dataType)), new FutureCallback<>() {

@Override

public void onSuccess(List<String> data) {

String presignedUrl = fileService.compose(data, filename);

jobService.finishJob(tenantId, jobId, JobStatus.SUCCEEDED, presignedUrl);

}

@Override

public void onFailure(Throwable e) {

log.error("Failed to export mdf data", e);

jobService.finishJob(tenantId, jobId, JobStatus.FAILED, "");

}

}, MoreExecutors.directExecutor());

return jobId;

} catch (Exception ex) {

log.info("Error writing file to output stream.", ex);

throw new RuntimeException("IOError writing file to output stream");

}

}

@ApiOperation(value = "下载lin mf4文件", tags = "GRCE-Controller")

@RequestMapping(value = "/linframe/export", method = RequestMethod.POST)

@ResponseBody

public JobId uploadLinFrameFile(@ApiParam(value = "一个可以代表导出的 Lin Frame的json格式数据")

@RequestBody GrcCanFrameDataFilter conditions,

@ApiParam(value = "导出的文件类型", defaultValue = "MF4") @RequestParam(required = false, defaultValue = "MF4") ExportFileType fileType) throws ThingsboardException {

try {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

UUID deviceId = UUID.fromString(deviceIdStr);

final TenantId tenantId = getTenantId();

String[] channelIds = Optional.ofNullable(conditions.getChannelIds()).orElse(new String[]{});

List<String> channelIndexIds = Arrays.stream(channelIds).map(ModelConstants::convertFromCanNameToCanIndex).collect(Collectors.toList());

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

String ts = startTs > 0 ? LocalDateTime.ofInstant(Instant.ofEpochMilli(startTs / 1000), ZoneId.systemDefault()).format(formatter) : LocalDateTime.now().format(formatter);

final Job job = new Job();

final String filename = ts + "." + fileType.getFileSuffix();

job.setEntityId(UUID.fromString(deviceIdStr));

job.setDownloadName(filename);

JobId jobId = jobService.createJob(tenantId, job);

CanMdfQuery query = new CanMdfQuery(deviceId, channelIndexIds, startTs, endTs);

Futures.addCallback(linFrameTimeseriesDao.findPkg(tenantId, deviceId, query, filename, fileType), new FutureCallback<>() {

@Override

public void onSuccess(List<String> data) {

String presignedUrl = fileService.compose(data, filename);

jobService.finishJob(tenantId, jobId, JobStatus.SUCCEEDED, presignedUrl);

}

@Override

public void onFailure(Throwable e) {

log.error("Failed to export lin mdf data", e);

jobService.finishJob(tenantId, jobId, JobStatus.FAILED, "");

}

}, MoreExecutors.directExecutor());

return jobId;

} catch (Exception ex) {

log.info("Error writing file to output stream.", ex);

throw new RuntimeException("IOError writing file to output stream");

}

}

@ApiOperation(value = "trace导出文件", tags = "GRCE-Controller")

@RequestMapping(value = "/trace/canframe/export", method = RequestMethod.POST)

@ResponseBody

public ResponseEntity<JobDownloadReturnData> exportTraceCanFile(@ApiParam(value = "传入的json格式的can数据") @RequestBody List<TraceCanFrame> data,

@ApiParam(value = "导出的文件类型", defaultValue = "MF4") @RequestParam(required = false, defaultValue = "MF4") ExportFileType fileType) throws ThingsboardException {

try {

String presignedUrl = "";

final String ts = LocalDateTime.now().format(formatter);

final String filename = ts + "." + fileType.getFileSuffix();

List<CanFramePackage> canFramePackages = canFrameTimeseriesDao.convertFromTraceDataToPkg(data);

switch (fileType) {

case ASC:

presignedUrl = vehicleGrpcClient.saveCanFramesToAscFile(canFramePackages, filename, false);

break;

case BLF:

presignedUrl = vehicleGrpcClient.saveCanFramesToBlfFile(canFramePackages, filename, false);

break;

default:

presignedUrl = vehicleGrpcClient.saveCanFramesToMf4File(canFramePackages, filename, false);

}

return new ResponseEntity<>(JobDownloadReturnData.builder().jobId(UUID.randomUUID())

.downloadLink(presignedUrl).fileName(filename).build(), HttpStatus.OK);

} catch (InterruptedException e) {

log.error("Failed to export mdf data", e);

return null;

}

}

@ApiOperation(value = "根据页码和当前页条目数查出业务相关数据", tags = "GRCE-Controller")

@PreAuthorize("hasAnyAuthority('SYS\_ADMIN', 'TENANT\_ADMIN', 'CUSTOMER\_USER')")

@RequestMapping(value = "/eth/timeseries", method = RequestMethod.POST)

@ResponseBody

public DeferredResult<ResponseEntity> getEthTimeseries(

@ApiParam(value = "一个代表Eth过滤条件的json数据") @RequestBody GrcCanFrameDataFilter conditions,

@ApiParam(value = PAGE\_NUMBER\_DESCRIPTION) @RequestParam(required = false, defaultValue = "0") int page,

@ApiParam(value = PAGE\_SIZE\_DESCRIPTION) @RequestParam(required = false, defaultValue = "1000") int pageSize,

@ApiParam(value = SORT\_ORDER\_DESCRIPTION, allowableValues = SORT\_ORDER\_ALLOWABLE\_VALUES)

@RequestParam(name = "sortOrder", defaultValue = "DESC") String orderBy,

@ApiParam(value = SORT\_PROPERTY\_DESCRIPTION, allowableValues = GRC\_TS\_DATA\_SORT\_PROPERTY\_ALLOWABLE\_VALUES)

