DCWriter 五代编辑器用户手册

撰写者：李应祥

撰写时间：2024-06-15

一. 前言

二. DCWriter 入门与部署

用户集成 DCWriter 到电子病历系统之前，需要满足客户端硬性条件、服务器资源部署两个条件才能集成到电子病历系统中。

2.1.客户端硬性条件

DCWriter 是纯前端控件,能运行在大多数主流浏览器上,内核版本需要满足:
Chrome 内核：73 及以上才行
360 浏览器
火狐浏览器内核：78 及以上
Edge 浏览器内核：98 及以上
IE 浏览器不支持
WebView 内核：86 及以上
客户端推荐使用 64 位操作系统

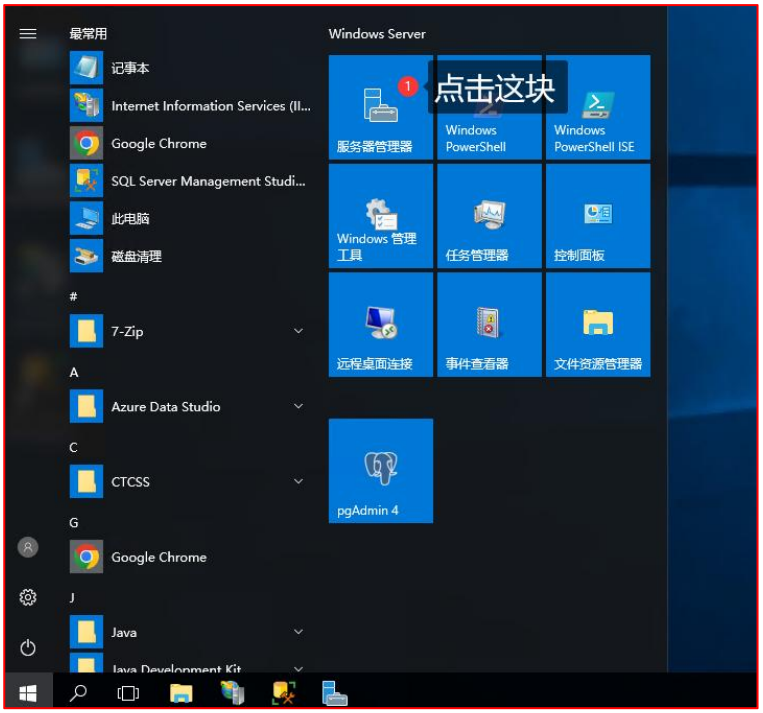
2.2.服务器资源搭建部署

DCWriter 支持 Windows-Linux 服务器搭建部署,符合信创要求,在银河麒麟、统信服务器已经认证成功。

Windows Server 2012 及以上部署步骤：

2.2.1. Internet Information Serices 环境搭建（简称 IIS）

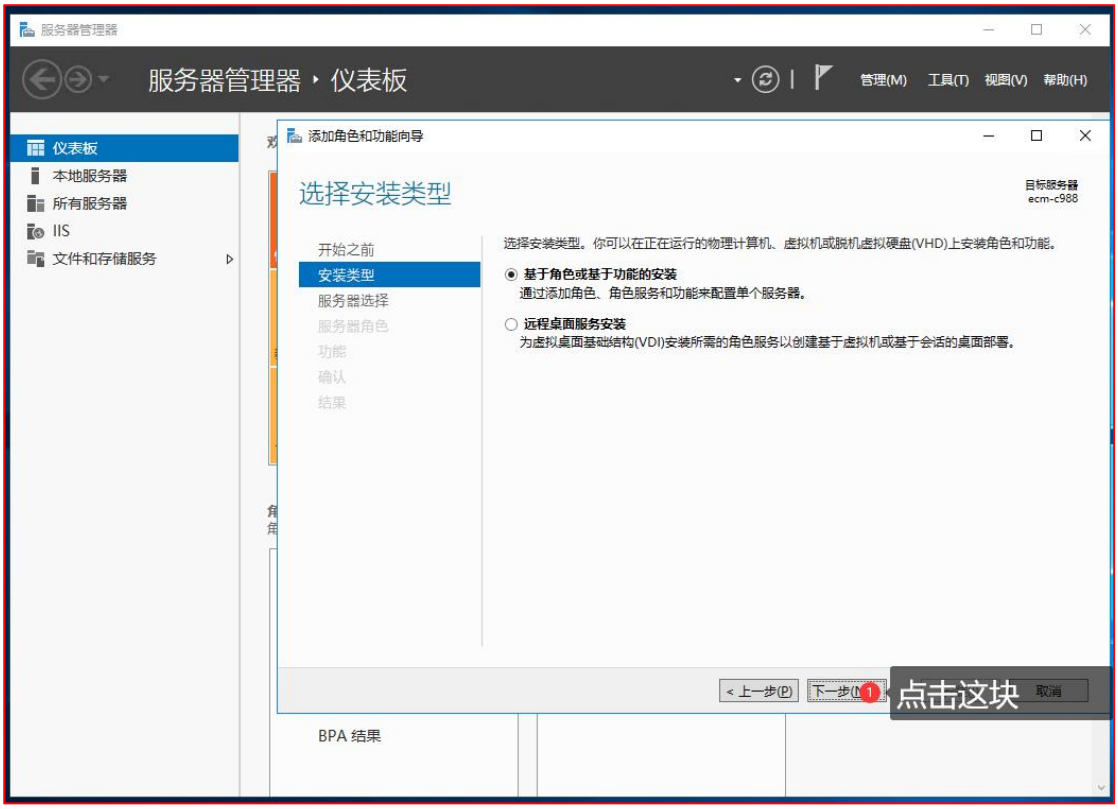
2.2.1.1.打开服务器管理器



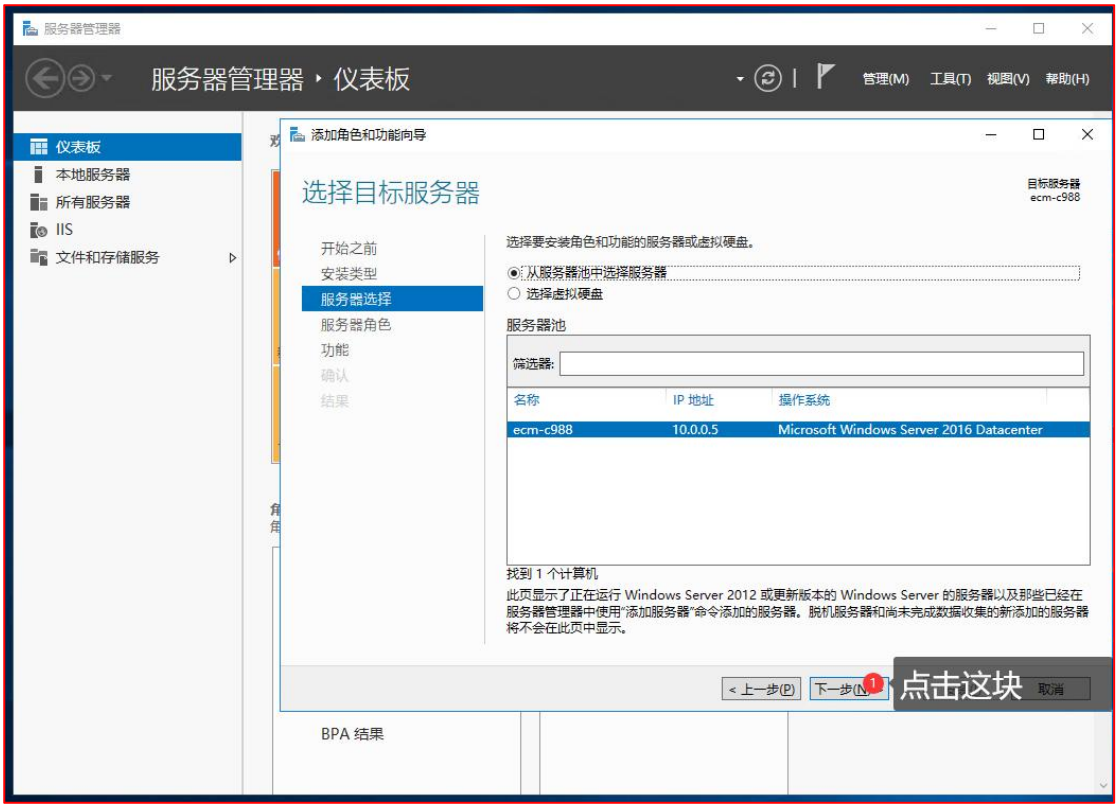
2.2.1.2.添加角色和功能

2.2.1.3.开始之前界面直接点击下一步

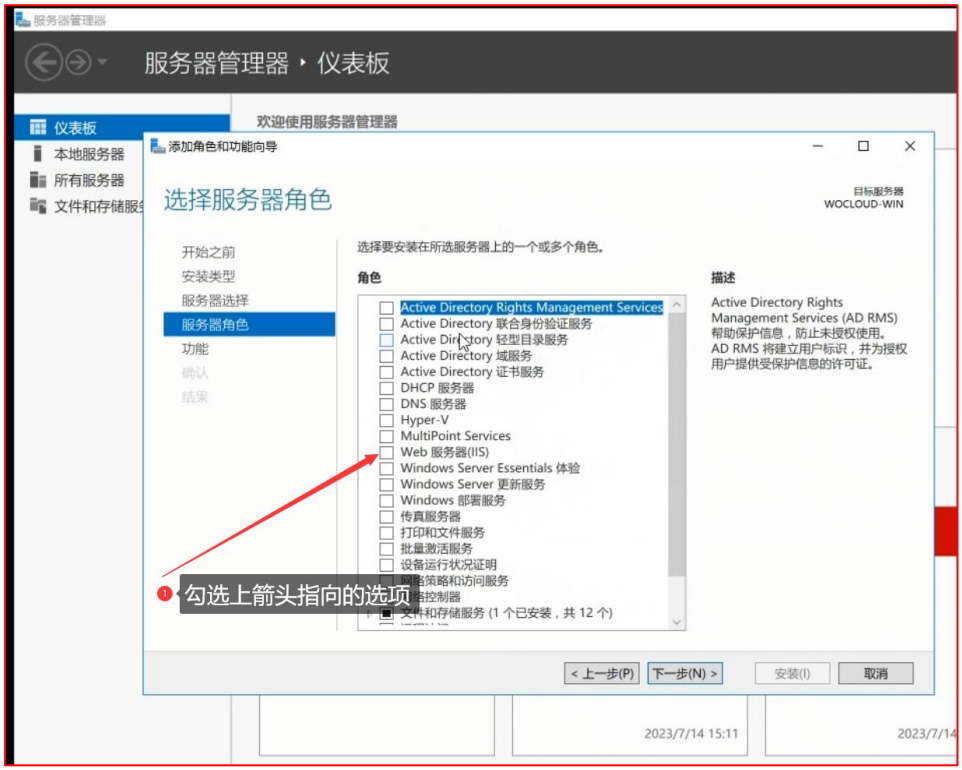
2.2.1.4.安装类型界面直接点击下一步



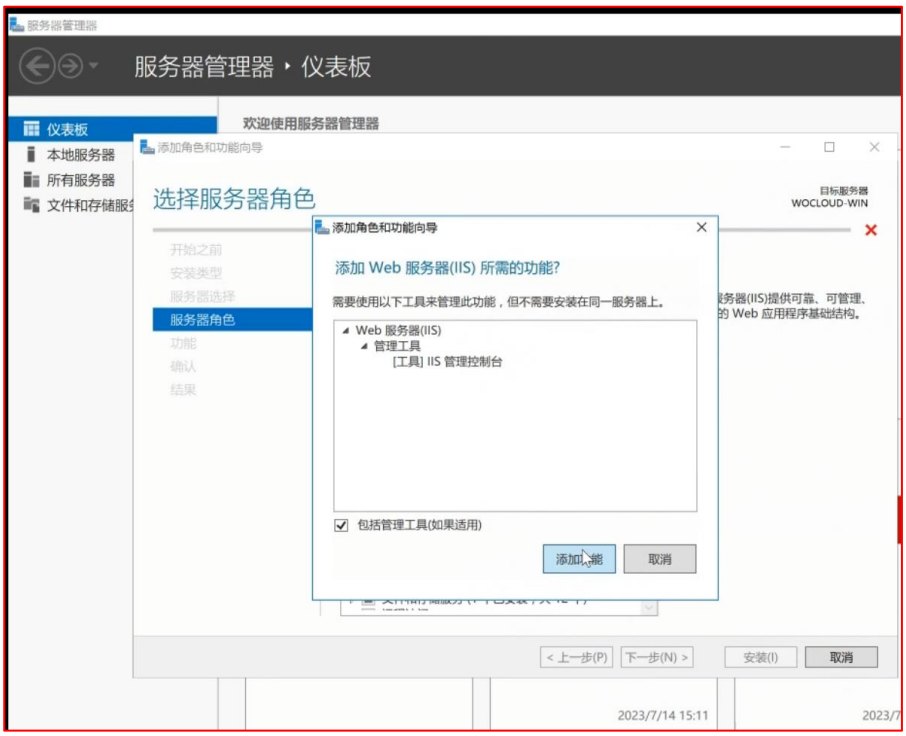
2.2.1.5.服务器选择界面直接点击下一步



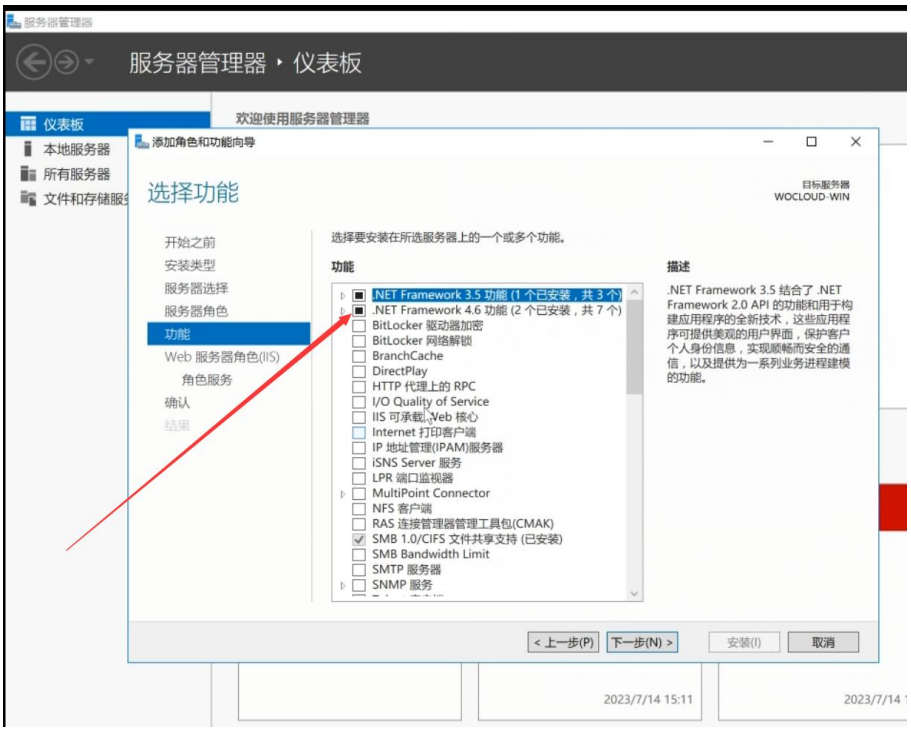
2.2.1.6.服务器角色



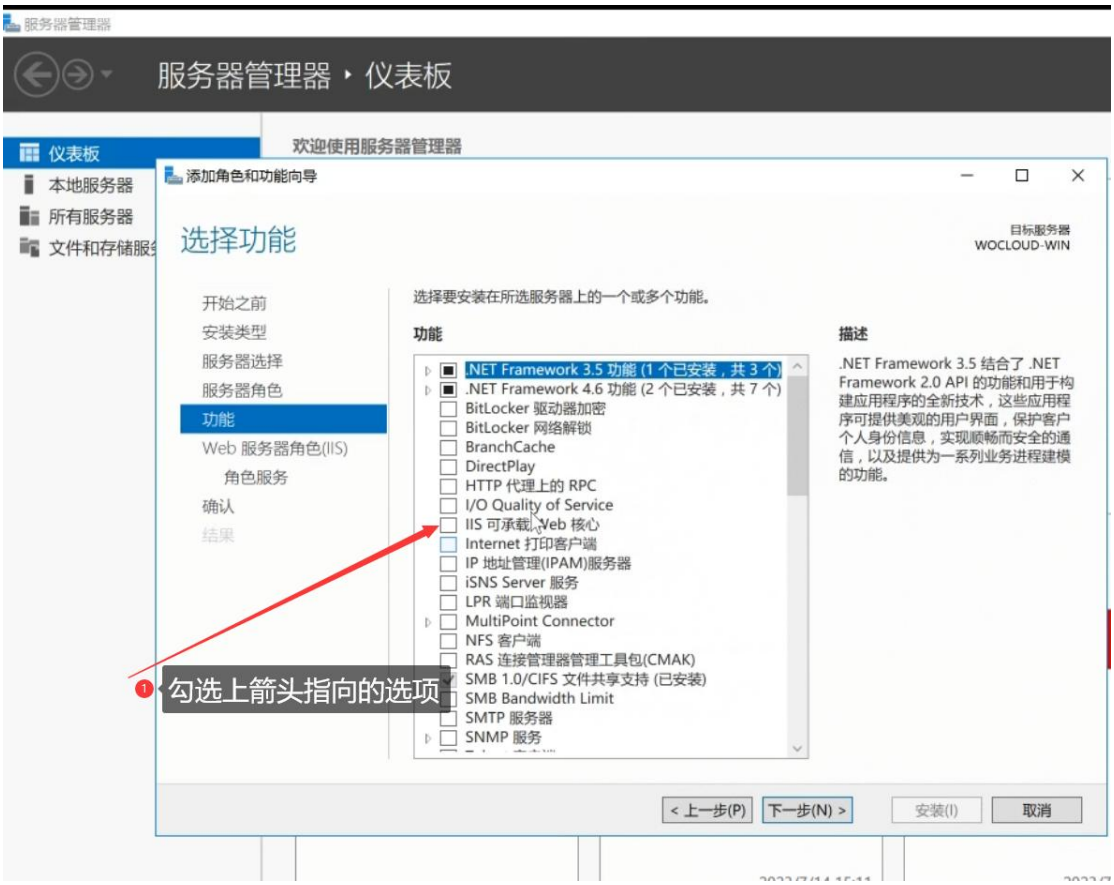
2.2.1.7.出现界面之后点击添加功能，然后点击下一步



2.2.1.8.功能界面，注意：.net framework 是否安装了 4.0 以上版本

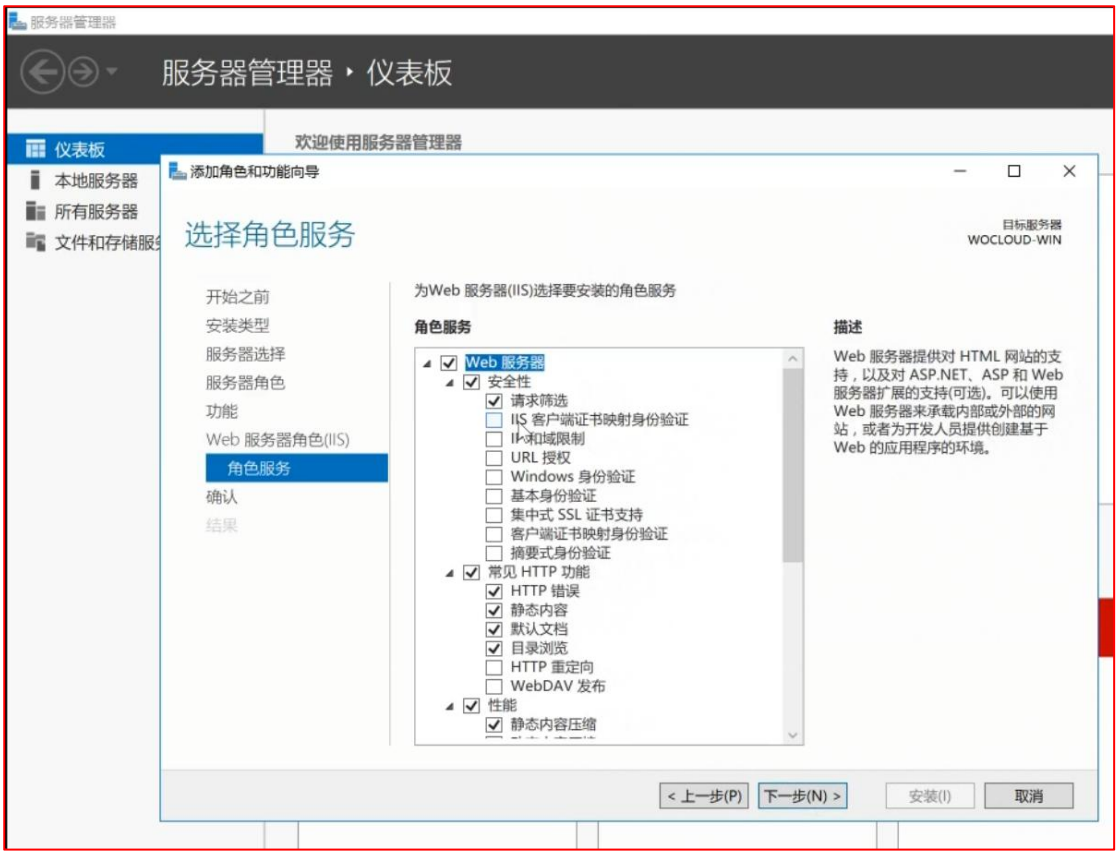


安装了.net framework4.0 以上的版本，继续向下操作，没有安装，先把这个.net framework 框架版本安装好，在重复上面的操作到这步，往下操作。

