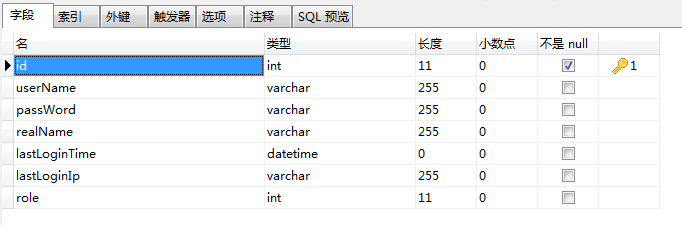
# 技术文档

## 1.1 多角色管理

角色表设计如下：



用户登录时，系统根据数据库中用户的角色role字段的值判断用户的级别，当role为1时，表示管理员，role为2时，表示一般人员，并分别赋予不同的操作权限。

## 1.2 船舶管理

本系统中船舶管理主要有以下几部分功能，船舶添加，船舶修改，船舶运动模型的加载，船舶轨迹回放。

船舶的类结构如下：

public int shipMMSI { get; set; } //船ID mmsi

public int shipID { get; set; } //船索引ID

public String shipName { get; set; } //船名称

public CShipType shipType { get; set; } //船类型

public double shipSpeed { get; set; } //航速knot

public double shipCourse { get; set; } //航向

public double shipHeading { get; set; } //航首向

public int shipLatitude { get; set; } //维度

public int shipLongitude { get; set; } //经度

public int shipDataFrom { get; set; } //船数据来源

public int shipMapID { get; set; } //船在海图上的id

public int shipMapPos { get; set; } //船在海图上的pos

public int shipLeftEngineSpeed { get; set; } //船左车钟档位

public int shipRightEngineSpeed { get; set; } //船右车钟档位

public int shipModel { get; set; } //模型ID

public float shipRudderAngle { get; set; } //舵角

public double shipRateOfTurn { get; set; } //转速率

public double shipDualDist { get; set; } //航程

public double shipResetDist { get; set; } //重置后的航程

public int shipUseOrNot { get; set; } //船舶是否可用

public M\_POINT[] shipPoint = new M\_POINT[5];

public bool ISStartTrackPlay = false; //是否开启船舶轨迹显示

[NonSerialized]

public List<int> shipScanByAis = null; //进去AIS区域 -1-无，0-x-进入radar编号

[NonSerialized]

public List<int> shipScanByRadar = null; //进入雷达区域 -1-无 0-x-进入ais编号

[NonSerialized]

public CShipTrack shipTrack = null; //船舶轨迹

[NonSerialized]

public CShipShowBasicShape shipBasicShape = null; //船舶显示区域

[NonSerialized]

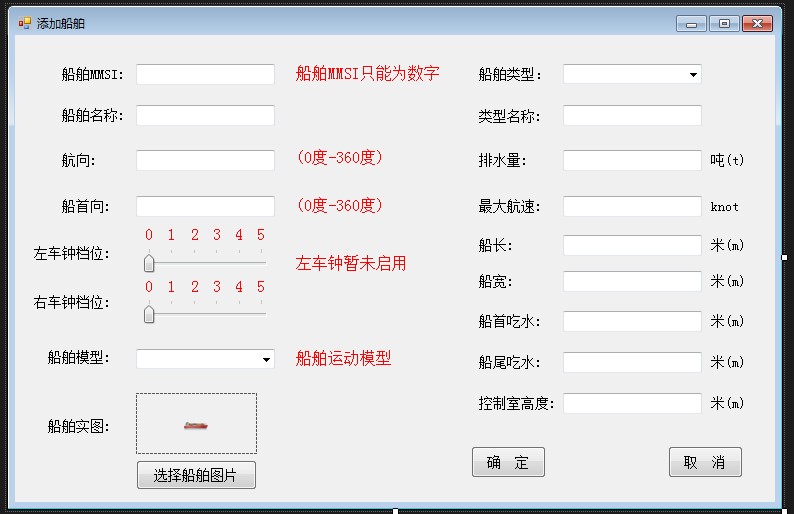
public CShipMotion shipMotion = null; //船舶模型

[NonSerialized]

public bool shipShowTrackOrNot = true; //显示或隐藏航迹

1.2.1 船舶添加与显示

添加船舶时需要填写的船舶信息如下：



填写完船舶的信息之后，初始化船舶对象，当在海图上点击鼠标左键时，根据获取到的数据屏幕坐标，使用函数GetGeoPoFromScrnPo(mMouseScrnPo.x, mMouseScrnPo.y, ref mMouseGeoPo.x, ref mMouseGeoPo.y);获取地理坐标信息，确定船舶的地理位置。在显示船舶的时候，先根据船舶的宽高信息，计算船舶的像素大小，船舶的中心点坐标和前后左右5个边缘点的坐标，绘制出船舶的轮廓信息。