@RequestParam(required = false, defaultValue = "") String sortProperty) throws ThingsboardException {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

String[] channelIds = Optional.ofNullable(conditions.getChannelIds()).orElse(new String[]{});

List<String> channelIndexIds = Arrays.stream(channelIds).map(ModelConstants::convertFromCanNameToCanIndex).collect(Collectors.toList());

String channelId = channelIndexIds.size() > 0 ? channelIndexIds.get(0) : "";

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

return accessValidator.validateEntityAndCallback(getCurrentUser(), Operation.READ\_TELEMETRY, DEVICE, deviceIdStr,

(result, tenantId, deviceId) -> {

ReadEthQuery query = new BaseReadEthQuery(deviceId.getId(), startTs, endTs, orderBy, page, pageSize, channelIndexIds, channelId);

Futures.addCallback(grcTimeseriesService.findEthFrame(tenantId, (DeviceId) deviceId, query), getGrcTsListCallback(result), MoreExecutors.directExecutor());

});

}

@ApiOperation(value = "上传mf4文件并解析", tags = "GRCE-Controller")

@PreAuthorize("hasAnyAuthority('TENANT\_ADMIN', 'CUSTOMER\_USER')")

@RequestMapping(value = "/graph/upload", method = RequestMethod.POST)

@ResponseStatus(value = HttpStatus.OK)

public ResponseEntity<GraphTsResponseData> parseDbcAndSave(@ApiParam(value = "加载文件") @RequestPart("multipartFile") MultipartFile multipartFile,

@ApiParam(value = "设备Id") @RequestParam(value = "deviceId") String deviceId,

@ApiParam(value = "文件类型") @RequestParam(value = "fileType", defaultValue = "MF4") ExportFileType fileType) throws Exception {

if (multipartFile != null) {

List<FramePackage> canFramePackages;

switch (fileType) {

case MF4:

canFramePackages = vehicleGrpcClient.saveMf4FileToCanFrames(multipartFile);

break;

default:

canFramePackages = vehicleGrpcClient.saveMf4FileToCanFrames(multipartFile);

}

return new ResponseEntity<>(graphService.parseSignals(canFramePackages, UUID.fromString(deviceId)), HttpStatus.OK);

}

return new ResponseEntity<>(new GraphTsResponseData(new HashMap<>()), HttpStatus.OK);

}

@ApiOperation(value = "导出xcp的mf4文件", tags = "GRCE-Controller")

@RequestMapping(value = "/xcp/measurement/export", method = RequestMethod.GET)

@ResponseBody

public JobId uploadXcpFile(@ApiParam(value = "设备Id") @RequestParam(value = "deviceId") String deviceIdStr,

@ApiParam(value = "xcp规则Id") @RequestParam(value = "ruleId") String ruleId) throws ThingsboardException {

try {

UUID deviceId = UUID.fromString(deviceIdStr);

final TenantId tenantId = getTenantId();

long ts = System.currentTimeMillis();

final String format = LocalDateTime.now().format(formatter);

final Job job = new Job();

final String filename = format + ".mf4";

job.setEntityId(deviceId);

job.setDownloadName(filename);

JobId jobId = jobService.createJob(tenantId, job);

Long startTimeVar = xcpMeasurementService.stopMeasurement(ruleId);

long startTime = startTimeVar != null ? startTimeVar : ts \* 1000 - 10000 \* 1000;

Futures.addCallback(xcpTimeseriesDao.findSignals(tenantId, new DeviceId(deviceId), 1651176866149295L, 1651176874149224L), new FutureCallback<>() {

@Override

public void onSuccess(List<SignalPackage> signalPackages) {

try {

String url = vehicleGrpcClient.saveSignalsToMF4File(signalPackages, filename, false);

jobService.finishJob(tenantId, jobId, JobStatus.SUCCEEDED, url);

} catch (InterruptedException e) {

log.error("error download xcp mf4" + e.getMessage());

jobService.finishJob(tenantId, jobId, JobStatus.FAILED, "");

}

}

@Override

public void onFailure(Throwable e) {

log.error("Failed to export mdf data", e);

jobService.finishJob(tenantId, jobId, JobStatus.FAILED, "");

}

}, MoreExecutors.directExecutor());

return jobId;

} catch (Exception ex) {

log.info("Error writing file to output stream.", ex);

throw new RuntimeException("IOError writing file to output stream");

}

}

@ApiOperation(value = "导出打点zip文件", tags = "GRCE-Controller")

@RequestMapping(value = "/zip/export", method = RequestMethod.POST)

@ResponseBody

public JobId uploadZipFile(@ApiParam(value = "一个可以代表导出的zip的json格式数据")

@RequestBody DotDataFilter conditions) throws ThingsboardException {

try {

String deviceIdStr = Optional.ofNullable(conditions.getDeviceId()).orElse("");

UUID deviceId = UUID.fromString(deviceIdStr);

final TenantId tenantId = getTenantId();

long startTs = Optional.ofNullable(conditions.getStartTime()).orElse(0L);

long endTs = Optional.ofNullable(conditions.getEndTime()).orElse(0L);

String ss = startTs > 0 ? LocalDateTime.ofInstant(Instant.ofEpochMilli(startTs / 1000), ZoneId.systemDefault()).format(formatter) : LocalDateTime.now().format(formatter);