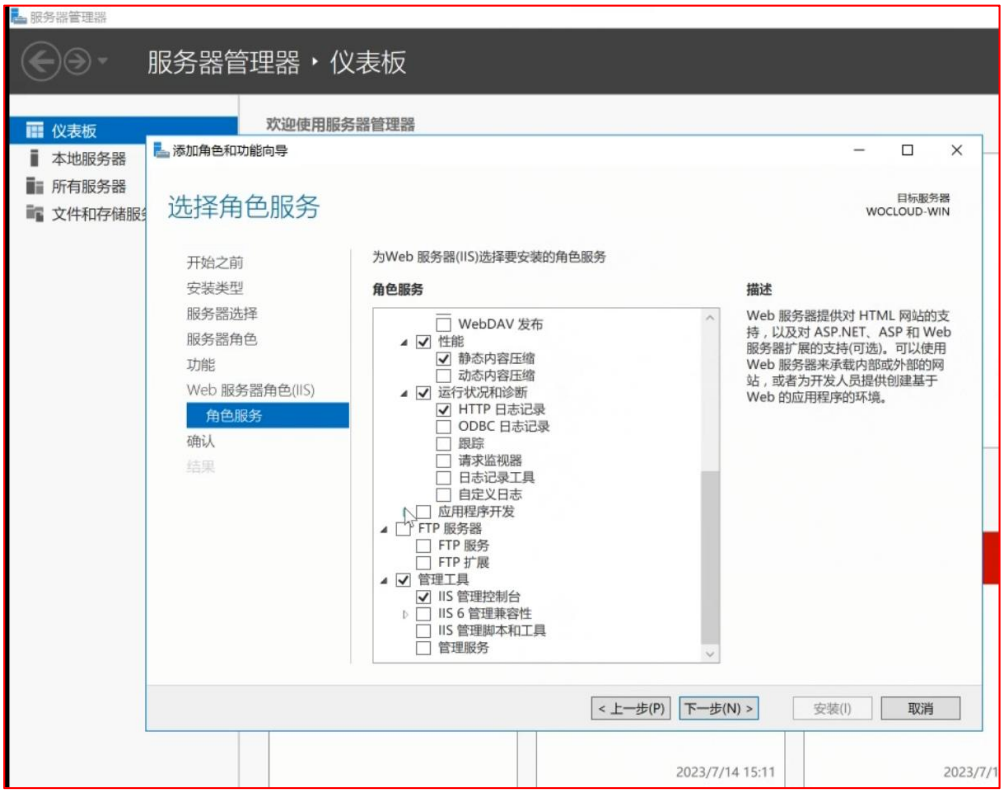


2.2.1.9. 功能界面点击下一步

2.2.1.10. 下面的界面只点击下一步到达角色服务界面



2.2.1.11.Web 服务器—>应用程序开发



检查应用程序开发功能中是否存在.net Extensibility 3.5、.net Extensibility 4.8 选项，存在把全部选项进行勾选，勾选完成之后，点击下一步。（2008 版本服务器，推荐使用 2.0 服务部署）

2.2.1.12.确定界面点击安装按钮进行安装

之后等安装完成即可，IIS 的环境配置成功。

2.2.2.搭建 DCWriter 网站

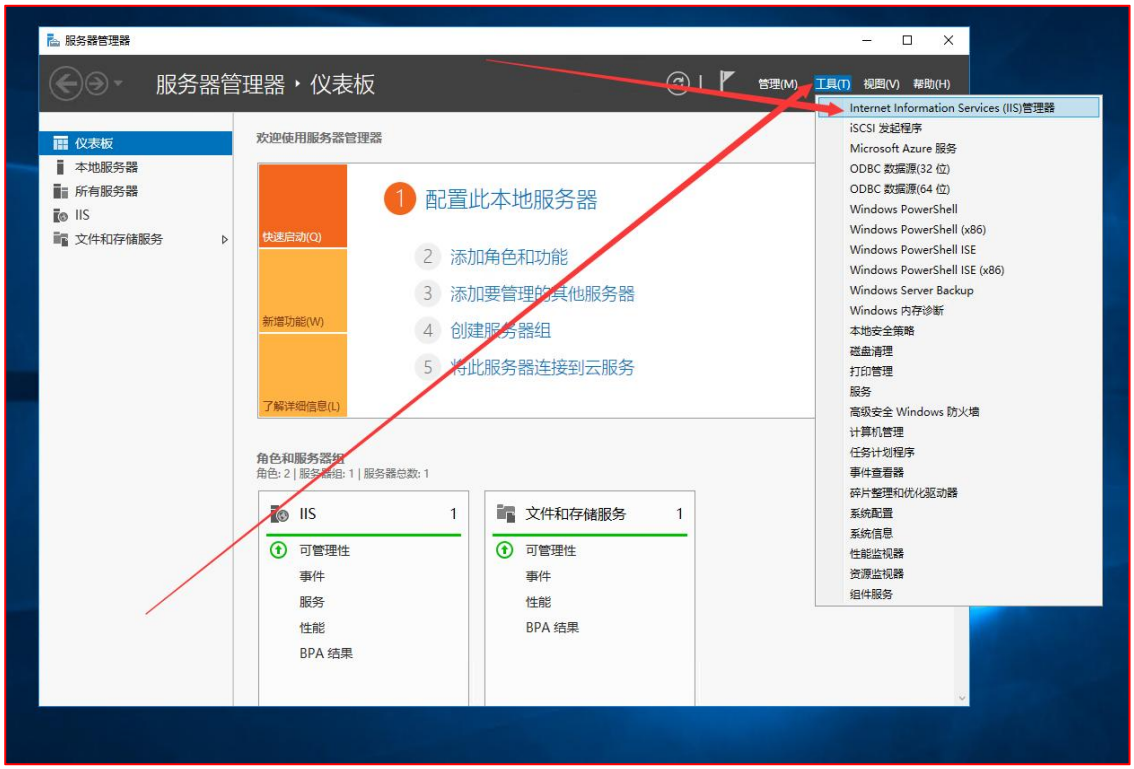
2.2.2.1.五代资源压缩包上传服务器

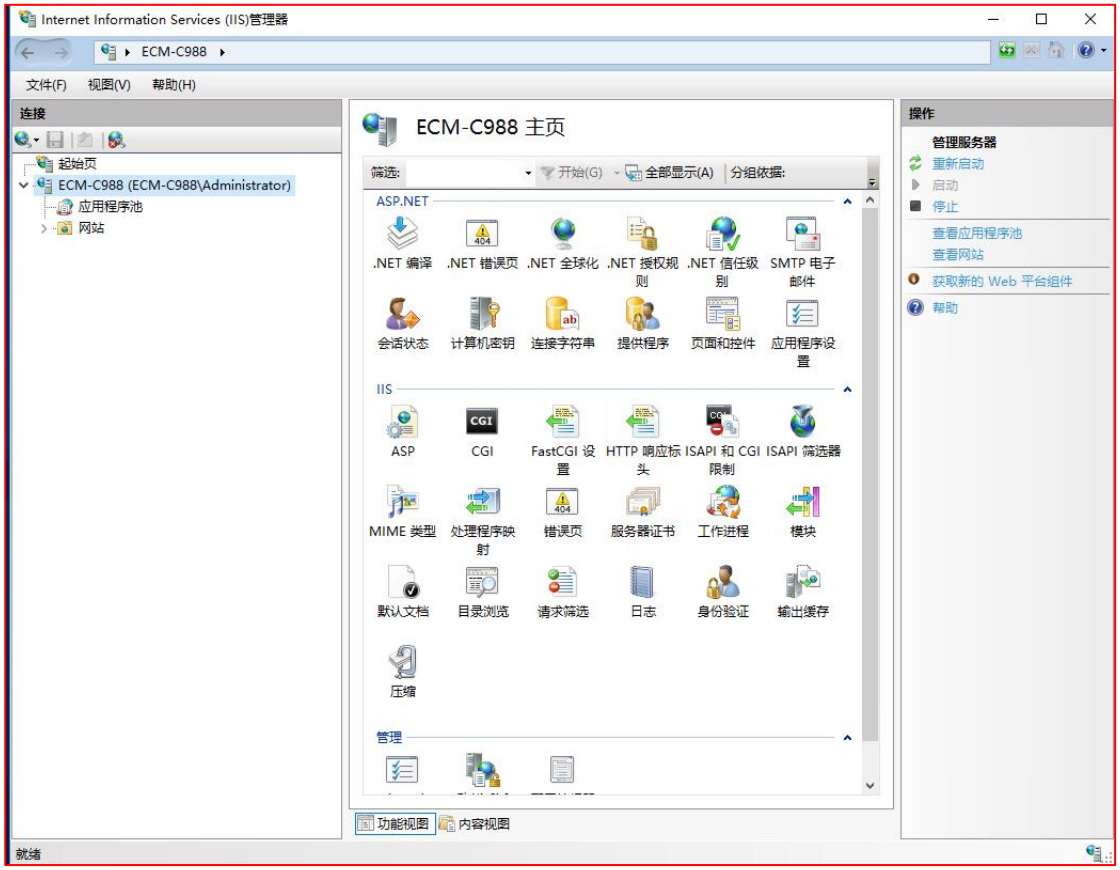
在**除 C 盘外**的盘根目录下创建文件夹（dcwriter），把五代资源压缩包进行上传，然后进行解压，DCWriter 默认实现 PDF 功能是需要进行服务器交互，当不需要服务器交互，纯前端实现 PDF 功能时，需要把字体压缩包（Fonts.zip）上传到五代资源包 dcwriter5files 文件夹里面在进行解压。

五代资源包解压效果:

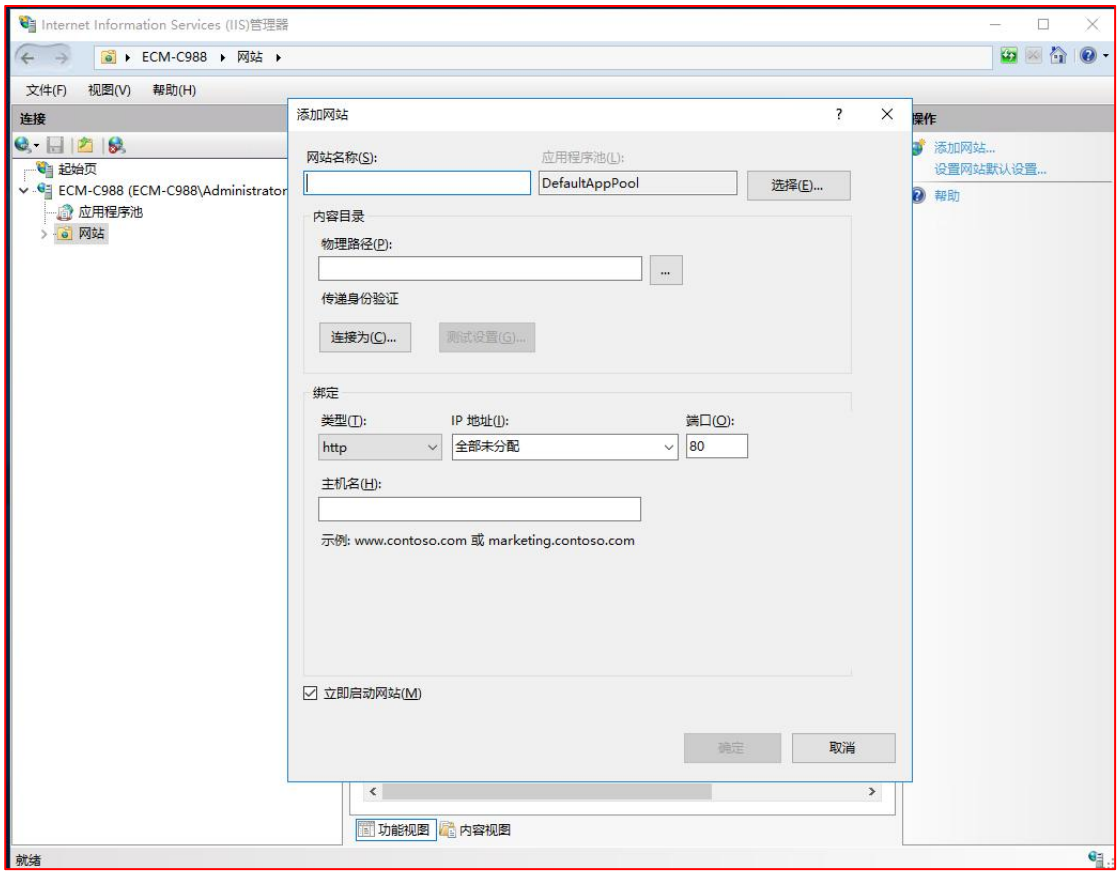
2.2.2.2.运行 IIS 管理器

在服务器管理器界面右上角工具下 IIS 管理器打开 IIS 管理器运行界面:

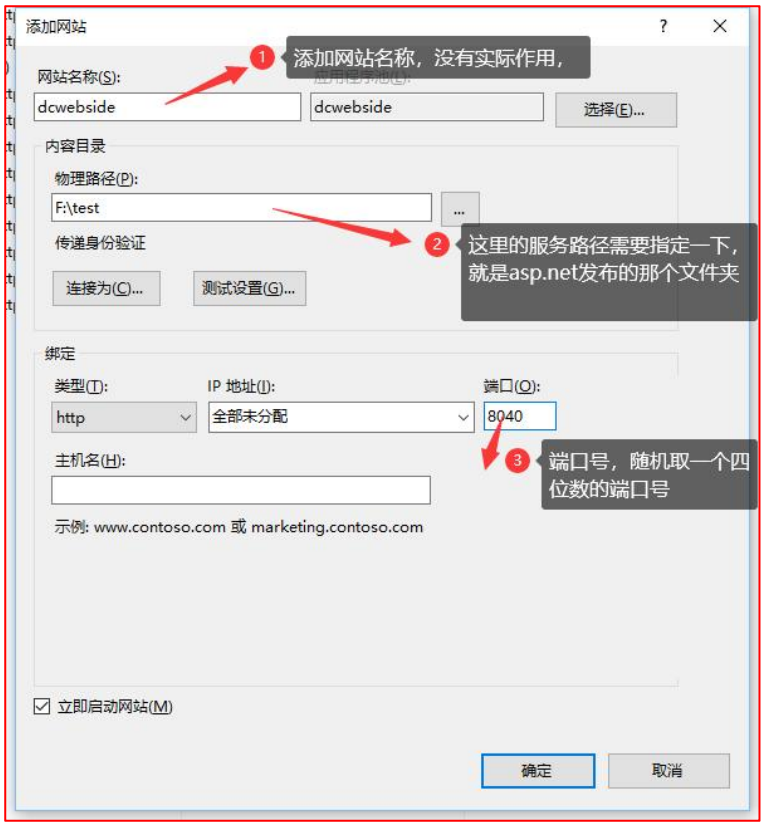




2.2.2.3. IIS 界面找到网站节点然后鼠标右键添加网站



2.2.2.4.配置网站名称、物理地址（服务上传的路径）、端口号



2.2.2.5.确定服务部署成功

通过服务器 IP:网站端口号/ServicePage.aspx?wasmres=dcwriter5.js。出现下图效果，即五代编辑器环境搭建成功。该浏览器访问地址就是五代编辑器需要引用的地址。

Linux 服务器部署步骤：

Linux 服务器支持两种部署方式：两种部署方式区别，.net core 部署能实现 DCWriter 下需要进行服务器交互功能，但是部署起来困难，Java 部署 DCWriter 不进行服务器交互，部署起来简单方便。

以 CentOS-7 服务器能连接外网-管理员账号为例进行解释部署步骤

2.2.3..net core 服务部署步骤：

2.2.3.1. .net core 3.1 环境安装

➢ 注册 Microsoft 密钥和源（执行一次）

sudo rpm -Uvh https://packages.microsoft.com/config/centos/7/packages-microsoft-prod.rpm

➢ 安装.NET Core SDK

yum install dotnet-sdk-3.1

- 安装 ASP.NET Core

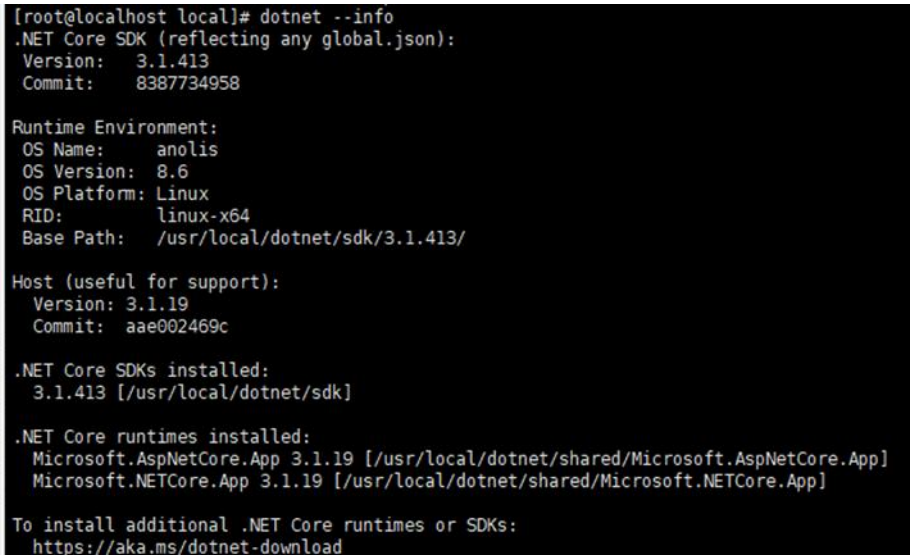
yum install aspnetcore-runtime-3.1

- 安装.NET Core

yum install dotnet-runtime-3.1

- 验证.net core 安装成功

dotnet --info



2.2.3.2.安装依赖包

yum -y install gcc gcc-c++ bison pkgconfig glib2-devel gettext make libpng-devel libjpeg-devel libtiff-devel libexif-devel giflib-devel libX11-devel freetype-devel fontconfig-devel cairo-devel

2.2.3.3.安装 libgdiplus

- 创建 libgdiplus 包存放文件夹

mkdir /usr/local/mono

- 进入到创建文件夹地址

cd /usr/local/mono

- 下载 libgdiplus 资源包

wget http://download.mono-project.com/sources/libgdiplus/libgdiplus-4.2.tar.gz

- 解压 libgdiplus-4.2.tar.gz

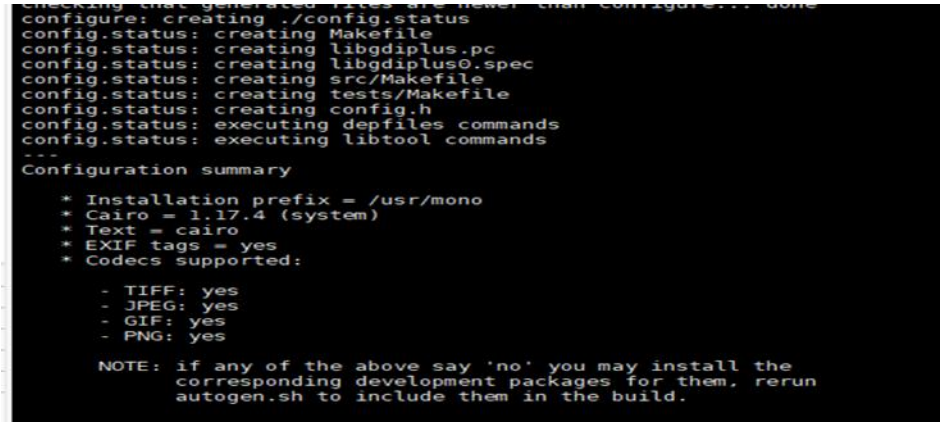
tar zxf libgdiplus-4.2.tar.gz

- 进入 libgdiplus-4.2.tar.gz 文件夹

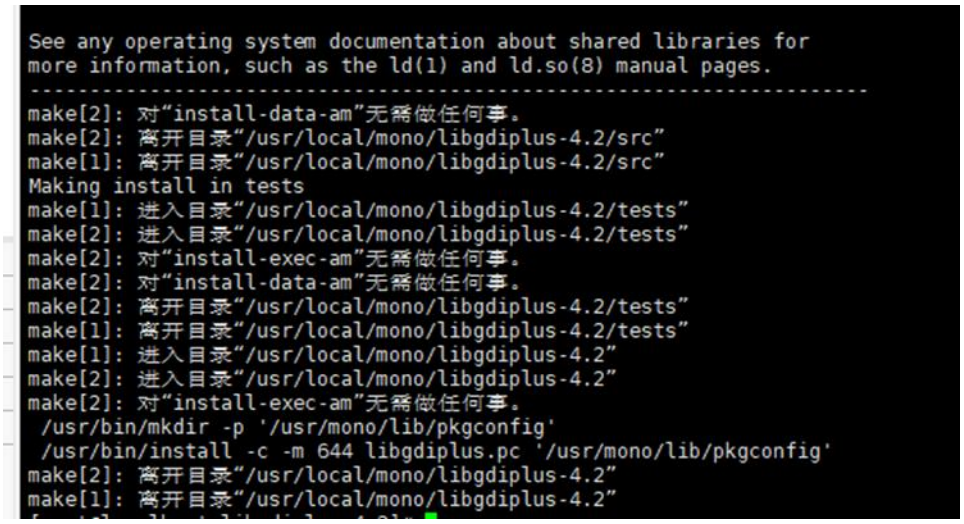
cd libgdiplus-4.2

- 安装 libgdiplus

./configure --prefix=/usr/mono



make && make install



- 配置输出文本

echo “/usr/mono/lib” > /etc/ld.so.conf.d/mono.conf

2.3.3.4. 安装 Mono

- 调整到 mono 文件夹

cd /usr/local/mono

- 下载 mono 资源

wget http://download.mono-project.com/sources/mono/mono-4.6.0.125.tar.bz2

- 解压 mono-4.6.0.125.tar.bz2

tar jxf mono-4.6.0.125.tar.bz2

- 进入 mono-4.6.0.125.tar.bz2

cd mono-4.6.0

- 安装 mono

./configure --prefix=/usr/mono/

```
checking compiler server... yes

       mcs source:    mcs
       C# Compiler:   roslyn
       CompilerServer:

   Engine:
       Host:          x86_64-pc-linux-gnu
       Target:        x86_64-pc-linux-gnu
       GC:            sgen (concurrent by default) and Included Boehm GC with typed GC and parallel mark
       Suspend:       Hybrid
       TLS:           __thread
       SIGALTSTACK:   yes
       Engine:        Building and using the JIT
       BigArrays:     no
       DTrace:        no
       LLVM Back End: no (dynamically loaded: no, built in-tree: no, assertions: no, msvc only: no)
       Spectre:       no mitigation
       Mono.Native:   Linux

   Libraries:
       .NET 4.x:        yes
       Xamarin.Android: no
       Xamarin.iOS:     no
       Xamarin.WatchOS: no
       Xamarin.TVOS:    no
       Xamarin.Mac:     no
       Windows AOT:     no
       Orbis:           no
       Unreal:          no
       WebAssembly:     no
       Test profiles:   AOT Full (no), AOT Hybrid (no), AOT Full Interp (no), Windows Full AOT Interp (no)
       JNI support:     IKVM Native
       libgdiplus:      assumed to be installed
       zlib:            system zlib
       BTLS:            yes (x86_64)
       jemalloc:        no (always use: no)
       crash reporting: yes (private crashes: yes)
       .NET Core:       no
```