1.2.3 船舶修改

当点击船舶修改时，将当前船舶数据回传到船舶添加窗口界面，并动态赋值，修改时，只需要根据需要修改上述船舶添加窗口的内容，并修改船舶对象信息即可完成船舶的修改工作。

1.2.4 船舶运动模型的加载

船舶的运动需要依靠船舶运动模型，根据不同类型的船和运行速度，不断计算出船的行驶距离，并更新船的实时位置，即可完成船舶的运动。本系统中每条船绑定一个船舶模型。

1.2.5 船舶轨迹回放

在船舶运动时，实时的保存船舶的运动节点到轨迹列表中，当需要轨迹回放时，循环遍历该轨迹列表，并根据列表数据实时更新船舶的位置，即可完成轨迹回放。

## 1.3 雷达管理

本系统中雷达管理主要有以下几部分功能，雷达添加，雷达修改，雷达与电子海图的叠加。

雷达的类结构如下：

public int radarID { get; set; } //雷达ID

public int radarMapID { get; set; } //雷达在海图中的ID

public String radarName { get; set; } //雷达名称

public double radarScanRadius { get; set; } //雷达扫描半径

public double radarPixelRadius { get; set; } //雷达基站像素半径

public int radarHeight { get; set; } //雷达高度

public double radarAngle { get; set; } //雷达安装角度

public int radarGeoPosX { get; set; } //雷达经纬度坐标X

public int radarGeoPosY { get; set; } //雷达经纬度坐标Y

public int radarMapPos { get; set; } //雷达海图索引

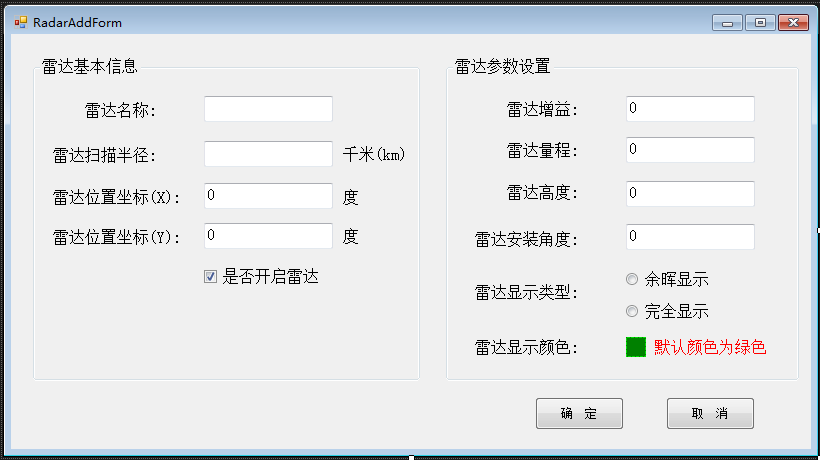
public int radarUseOrNot { get; set; } //雷达是否启用

public int radarShowType { get; set; } //0 完全显示 1余晖显示

public int radarShowColor { get; set; } //雷达显示颜色

1.3.1 雷达添加与显示

添加雷达时需要填写的雷达信息如下：



填写完雷达的信息之后，初始化雷达对象，当在海图上点击鼠标左键时，根据获取到的数据屏幕坐标，使用函数GetGeoPoFromScrnPo(mMouseScrnPo.x, mMouseScrnPo.y, ref mMouseGeoPo.x, ref mMouseGeoPo.y);获取地理坐标信息，确定雷达的地理位置。具体代码如下：

//添加雷达物标

this.ymEncCtrl.tmAppendObjectInLayer(0, (int)M\_GEO\_TYPE.TYPE\_POINT);

currentMouseRadar.radarMapPos = this.ymEncCtrl.tmGetLayerObjectCount(0) - 1;

this.ymEncCtrl.tmSetPointObjectCoor(0, currentMouseRadar.radarMapPos, currentMouseRadar.radarGeoPosX, currentMouseRadar.radarGeoPosY);

//设置雷达显示的标识

this.ymEncCtrl.tmSetPointObjectStyleRefLib(0, currentMouseRadar.radarMapPos, 57, false, 0, 1, 0);

//设置ym雷达对象

currentMouseRadar.radarMapID = this.ymEncCtrl.AddOneRadar();

//通过雷达ID获取雷达POS

int radarMapPos = ymEncCtrl.GetRadarPosById(currentMouseRadar.radarMapID);