String se = startTs > 0 ? LocalDateTime.ofInstant(Instant.ofEpochMilli(endTs / 1000), ZoneId.systemDefault()).format(formatter) : LocalDateTime.now().format(formatter);

String name = ss + "\_" + se;

final Job job = new Job();

final String filename = name + ".zip";

final String linFilename = name + "\_lin.asc";

final String canFilename = name + "\_can.blf";

final String ethFilename = name + "\_eth.pcap";

job.setEntityId(UUID.fromString(deviceIdStr));

job.setDownloadName(filename);

JobId jobId = jobService.createJob(tenantId, job);

ListenableFuture<List<String>> exportLinAsync = linFrameTimeseriesDao.findPkg(tenantId, deviceId,

new CanMdfQuery(deviceId, Collections.emptyList(), startTs, endTs), linFilename, ExportFileType.ASC);

ListenableFuture<List<String>> exportCanAsync = canFrameTimeseriesDao.findPkg(tenantId, deviceId,

new CanMdfQuery(deviceId, Collections.emptyList(), startTs, endTs, null), canFilename, ExportFileType.BLF, false, false);

ListenableFuture<List<Pair<String, Integer>>> exportEthAsync = ethFrameTimeseriesDao.findPkg(tenantId, deviceId,

new EthPcapQuery(deviceId, Collections.emptyList(), startTs, endTs), ethFilename);

ListenableFuture<List<Object>> allExports = Futures.allAsList(exportLinAsync, exportCanAsync, exportEthAsync);

Futures.addCallback(allExports, new FutureCallback<>() {

@Override

public void onSuccess(List<Object> result) {

List<String> linFileData = (List<String>) result.get(0);

List<String> canFileData = (List<String>) result.get(1);

List<Pair<String, Integer>> ethFileData = (List<Pair<String, Integer>>) result.get(2);

List<String> sourceList = new ArrayList<>();

List<String> minioList = new ArrayList<>();

try {

fileService.compose(linFileData, linFilename);

minioList.add(linFilename);

} catch (Exception e1) {

log.error("error export lin, {}", e1.getMessage());

}

try {

ethService.saveEthPayloadToPcapFile(ethFileData, ethFilename);

minioList.add(ethFilename);

} catch (Exception e2) {

log.error("error export eth, {}", e2.getMessage());

}

if (canFileData != null && canFileData.size() > 0) {

// merge blf

FrameConverter converter = new FrameConverter();

if (converter.mergeBlfFiles(canFilename, canFileData.toArray(String[]::new)) == 0) {

log.info("merge blf file success");

} else {

log.error("merge blf file failed");

}

sourceList.add(canFilename);

}

if (sourceList.isEmpty() && minioList.isEmpty()) {

jobService.finishJob(tenantId, jobId, JobStatus.FAILED, "");

} else {

String presignedUrl = fileService.zipCompose(sourceList, minioList, filename);

jobService.finishJob(tenantId, jobId, JobStatus.SUCCEEDED, presignedUrl, filename);

fileService.deleteLocalFiles(canFileData);

}

}

@Override

public void onFailure(Throwable t) {

log.error("Failed to export zip file", t);

jobService.finishJob(tenantId, jobId, JobStatus.FAILED, "");

}

}, MoreExecutors.directExecutor());

return jobId;

} catch (Exception ex) {

log.info("Error writing file to output stream.", ex);

throw new RuntimeException("IOError writing file to output stream");

}

}

private FutureCallback<GrcTsResult<?>> getGrcTsListCallback(final DeferredResult<ResponseEntity> response) {

return new FutureCallback<>() {

@Override

public void onSuccess(GrcTsResult data) {

response.setResult(new ResponseEntity<>(data, HttpStatus.OK));

}

@Override

public void onFailure(Throwable e) {

log.error("Failed to fetch grc timeseries data", e);

AccessValidator.handleError(e, response, HttpStatus.INTERNAL\_SERVER\_ERROR);

}

};

}

private FutureCallback<GraphTsResponseData> getGraphListCallback(final DeferredResult<ResponseEntity> response) {

return new FutureCallback<>() {

@Override

public void onSuccess(GraphTsResponseData data) {

response.setResult(new ResponseEntity<>(data, HttpStatus.OK));

}

@Override

public void onFailure(Throwable e) {

log.error("Failed to fetch grc timeseries data", e);

AccessValidator.handleError(e, response, HttpStatus.INTERNAL\_SERVER\_ERROR);

}

};

}

}

<mat-drawer-container class="tb-absolute-fill">

<mat-drawer class="tb-details-drawer " disableClose="true" #drawer mode="over" position="end"

[opened]="isDetailsOpen">

<tb-dtc-code [config]="dtcConfig" \*ngIf="isDetailsOpen" style="width: 100%; height: 100%; overflow: hidden;"

[close]="closeDtcCode"></tb-dtc-code>

</mat-drawer>

<mat-drawer-content>

<div class="eol-page-content-wrapper">

<mat-toolbar>

<div class="option-wrapper">

<mat-form-field class="base-filed-wrapper" subscriptSizing="dynamic"

style="max-width: 140px; min-width: 100px;">

<mat-label>全部设备:</mat-label>

<input type="text" #deviceInput matInput [formControl]="deviceControl" aria-label="explaination"

(input)="filter()" (focus)="filter()" [matAutocomplete]="auto">

<mat-autocomplete #auto="matAutocomplete" (optionSelected)="handleSeleectDevice($event)"

[displayWith]="displayFn">

<mat-option \*ngFor="let item of filteredOptions" class="autocomplete-option" [matTooltip]='item.name'