make && make install

```
 /usr/bin/mkdir -p '/usr/mono/bin'
  ../../doltlibtool   --mode=install /usr/bin/install -c mono-hang-watchdog '/usr/mono/bin'
libtool: install: /usr/bin/install -c mono-hang-watchdog /usr/mono/bin/mono-hang-watchdog
make[3]: 对“install-data-am”无需做任何事。
make[3]: 离开目录“/usr/local/mono/mono-6.12.0.122/tools/mono-hang-watchdog”
make[2]: 离开目录“/usr/local/mono/mono-6.12.0.122/tools/mono-hang-watchdog”
make[2]: 进入目录“/usr/local/mono/mono-6.12.0.122/tools”
make[3]: 进入目录“/usr/local/mono/mono-6.12.0.122/tools”
make[3]: 对“install-exec-am”无需做任何事。
make[3]: 对“install-data-am”无需做任何事。
make[3]: 离开目录“/usr/local/mono/mono-6.12.0.122/tools”
make[2]: 离开目录“/usr/local/mono/mono-6.12.0.122/tools”
make[1]: 离开目录“/usr/local/mono/mono-6.12.0.122/tools”
make[1]: 进入目录“/usr/local/mono/mono-6.12.0.122/docs”
make[2]: 进入目录“/usr/local/mono/mono-6.12.0.122/docs”
make[2]: 对“install-exec-am”无需做任何事。
 /usr/bin/mkdir -p '/usr/mono/lib/monodoc/sources'
 /usr/bin/install -c -m 644 mono-file-formats.source mono-tools.source monoapi.source mono-file-formats.tree mono-file
make[2]: 离开目录“/usr/local/mono/mono-6.12.0.122/docs”
make[1]: 离开目录“/usr/local/mono/mono-6.12.0.122/docs”
make[1]: 进入目录“/usr/local/mono/mono-6.12.0.122/msvc”
make[2]: 进入目录“/usr/local/mono/mono-6.12.0.122/msvc”
make[2]: 对“install-exec-am”无需做任何事。
make[2]: 对“install-data-am”无需做任何事。
make[2]: 离开目录“/usr/local/mono/mono-6.12.0.122/msvc”
make[1]: 离开目录“/usr/local/mono/mono-6.12.0.122/msvc”
make[1]: 进入目录“/usr/local/mono/mono-6.12.0.122/acceptance-tests”
make[2]: 进入目录“/usr/local/mono/mono-6.12.0.122/acceptance-tests”
make[2]: 对“install-exec-am”无需做任何事。
make[2]: 对“install-data-am”无需做任何事。
make[2]: 离开目录“/usr/local/mono/mono-6.12.0.122/acceptance-tests”
make[1]: 离开目录“/usr/local/mono/mono-6.12.0.122/acceptance-tests”
```

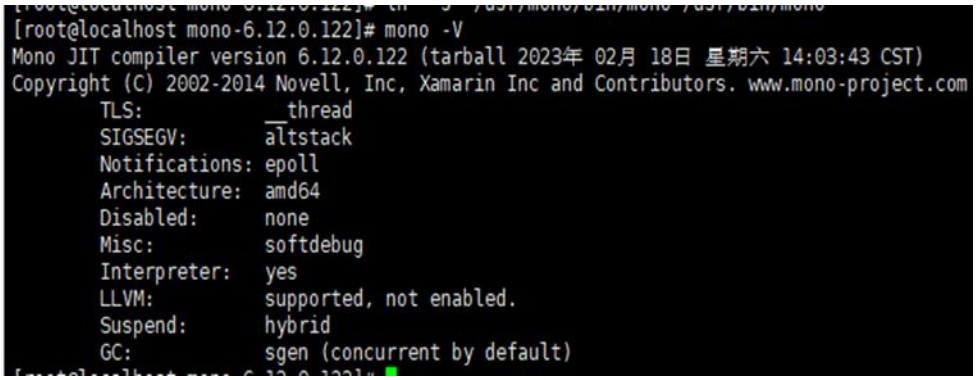
- 配置属性

ln -s /usr/mono/bin/mono /usr/bin/mono

ln -s /usr/mono/lib/libgdiplus.so /usr/lib64/libgdiplus.so

- 验证 mono 安装成功

mono -V



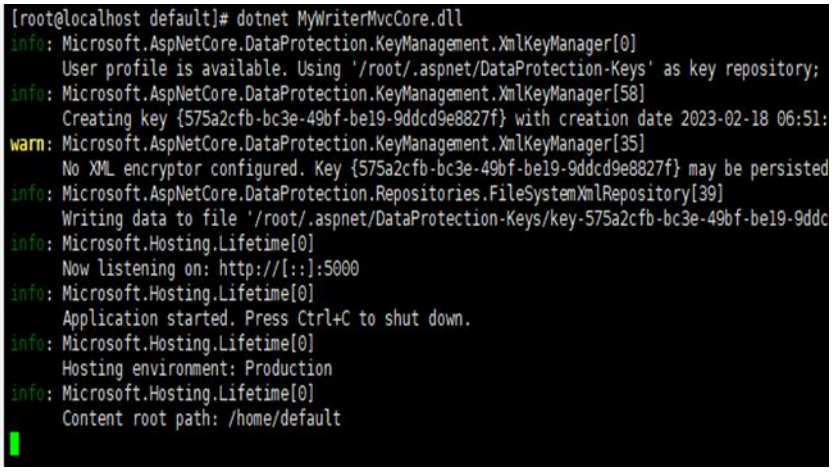
2.3.3.5.搭建 DCWriter 服务

- 创建文件夹

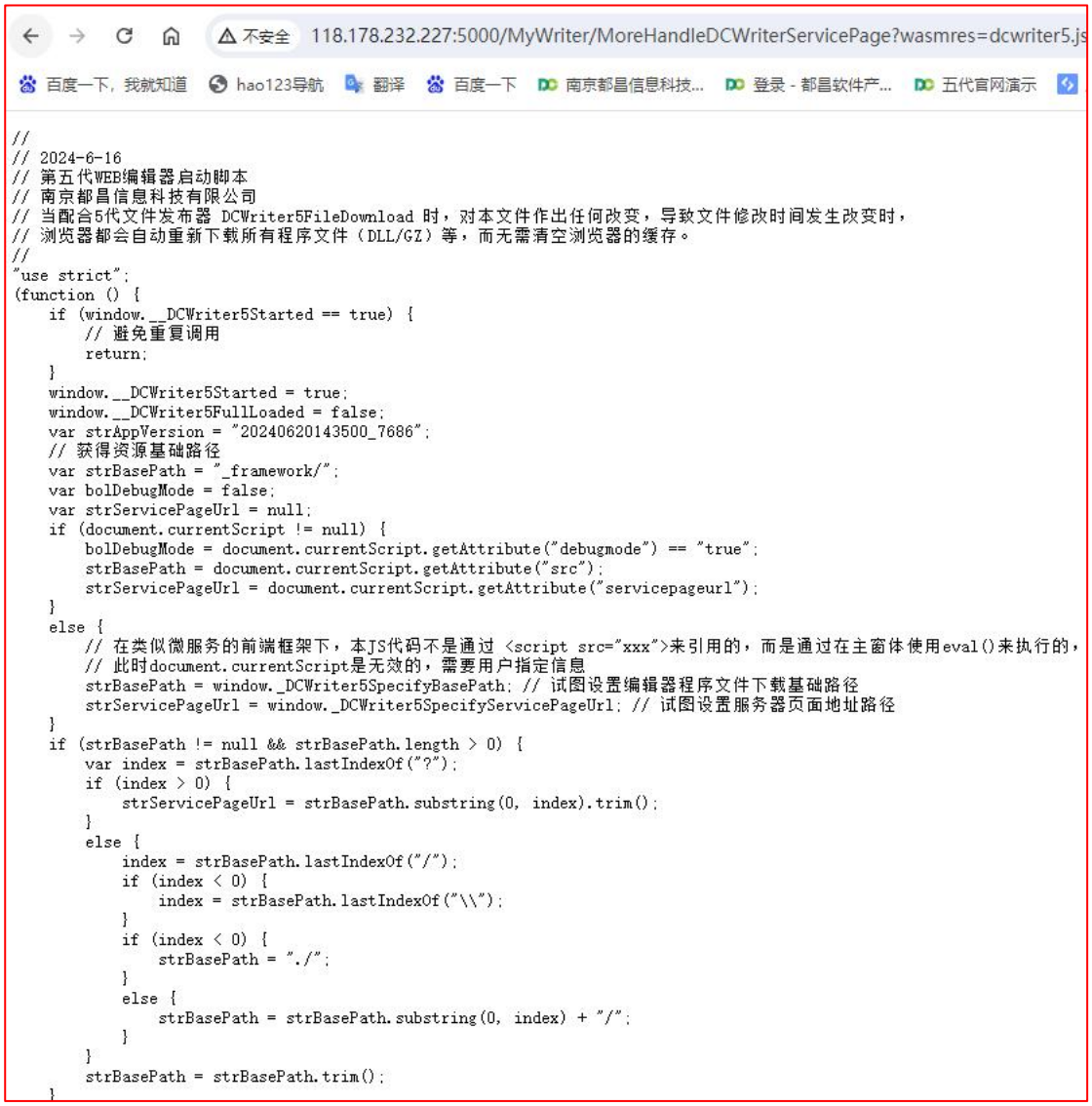
mkdir /home/default

- 上传.net core 五代资源包到 default 目录下
- 启动服务

dotnet MyWriterMvcCore.dll



通过服务器IP:网站端口号/MyWriter/MoreHandleDCWriterServicePage?wasmres=dcwriter5.js 访问出现下图 js 效果表示.net core 环境搭建完毕，该浏览器访问地址就是五代编辑器需要引用的地址。




```
//
// 2024-6-16
// 第五代WEB编辑器启动脚本
// 南京都昌信息科技有限公司
// 当配合5代文件发布器 DCWriter5FileDownload 时，对本文件作出任何改变，导致文件修改时间发生改变时，
// 浏览器都会自动重新下载所有程序文件（DLL/GZ）等，而无需清空浏览器的缓存。
//
"use strict";
(function () {
    if (window.__DCWriter5Started == true) {
        // 避免重复调用
        return;
    }
    window.__DCWriter5Started = true;
    window.__DCWriter5FullLoaded = false;
    var strAppVersion = "202406201435OO_7686";
    // 获得资源基础路径
    var strBasePath = "_framework/";
    var bolDebugMode = false;
    var strServicePageUrl = null;
    if (document.currentScript != null) {
        bolDebugMode = document.currentScript.getAttribute("debugmode") == "true";
        strBasePath = document.currentScript.getAttribute("src");
        strServicePageUrl = document.currentScript.getAttribute("servicepageurl");
    }
    else {
        // 在类似微服务的前端框架下，本JS代码不是通过 <script src="xxx">来引用的，而是通过在主窗体使用eval()来执行的，
        // 此时document.currentScript是无效的，需要用户指定信息
        strBasePath = window._DCWriter5SpecifyBasePath; // 试图设置编辑器程序文件下载基础路径
        strServicePageUrl = window._DCWriter5SpecifyServicePageUrl; // 试图设置服务器页面地址路径
    }
    if (strBasePath != null && strBasePath.length > 0) {
        var index = strBasePath.lastIndexOf("?");
        if (index > 0) {
            strServicePageUrl = strBasePath.substring(0, index).trim();
        }
        else {
            index = strBasePath.lastIndexOf("/");
            if (index < 0) {
                index = strBasePath.lastIndexOf("\\");
            }
            if (index < 0) {
                strBasePath = "./";
            }
            else {
                strBasePath = strBasePath.substring(0, index) + "/";
            }
        }
        strBasePath = strBasePath.trim();
    }
```

2.2.4. Java 服务部署步骤：

2.2.4.1. 验证 jdk 版本

java -version

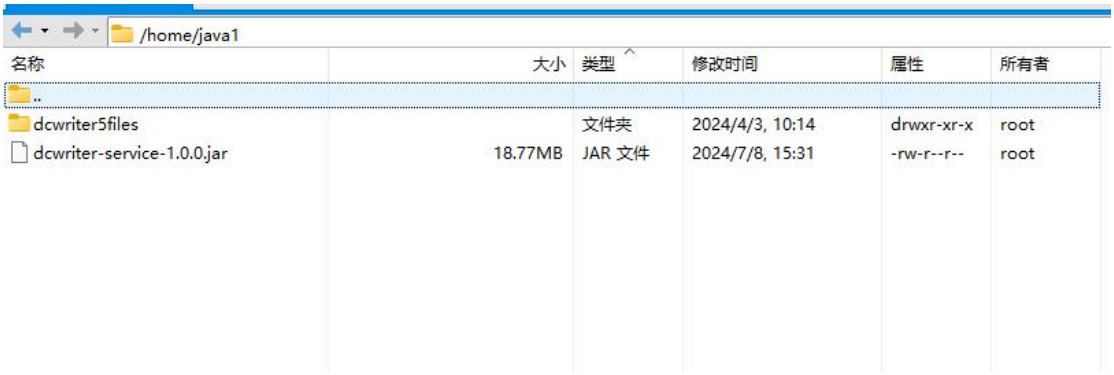
```
[root@localhost java1]# java -version
openjdk version "1.8.0_362"
OpenJDK Runtime Environment (build 1.8.0_362-b08)
OpenJDK 64-Bit Server VM (build 25.362-b08, mixed mode)
[root@localhost java1]#
```

注意：没有对应版本，需要先安装 jdk 版本

2.2.4.2. 创建目录

mkdir /home/java1

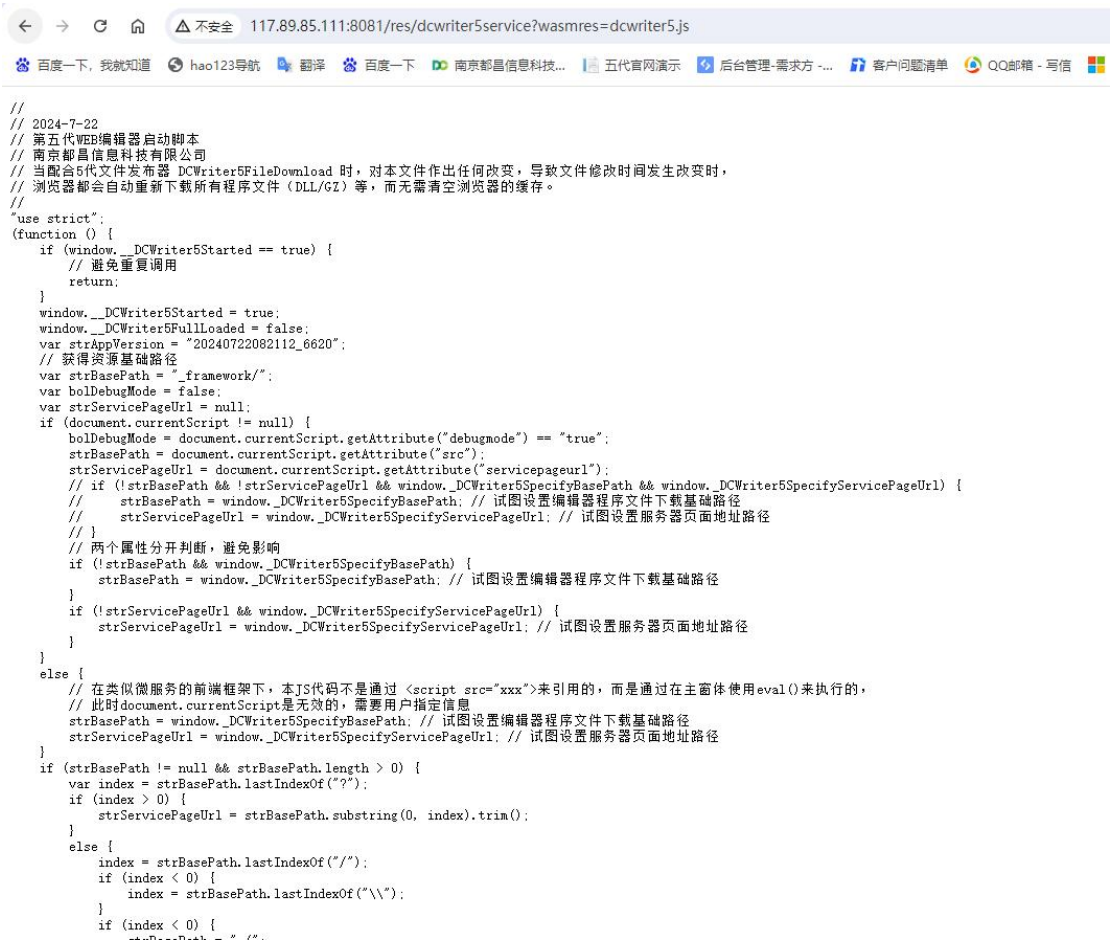
2.2.4.3.上传 java 五代资源包



2.2.4.4.启动 java 服务

java -jar dcwriter-service-1.0.0.jar --server.port=8081

通过服务器 IP:8081/res/dcwriter5service?wasmres=dcwriter5.js 访问出现下图 js 效果表示 Java 环境搭建完毕，该浏览器访问地址就是五代编辑器需要引用的地址。



2.3.DCWriter 入门测试

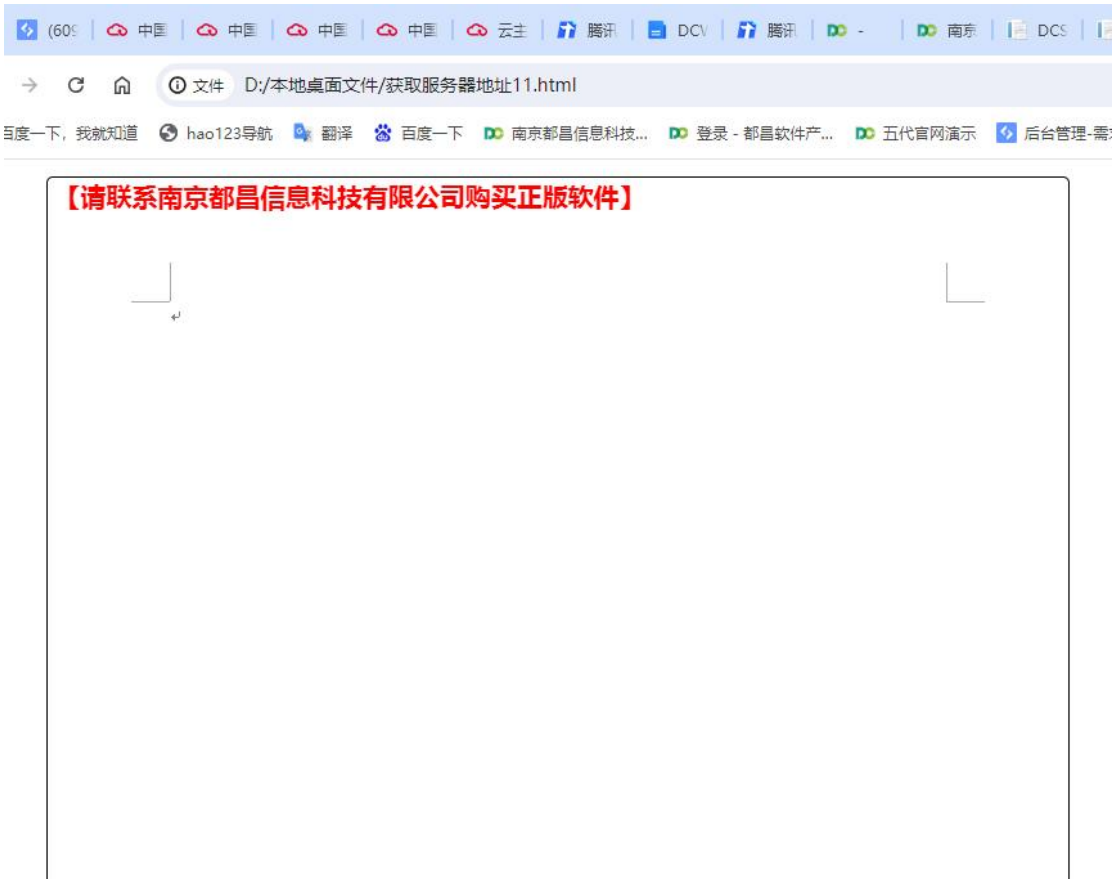
当满足客户端硬性条件和服务器部署 DCWriter 成功后，就可以简易创建 DCWriter 进行一些功能性的验证，比如加载文档、保存文档、元素赋值、取值、打印等电子病历核心等功能。

2.3.1.搭建静态 HTML 测试

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8" />
   <title>DCWriter 测试</title>
</head>
<body>
   <div id="myWriterControl" dctype="WriterControlForWASM">
      正在加载...
   </div>
<script
src="http://www.dcwriter.cn:8023/Jquery/jquery-1.7.2.min.js"></script>
<script
src="http://www.dcwriter.cn:8023/ServicePage.aspx?wasmres=dcwriter5.js"></sc
```