1.3.3 雷达修改

当点击雷达修改时，将当前船舶数据回传到雷达添加窗口界面，并动态赋值，修改时，只需要根据需要修改上述雷达添加窗口的内容，并修改雷达对象信息即可完成雷达的修改工作。

1.3.4 雷达与海图的叠加显示

祥见接口文档，核心代码如下：

|  |
| --- |
| for (int i = 0; i < 2048; i++)  {  if (i < 512)  {  for (int j = 0; j < 512; j++)  {  //第一象限 x+ y-  tmp\_degree = 90.0 - ((i + 1) \* degree);  tmp\_r = package.radarPixelWidth \* (j +1);  x = tmp\_r \* Math.Cos(tmp\_degree \* Math.PI / 180); //求出来的x长度  y = tmp\_r \* Math.Sin(tmp\_degree \* Math.PI / 180); //求出来的y长度  color = radarBitmap.GetPixel(Convert.ToInt32(package.radarCenterX + x), Convert.ToInt32(package.radarCenterY - y));  if (j > 37)  {  if (color.R > color.G)  {  dataArray[j] = 0; //红色遮挡0  }  else  {  dataArray[j] = 1; //绿色无遮挡1  }  }  else  {  dataArray[j] = 0; //红色遮挡0  }  }  }  else if (i >= 512 && i < 1024)  {  for (int j = 0; j < 512; j++)  {  //第二象限 x+ y+  tmp\_degree = ((i + 1) \* degree) - 90.0;  tmp\_r = package.radarPixelWidth \* (j + 1);  x = tmp\_r \* Math.Cos(tmp\_degree \* Math.PI / 180); //求出来的x长度  y = tmp\_r \* Math.Sin(tmp\_degree \* Math.PI / 180); //求出来的y长度  color = radarBitmap.GetPixel(Convert.ToInt32(package.radarCenterX + x), Convert.ToInt32(package.radarCenterY + y));  if(j > 37)  {  if (color.R > color.G)  {  dataArray[j] = 0; //红色遮挡0  }  else  {  dataArray[j] = 1; //绿色无遮挡1  }  }  else  {  dataArray[j] = 0; //红色遮挡0  }  }  }  else if (i >= 1024 && i < 1536)  {  for (int j = 0; j < 512; j++)  {  //第三象限 x- y+  tmp\_degree = ((i + 1) \* degree) - 180.0;  tmp\_r = package.radarPixelWidth \* (j + 1);  x = tmp\_r \* Math.Cos(tmp\_degree \* Math.PI / 180); //求出来的x长度  y = tmp\_r \* Math.Sin(tmp\_degree \* Math.PI / 180); //求出来的y长度  color = radarBitmap.GetPixel(Convert.ToInt32(package.radarCenterX - x), Convert.ToInt32(package.radarCenterY + y));  if (j > 37)  {  if (color.R > color.G)  {  dataArray[j] = 0; //红色遮挡0  }  else  {  dataArray[j] = 1; //绿色无遮挡1  }  }  else  {  dataArray[j] = 0; //红色遮挡0  }  }  }  else if (i >= 1536)  {  for (int j = 0; j < 512; j++)  {  //第四象限 x- y-  tmp\_degree = ((i + 1) \* degree) - 270.0;  tmp\_r = package.radarPixelWidth \* (j + 1);  x = tmp\_r \* Math.Cos(tmp\_degree \* Math.PI / 180); //求出来的x长度  y = tmp\_r \* Math.Sin(tmp\_degree \* Math.PI / 180); //求出来的y长度  color = radarBitmap.GetPixel(Convert.ToInt32(package.radarCenterX - x), Convert.ToInt32(package.radarCenterY - y));  if (j > 37)  {  if (color.R > color.G)  {  dataArray[j] = 0; //红色遮挡0  }  else  {  dataArray[j] = 1; //绿色无遮挡1  }  }  else  {  dataArray[j] = 0; //红色遮挡0  }  }  }  String data = InteropEncDotNet.GetStringFromIntArray(dataArray, 512);  this.ymEncCtrl.SetRadarLineDataByPos(radarYMPos, i, ref data, 512);//设置雷达数据 |

## 1.4 AIS管理同雷达管理，不再赘述。

## 1.5海图操作相关功能，见意码接口文档。

## 1.6 数据传输相关功能

系统所有的数据交互采用UDP协议网络通信的形式传输，具体的传输以雷达数据为例：

1. 封装雷达对象，网络通信对象初始化
2. 雷达对象序列化
3. 雷达对象网络发送

接收端只需要根据同样雷达数据结构接收网络序列化雷达对象即可，然后进行雷达对象的反序列化，保存为本地雷达对象即完成雷达数据的传输。