[value]="item">

<span class="item-value">{{item.name}}</span>

</mat-option>

</mat-autocomplete>

<button class="clear-device-btn" mat-icon-button matSuffix \*ngIf="deviceId"

(click)="handleClearDevice($event)">

<mat-icon>close</mat-icon>

</button>

</mat-form-field>

<mat-form-field subscriptSizing="dynamic" class="base-filed-wrapper" style="margin-left: 12px; width: 140px;">

<mat-label>测试类型:</mat-label>

<mat-select [(ngModel)]="eolType">

<mat-option \*ngFor="let item of eolTypeOptions" [value]="item.value">

{{item.label}}

</mat-option>

</mat-select>

<button class="clear-device-btn" mat-icon-button matSuffix \*ngIf="eolType"

(click)="handleClearCondition($event, 'eolType')">

<mat-icon>close</mat-icon>

</button>

</mat-form-field>

<tb-docap-autocomplete #docapInput class="base-filed-wrapper"

style="margin-left: 12px; width: 210px; min-width: 100px;" [otherProps]='{ isError: false }'

[optionSelected]='handleConditionSelectVariable' [sortOrder]="{}" uniqKey="label"

[getAutoCompleteOptions]='getAutoCompleteList' [value]="vinCode ?? ''" autoLabel='VIN' placeholder='VIN'

inputTooltip='输入的变量不在当前列表中'>

</tb-docap-autocomplete>

<div style=" display: flex;

align-items: center;">

<section style="margin-left: 12px;">

<mat-form-field class="base-filed-wrapper" subscriptSizing="dynamic">

<mat-label translate>开始时间</mat-label>

<span style="display: inline-flex;">

<input [(ngModel)]="startTime" matInput [owlDateTime]="dt1">

<mat-icon style=" overflow: visible;" openOnFocus="true"

[owlDateTimeTrigger]="dt1">access\_time</mat-icon>

<owl-date-time #dt1 [showSecondsTimer]="true"></owl-date-time>

</span>

</mat-form-field>

</section>

<span style=" display: inline-block;

width: 8px;

height: 1px;

border-top: 1px solid;

margin: 4px;"></span>

<section>

<mat-form-field class="base-filed-wrapper" subscriptSizing="dynamic">

<mat-label translate>结束时间</mat-label>

<span style="display: inline-flex;">

<input [(ngModel)]="endTime" matInput [owlDateTime]="dt2">

<mat-icon style=" overflow: visible;" openOnFocus="true"

[owlDateTimeTrigger]="dt2">access\_time</mat-icon>

<owl-date-time #dt2 [showSecondsTimer]="true"></owl-date-time>

</span>

</mat-form-field>

</section>

</div>

<mat-form-field subscriptSizing="dynamic" style="margin-left: 12px; width: 140px" class="base-filed-wrapper">

<mat-label>类型:</mat-label>

<mat-select [(ngModel)]="status">

<mat-option \*ngFor="let item of statusOptions" [value]='item.value'>

{{item.label}}

</mat-option>

</mat-select>

<button class="clear-device-btn" mat-icon-button matSuffix \*ngIf="status"

(click)="handleClearCondition($event, 'status')">

<mat-icon>close</mat-icon>

</button>

</mat-form-field>

<mat-form-field subscriptSizing="dynamic" style="margin-left: 12px; width: 140px" class="base-filed-wrapper">

<mat-label>DTC:</mat-label>

<mat-select [(ngModel)]="hasDtcCode">

<mat-option \*ngFor="let item of dtcOptions" [value]='item.value'>

{{item.label}}

</mat-option>

</mat-select>

<button class="clear-device-btn" mat-icon-button matSuffix \*ngIf="status"

(click)="handleClearCondition($event, 'hasDtcCode')">

<mat-icon>close</mat-icon>

</button>

</mat-form-field>

</div>

<div class="header-button-wrapper" style="flex-shrink: 0; margin-left: 12px">

<button color="primary" mat-raised-button \*ngIf="!getDtcExportDisabled()" [disabled]="exportDtcLoading"

[matMenuTriggerFor]="menu1" style="margin-right: 8px;">

<div style="display: flex; align-items: center;">

DTC导出 <mat-spinner \*ngIf="exportDtcLoading"

style="width: 14px; height: 14px; margin: auto; margin-left: 8px;">

</mat-spinner>

</div>

</button>

<mat-menu #menu1="matMenu">

<button \*ngFor="let item of dtcExportOptions" mat-menu-item

(click)="handleDownloadDtcPdf(item.value)">

<span>{{item.label}}</span>

</button>

</mat-menu>

<button color="primary" mat-raised-button

[disabled]="getExportCsvDisabled() || (dataSource.isEmpty() | async)" (click)="handleDownloadCsv()"

style="margin-right: 8px;">

<div style="display: flex; align-items: center;">

记录导出 <mat-spinner \*ngIf="exportCsvLoading"

style="width: 14px; height: 14px; margin: auto; margin-left: 8px;">

</mat-spinner>

</div>

</button>

<button color="primary" mat-raised-button [disabled]=" deleteLoading" (click)="onOkClick()">

<div style="display: flex; align-items: center;">

查询 <mat-spinner \*ngIf="loading" style="width: 14px; height: 14px; margin: auto; margin-left: 8px;">