```
ript>
</body>
</html>
```

上述代码 div 中加入属性 dctype="WriterControlForWASM"就是为第五代编辑器，下面引用两个 js 文件，第一个为 jquery-js 文件，第二个为服务器部署网站（服务）成功引用地址，使用浏览器加载，DCWriter 自动初始化进行渲染编辑器。



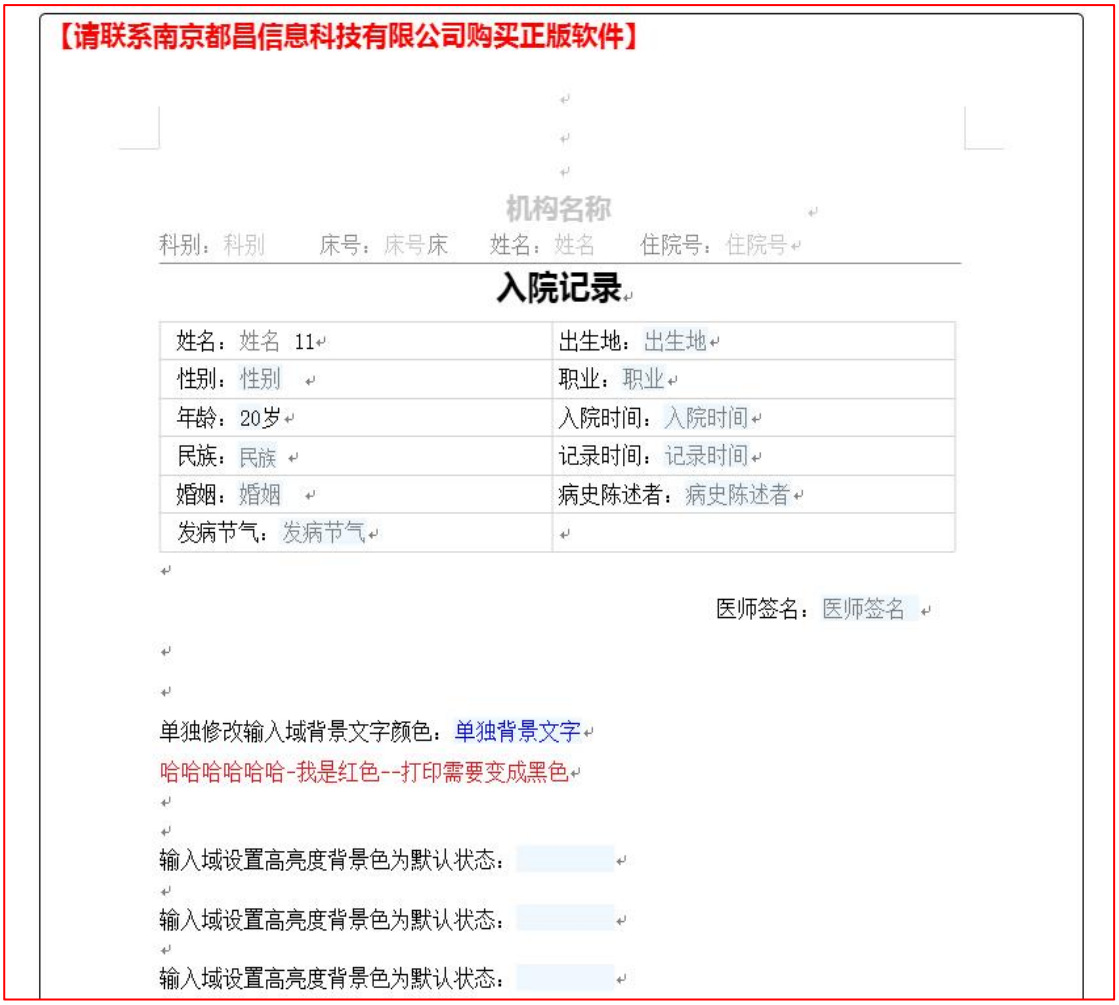
2.3.2.功能测试

2.3.2.1.加载-保存文档

编辑器只是一个控件，没有数据库，病历数据需要用户自行存储，比如存入数据库或者存储到文件服务器中，当调用编辑器内置加载文档方法，需要用户从前端发送请求到业务层连接数据库获取到病历数据，然后返回数据到前端由编辑器进行渲染显示。

编辑器加载病历文档：

```
var ctl=document.getElementById("myWriterControl")
ctl.LoadDocumentFromString(xmlstr)
```



编辑器保存文档:

```
var ctl=document.getElementById("myWriterControl")
var xmlstr=ctl.SaveDocumentToString()
console.log("病历数据：",xmlstr)
```

调用编辑器保存方法，编辑器会返回当前病历的 xml 字符串数据，前端获取该数据发送到业务层由用户自行保存到数据库或者文件服务器。

2.3.2.2.结构化元素赋值-取值

文本域元素赋值：

```
var ctl=document.getElementById("myWriterControl")
ctl.SetElementTextByID("姓名","张三")
```

文本域元素取值：

```
var ctl=document.getElementById("myWriterControl")
var fieldtext=ctl.GetElementTextByID("姓名")
console.log("姓名输入域的内容：",fieldtext)
```

2.3.2.3.浏览器打印

```
var ctl=document.getElementById("myWriterControl")
ctl.PrintDocument()
```

2.3.3.Vue2-DCWriter 测试

三．核心功能开发

5.1.加载病历文档

加载病历文档分为两块，第一块就是病人是医生第一次接入的，不管是书写门诊病历还是住院病历，都是需要用户从数据库中获取到对应的病历模板。第二块就是医生查看已经书写完成的病历文档。虽然应用场景不同，但是编辑器把病历数据渲染成可视界面调用的接口都是一致的。

病历数据格式分为三种，第一个就是默认的明文的 xml 字符串数据、第二个就是 json 格式数据，第三个就是 base64 数据。根据不同类型的数据，调用不同的接口进行加载。

```
//加载 xml 数据的方法
var ctl = document.getElementById("myWriterControl");
ctl.LoadDocumentFromString('xml 字符串数据','xml'); //xml 字符串数据需要客户通业务层从数据库取到数据，然后发送到前端，通过编辑器接口加载。
//加载 json 数据的方法
var ctl = document.getElementById("myWriterControl");
ctl.LoadDocumentFromString('json 字符串数据','json');
//加载 base64 数据的方法
var ctl = document.getElementById("myWriterControl");
ctl.LoadDocumentFromBase64String('base64 字符串数据','xml');
```

注意：当编辑器加载文档完成，会触发 DocumentLoad 事件（文档加载完成事件），但是该事件定位的位置一定得在 LoadDocumentFrom...方法的前面才行，否则无法触发。

```
var ctl = document.getElementById("myWriterControl");
ctl.DocumentLoad=function(){
console.log("1.加载文档成功之后触发的事件")
}
ctl.LoadDocumentFromString(xmlstr,"xml")
```

5.1.1.什么是文档加载完成事件

文档加载完成事件就是把文档成功渲染到编辑器上面，然后需要根据业务的不同，可能需要修改某个结构化元素属性，或者是需要把当前病人的基本信息（姓名、年龄、性别、电话号码等）自动带入到文档对应的结构化元素里面，或者还有一个业务时需要在加载文档进行处理，都可以在这个事件上进行处理。

5.2.保存文档

用户书写完病历执行完保存接口，电子病历的流程就算走完，编辑器支持把病历保存为 xml 数据格式、json 数据格式、base64 数据格式。然后有额外的业务要求也能生成 pdf 文件、rtf 文档、图片等格式的数据。

```
//保存 xml 数据的方法
var ctl = document.getElementById("myWriterControl");
var result=ctl.SaveDocumentToString('xml'); //前端返回 xml 字符串数据，用户发送到业
```

```
务端自行保存。
//保存 json 数据的方法
var ctl = document.getElementById("myWriterControl");
var result=ctl.SaveDocumentToString('json);
//保存 base64 数据的方法
var ctl = document.getElementById("myWriterControl");
var result=ctl.SaveDocumentToBase64String('xml');
```

保存前也能调用文档校验的接口全篇排查下文档中是否出现不满足电子病历规范的数据。例如：主诉不能为空、且输入的内容不能超过 20 个字符。男性病历不能出现女性的疾病。出院时间不能低于入院时间等等。保存前都是可以校验出来的，有的校验规则在制作模板中就可以实现，有的校验规则通过代码就能判断。

```
//批注式校验
var ctl = document.getElementById("myWriterControl");
ctl. DocumentValueValidateWithCreateDocumentComments()
```

当出现文档校验失败，就可以提醒用户修改，否则不允许保存。

5.3.文档元素取值赋值

当文档成功渲染到编辑器上，那肯定是需要把文档中结构化元素进行赋值，比如可以从 HIS 系统里面获取到用户的基本信息，那需要把这些信息进行同步到病历中，就需要调用编辑器内置接口进行结构化元素的一对一赋值。

当执行到这步时，模板中的元素设置就需要进行规范性设置，否则当文档中存在 ID 相同的元素，或者是某个元素属性没有控制好，开发过程就容易出现问题，如何规范的设置模板就需要清晰了解结构化元素的介绍和设计了。

5.3.1.结构化元素设计与介绍

5.3.1.1.什么是病历结构化

病历结构化可以用于电子病历系统中，帮助医生快速记录和查找病历信息，提高医疗工作效率。病历结构化还可以用于医学研究和临床决策支持系统中，帮助研究人员和医生分析和挖掘大量的医疗数据，提供科学依据和指导

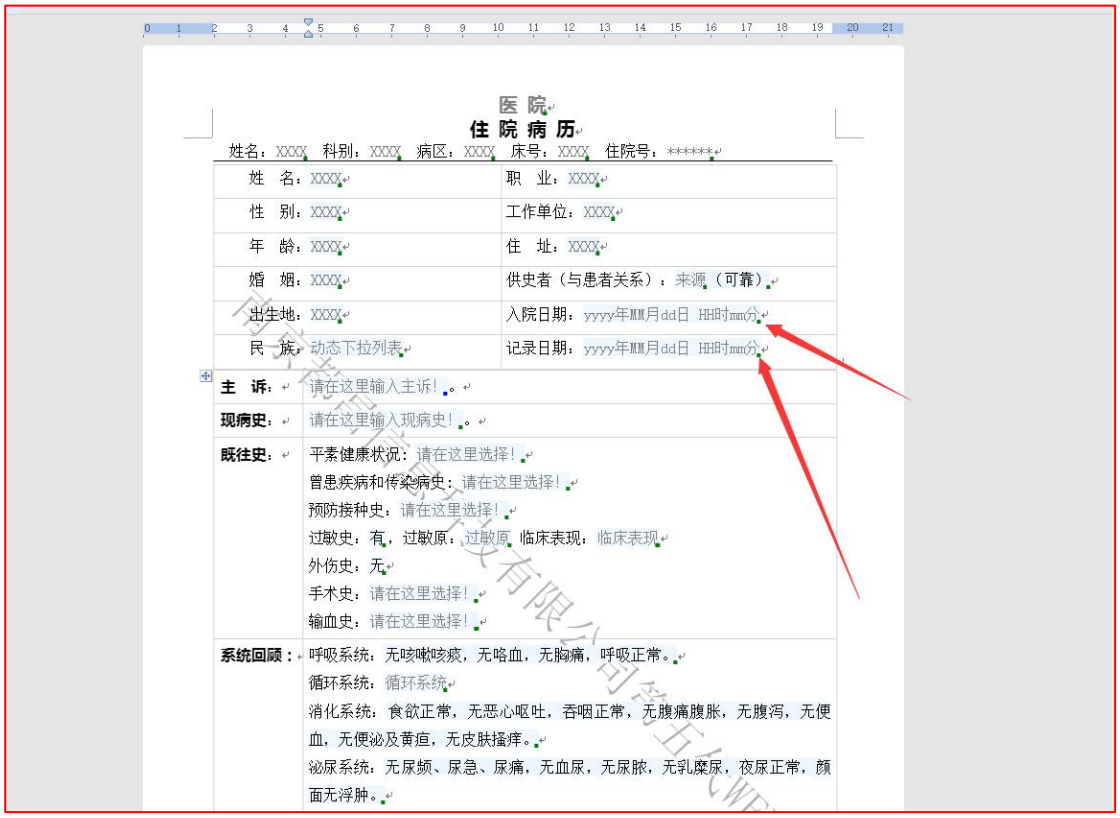
那都昌编辑器就可以通过创建不用的结构化元素用来组成一个一个的病历结构化文档，帮助友商更好的实现医院结构化的要求。

5.3.1.2.什么是结构化元素

结构化元素就是可以用来解析从而获取到医院需要的数据，帮助友商能更好的满足数据共享和分析等功能。

5.3.1.3.结构化元素有哪些

5.3.1.3.1.输入域元素



医 院

住 院 病 历

姓名：XXXX 科别：XXXX 病区：XXXX 床号：XXXX 住院号：******

姓 名：XXXX	职 业：XXXX
性 别：XXXX	工作单位：XXXX
年 龄：XXXX	住 址：XXXX
婚 姻：XXXX	供史者（与患者关系）：来源（可靠）
出生地：XXXX	入院日期：yyyy年MM月dd日 HH时mm分
民 族：动态下拉列表	记录日期：yyyy年MM月dd日 HH时mm分
主 诉：	请在这里输入主诉！
现病史：	请在这里输入现病史！
既往史：	平素健康状况：请在这里选择！ 曾患疾病和传染病史：请在这里选择！ 预防接种史：请在这里选择！ 过敏史：有，过敏原：过敏原 临床表现：临床表现 外伤史：无 手术史：请在这里选择！ 输血史：请在这里选择！
系统回顾：	呼吸系统：无咳嗽咳痰，无咯血，无胸痛，呼吸正常。 循环系统：循环系统 消化系统：食欲正常，无恶心呕吐，吞咽正常，无腹痛腹胀，无腹泻，无便血，无便泌及黄疸，无皮肤搔痒。 泌尿系统：无尿频、尿急、尿痛，无血尿，无尿脓，无乳糜尿，夜尿正常，颜面无浮肿。

右下角带有蓝色圆点或者有淡蓝色高亮度的元素，我们统称为输入域元素，且在一个病历文书中输入域是一个非常重要的组成部分。

1.如何创建输入域元素

创建输入域元素常用的有两个方式：

第一种：通过对话框配置好输入域属性进行插入

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'InsertInputField' ,true,null)
```

效果图：

第二种：先配置好创建输入域元素需要的属性，然后插入

```
var  ctl=document.getElementById("myWriterControl");
var options={
     ID:'field1',
      ·····,//此处省略多个属性
}
ctl.DCExecuteCommand( 'InsertInputField' ,false,options)
```

2.输入域属性对话框

输入域内置属性对话框功能，把光标放在输入域里面调用编辑器内置命令:

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'ElementProperties' ,true,null)
```

1.编号（ID）

编号（ID）对应的是输入域属性中的 ID，在设计该 ID 值时，需要满足标识符规范，不要带有小数点或者纯数字来进行命名，且尽量不要出现相同的元素 ID，因为当给输入域元素赋值的时候可以通过对指定 ID 的元素进行赋值（SetElementTextById），这样当一个文档中存在多个相同 ID 的元素时，就只会对第一个元素进行赋值了，且元素的可见性表达式、数值表达式也是通过元素 ID 来进行逻辑判断的。

当文档中存在多个相同 ID 的元素时，需要分别赋值时，可以使用一下组合方法进行处理：

```
var ctl=d ocument.getElementById("myWriterControl");
var input = ctl.GetElementsById('field1')//获取 ID 为 field1 的元素集合
        input.forEach(element => {
          var ele= ctl.GetElementProperties(element)//循环获取每一个输入域的属性
         //这个地方可以加个判断，比如需要对设置了特定条件的元素进行赋值
          ctl.SetElementTextByID(ele.NativeHandle,"赋值 ID")//通过元素属性 NativeHandle 值进行赋值，该值在一个文档中是唯一的。
        });
```

2.名称（Name）

名称对应的是输入域属性中的 Name 值，该值类似于 ID，也可通过该属性值获取到指定的输入域。也需要符合标识符规范。

3. 背景文字

背景文字对应的是输入域属性中的 BackgroundText，设置该值的在界面上效果为：

需要注意的时，当输入域没有输入内容的时候，打印该文档的时候，是否需要把背景文字打印出来，要是不打印背景文字，那背景文字是否需要占位。

默认是不打印背景文字，需要打印的话，需要调用文档选项进行控制：

```
var ctl=d ocument.getElementById("myWriterControl");
ctl. DocumentOptions.ViewOptions.PrintBackgroundText=true;
ctl.ApplyDocumentOptions();
```

默认是不打印背景文字，但是占位的，要是不需要占位，需要调用文档选项进行控制：

```
var ctl=d ocument.getElementById("myWriterControl");
ctl. DocumentOptions.ViewOptions.PreserveBackgroundTextWhenPrint=false;
ctl.ApplyDocumentOptions();
```

4. 左边框

左边框对应的是输入域属性中的 StartBorderText。用户需要自定义左边框，修改该属性就可以。

5. 右边框

右边框对应的是输入域属性中的 EndBorderText。用户需要自定义左边框，修改该属性就可以。

6. 固定宽度

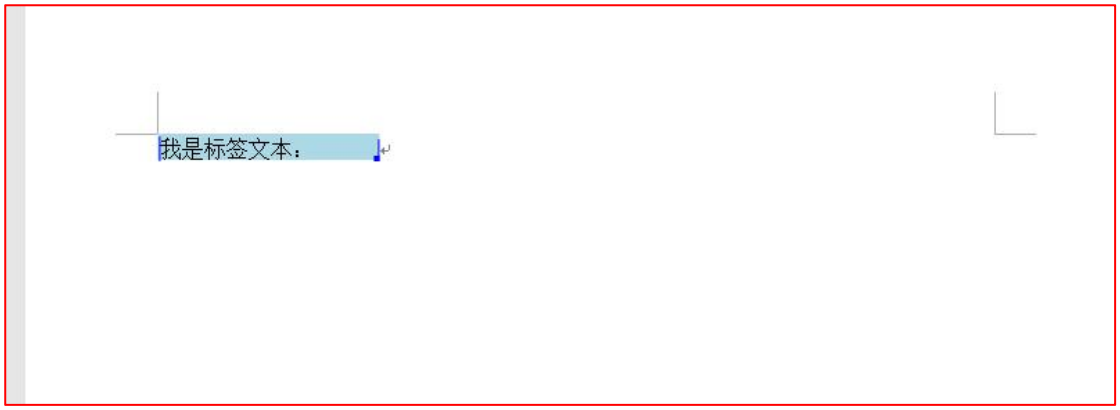
固定宽度对应的是输入域属性中的 SpecifyWidth（单位 1/300 英寸）。可以设置正数和负数，当处于正数的时候，输入域会设置宽度，然后里面输入的内容宽度超过输入域设置的宽度，输入域宽度会随着内容的宽度变化而变化，当为负数的时候，输入域永远是这么宽，输入的内容超过了该宽度，输入域的宽度也不会有变化。

7. 焦点快捷键

焦点快捷键对应的是输入域属性中的 MoveFocusHotKey。当用户需要把光标从一个输入域调整到另外一个输入域里面，就可以通过设置该属性来进行处理。默认支持：Tab 键、Enter 键切换。注意：当用户需要纯键盘操作的话，通过快捷键进行元素直接的快速切换，需要处于严格表单模式才行。且该值需要区分大小写。

8. 标签文本

标签文本对应的是输入域属性中的 LabelText。注意：设置了标签文本值，获取元素内容（GetElementTextByID）的时候，也会把标签文本带出来。



9. 单位文本

单位文本对应的是输入域属性中的 UnitText。注意：设置了单位文本值，获取元素内容（GetElementTextByID）的时候，也会把单位文本带出来。

10. 激活模式

激活模式对应的是输入域属性中的 EditorActiveMode，该属性只有当输入域不是纯文本元素才有效。默认都是当鼠标双击的时候才激活下拉框。根据不同业务可以进行控制，比如修改成单击激活，或者当用户需要纯键盘操作，然后当光标进入到输入域中的时候，就需要设置成当元素获取到输入焦点就需要激活。

11. 高亮度状态

高亮度状态对应的是输入域属性中的 EnableHighlight。

不需要显示高亮度状态可以修改值为 Disabled

12. 打印显示

打印显示对应的是输入域属性中的 PrintVisibility，通过该属性可以让指定元素是否需要打印，但是在书写过程中该元素还是可见的，只不过打印预览、阅读模式、打印下该元素隐藏。详细枚举值，请参考接口文档。

13. 输入最大字符数

输入最大字符数对应的是输入域属性中的 MaxInputLength，该属性可以控制用户最大只能输入几个字符。

14. 是否可以直接编辑修改内容

是否可以直接编辑修改内容对应的是输入域属性中的 UserEditable，通过控制该属性，能影响用户是否能允许键盘输入内容。比如当元素设置成下拉输入域、时间输入域、数值输入域。就可以让用户只能通过下拉进行选择内容，而不让用户手动输入。注意：当设置了不允许用户编辑的时候，输入域的边框会变颜色。通过边框颜色就能知道那些元素设置成了不允许用户修改。处于管理员模式下该属性无效

15. 是否只读

是否只读对应的是输入域属性中的 ContentReadonly，该属性可以控制当一个容器元素中存在多个子元素，然后容器元素需要不允许编辑，但是里面的子元素有的需要能编辑有的不需要编辑，就可以把父容器的这个属性设置成 true，需要能编辑的元素设置成 false，这样父元素和子元素就可以互不干扰。注意：当设置了只读的时候，输入域的背景色会变成浅灰色。处于管理员模式下该属性无效。

16. 是否允许删除

是否允许删除对应的是输入域属性中的 Deleteable，该属性可以控制元素是否允许被用户键盘删除掉。处于管理员模式下该属性无效。

17. 加密显示

加密显示对应的是输入域属性中的 ViewEncryptType，该输入域可以控制用户输入的内容是否进行脱敏展示。分为两种情况，全部脱敏和部分脱敏。详细枚举值，请参数接口文档。

18. 能接受字符清单

能接受字符清单对应的是输入域属性中的 LimitedInputChars，该输入域可以控制用户输入的内容，比如需要控制用户只能输入纯数字，那就可以设置成’0,1,2,3,4,5,6,7,8,9’，

这样用户只能输入 0-9 之内的字符了。

19. 数据源名

在解释数据源名称的时候，需要解释下编辑器对病历文档中结构化元素赋值的两种模式：

第一种的单一元素赋值，也就是提供一个元素 ID 或者是其他的条件，让编辑器找到对应的元素，然后进行赋值，比如常用的 SetElementTextByID 方法就是传一个元素的 ID 传一个对应的内容。那需要一次性赋值 10 个元素，就需要调用 10 次这个方法。

第二种就是输入域的数据源绑定功能，需要再制作病历模板的时候优先配置好每个结构化元素的数据源名称和绑定路径，这样只需要前端生成一个整体的 json 数据，就可以调用编辑器接口进行批量的有对应关系的方式赋值。

数据源名称对应的是输入域属性中的 ValueBinding 对象下的 DataSource，该属性可以随意定义（需要符合标识符规范），然后可以把相同类型的结构化元素归为一类，设置成同一个数据源名称，比如：姓名输入域、性别输入域、年龄输入域、住址输入域等等。

注意：对元素设置数据源名称是，不区分大小写。不建议用户设置两个这样的数据源名称（AAA，aaa）

20. 绑定路径

绑定路径就是对应输入域属性中的 ValueBinding 对象下的 BindingPath，该属性可以随意定义（需要符合标识符规范），该属性定义最好能跟数据库存储的数据字段有对应关系。然后通过前端生成的 json 数据中 key 值需要进行对应。注意：通过设置这个值，绑定的是输入域的 InnerValue 值，当输入域为纯文本输入域时，界面显示的内容（Text）默认就是 InnerValue 值。

注意：绑定路径的命名一定得符合标识符的规范（不允许出现类似：数字+英文）等命令

21. Text 绑定路径

Text 绑定路径就是对应输入域属性中的 ValueBinding 对象下的 BindingPathForText，该属性可以随意定义(需要符合标识符规范),该属性才是绑定病历界面上显示的内容(Text)，该属性一般是应用于下拉输入域中，比如性别输入域：男—0、女—1，这个时候就需要同时绑定两个值，绑定路径绑定 InnerValue 值，Text 绑定路径绑定 Text 值。

22. 元素类型

元素类型就是对应输入域属性中的 InnerEditStyle，该属性就是控制输入域是什么类型，大致分为以下几类：

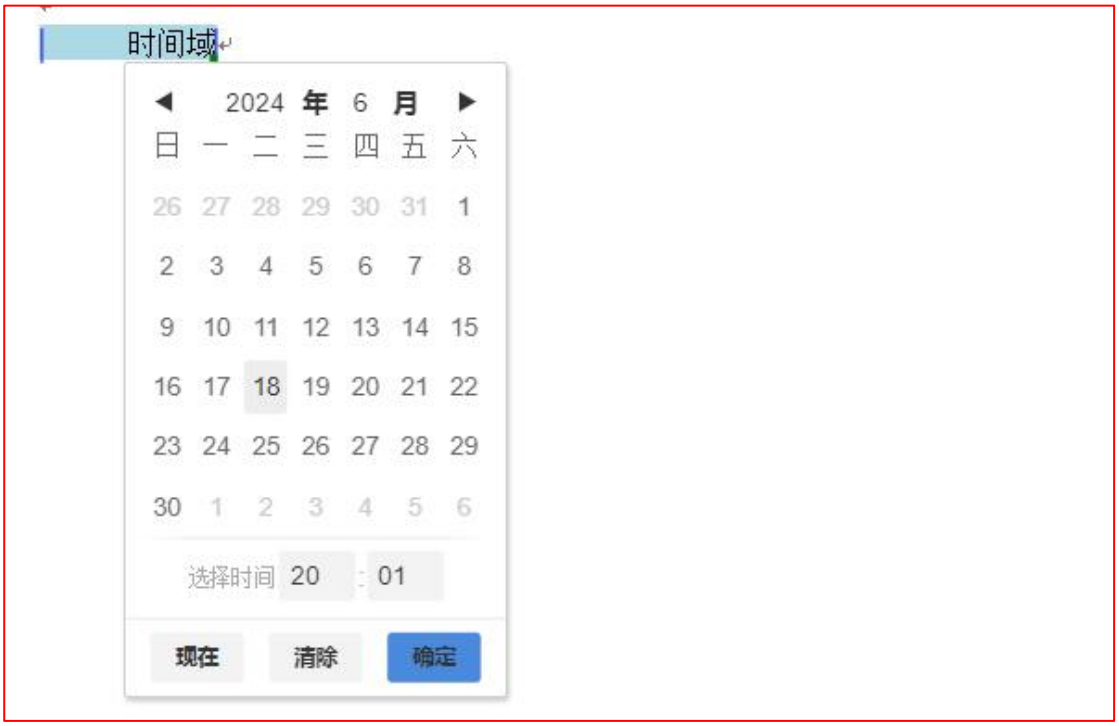
Text：纯文本输入域

DropdownList：下列列表

Date：日期类型

DateTime：日期时间类型

DateTimeWithoutSecond：不含秒的日期时间类型



Time：时间类型

Numeric：数值类型

其中当输入域类型为下拉列表输入域类型时，支持以下功能：

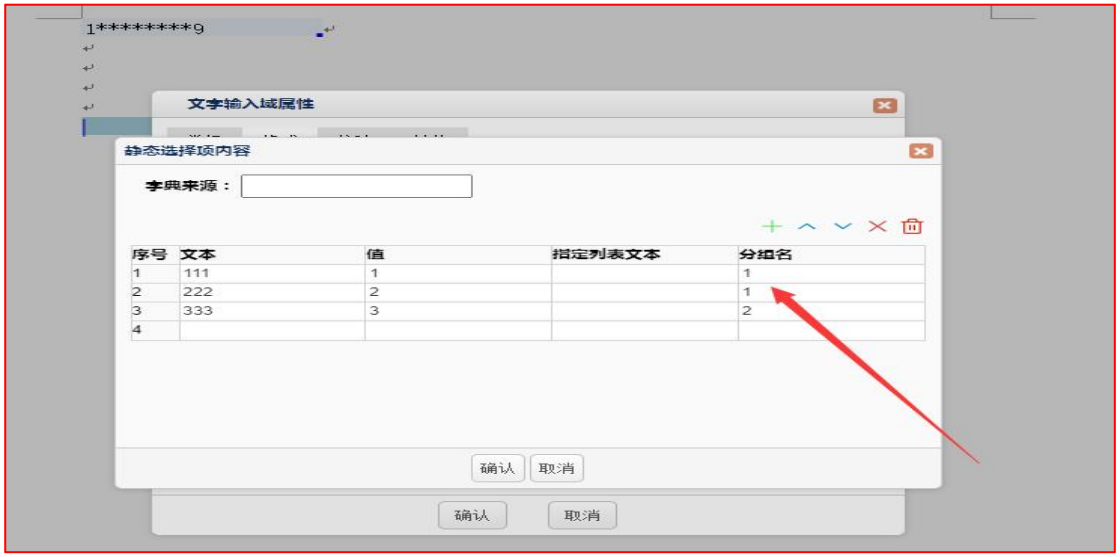
1.是否允许多选

是否允许多选对应的是输入域属性中的 InnerMultiSelect，该属性可以控制用户可以选

择多个选项

2.分组互斥

分组互斥对应的是输入域属性中的 RepulsionForGroup，该属性需要结合 ListItems 对象下的 Group 属性使用。



设置分组互斥，当选中不同的组的时候，会把原来组的内容全部清空，只保留当组的内容，该功能基本在多选下拉模式下使用。

3.静态下拉

静态下拉对应的是输入域属性中的 ListItems,通过该属性用户可以自己把下拉项维护到模板中，用户引用对应的模板，直接选择对应的下拉项填充内容即可。

4.动态下拉

动态下拉对应的是输入域属性中的 DynamicListItems，当设置了该属性时，激活模式中对应的方法激活该输入域时，会触发前端动态下拉列表事件（QueryListItems），在这个事件中进行下拉项的填充。该下拉数据不会存在文档中，数据都是动态生成。

23.输出格式

输出格式对应的是输入域属性中的 DisplayFormat，通过该属性可以控制界面上最终显

示出来的效果。

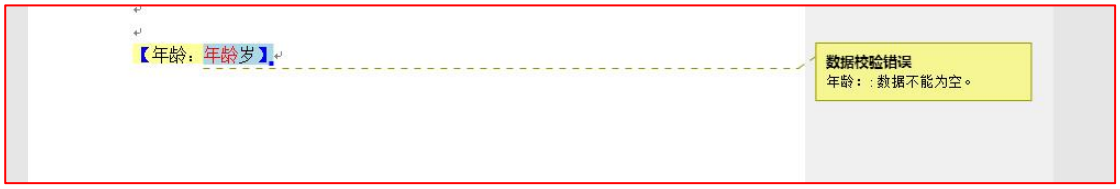
其中常用分为两块：

第一块：数值格式，设置该输出格式，可以控制保留小数位等要求。

第二块：时间格式，设置该输出格式，可以用户自由控制年、月、日、时、分、秒的格式，比如 yyyy 年-MM 月-dd 日等进行输出。

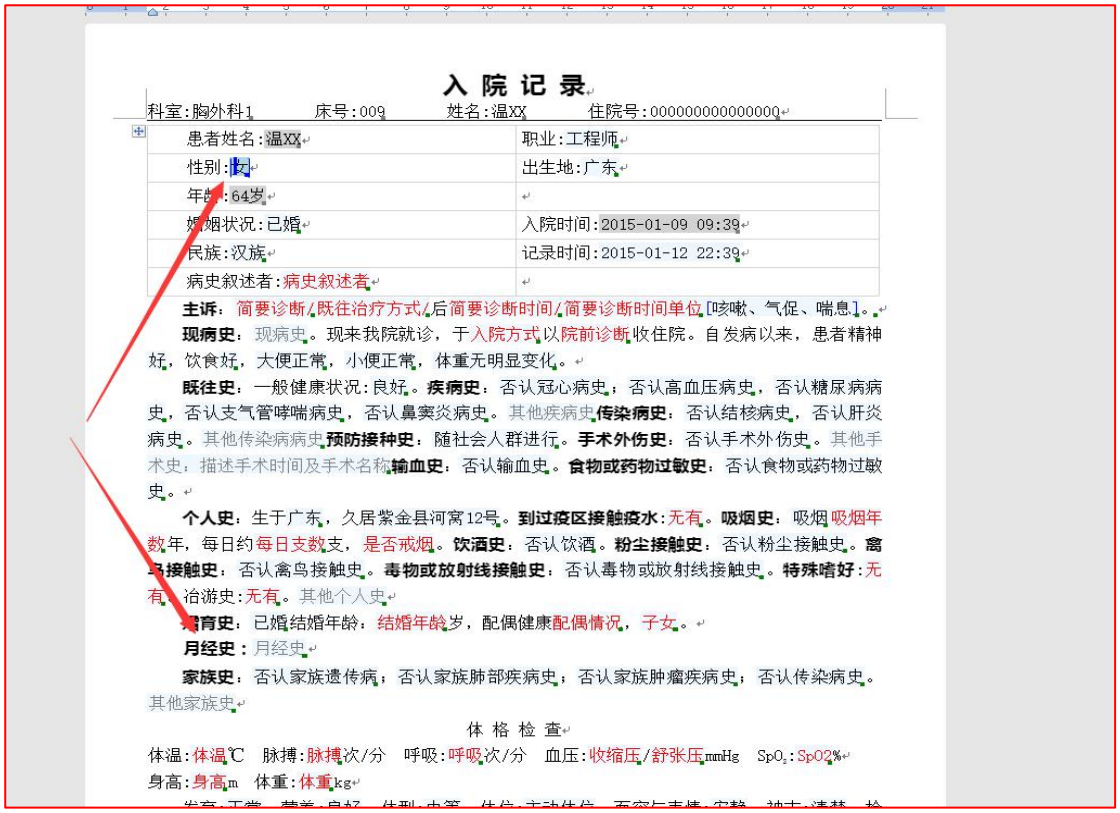
24. 数据校验

数据校验对应的是输入域属性中的 ValidateStyle，通过该属性可以对文档进行一些简单的逻辑质控，比如元素是否必填、长度校验、数值校验、时间校验、违禁关键字校验、正则表达式校验（身份证、电话号码录入是否有误）



25. 可见性表达式

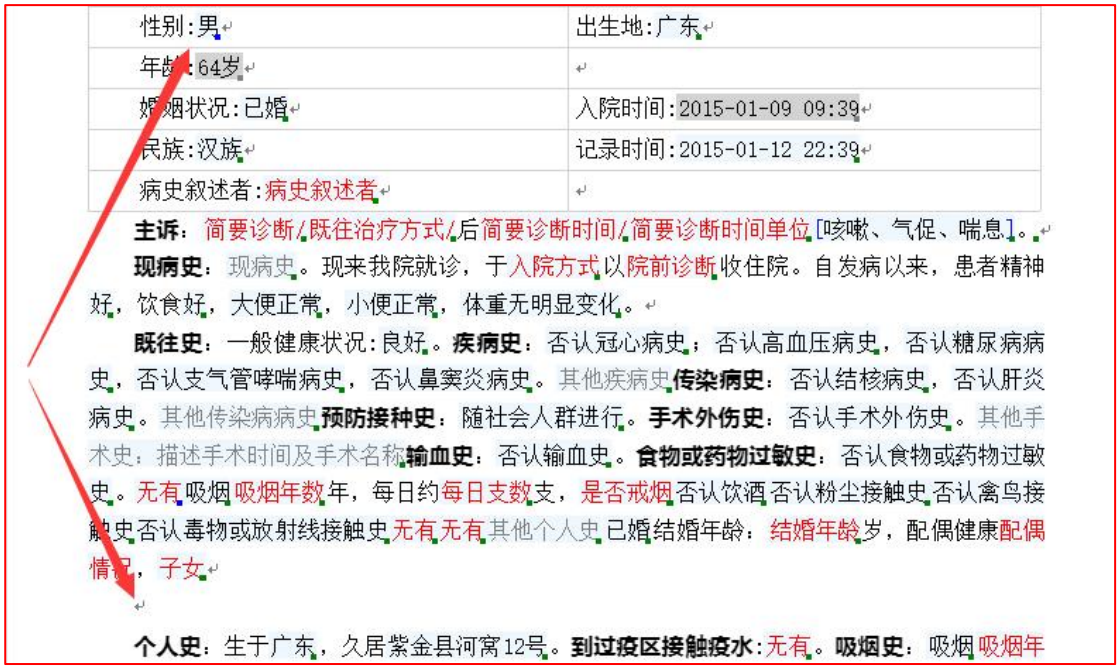
可见性表达式对应的是输入域属性中的 VisibleExpression，通过该属性可以根据某个条件把元素进行隐藏显示，比如入院记录分为男性和女性，那只有女性病人才有月经史，男性病人没有，这个时候就可以通过可见性表达式进行配置，在月经史输入域里面设置可见性表达式当性别输入域元素为女性的时候，月经史就显示，不为女性的时候就隐藏。



如上图所示入院记录中的性别输入域和月经史输入域直接的关联关系，就可以在月经史输入域属性对话框设置可见性表达式：[xx]=0（其中 xx 表示的就是性别输入域的编号（ID），那 0 就是表示该下拉输入域中女选项的 Value 值，当没有 Value 值的时候，可以直接写出[xx]='女'），这样当性别输入域为女的时候，月经史输入域显示，性别输入域不为女的时候，月经

史输入域隐藏。

注意：当月经史隐藏之后，该输入域占据的一行会变成空白行。



性别:男	出生地:广东
年龄:64岁	
婚姻状况:已婚	入院时间:2015-01-09 09:39
民族:汉族	记录时间:2015-01-12 22:39
病史叙述者:病史叙述者	

主诉：简要诊断/既往治疗方式/后简要诊断时间/简要诊断时间单位[咳嗽、气促、喘息]。

现病史：现病史。现来我院就诊，于入院方式以院前诊断收住院。自发病以来，患者精神好，饮食好，大便正常，小便正常，体重无明显变化。

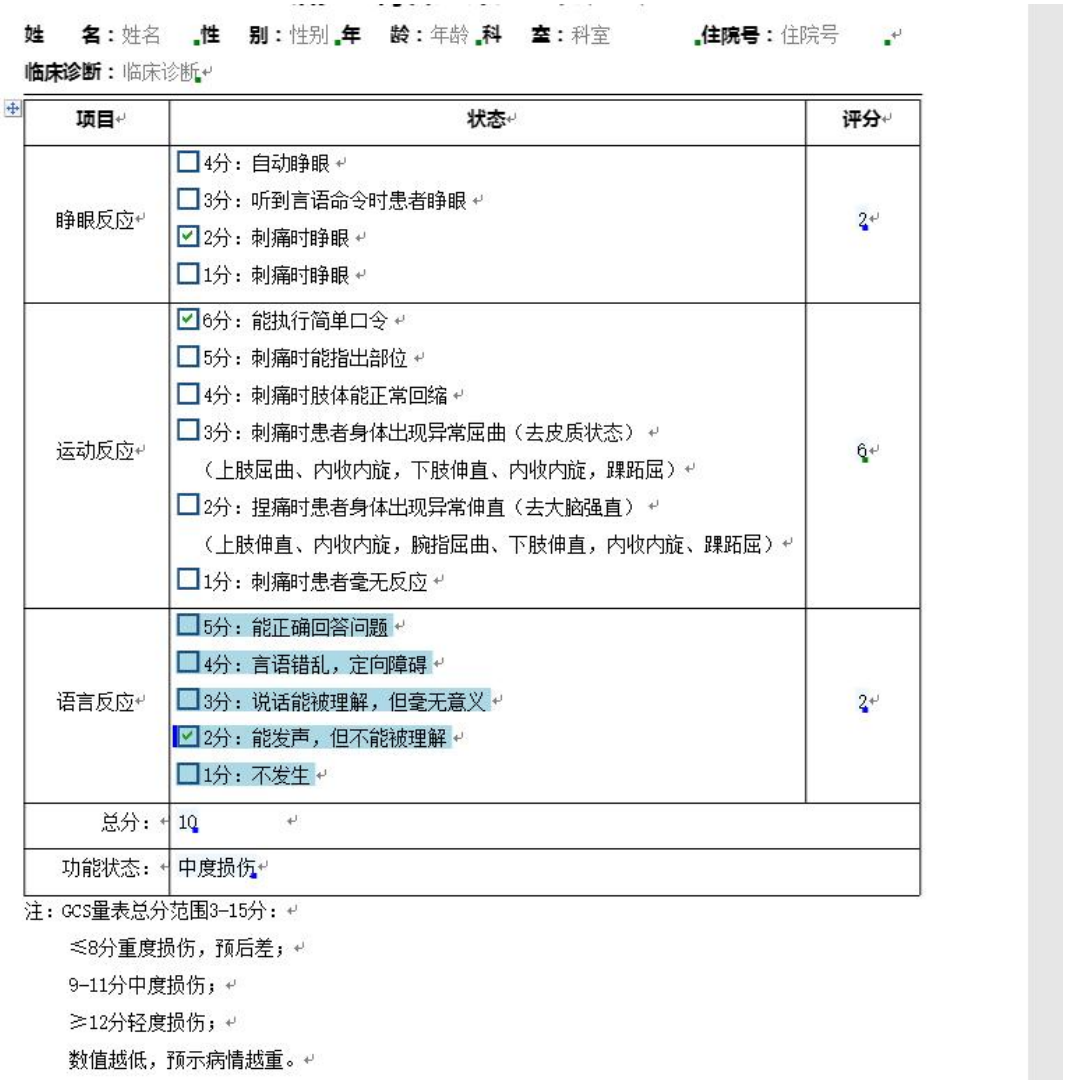
既往史：一般健康状况:良好。**疾病史**：否认冠心病史；否认高血压病史，否认糖尿病病史，否认支气管哮喘病史，否认鼻窦炎病史。其他疾病史**传染病史**：否认结核病史，否认肝炎病史。其他传染病病史**预防接种史**：随社会人群进行。**手术外伤史**：否认手术外伤史。其他手术史：描述手术时间及手术名称**输血史**：否认输血史。**食物或药物过敏史**：否认食物或药物过敏史。无有吸烟吸烟年数年，每日约每日支数支，是否戒烟否认饮酒否认粉尘接触史否认禽鸟接触史否认毒物或放射线接触史无有无有其他个人史已婚结婚年龄：结婚年龄岁，配偶健康配偶情况，子女

个人史：生于广东，久居紫金县河寓12号。**到过疫区接触疫水**:无有。**吸烟史**：吸烟吸烟年

两个解决方案，第一个解决方案就是把从主诉、现病史、既往史、月经史等内容放在表格中的每一行里面，然后对表格行设置可见性表达式，就不会出现空白行的情况。第二个解决方案就是把月经史输入域的 EndingLineBreak 属性为 true，这样该输入域隐藏，后面的内容会自动往上面补充。

26. 数值表达式

数值表达式对应的是输入域数值中的 ValueExpression，通过该属性能实现护理评估量表功能，满足 Execl 里面常用的函数、SUM、Mix、Max、LOOKUP 等。



姓　名：姓名　性　别：性别 年　龄：年龄 科　室：科室　住院号：住院号

临床诊断：临床诊断

项目	状态	评分
睁眼反应	☐4分：自动睁眼 ☐3分：听到言语命令时患者睁眼 ☑2分：刺痛时睁眼 ☐1分：刺痛时睁眼	2
运动反应	☑6分：能执行简单口令 ☐5分：刺痛时能指出部位 ☐4分：刺痛时肢体能正常回缩 ☐3分：刺痛时患者身体出现异常屈曲（去皮质状态） （上肢屈曲、内收内旋，下肢伸直、内收内旋，踝跖屈） ☐2分：捏痛时患者身体出现异常伸直（去大脑强直） （上肢伸直、内收内旋，腕指屈曲、下肢伸直，内收内旋、踝跖屈） ☐1分：刺痛时患者毫无反应	6
语言反应	☐5分：能正确回答问题 ☐4分：言语错乱，定向障碍 ☐3分：说话能被理解，但毫无意义 ☑2分：能发声，但不能被理解 ☐1分：不发生	2
总分：	10	
功能状态：	中度损伤	

注：GCS量表总分范围3-15分：

≤8分重度损伤，预后差；

9-11分中度损伤；

≥12分轻度损伤；

数值越低，预示病情越重。

已上面截图为例，通过勾选单选框组得到单个评分，然后根据各个评分得到总分，然后根据总分不同区间，自动进行状态评估。

27. 内容复制来源

内容复制来源对应的是输入域属性中的 CopySource，通过设置改属性，就可以控制一个文档中存在相同元素直接的内容进行同步，比如文档中存在两个姓名元素，需要当第一个姓名输入域元素输入了姓名，第二个姓名输入域元素自动带入上一个姓名输入域中的内容，反之依然，只需要在当前输入域属性对话框中的复制来源（SourceID）填入对方元素的编号即可。

28. 自定义属性

自定义属性对应的是输入域属性中的 Attributes，当输入域中自带的属性满足不了客户业务需求的时候，就可以在自定义属性中定义业务需要的数据，然后通过获取自定义属性来进行不同业务方面的处理。

上述就是关于输入域元素常用的功能介绍，需要更细更全的属性介绍，请参考五代编辑器文档 10.3 附录.输入域篇，或者联系南京都昌技术工作人员进行反馈。

3. 输入域元素常用方法

1. 通过编号获取输入域对象