</mat-spinner>

</div>

</button>

</div>

</mat-toolbar>

<mat-toolbar style="padding-top: 0;">

<div class="option-wrapper">

<div style=" display: flex;

align-items: center;">

<section>

<mat-form-field class="base-filed-wrapper" subscriptSizing="dynamic">

<mat-label translate>检测开始时间</mat-label>

<span style="display: inline-flex;">

<input [(ngModel)]="checkStartTime" matInput [owlDateTime]="dt3">

<mat-icon style=" overflow: visible;" openOnFocus="true"

[owlDateTimeTrigger]="dt3">access\_time</mat-icon>

<owl-date-time #dt3 [showSecondsTimer]="true"></owl-date-time>

</span>

</mat-form-field>

</section>

<span style=" display: inline-block;

width: 8px;

height: 1px;

border-top: 1px solid;

margin: 4px;"></span>

<section>

<mat-form-field class="base-filed-wrapper" subscriptSizing="dynamic">

<mat-label translate>检测结束时间</mat-label>

<span style="display: inline-flex;">

<input [(ngModel)]="checkEndTime" matInput [owlDateTime]="dt4">

<mat-icon style=" overflow: visible;" openOnFocus="true"

[owlDateTimeTrigger]="dt4">access\_time</mat-icon>

<owl-date-time #dt4 [showSecondsTimer]="true"></owl-date-time>

</span>

</mat-form-field>

</section>

</div>

</div>

</mat-toolbar>

<div class="tb-content-wrapper ">

<mat-table #table [dataSource]="dataSource" matSort [matSortActive]="pageLink.sortOrder.property"

[matSortDirection]="pageLink.sortDirection()">

<ng-container matColumnDef="select" sticky>

<mat-header-cell \*matHeaderCellDef style="width: 30px;">

<mat-checkbox [disabled]="getMasterDisabled()" (change)="handleCheckMasterBox($event)"

[checked]="getMasterCheckBox()" [indeterminate]="getMasterIndeterminateBox()">

</mat-checkbox>

</mat-header-cell>

<mat-cell \*matCellDef="let relation">

<mat-checkbox [disabled]="!relation?.eolDtcCode" (change)="handleChangeItemBox($event, relation)"

[checked]="handleCheckSelect(relation)">

</mat-checkbox>

</mat-cell>

</ng-container>

<ng-container matColumnDef="position">

<th mat-header-cell \*matHeaderCellDef> No. </th>

<td mat-cell \*matCellDef="let element ;let i = index;"> {{ i + 1 }} </td>

</ng-container>

<ng-container \*ngFor="let item of columnOptions" [matColumnDef]="item.key">

<div \*ngIf="['deviceId'].includes(item.key)" [style.width.px]="item?.width">

<th mat-header-cell \*matHeaderCellDef>

<div class="line-is" [style.width.px]="item?.width"> {{ item.label}} </div>

</th>

<td mat-cell \*matCellDef="let element ;" [style.width.px]="item?.width">

<div [style.width.px]="item?.width" style=" word-wrap: break-word;"> {{ getRowValue(element,

item.key) }}</div>

</td>

</div>

<div \*ngIf="['eolDtcCode'].includes(item.key)" [style.width.px]="item?.width">

<th mat-header-cell \*matHeaderCellDef>

<div class="line-is" [style.width.px]="item?.width"> {{ item.label}} </div>

</th>

<td mat-cell \*matCellDef="let element ;" [class]="element?.eolDtcCode ? 'is-green' : ''"

[style.width.px]="item?.width">

<div (click)="handleShowDtcCode(element)"> {{ getRowValue(element, item.key) }}</div>

</td>

</div>

<th mat-header-cell \*matHeaderCellDef [style.width.px]="item?.width">

{{ item.label}}

</th>

<td mat-cell \*matCellDef="let element ;" [style.width.px]="item?.width"> {{ getRowValue(element, item.key)

}} </td>

</ng-container>

<ng-container matColumnDef="action" stickyEnd>

<mat-header-cell \*matHeaderCellDef> 操作 </mat-header-cell>

<mat-cell \*matCellDef="let element ;let i = index;">

<div style="display: flex; align-items: center; " class="row-wrapper">

<ng-container>

<button [disabled]="getExportDisabled(element, i)" [matMenuTriggerFor]="menu"

style="margin-left: 4px;" mat-raised-button>

<div style="display: flex; align-items: center;">

{{downLoading && currentIndex == i ? '' : 'common.export' | translate}} <mat-spinner

\*ngIf="downLoading && currentIndex == i "

style="width: 14px; height: 14px; margin: auto; margin-left: 8px;">

</mat-spinner>

</div>

</button>

<mat-menu #menu="matMenu">

<button \*ngFor="let item of fileOptions" mat-menu-item

(click)="handleDownloadFileByType(element, item.value, i)">

<span>{{item.label}}</span>

</button>

</mat-menu>

</ng-container>

<button mat-icon-button

style="margin-left: 12px; box-shadow: var(--mdc-protected-button-container-elevation, 0px 3px 1px -2px rgba(0, 0, 0, 0.2), 0px 2px 2px 0px rgba(0, 0, 0, 0.14), 0px 1px 5px 0px rgba(0, 0, 0, 0.12));"

[disabled]="getDeleteDisabled(element, i)" matTooltip="逻辑删除"

(click)="handleDeleteLoginByTestId(element.testId)"><mat-icon>delete</mat-icon></button>

<button mat-icon-button

style="margin-left: 12px; background: #dbdbdb; box-shadow: var(--mdc-protected-button-container-elevation, 0px 3px 1px -2px rgba(0, 0, 0, 0.2), 0px 2px 2px 0px rgba(0, 0, 0, 0.14), 0px 1px 5px 0px rgba(0, 0, 0, 0.12));"

matTooltip="物理删除" [disabled]="getDeleteDisabled(element, i)"