```
var ctl = document.getElementById("myWriterControl");
```

```
let result = ctl.GetElementById("姓名 2")//通过传元素编号获取指定输入域对象
//然后可以根据返回值对指定元素进行取值赋值操作
```

2. 获取文档中所有的输入域对象

```
var ctl = document.getElementById("myWriterControl");
let result = ctl.GetAllInputFields(false, false, "ZSK_YiBanQK")
        //false  非必填   Bool      是否排除只读元素
      * false      非必填   Bool      是否排除隐藏元素
      * ZSK_YiBanQK     非必填    可以传父容器对象，获取指定容器下的所有输入域（病程）
```

3.对文档中相同 ID 的输入域元素赋值

```
var ctl = document.getElementById("myWriterControl");
let result = ctl.GetElementsById("姓名 2")//获取到 id 相同的输入域集合
var resultj=ctl.GetElementProperties(resultj[0]）//获取集合中某一个的元素属性
let result2 = ctl.SetElementTextByID(resultj.NativeHandle, "新的内容") //设置元素内容
NativeHandle 属性是元素的唯一值，确保修改是需要元素
```

4.获取指定元素的内容（Text 值）

```
  var ctl = document.getElementById("myWriterControl");
  let result = ctl.GetElementTextByIDPro('field1') //获取元素的内容，只返回用户输入的内
容
```

5.通过给下拉输入域赋上对应选项的 Value 界面显示 Text 值

```
var ctl = document.getElementById("myWriterControl");
let result = ctl.SetFieldDropListItemByValue('yyds', '2222')//id,value 值.下拉输入域赋值
```

6.对输入域进行赋值（Text 和 Value）

```
  let ctl = document.getElementById("myWriterControl");
  var result = ctl.SetElementInnerValueStringByID("field2", "1", "男");
//下拉输入域一般都是有两个值，其中一个是界面上显示的内容 2，一个是实际存储在数
据库的数值。
```

7. 删除指定输入域元素

```
var ctl = document.getElementById("myWriterControl");
1.ctl.DeleteElement("txtAge");//通过传元素编号删除指定输入域
```

8. 输入域元素的数据同步（入院记录中的主诉内容，同步到病程记录中的主诉上）

```
var ctl = document.getElementById("myWriterControl");
var result = ctl.GetElementInnerXmlByID("field1");//获取主诉输入域下对应内容的 xml
   var obj = {"file": xmlstr,//xml 文档
              "format": "xml",//xml 文档格式
               "base64": "false",//是否是 base64 字符串
               "position": "start" }
   ctl.InsertXmlById(obj, 'field1');
//当需要把病程记录下主诉输入域数据同步，通过上述的方法把保存的 xml 数据进行插入
```

9. 获取指定容器下的指定输入域元素

```
var ctl = document.getElementById("myWriterControl");
var result = ctl.GetElementByIdExt("field1", "Sub2");//子元素 id,指定的父元素的 ID
//返回值为元素的唯一值，可以通过返回值对元素进行取值赋值操作
```

10. 设置输入域元素自定义属性

```
var myWriterControl = document.getElementById("myWriterControl")
 var opts = { aa: "DDDDDD",
                    d2: "valuevalue",  };
var result = myWriterControl.SetElementCustomAttributes("field1",opts);//通过元素 id 以及自定义属性对象进行设置自定义属性
```

11. 获取输入域元素自定义属性

```
 var ctl = document.getElementById("myWriterControl");
 let result = ctl.GetElementCustomAttributes("field1");、/通过传元素编号获取指定输入域的自定义属性
```

12. 控制输入域元素编辑显示，但是不进行打印

```
var ctl = document.getElementById("myWriterControl");
var result = ctl.SetElementPrintVisibility("input", "Hidden");//通过元素的 id,可见性的参数如：Visible：打印可见、Hidden：打印不可见、None：打印不可见。进行控制
```

13. 获取指定输入域元素状态（是否被修改）

```
var ctl = document.getElementById("myWriterControl");
let result = ctl.GetElementById("姓名 2")//通过传元素编号获取指定输入域对象
var resultj=ctl.GetElementProperties(result).Modified//获取元素的 Modified 属性查看输入域元素状态
```

14. 控制输入域元素隐藏-显示

```
 var myWriterControl = document.getElementById("myWriterControl");
 var result = myWriterControl.SetElementVisibility("field1", false);//通过 id，visible 是否隐藏，控制
myWriterControl.RefreshDocument()刷新文档进行排版
```

5.3.1.3.2.表格元素

编辑器中创建的表格跟 word 文档中的表格操作类型，都支持插入表格、新增行、删除行、新增列、删除列、单元格合并、拆分、表格行、列拖拽、斜分割线、单元格段落对齐方式等功能，然后在此基础上添加了大量的符合电子病历要求的功能，比如量表的计算、单元格内容跨页镜像（表格行跨页之后，上一页下一页中的单元格内容可以相同）、单元格网格线、表格数据源绑定等功能。

1. 如何创建表格元素

创建表格元素常用的有两个方式：

第一种：通过对话框配置好表格属性进行插入

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand('InsertTable',true,null)
```

效果图：

第二种：先配置好创建表格元素需要的属性，然后插入

```
var  ctl=document.getElementById("myWriterControl");
var options={
     ID:'table1',
     RowCount:2,//行数
     ColumnCount:3//列数
}
ctl.DCExecuteCommand( 'InsertTable' ,false,options)
```

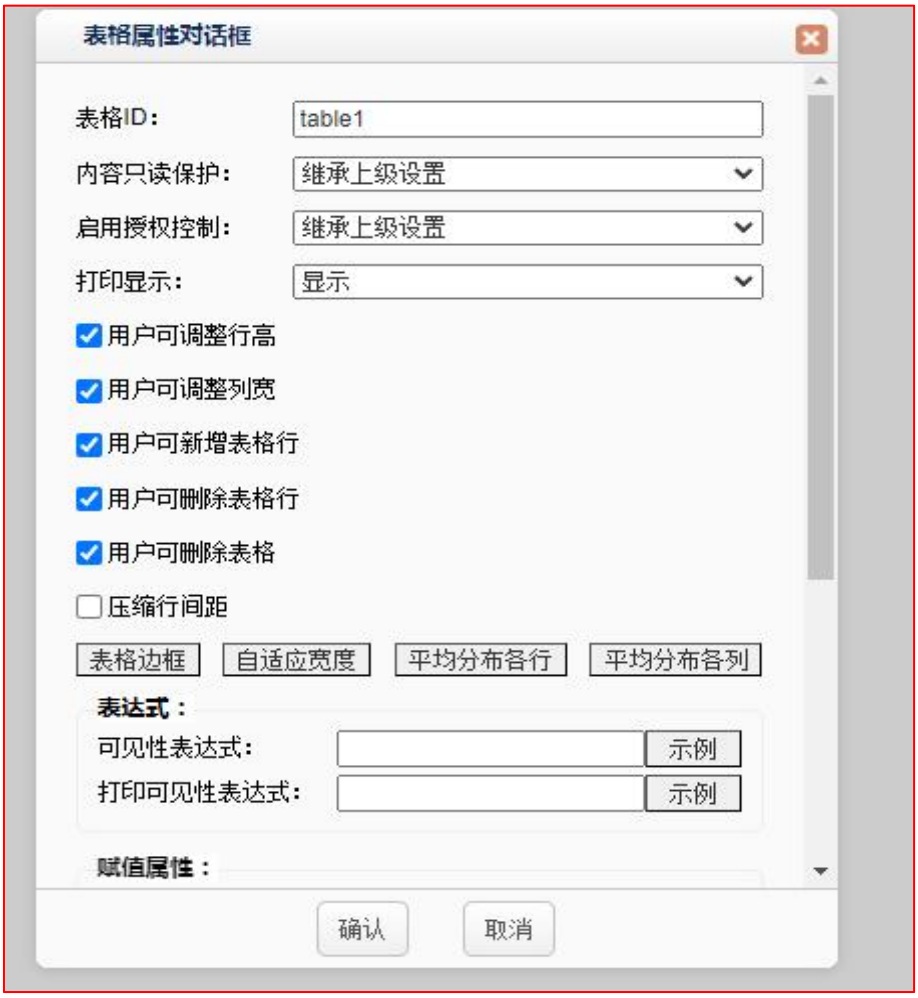
注意：当插入表格时，表格后面多出了一个空白行，需要调用文档选项来把空白行进行删除

```
var  ctl=document.getElementById("myWriterControl")
ctl.DocumentOptions.BehaviorOptions.ParagraphFlagFollowTableOrSection=true;
ctl.ApplyDocumentOptions();
```

2.表格属性对话框

表格内置属性对话框功能，把光标放在表格里面调用编辑器内置命令:

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'TableProperties' ,true,null)
```



1.表格 ID

表格 ID 对应的是表格属性中的 ID,通过 ID 值可以通过接口获取到表格对象。

2. 内容只读保护

内容只读保护对应的是表格属性中 ContentReadonly，效果与输入域的 ContentReadonly 效果一致。处于管理员模式下该属性无效

3. 打印显示

打印显示对应的是表格属性中的 PrintVisibility，效果与输入域的 PrintVisibility 效果一致

4. 用户可调整行高

用户可调整行高对应的是表格属性中的 AllowUserToResizeRows，当该属性为 false 时，无法通过鼠标进行拖拽表格行的高度

5. 用户可调整列宽

用户可调整列宽对应的是表格属性中的 AllowUserToResizeColumns，当该属性为 false 时，无法通过鼠标进行拖拽表格列的宽度

6. 用户可新增表格行

用户可新增表格行对应的是表格属性中的 AllowUserInsertRow，当该属性为 false 的时候，用户无法新增表格行。

7. 用户可删除表格行

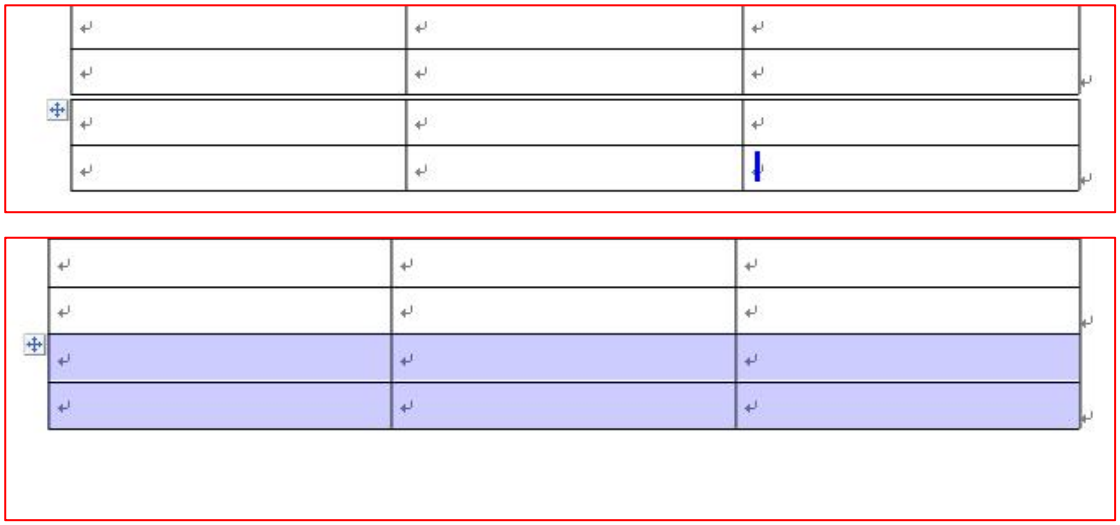
用户可删除表格行对应的是表格属性中的 AllowUserDeleteRow，当该属性为 false 的时候，用户无法删除表格行

8. 用户可删除表格

用户可删除表格对应的是表格属性中的 Deleteable，当该属性为 false 的时候，用户无法通过键盘操作删除表格

9. 压缩行间距

压缩行间距对应的是表格属性中的 CompressOwnerLineSpacing。当该值为默认值 false 的时候，文档中存在两个相邻的表格的时候，会出现一个小的间隙，然后通过这个属性，间隙就会消失，两个表格会连接在一起显示。



10. 可见性表达式：

可见性表达式对应的是表格中的 VisibleExpression，通过该属性可以设置逻辑关系来对表格进行显示隐藏，效果跟输入域一致。

11. 表格自定义属性

表格自定义属性对应的是表格属性中的 Attributes，使用跟输入域的自定义属性效果一致。

12. 表格边框设置

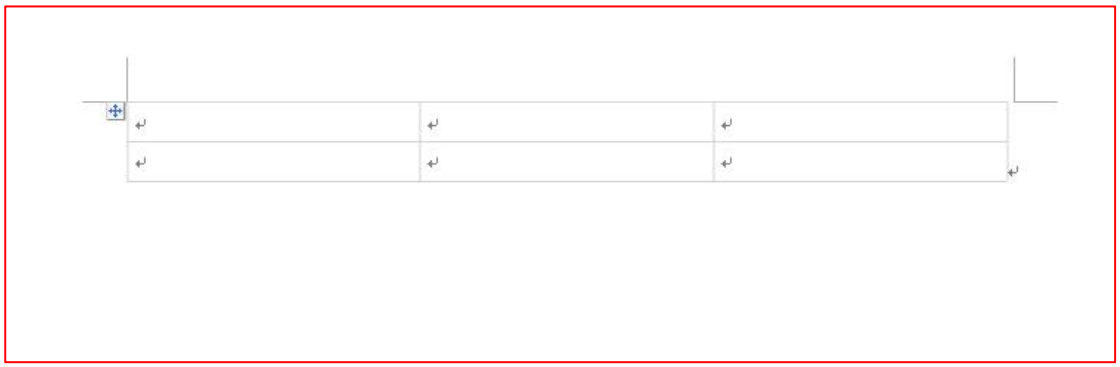
表格边框设置修改的是表格属性中的 Style 样式，编辑器内置表格边框对话框进行修改表格的边框线样式。

调用编辑器内置方法：

```
var ctl=document.getElementById("myWriterControl");
ctl.bordersShadingDialog();
```

表格边框主要分为三块，第一块就是表格边框的四条边框线（上、下、左、右），第二块就是四条边框线的颜色、第三块就是表格边框线的粗细（宽度-单位为 1/300 英寸）。

当用户需要使用表格来进行排版的时候，但是打印的时候不需要打印表格边框线，这个就可以把表格四条边框线的边框不进行勾选，且宽度为 0，这样表格边框线就会隐藏，从而不进行打印边框线。效果图：



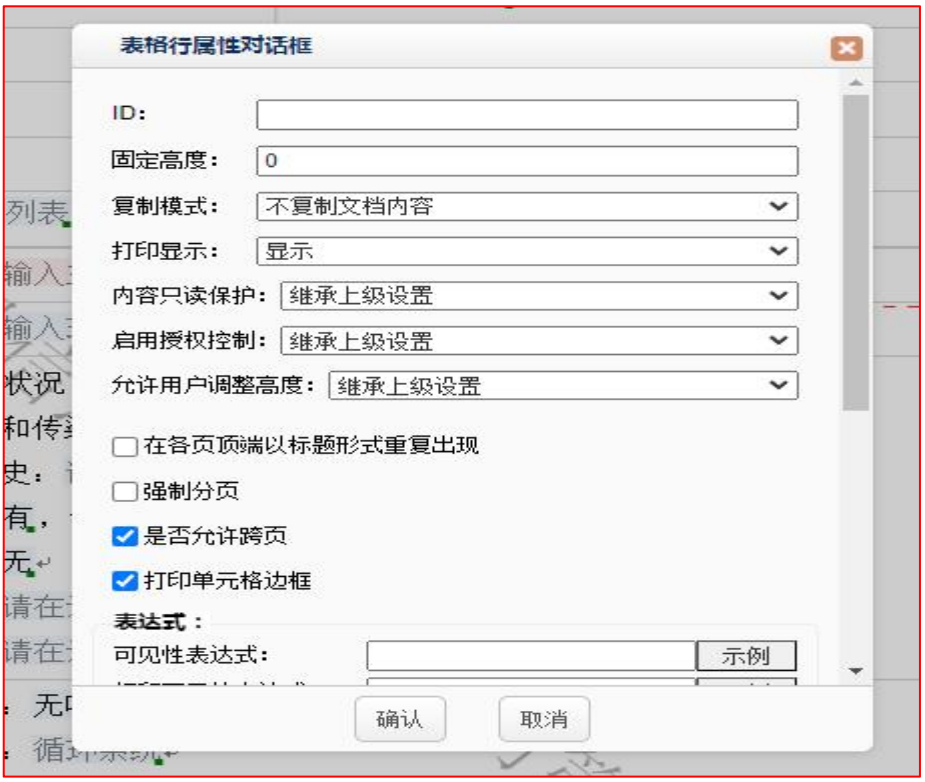
上图就是表格设置成了虚边框，在编辑模式下还是能看到浅灰色的表格边框线，这个是为了提醒用户这块是一个表格，其实是不参与打印的。也可以通过文档选项来控制不显示浅灰色边框。设置文档选项的代码为：

```
var ctl=document.getElementById("myWriterControl");
ctl.DocumentOptions.ViewOptions.ShowCellNoneBorder=false;
ctl.ApplyDocumentOptions()
```

3. 表格行属性对话框

表格行内置属性对话框功能，把光标放在表格行里面调用编辑器内置命令：

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'TableRowProperties' ,true,null)
```



1. ID

ID 对应的是表格行对象下属性的 ID，用户可以通过这个 ID 获取到指定行，然后通过行对象获取到所有的单元格对象，然后根据需要实现不同的业务功能。

2. 固定宽度

固定宽度对应的是表格行对象下属性的 SpecifyHeight，该值可以影响表格行的默认高度，当值为正数的时候，表格行的高度会随着里面的内容的高度变化而变化，当内容过多的时候会自动撑大，内容的高度没有行高大，那表格行的高度不变，但是当值为负数的时候，表格行的高度就固定了，不管里面的内容多少，高度永远不变，内容过多在表格行里面显示不全。

3. 复制模式

复制模式对应的是表格行对象下属性的 CloneType，通过控制该属性中的值，可以当用户在新增表格行的时候，把当前行的样式给同步复制到新行上面。

正常情况下新增的表格行，是一个空白行。

当表格行设置属性 CloneType 为 ContentWithClearField 时，保留输入域样式，但是清空输入域里面的内容，需要保留内容的话，请设置值为 Complete，就是行进行克隆进行新增。

4. 打印显示

打印显示对应的是表格行对象下属性的 PrintVisibility，效果与表格的打印显示效果一致。

5. 内容只读保护

内容只读保护对应的是表格行对象下属性的 ContentReadonly，效果与表格的内容只读效果一致。处于管理员模式下该属性无效

6. 在各页顶端以标题形式重复出现

在各页顶端以标题形式重复出现对应的是表格行对象下属性的 HeaderStyle，表格行设置这个属性之后，当表格进行分页的时候，设置了该属性的表格行在另外页以标题行的方式显示。常用于护理记录单。

7. 强制分页

强制分页对应的是表格行对象下属性的 NewPage，当设置该属性为 true 的时候，改行就会进行强制的换页显示。

8. 是否允许跨页

是否允许跨页对应的是表格行对象下属性的 CanSplitByPageLine，当设置该属性为 false 的时候，当表格行处于页底且内容过多的时候，默认是会上下分离展示，设置成 false，表格行会整体到下一页展示。

9. 可见性表达式

可见性表达式对应的是表格行对象下属性的 VisibleExpression，通过该属性可以设置逻辑关系来对表格行进行显示隐藏，效果跟表格一致。

10. 使用快捷键在下面插入新增行

使用快捷键在下面插入新增行对应的是表格行对象下属性的 AllowInsertRowDownUseHotKey，默认是只能在表格最后一行最后一个单元格按下 tab 键才能新增行，可以修改属性值为 EnableInAllCases，这样在表格任意一行最后一个单元格按下 tab 键都能进行新增表格行。