(click)="handleDeletePhyByTestId(element.testId)"><mat-icon>delete</mat-icon></button>

</div>

</mat-cell>

</ng-container>

<tr mat-header-row \*matHeaderRowDef="this.getDisplayedColumns(); sticky: true"></tr>

<tr mat-row \*matRowDef="let row; columns: this.getDisplayedColumns();"></tr>

</mat-table>

<span [fxShow]="dataSource?.dataLoading" fxLayoutAlign="center center" class="no-data-found"

translate>正在加载中...</span>

</div>

<mat-paginator [length]="dataSource.total() | async" [disabled]="dataSource?.dataLoading"

[pageIndex]="pageLink.page" [pageSize]="pageLink.pageSize" [pageSizeOptions]="[10 , 30 , 50 ]"

showFirstLastButtons></mat-paginator>

</div>

</mat-drawer-content>

</mat-drawer-container>

@Component({

selector: 'tb-eol-page',

templateUrl: './eol-page.component.html',

styleUrls: ['./eol-page.component.scss']

})

export class EolPageComponent {

@ViewChild(MatPaginator) paginator: MatPaginator;

@ViewChild(MatSort) sort: MatSort;

@ViewChild('docapInput') docapInput!: ElementRef<HTMLInputElement> & { listOptions: string[], inputValue: string };

deviceId = ''

deviceName = ''

deviceLists = []

loading = false;

dataSource: DatasourceTable;

pageLink: PageLink;

eolType

startTime

endTime

checkStartTime

checkEndTime

deleteTime

eolTypeOptions = eolTestOptions

statusOptions = statusOptions

dtcOptions = dtcOptions

columnOptions = columnConfig

deviceMap = new Map();

vinCode = null

status: StatusType

deviceIdsMap = new Map();

activeIdsMap = new Map();

fileOptions = [{ label: FileTypeEnum.BLF, value: FileTypeEnum.BLF }, { label: FileTypeEnum.MDF, value: FileTypeEnum.MF4 }, { label: FileTypeEnum.ASC, value: FileTypeEnum.ASC }];

dtcExportOptions = [{ label: OrdinaryFileTypeEnum.PDF, value: OrdinaryFileTypeEnum.PDF }, { label: OrdinaryFileTypeEnum.Excel, value: OrdinaryFileTypeEnum.Excel }];

exportLoading = false;

exportCsvLoading = false;

downLoading = false;

deleteLoading = false;

currentIndex = null;

isTenantAdmin = false

@ViewChild('deviceInput') deviceInput: ElementRef<HTMLInputElement>;

deviceControl = new FormControl('');

filteredOptions: any[];

debug = false;

isDetailsOpen = false

dtcConfig = null

hasDtcCode = false

exportDtcLoading = false;

filter(): void {

const filterValue = this.deviceInput.nativeElement.value.toLowerCase();

this.filteredOptions = this.deviceLists.filter(o => o.name.toLowerCase().includes(filterValue));

}

displayFn = (item) => item ? item?.name : ''

constructor(

private deviceService: DeviceService,

private deviceSettingService: DeviceSettingService,

public dialog: MatDialog,

public dialogService: DialogService,

private \_snackBar: MatSnackBar,

private store: Store<AppState>,

private route: ActivatedRoute

) {

this.dataSource = new DatasourceTable();

const sortOrder: SortOrder = { property: 'startTime', direction: Direction.DESC };

this.pageLink = new PageLink(100, 0, null, sortOrder);

this.debug = this.route.snapshot.routeConfig.path.includes('debug')

this.store.pipe(select(selectIsAuthenticated)).subscribe(

async (authenticated: boolean) => {

if (authenticated) {

this.store.pipe(select(selectAuth), take(1)).subscribe(

async (authState: AuthState) => {

if (authState.authUser) {

this.isTenantAdmin = authState.authUser.authority === Authority.TENANT\_ADMIN

}

}

);

}

}

);

}

ngOnInit() { this.getdeviceLists() }

ngAfterViewInit() {

merge(this.paginator.page, this.sort.sortChange)

.pipe(

tap(() => {

this.getTableData()

})

)

.subscribe();

}

getdeviceLists = () => {

this.deviceService

.getAllDeviceList()

.pipe(

catchError((error: HttpErrorResponse) => {

return handleError(error)

})

)

.subscribe((response: Array<any>) => {

this.deviceLists = (response || [])?.map(item => ({ ...item, value: item?.id?.id }))

this.deviceMap = getTargetValueMap(response, 'name')

this.deviceIdsMap = getTargetValueMap(response, 'id.id')

this.getTableData()

this.filteredOptions = this.deviceLists.slice();

});

this.getActiveLists()

}

getActiveLists = () => {

this.deviceService

.getDeviceInfosActiveAll()

.pipe(

catchError((error: HttpErrorResponse) => {

return handleError(error)

})

)

.subscribe((response: Array<any>) => {

this.activeIdsMap = getTargetValueMap(response, 'id.id')

console.log(this.activeIdsMap)

});

}

getDisplayedColumns = () => ['select', 'position'].concat([...columnConfig.map(item => item.key), 'action']);

getRowValue = (item, key) => {

if (key === 'status') return StatusNameType[item?.status];

if (key === 'testType') return EolNameType[item?.testType];

if (key === 'deviceId') {

if (!this.deviceIdsMap?.get(item?.deviceId)) return '';

let deviceName = this.deviceIdsMap?.get(item?.deviceId)?.name

if (this.activeIdsMap?.get(item?.deviceId)?.active) {

deviceName += '(在线)'

} else {

deviceName += '(离线)'

}

return deviceName;

}

if (['startTime', 'endTime', 'uploadEndTime', 'checkStartTime', 'checkEndTime','deleteTime'].includes(key)) {

const labelKey = Reflect.has(triggerTransformConfig, key) ? get(item, triggerTransformConfig[key]) : ''

const value = labelKey ? TriggerEnum[labelKey] : ''

const label = value ? `(${value})` : ''

return item?.[key] > 0 ? `${formatDate(item?.[key], false)}${label}` : '--'

}

if (key === 'eolDtcCode') {

return item?.[key] ? '有' : '无'

}

return item?.[key]

};

onOkClick = () => {

this.paginator.pageIndex = 0;

this.paginator.pageSize = 100;

if (this.startTime && this.endTime && this.startTime.getTime() > this.endTime.getTime()) {

this.showSnackBarMessage('结束日期要大于开始日期', 'error')

return;

}

//

this.getTableData();

}

handleClearDevice = $event => {

$event.stopPropagation();

this.deviceControl.setValue(null)

this.deviceId = ''

this.deviceName = ''

this.docapInput.listOptions = [];

this.vinCode = null;

this.docapInput.inputValue = null;

}

// 设备切换

handleSeleectDevice = async ($event) => {

const deviceName = (this.deviceControl.value as any)?.name

this.deviceId = this.deviceMap.get(deviceName)?.id?.id;

this.deviceName = deviceName

this.vinCode = null

this.docapInput.listOptions = [];

this.docapInput.inputValue = null

}

handleConditionSelectVariable = (item) => {

this.vinCode = item.label;

}

getAutoCompleteList = async (params) => {

const res = await this.deviceSettingService.findAllEolVin(omit({ ...params, deviceId: this.deviceId }, ['sortOrder'])).toPromise();

const { data = [] } = res || {}

return data.length > 0 ? {

...res, data: compact(data).map(label => ({ value: label, label }))

} : res

}

getPageParams = () => {

return {

deviceId: this.deviceId,

vin: this.vinCode || null,

testType: this.eolType,

startTime: expandTsTo16digits(this.startTime?.getTime()),

endTime: expandTsTo16digits(this.endTime?.getTime()),

checkStartTime: expandTsTo16digits(this.checkStartTime?.getTime()),

checkEndTime: expandTsTo16digits(this.checkEndTime?.getTime()),

status: this.status,

deleteTime: expandTsTo16digits(this.deleteTime?.getTime()),

isDtcCode: this.hasDtcCode

}

}

getTableData = async () => {

//重新刷新激活列表

this.getActiveLists()

const params = this.getPageParams();

this.pageLink.page = this.paginator.pageIndex;

this.pageLink.pageSize = this.paginator.pageSize;

this.pageLink.sortOrder.property = this.sort.active;

this.pageLink.sortOrder.direction = Direction[this.sort.direction.toUpperCase()];

this.dataSource.loadEntityMeasurements(this.deviceSettingService.getEolTableData(this.pageLink, params, this.debug))

}

handleDownloadCsv = async () => {

this.exportCsvLoading = true;

const params = this.getPageParams();

const response = await this.deviceSettingService.downloadEolCsv(params).toPromise();

this.exportCsvLoading = false;

if (response?.data?.fileName) {

this.deviceService.download(response.data.fileName)

}

}

handleDownloadFileByType = async (item, type, index) => {

const { testId } = item

this.currentIndex = index;

this.exportLoading = true;

const response = await this.deviceSettingService.getEolVinJobId(testId, type).toPromise();

this.exportLoading = false;

if (Object.prototype.toString.call(response) === '[object Object]' && response?.id) {

this.downLoading = true;

this.checkFileLoad(response?.id)

}

}

checkFileLoad = (id) => {

let timer = setInterval(async () => {

const func = () => {

this.deviceSettingService.findJobStatusById(id).pipe(

catchError((error: HttpErrorResponse) => {

clearInterval(timer);

this.downLoading = false

this.store.dispatch(new ActionLoadFinish());

this.currentIndex = null;

return throwError('Something went wrong, please try again later.');

})

).subscribe((res: any) => {

this.store.dispatch(new ActionLoadFinish());

if (res.status === 'SUCCEEDED') {

this.store.dispatch(new ActionLoadFinish());

clearInterval(timer);

this.deviceSettingService.findJobFilesByStatus(id).pipe(

catchError((error: HttpErrorResponse) => {

this.downLoading = false

this.currentIndex = null;

this.store.dispatch(new ActionLoadFinish());

return throwError('Something went wrong, please try again later.');

})

).subscribe((res: any) => {

this.store.dispatch(new ActionLoadFinish());

const { fileName } = res || {}

this.downLoading = false

this.currentIndex = null;

this.deviceService.download(fileName)

})

} else {

this.store.dispatch(new ActionLoadFinish());

}

})

}

func()

}, 5000)

}

getExportCsvDisabled = () => {

return this.exportCsvLoading

}

getExportDisabled = (item, index) => {

const { status, startTime, endTime } = item

return this.exportLoading || this.downLoading || startTime <= 1 || endTime <= 1 || this.deleteLoading