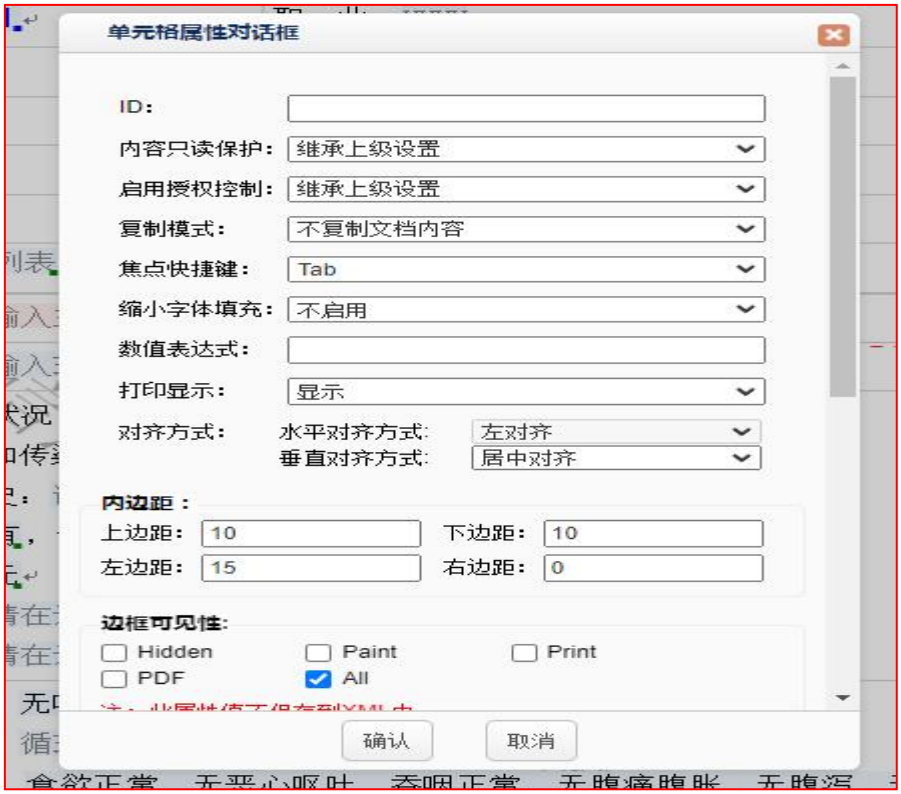
11. 表格行自定义属性

表格行自定义属性对应的是表格行对象下属性的 Attributes，效果使用跟表格自定义属性一致。

4. 单元格属性对话框

单元格内置属性对话框功能，把光标放在单元格里面调用编辑器内置命令：

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'TableCellProperties' ,true,null)
```



1. ID

ID 对应的是单元格对象下属性的 ID，用户可以通过这个 ID 获取到指定单元格，然后通过单元格对象根据需要实现不同的业务功能。

2. 复制模式

复制模式对应的是单元格对象下属性的 CloneType，通过控制该属性中的值，可以当用户在新增表格行的时候，把当前单元格的样式给同步复制到新行对应的单元格上面。

正常情况下新增的表格行，单元格里面是空白的。

【年龄：年龄岁】	【年龄：年龄岁】	【年龄：年龄岁】

当单元格设置属性 CloneType 为 ContentWithClearField 时，保留输入域样式，但是清空输入域里面的内容，需要保留内容的话，请设置值为 Complete，就是单元格进行克隆进行新增。

【年龄：111岁】	【年龄：111岁】	【年龄：111岁】
【年龄：年龄岁】	【年龄：年龄岁】	【年龄：年龄岁】

3. 打印显示

打印显示对应的是单元格对象下属性的 PrintVisibility，效果与表格的打印显示效果一致。

4. 内容只读保护

内容只读保护对应的是单元格对象下属性的 ContentReadonly，效果与表格的内容只读效果一致。处于管理员模式下该属性无效

5. 焦点快捷键

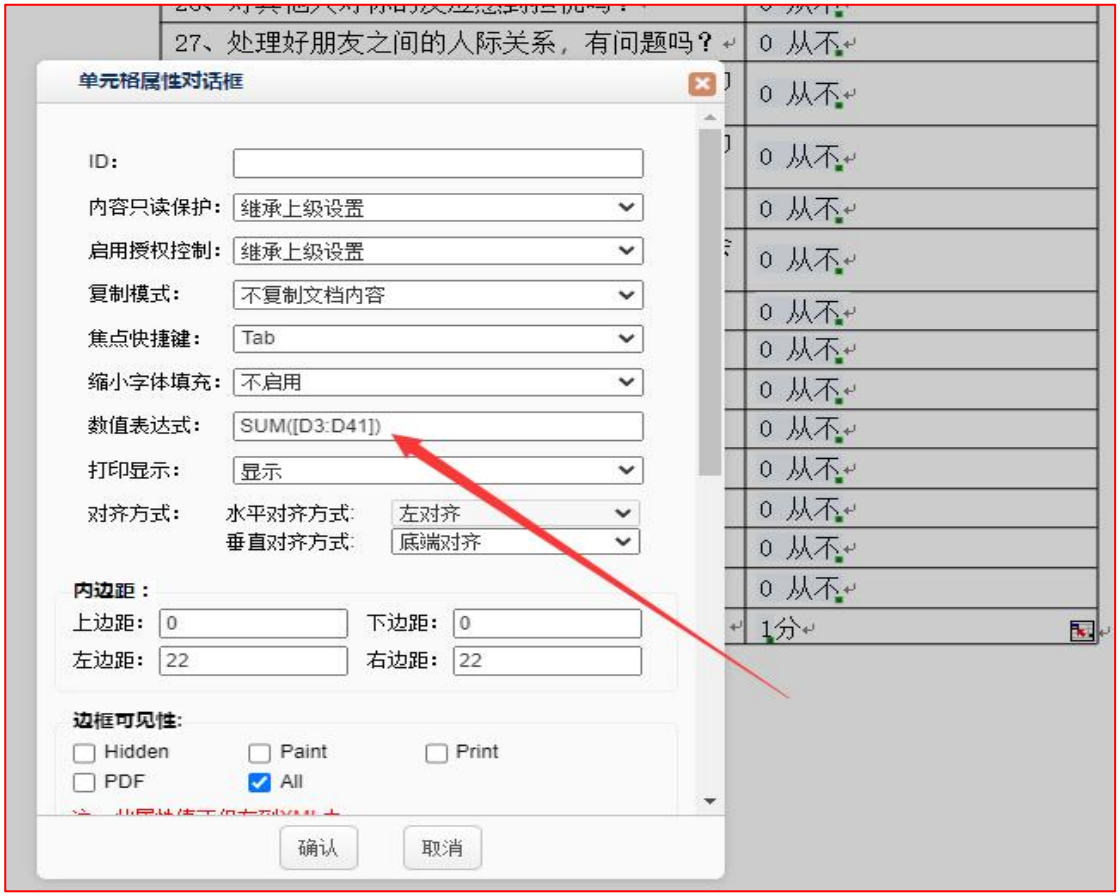
焦点快捷键对应的是单元格属性中的 MoveFocusHotKey。当用户需要把光标从一个单元格调整到下一个单元格里面，就可以通过设置该属性来进行处理。默认支持：Tab 键、Enter 键切换。很多用户需要使用 enter 键来进行切换的时候，就可以修改值为 Enter 键。

6. 缩小字体填充

缩小字体填充对应的是单元格属性中的 AutoFixFontSizeMode，当用户需要再对应的单元格中输入内容，过多的时候也不需要撑大单元格的高度，比如检查报告单有要求只能是一页展示，当检查过多的时候，单元格的高度就会撑高，满足不了一页展示的需求，这个时候就可以设置该属性来自动缩小输入的内容。

7. 数值表达式

数值表达式对应的是单元格对象下属性的 ValueExpression，用户通过这个属性可以配置好护理评估量表对应的模板要求。



通过 SUM([D3:D41])，就是表示从第四列中的第 3 个单元格一直计算到第四列的第 41 一个单元格，D 表示单元格的背景编号，后面的数据表示这列的第几个单元格。通过调用命令可以在界面上显示单元格背景编号：

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand(‘ShowBackgroundCellID’,true,null)
```

8. 单元格内容跨页镜像

单元格内容跨页镜像对应的是单元格对象下属性的 MirrorViewForCrossPage，通过设置该属性当表格行进行了跨页是，可以把上一页单元格的内容，复制一份到下一页的相同单元格里面，可以让用户感觉到上下页的表格行其实是一个。

9. 单元格边框设置

单元格边框设置修改的是当前光标处单元格属性中的的 Style 样式，编辑器内置单元格边框对话框进行修改单元格的边框线样式。

调用编辑器内置方法：

```
var ctl=document.getElementById("myWriterControl");
ctl.borderShadingcellDialog();
```

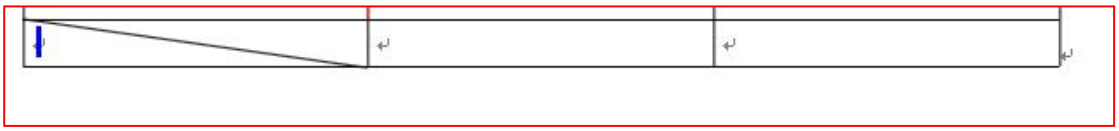
可以根据不同的需求，对单元格四条边框线进行边框线的样式修改，但是主要设置的单

元格存在相邻的边框线，需要另外一个单元格也需要设置相应的边框线样式，否则无效

提示：用户可以通过鼠标选中多个单元格设置单元格的边框样式。

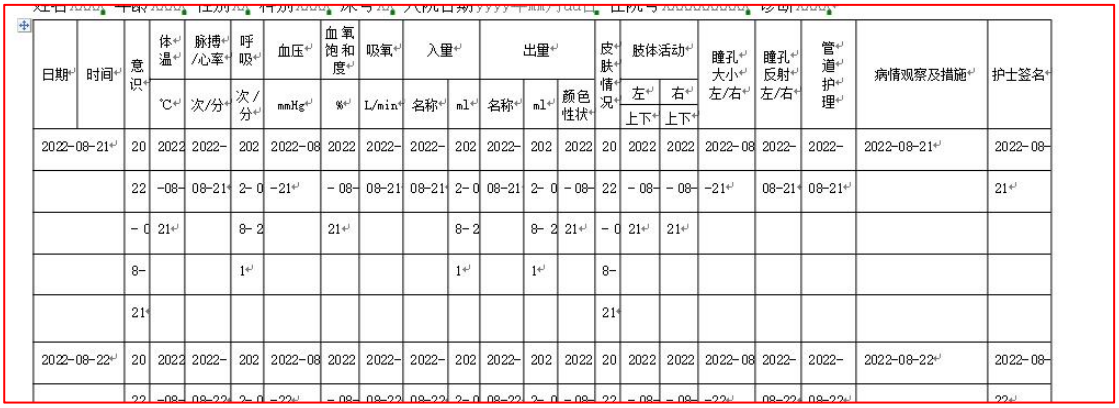
10. 单元格斜分线

单元格斜分线对应的是单元格对象下的属性 SlantSplitLineStyle，通过设置该属性可以设置单元格的斜分线的样式。



11. 单元格网格线

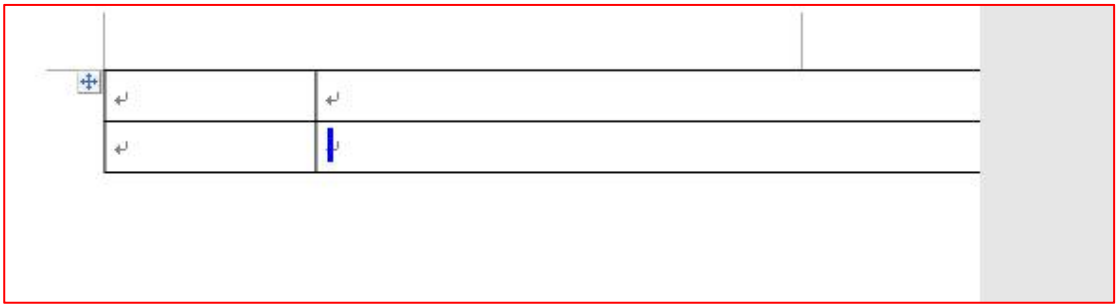
单元格单元格网格线对应的是单元格对象下的属性 GridLine，通过设置该属性，当单元格里面的内容宽度多于单元格的宽度的时候，会通过网格线模拟出一个新的单元格，当整行中所有的单元格都设置了网格线的话，这样当内容换行的时候，看起来是创建了一个新行，但是其实内容还是处于一个单元格里面的。常用于护理记录单和医嘱单上面



日期	时间	意识	体温	脉搏/心率	呼吸	血压	血氧饱和度	吸氧	入量		出量			皮肤情况	肢体活动		瞳孔大小左/右	瞳孔反射左/右	管道护理	病情观察及措施	护士签名
			℃	次/分	次/分	mmHg	%	L/min	名称	ml	名称	ml	颜色性状		左上下	右上下					
2022-08-21		20	2022	2022-	202	2022-08	2022	2022-	2022-	202	2022-	202	2022	20	2022	2022	2022-08	2022-	2022-	2022-08-21	2022-08-
		22	-08-	08-21	2-0	-21	-08-	08-21	08-21	2-0	08-21	2-0	-08-	22	-08-	-08-	-21	08-21	08-21		21
		-0	21		8-2		21			8-2		8-2	21	-0	21	21					
		8-			1					1		1		8-							
		21												21							
2022-08-22		20	2022	2022-	202	2022-08	2022	2022-	2022-	202	2022-	202	2022	20	2022	2022	2022-08	2022-	2022-	2022-08-22	2022-08-
		22	-08-	08-22	2-0	-22	-08-	08-22	08-22	2-0	08-22	2-0	-08-	22	-08-	-08-	-22	08-22	08-22		22

12. 单元格自适应宽度

单元格自适应宽度功能适用于当模板为 A4 的时候里面带有表格，带有把 A4 模板修改成 A5 模板，这个时候表格的宽度不会自适应的，有的单元格就会超出页面的边距，这个时候就需要调用还方法进行自适应宽度。



调用的代码为：

```
var ctl=document.getElementById("myWriterControl");
  ctl.AutoFixTableWidth();
```

对应表格行新增、删除。表格列新增、删除、单元格合并、拆分、单元格对齐方式等功能，参考 9.前端命令。

5.表格常用方法

1. 设置表格边框

1.可以通过弹出表格边框对话框设置表格边框

```
let ctl = document.getElementById("myWriterControl")
 ctl.bordersShadingDialog();
```

2. 修改表格边框样式设置表格边框。如果第一个参数为空，表示设置当前表格。

```
let ctl = document.getElementById("myWriterControl")
var ele=ctl.CurrentTable();//获取当前光标处表格对象
let Style = {
    BorderBottom: true,
    BorderBottomColorString: "#000000",
    BorderLeft: true,
    BorderLeftColorString: "#000000",
    BorderRight: true,
    BorderRightColorString: "#000000",
    BorderStyle: "Solid",
    BorderTop: true,
    BorderTopColorString: "#000000",
    BorderWidth: 0,
}
ctl.SetTableBorder(ele, Style)
```

2. 设置单元格边框

1.可以通过弹出单元格边框对话框设置边框样式

```
let ctl = document.getElementById("myWriterControl")
 ctl.borderShadingcellDialog();
//修改当前光标处单元格边框，也可以先选中多个单元格，在弹出单元格对话框进行设置
```

2.设置指定单元格的边框。如果根据第一个参数无法获取单元格，表示设置当前单元格。

```
 let ctl = document.getElementById("myWriterControl")
var ele=ctl.CurrentTableCell();//获取当前光标处单元格对象
let Style = {
    BorderBottom: true,
    BorderBottomColorString: "#000000",
    BorderLeft: true,
    BorderLeftColorString: "#000000",
    BorderRight: true,
    BorderRightColorString: "#000000",
    BorderStyle: "Solid",
    BorderTop: true,
    BorderTopColorString: "#000000",
    BorderWidth: 0,
}
```

```
ctl.SetTableCellBorder(ele, Style)
```

3. 新增表格行（根据行索引处插入）

在指定表格行下面新增表格行（根据第三个参数可以插入多行）

```
let ctl = document.getElementById("myWriterControl")
ctl.DCExecuteCommand("Table_InsertRowDown", false, {
      tableid: "table1",//表格 id
      rowindex: 1,//表格的索引 0 开始计算
      rowcount: 3//插入的行数
});
```

4. 获取表格属性

获取当前表格的后台引用对象，通过后台引用对象获取当前表格属性，也可以传入表格的 id

```
 var myWriterControl = document.getElementById("myWriterControl");
  var currenttable = myWriterControl.CurrentTable();//获取当前光标处的表格对象
 var result = myWriterControl.GetElementProperties(currenttable);
```

5. 获取表格行属性

获取当前表格行的后台引用对象，通过后台引用对象获取当前表格行属性，也可以传入表格行的 id

```
var myWriterControl = document.getElementById("myWriterControl");
var currentrow = myWriterControl.CurrentTableRow();//获取当前光标处的表格行对象
 var result = myWriterControl.GetElementProperties(currentrow);
```

6. 获取单元格属性

获取当前单元格的后台引用对象，通过后台引用对象获取当前单元格属性，也可以传入单元格的 id

```
var myWriterControl = document.getElementById("myWriterControl");
var currentcell = myWriterControl.CurrentTableCell();//获取当前光标处的单元格对象
var result = myWriterControl.GetElementProperties(currentcell);
```

7. 对指定单元格进行赋值

将文本内容传入到指定的单元格中

```
var ctl = document.getElementById("myWriterControl");
let res= ctl. SetTableCellText("table1", 0, 0, "newtext")
/方法第一个参数为表格编号，第二个参数为行索引、第三个参数为列索引，第三个参数为单元格赋值的内容
```

8.对表格、表格行、单元格-设置-获取自定义属性

1. 获取当前表格的后台引用对象，通过后台引用对象获取当前表格的自定义属性，也可以传入表格的 id

```
var ctl = document.getElementById("myWriterControl");
var table = ctl.CurrentTable();
var result = ctl.GetTableAttribute(table);
```

2。获取当前表格行的后台引用对象，通过后台引用对象获取当前表格行属性，也可以传入表格行的 id

```
var ctl = document.getElementById("myWriterControl");
```

```
var row = ctl.CurrentTableRow();
 var result = ctl.GetTableRowAttribute(row);
```

3. 获取当前单元格的后台引用对象，通过后台引用对象获取当前单元格属性，也可以传入单元格的 id

```
var ctl = document.getElementById("myWriterControl");
var cell = ctl.CurrentTableCell();
var result = ctl.GetTableCellAttribute(cell)
```

4. 获取当前表格行的后台引用对象，通过后台引用对象设置当前表格自定义属性，也可以传入表格行的 id

```
var ctl = document.getElementById("myWriterControl");
var table = ctl.CurrentTable();
var opts = {
   t1: "TDDDDDD",
   t2: "valuevalue",
   tA: "myAA"
};
var result = ctl.SetTableAttribute(table, opts);
```

5.获取当前表格行的后台引用对象，通过后台引用对象设置当前表格行自定义属性，也可以传入表格行的 id

```
var ctl = document.getElementById("myWriterControl");
var row = ctl.CurrentTableRow();
var opts = {
   R1: "rDDDDDD",
   r2: "valuevalue",
   rA: "myAA"
};
var result = ctl.SetTableRowAttribute(row, opts);
```

6. 获取当前表格行的后台引用对象，通过后台引用对象设置当前单元格自定义属性，也可以传入表格行的 id

```
var ctl = document.getElementById("myWriterControl");
var cell = ctl.CurrentTableCell();
var opts = {
    c1: "cDDDDDD",
    c2: "valuevalue",
    cA: "myAA"
};
var result = ctl.SetTableCellAttribute(cell, opts);
```

9. 显示单元格背景编号（用来单元格计算数值表达式）

1.显示单元格背景编号。执行后会在文档中的单元格内容显示单元格的编号信息

```
var ctl = document.getElementById("myWriterControl");
ctl.ShowBackgroundCellID ()
```

10. 设置表格-表格行书写过程中显示，打印隐藏

获取当前表格行的后台引用对象，设置元素是否打印时可见。打印预览、打印该元素就会隐藏不可见

```
var ctl = document.getElementById("myWriterControl");
var row = ctl.CurrentTableRow();
let res=ctl.SetElementPrintVisibility(row, "Hidden");
```

5.3.1.3.3. 单选框元素

1. 如何创建单选框元素

通过以下代码插入单选框元素

```
var ctl=document.getElementById("myWriterControl");
    var options = {
        "Name": "name001", //单选框的 Name 属性相同
        "Type": "radio",
        "ListItems": [
          {
            "ID": "name001-1",
            "ToolTip": "提示信息",
            "Text": "内容 1", "Value": "值 1"
          },
          {
            "ID": "name001-2",
            "ToolTip": "提示信息", "Text": "内容 2",
            "Value": "值 2"
          },
          {
            "ID": "name001-3",
            "ToolTip": "提示信息", "Text": "内容 3",
            "Value": "值 3"
          }
        ]
      };
ctl.DCExecuteCommand("insertcheckboxorradio", false, options)
```

上述的代码是插入一个单选框组，多个单选框元素中的 Name 属性使用的是相同值，那么这几个单选框就表示一组，可以进行互斥。

2. 单选框属性对话框

单选框内置属性对话框功能，鼠标左击下单选框元素然后调用编辑器内置命令：

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'ElementsProperties' ,true,null)
```

1. 编号

编号对应的是单选框属性中的 ID（命名的时候需要符合标识符规范，因为有的数值表达式会通过单选框的 ID 进行计算的，不符合规范的话，表达式会不生效），然后也可以通过元素的 ID 进行勾选或者取消勾选。

2. 名称

名称对应的是单选框属性中的 Name，多个单选框的 Name 值相同表示为一组，然后勾选会产生互斥效果，且对量表计算只需要写对应的 Name 值就行。比如输入域中数值表达式填写[Sex]值，那勾选不同的性别，得出对应数值。

3. 文本

文本对应的是单选框属性中的 Text，设置该属性也就是界面上显示的内容。

4. 数值

数值对应的是单选框属性中的 InnerValue，用户实际存储的值。

5. 可以删除

可以删除对应的是单选框属性中的 Deleteable，设置该属性就可以控制该元素是否能被键盘删除。处于管理员模式下该属性无效。

6. 对象是否可用

对象是否可用对应的是单选框属性中的 Enabled，设置该属性为 false，用户无法进行勾选

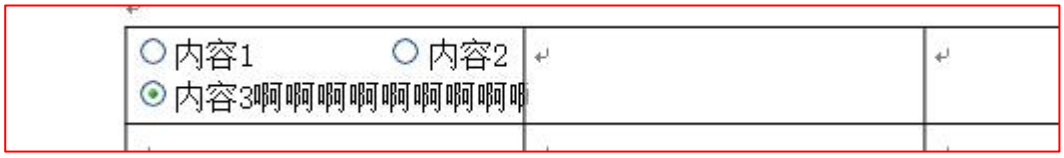
7. 必勾项

必勾项对应的是单选框属性中的 Requried，设置该属性为 true 时，该元素没有勾选的

时候，调用文档校验的方法会校验出来。注意：单选框需要设置 Name 值才能起效果

8. 文档参与流式排版

文档参与流式排版对应的是单选框属性中的 CaptionFlowLayout，当没有设置该属性的时候，单选框的文本长度超出了父容器对象的宽度的时候，就会显示不全.



9. 数据源

数据源对应的是单选框属性中的 ValueBinding 对象下的 DataSource，该属性可以随意定义（需要符合标识符规范），然后可以把相同类型的结构化元素归为一类，设置成同一个数据源名称。

10. 绑定路径

绑定路径就是对应单选框属性中的 ValueBinding 对象下的 BindingPath，该属性可以随意定义（需要符合标识符规范），该属性定义最好能跟数据库存储的数据字段有对应关系。然后通过前端生成的 json 数据中 key 值需要进行对应。绑定的方式跟输入域一模一样，当绑定的值为某个单选框的 InnerValue 值，那这个单选框就自动勾选。

11. 自定义属性

自定义属性对应的是单选框属性中的 Attributes，使用效果跟输入域的 Attributes 一样。

3. 单选框常用方法

1. 自定义单选框勾选框样式

默认样式：