}

handleDeleteLoginByTestId = testId => {

this.dialogService.confirm('确定删除么？', '该操作会删除任务数据保留原始数据！', '取消', '确认').subscribe((res) => {

if (res) {

this.deleteLoading = true;

this.deviceSettingService.deleteVinLogic(testId).pipe(catchError((error: HttpErrorResponse) => {

this.deleteLoading = false

return throwError('Something went wrong, please try again later.');

})).subscribe((response) => {

this.deleteLoading = false;

if (typeof response === 'string') {

this.onOkClick()

}

const isSuccess = typeof response === 'string';

this.showSnackBarMessage(typeof response === 'string' ? '删除成功！' : '删除失败！', isSuccess ? 'success' : 'error')

});

}

})

}

showSnackBarMessage = (message, type?: NotificationType) => {

const data: ToastPanelData = {

notification: {

message, type: type ?? 'success', horizontalPosition: 'center', verticalPosition: 'top',

},

parent: { nativeElement: document.body },

panelClass: [], destroyToastComponent: () => { }

};

const config: any = {

horizontalPosition: 'center', verticalPosition: 'top', data, duration: 1000

};

this.\_snackBar.openFromComponent(

TbSnackBarComponent, config

)

}

handleDeletePhyByTestId = (testId) => {

this.dialogService.confirm('确定删除么？', '该操作会将任务数据以及原始数据同步删除', '取消', '确认').subscribe((res) => {

if (res) {

this.deleteLoading = true;

this.deviceSettingService.deleteVinPhy(testId).pipe(catchError((error: HttpErrorResponse) => {

this.deleteLoading = false

return throwError('Something went wrong, please try again later.');

})).subscribe((response) => {

this.deleteLoading = false;

if (typeof response === 'string') {

this.onOkClick()

}

const isSuccess = typeof response === 'string';

this.showSnackBarMessage(typeof response === 'string' ? '删除成功！' : '删除失败！', isSuccess ? 'success' : 'error')

});

}

})

}

getDeleteDisabled = (item, index) => {

if (!this.isTenantAdmin) return true

return this.deleteLoading || !item.testId || this.exportLoading || (this.downLoading && this.currentIndex == index)

};

handleClearCondition = ($event, key) => {

$event.stopPropagation();

if (this.hasOwnProperty(key)) {

this[key] = null;

}

}

handleShowDtcCode = (row) => {

const { eolDtcCode } = row

if (!eolDtcCode) return

let config = null

try {

config = JSON.parse(eolDtcCode)

} catch (e) {

}

if (!config) return

this.isDetailsOpen = true;

this.dtcConfig = Object.assign({}, row, { eolDtcCode: config });

}

closeDtcCode = () => {

this.isDetailsOpen = false;

this.dtcConfig = null;

}

handleCheckSelect(item: any): boolean {

const list = this.dataSource.selection.selected.map((vt: any) => get(vt, 'id.id'))

return list.includes(get(item, 'id.id'));

}

getCheckPageData = () => {

// 勾选只针对有DTC 的数据

const currentPageAllData = this.dataSource.pageDataSubject.value.data.filter(item => item.eolDtcCode);

const selectAllList = this.dataSource.selection.selected

const selectAllIds = selectAllList.map(item => get(item, 'id.id'))

const currentPageAllIds = currentPageAllData.map(item => get(item, 'id.id'))

const sMap = new Set(selectAllIds);

const intersect = currentPageAllIds.filter(name => sMap.has(name))

// 获取当前页面没有勾选的

const aIntersectData = currentPageAllData.filter(item => !sMap.has(get(item, 'id.id')))

const selectDataMap = new Map()

selectAllList.forEach(item => {

const id = get(item, 'id.id');

id && selectDataMap.set(id, item)

})

return { sIds: selectAllIds, caIds: currentPageAllIds, intersect, selectAllList, currentPageAllData, aIntersectData, selectDataMap };

}

handleChangeItemBox($event, relation: any) {

if (!$event) return

const { selectDataMap } = this.getCheckPageData();

const vk = selectDataMap.get(get(relation, 'id.id'))

if (vk) {

this.dataSource.selection.deselect(vk);

return;

}

this.dataSource.selection.select(relation)

}

getMasterCheckBox() {

const { intersect, caIds } = this.getCheckPageData();

return caIds.length === intersect.length && caIds.length > 0

}

getMasterIndeterminateBox() {

const { caIds, intersect } = this.getCheckPageData()

return intersect.length > 0 && caIds.length != intersect.length;

}

handleCheckMasterBox($event) {

if (!$event) return

const check = this.getMasterCheckBox();

const { currentPageAllData, aIntersectData, selectDataMap } = this.getCheckPageData()

if (check) {

currentPageAllData.forEach(item => {

const vk = selectDataMap.get(get(item, 'id.id'))

this.dataSource.selection.deselect(vk)

})

}

if (!check) {

aIntersectData.forEach(item => { this.dataSource.selection.select(item) })

}

}

getMasterDisabled = () => {

const { currentPageAllData } = this.getCheckPageData()

return currentPageAllData.length < 1

}

getDtcExportDisabled = () => {

const { sIds } = this.getCheckPageData()

return !sIds || sIds.length < 1

}

handleDownloadDtcPdf = async (type) => {

const { sIds } = this.getCheckPageData()

this.exportDtcLoading = true;

const response = await this.deviceSettingService.getEolDtcPdfPath(sIds.join(), type).toPromise().catch(() => {

this.exportDtcLoading = false;

});

this.exportDtcLoading = false;

if (response && response.success) {

const { data = {} } = response || {};

const { fileName, filePath } = data || {};

this.deviceService.download(fileName)

}

}

}