```
 var ctl = document.getElementById("myWriterControl");
    let keyName = "RadioBoxChecked"
let xmlText =
"iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAAf8/9hAAAAAXNSR0IArs4c6QAAA
SBJREFUOE+NU0GOgzAMHKs9VGou+UH7ki0vWXjJsi8p+5JmX1J+wCVIPYBcHEiaoFYhtyT
jGXtsE1anOxxO2O+/I+cLmE8ADIB/jKPRj0cbh1B86ZS6ALitScOduQVRo6399W+BoFNKAo
Vgy6k9iSPIKr+jZK503zeegLfIJhgpZxwL6o7HEkTXLMFS/4SrI2wtBFcQISujxPnXYW5135+XD
t2jHyME98nZEDCZQ51SP0HpczDA3Ap47b5z2JEwlx+UfRlug7SE+Su06U3acWlNmm7qZl1
h+JumMq45RTBX5BR2u1vsQ7YjHjAM53kOtrYyZS60tSYe5Zfz+RSMtrYQWLpMuUzmYapE
2WskBK6c2RNZqi/ni6wzUSN/8RZ6gieVU4ZOnRIsqwAAAABJRU5ErkJggg==";
    ctl.SetDomImageByBase64String(keyName, xmlText);//修改单选框勾选状态样式
        let keyName1 = "RadioBoxUnchecked";
        let xmlText1 =
"iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAAf8/9hAAAAAXNSR0IArs4c6QAAA
```

```
TFJREFUOE+NU0GSgyAQ7KlwSFW48APzko0v2fiSuC/Rn4R9SfwBF6zKQYrsIBg0bil30Onp6
e4hLI7Z7wsl8R2fT/C+AKAB/Ml5re73Li+h/GKkPAG4Tm/edyBigPGM91ZZ+5OeJgAjJRcyAJ
86/yljVS+/B4Css8YwVEuaqVsA2u2ugZX3ler7NgH48NMwHP8rnoEIcQvjOFeSORzOIGoS4I
LUtftUA9QM0IDovKX7CwtAM8CNZ/oTbebIJyZGSs9jUFDf+0L1/fFT0czysXH3HAEoIbUcmE
0nMmiZwYV9X3r/DiUXniZvucK5crONAFi3MQdPK4O3b4MkRBMTG0bOozyOEvPOy5M0
WYmyVtaW3Hy+TMwEuEwLtL5MVS72i/dRE16qr5j5gjeQu+ULlkR+AFzmry+yrwaLAAAAA
ElFTkSuQmCC";
                ctl.SetDomImageByBase64String(keyName1, xmlText1);//修改单选框取消勾
选状态样式
```

修改之后样式：

注意：自定义样式的图片数据必须为 16*16 尺寸的

2. 指定单选框进行勾选-或者取消勾选

```
//对指定 ID 的单选框进行勾选
var ctl=document.getElementById("myWriterControl");
ctl.SetElementChecked("单选框编号",true)
//对指定 ID 的单选框取消勾选
var ctl=document.getElementById("myWriterControl");
ctl.SetElementChecked("单选框编号",true)
```

3. 获取一组单选框对象列表

```
var ctl=document.getElementById("myWriterControl");
ctl.GetAllCheckboxOrRadio('radio','单选框 Name 值')
//获取一组指定 Name 的单选框组合列表,然后循环对象，实现逻辑功能
```

5.3.1.3.4. 复选框元素

1. 如何创建复选框元素

通过以下代码插入单选框元素

```
var ctl=document.getElementById("myWriterControl");
      var options = {
            "Name": "hobby", //复选框的 Name 属性相同
            "Type": "checkbox",
            "ListItems": [
              {
                  "ID": "hobby-1",
                  "ToolTip": "提示信息",
                  "Text": "篮球",
                  "Value": "1"
              },
              {
```

```
                "ID": "hobby-2",
                "ToolTip": "提示信息",
                 "Text": "足球",
                "Value": "2"
            },
            {
                "ID": "hobby-3",
                "ToolTip": "提示信息",
                "Text": "排球",
                "Value": "3"
            }
        ]
    };
ctl.DCExecuteCommand("insertcheckboxorradio", false, options)
```

上述的代码是插入一个复选框组，多个复选框元素中的 Name 属性使用的是相同值，那么这几个复选框就表示一组。

2. 复选框属性对话框

复选框内置属性对话框功能，鼠标左击下单选框元素然后调用编辑器内置命令：

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'ElementsProperties' ,true,null)
```

1. 编号

编号对应的是复选框属性中的 ID（命名的时候需要符合标识符规范，因为有的数值表达式会通过复选框的 ID 进行计算的，不符合规范的话，表达式会不生效），然后也可以通过元素的 ID 进行勾选或者取消勾选。

2. 名称

名称对应的是复选框属性中的 Name，多个复选框的 Name 值相同表示为一组，然后勾选会产生互斥效果，且对量表计算只需要写对应的 Name 值就行。比如输入域中数值表达式填写 SUMINNERVALUE([Hobby])值，那勾选不同的爱好，会得出多个爱好对应的数值总和。

3. 文本

文本对应的是复选框属性中的 Text，设置该属性也就是界面上显示的内容。

4. 数值

数值对应的是复选框属性中的 InnerValue，用户实际存储的值。

5. 可以删除

可以删除对应的是复选框属性中的 Deleteable，设置该属性就可以控制该元素是否能通过键盘删除。处于管理员模式下该属性无效。

6. 对象是否可用

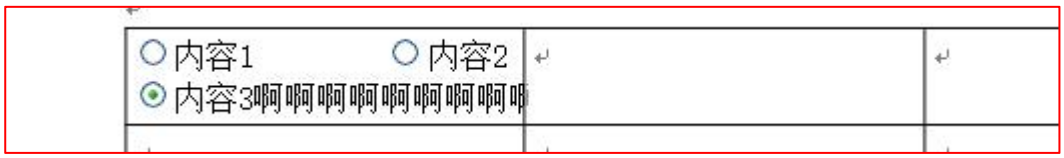
对象是否可用对应的是复选框属性中的 Enabled，设置该属性为 false，用户无法进行勾选

7. 必勾项

必勾项对应的是复选框属性中的 Requried，设置该属性为 true 时，该元素没有勾选的时候，调用文档校验的方法会校验出来。注意：复选框需要设置 Name 值才能起效果

8. 文档参与流式排版

文档参与流式排版对应的是复选框属性中的 CaptionFlowLayout，当没有设置该属性的时候，复选框的文本长度超出了父容器对象的宽度的时候，就会显示不全.



9. 数据源

数据源对应的是复选框属性中的 ValueBinding 对象下的 DataSource，该属性可以随意定义（需要符合标识符规范），然后可以把相同类型的结构化元素归为一类，设置成同一个数据源名称。

10. 绑定路径

绑定路径就是对应复选框属性中的 ValueBinding 对象下的 BindingPath，该属性可以随意定义（需要符合标识符规范），该属性定义最好能跟数据库存储的数据字段有对应关系。然后通过前端生成的 json 数据中 key 值需要进行对应。绑定的方式跟输入域一模一样，当绑定的值为多个复选框的 InnerValue 值通过英文逗号分隔开来，那对应的复选框就自动勾选。

11. 自定义属性

自定义属性对应的是复选框属性中的 Attributes，使用效果跟输入域的 Attributes 一样。

3. 复选框常用方法

1. 自定义复选框勾选框样式

默认样式：


```
var ctl = document.getElementById("myWriterControl");
        let keyName = "CheckBoxChecked";
                                                    let xmlText =
"iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAAf8/9hAAAAAXNSR0IArs4c6QAAA
LJJREFUOE/VU8ENgzAMPAseSOSTDRgI3aSZpDAJbNJu0myQD/khuXIEVavSQFA/9SeKdHf
x+WLyShkw9yBqkFPMDkBHXqkrAJPDfWKZnQjwIfJM+i+Bdu56OeN1nwVmq0MYfF2fQdS
/zmxN4IRpcijLewQuZIkbkMTe6lOAedAhWF9VDYrCxJe/kFMWWj2OnQBS5K0ZrA5t20Lmr
9qXQkL0JwLHIwm4SQeS7yV7I2WdiewD45dQr2m9ErYAAAAASUVORK5CYII=";
        ctl.SetDomImageByBase64String(keyName, xmlText);//修改复选框勾选样式
        let keyName1 = "CheckBoxUnchecked";
                                                    let xmlText1 =
"iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAAf8/9hAAAAAXNSR0IArs4c6QAAA
```

```
KNJREFUOE9jZKAQMML0v+fgUGBgYYln+P9fgaCZf/82Cv748QCkDmwAVPN8hj9/EmESu
AwBq2VmrmdgZFwo+OXLAYgB3NzzYQIEbUdYGC/45Usj3ADBr18TidEM9zI393yQnlEDhlE
Y8PDUMzAwHASILGLSwnseHgeG///jEekAkg/mA5N2lyFD4MkeqhY1M4HSOCHAyPiA4c+f
hSiZiZAefPIAdjZ7EaCuwKQAAAAASUVORK5CYII=";
            ctl.SetDomImageByBase64String(keyName1, xmlText1);//修改复选框取消勾选
样式
```

修改之后样式:

注意: 自定义样式的图片数据必须为 16*16 尺寸的

2. 指定复选框进行勾选-或者取消勾选

```
//对指定 ID 的复选框进行勾选
var ctl=document.getElementById("myWriterControl");
ctl.SetElementChecked("复选框编号",true)
//对指定 ID 的复选框取消勾选
var ctl=document.getElementById("myWriterControl");
ctl.SetElementChecked("复选框编号",true)
```

3. 获取一组复选框对象列表

```
var ctl=document.getElementById("myWriterControl");
ctl.GetAllCheckboxOrRadio('checkbox','复选框 Name 值')
//获取一组指定 Name 的复选框组合列表,然后循环对象, 实现逻辑功能
```

5.3.1.3.5. 医学表达式

编辑器内置多个医学表达式, 比如: 月经史公式、光定位图、恒牙牙位图、乳牙牙位图、光定位图、电活力测试牙位图、眼球突出度、病变上、下牙位图等等。编辑器支持的所有牙位图请登录五代演示程序官网上面进行详细的查看了解。

1. 如何创建不同的医学表达式

创建牙位图的方法只有一个, 然后根据方法里面参数的不同来插入不同类型的医学表达式

```
var ctl = document.getElementById("myWriterControl");
            var options = {
                "ID": "expression1",
                "ExpressionStyle": "FourValues1",
                "Type": "XTextNewMedicalExpressionElement",
                "FontSize": "12",
                "Width": "112px",
                "Height": "46px",
                "Values": "Value1:14;Value2:14;Value3:14;Value4:14",
                “AutoSize”:true
```

```
        };
        ctl.DCExecuteCommand("insertmedicalexpression", true, options);
```

其中 DCExecuteCommand 中的第一个参数就是插入医学表达式，然后根据 options 对象中的 ExpressionStyle 属性值来控制插入不同的医学表达式，然后 AutoSize 默认为 true，对医学表达式自适应整个宽度，不需要用户手动拖拽表达式的大小来进行适应。

其中 ExpressionStyle 是个枚举值

5.3.1.3.6.标签文本元素

有的用户需要把病历设置成全结构化元素。比如对于基本元素：姓名：姓名输入域。是这样的组合，但是姓名两个字是纯文本，且比较难于控制是否能被删除，这个时候就可以使用标签文本元素来代替。且还能使用在病程转科功能上面，在病程章进行详细介绍

1 如何创建标签文本元素

创建标签文本元素有两种方式：

第一种通过弹出标签文本元素对话框配置好对应属性进行插入

```
var ctl = document.getElementById("myWriterControl");
ctl.DCexecuteCommand('InsertLabelElement',true,null)
```

第二种就是通过代码配置好对应的属性，然后直接通过方法进行插入

```
var ctl = document.getElementById("myWriterControl");
var options = {
 ID: "label1",
 Text: "测试用标签",
 Bold: true,
 Deletable: false,
  };
 ctl.DCExecuteCommand("InsertLabelElement", false, options);
```

2.标签文本属性对话框

标签文本内置属性对话框功能，鼠标左击下标签文本然后调用编辑器内置命令：

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'ElementsProperties' ,true,null)
```

1. 编号

编号对应的是标签文本元素属性中的 ID（命名需要符合标识符规范），可以通过该值获取到对象，根据对象在不同业务场景下获取对应的属性值。

2. 名称

名称对应的是标签文本元素属性中的 Name。

3. 能否被删除

能否被删除对应的是标签文本元素属性中的 Deleteable,设置该值为 false，用户不允许键盘删除该元素。处于管理员模式下该属性无效

5. 连接模式设置

连接模式设置实际应用场景在病程记录的转科。

6. 文本

文本对应的是标签文本属性中的 Text，也就是界面上显示的内容。

3. 标签文本常用方法

1. 修改指定页标签文本的内容（不同页显示不同信息）

5.3.1.3.7.条形码元素

1 如何创建条形码元素

创建条形码元素有两个方式

第一个方式就是调用方法弹出条形码属性对话框，设置对应属性进行插入:

```
var ctl = document.getElementById("myWriterControl");
ctl.DCExecuteCommand("InsertBarcodeElement", true, null)
```

第二个方式就是通过代码配置好属性，然后调用方法直接插入:

```
var ctl = document.getElementById("myWriterControl")
var options = {
 ID: "barcode1",
 Text: "1234566",
 Width: "1000",
 Height: "300",
 ShowText: true,
 TextAlignment: "Right",
 ValueBinding: {
 DataSource: "datasourceforbarcode",
 BindingPath: "path1"
 }
 };
 ctl.DCExecuteCommand("InsertBarcodeElement", false, options);
```

2.条形码属性对话框

条形码内置属性对话框功能，鼠标左击下条形然后调用编辑器内置命令:

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'ElementsProperties' ,true,null)
```

1. 编号

编号对应的是条形码属性中的 ID（命名需要符合标识符规范），可以通过 ID 获取到该元素，然后实现不同的业务需求。

2. 名称

名称对应的是条形码属性中的 Name

3. 文本内容

文本内容对应的是条形码属性中的 Text，也就是通过硬件扫描出来的内容。

4. 条码样式

条码样式对应的是条形码属性中的 BarcodeStyle，通过设置不同的条码样式值，扫描出来的内容不同，常用的是条码样式为 Core128A(只支持存在数字、大写英文)、Code128B(只支持存在数字、小写英文、大写英文)、Code128C（只支持数字）内容为奇数时，扫描出来的内容会在最前面自动添加 0，变成偶数位。

5. 文本对齐方式

文本对齐方式对应的是条形码属性中的 TextAlignment，通过设置该属性来控制文本的显示对齐方式。

6. 是否显示文本

是否显示文本就是条形码属性中的 ShowText，默认为 true，插入的条形码上面会出现文本，设置成 false，文本隐藏。

7. 数据源

数据源对应的是条形码属性中的 ValueBinding 对象下的 DataSource，该属性可以随意定义（需要符合标识符规范）。

8.绑定路径

绑定路径就是对应条形码属性中的 ValueBinding 对象下的 BindingPath，该属性可以随意定义（需要符合标识符规范），该属性定义最好能跟数据库存储的数据字段有对应关系。然后通过前端生成的 json 数据中 key 值需要进行对应。绑定的方式跟输入域一模一样。

5.3.1.3.8.二维码元素

1.如何创建二维码元素

创建二维码元素有两个方式

第一个方式就是调用方法弹出条形码属性对话框，设置对应属性进行插入：

```
var ctl = document.getElementById("myWriterControl");
ctl.DCExecuteCommand("InsertTDBarcodeElement", true, null)
```

第二个方式就是通过代码配置好属性，然后调用方法直接插入：

```
var ctl = document.getElementById("myWriterControl")
 var tdoptions = {
 ID: 'tdbarcode1', //元素 ID
 Text: "http://www.dcwriter.cn:6788/", //二维码内容文本
 Width: 300, //像素宽度
 Height: 300, //像素高度
 };
 ctl.DCExecuteCommand('InsertTDBarcodeElement', false, tdoptions);
```

2.二维码属性对话框

条形码内置属性对话框功能，鼠标左击下条形然后调用编辑器内置命令：

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'ElementsProperties' ,true,null)
```

1. 编号

编号对应的是二维码属性中的 ID（命名需要符合标识符规范），可以通过 ID 获取到该元素，然后实现不同的业务需求。

2. 名称

名称对应的是二维码属性中的 Name

3. 文本内容

文本内容对应的是二维码属性中的 Text，可以传一个网址，里面记录记录当前病人的信息，也能是医院的介绍等。

4. 数据源

数据源对应的是二维码属性中的 ValueBinding 对象下的 DataSource，该属性可以随意定义（需要符合标识符规范）。

5. 绑定路径

绑定路径就是对应二维码属性中的 ValueBinding 对象下的 BindingPath，该属性可以随意定义（需要符合标识符规范），该属性定义最好能跟数据库存储的数据字段有对应关系。然后通过前端生成的 json 数据中 key 值需要进行对应。绑定的方式跟输入域一模一样。

5.3.1.3.9.按钮元素

1 如何创建按钮元素

创建按钮元素有两个方式

第一个方式就是调用方法弹出按钮属性对话框，设置对应属性进行插入:

```
var ctl = document.getElementById("myWriterControl");
ctl.DCExecuteCommand("InsertButton", true, null)
```

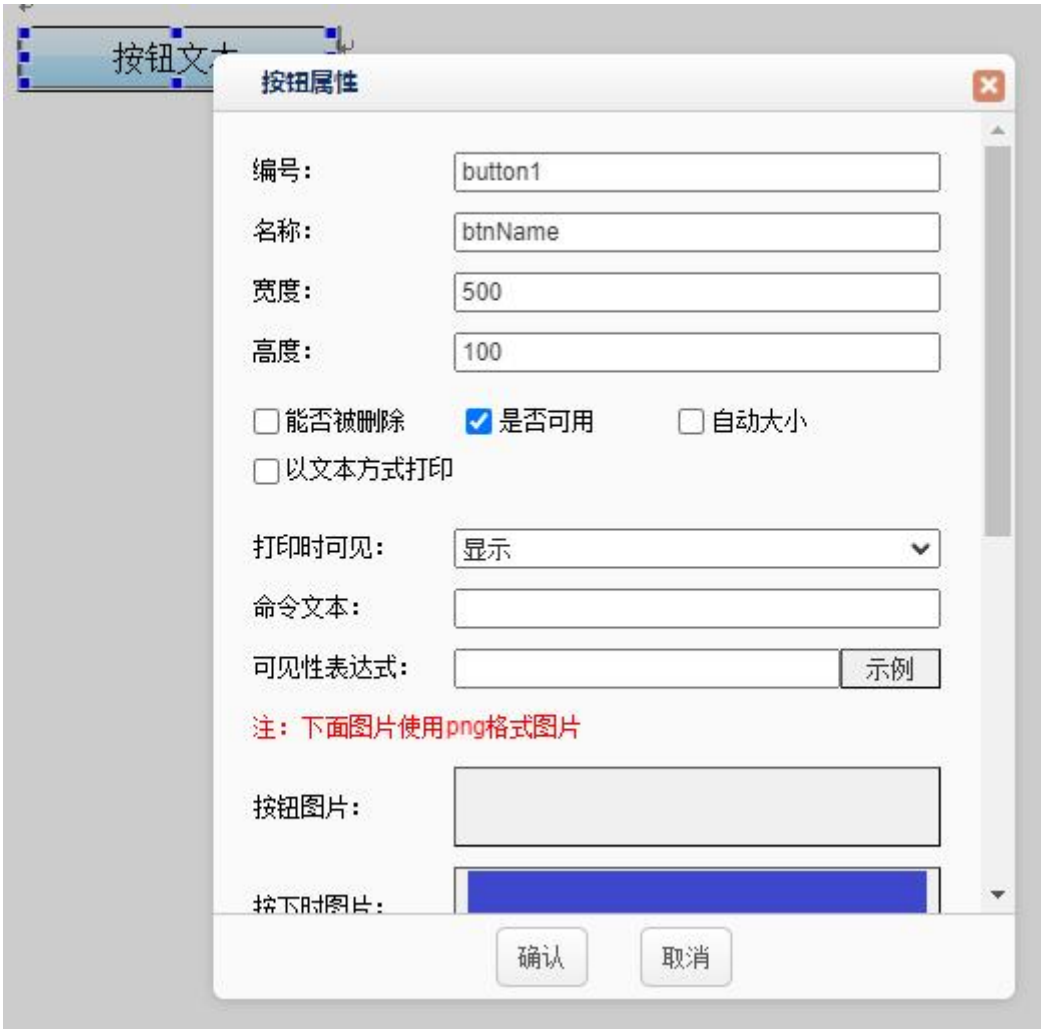
第二个方式就是通过代码配置好属性，然后调用方法直接插入:

```
var ctl = document.getElementById("myWriterControl")
 var options = {
 ID: "button1",
 Height: "100",
 Width: "500",
 Deleteable: false,
 Name: "btnName",
 Enabled: true,
 PrintAsText: false,
 Text: "按钮文本",
 };
 ctl.DCExecuteCommand('InsertButton', false, tdoptions);
```

2.按钮属性对话框

按钮内置属性对话框功能，鼠标选中按钮元素然后调用编辑器内置命令:

```
var  ctl=document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'ElementsProperties' ,true,null)
```



1. 编号

编号对应的是按钮属性中的 ID（命名需要符合标识符规范），可以通过 ID 获取到该元素，然后实现不同的业务需求。

2. 名称

名称对应的是按钮属性中的 Name

3. 能否被删除

能否被删除对应的是按钮属性中的 Deleteable，该属性可以控制元素是否允许被用户键盘删除掉。处于管理员模式下该属性无效。

4. 以文本方式打印

以文本方式打印对应的是按钮属性中的 PrintAsText，默认为 false，当设置成 true 的时候，只保留按钮文本进行打印。

5. 打印时可见

打印时可见对应的是按钮属性中的 PrintVisibility，效果与输入域效果一致。

6. 文本

文本对应的是按钮属性中的 Text,也就是界面上显示的内容。

注意：当需要按钮来实现业务需求的时候，可以定义按钮的点击事件 EventButtonClick 来进行处理。

5.3.1.3.10.图片元素

1.如何插入图片

编辑器支持插入图片元素，插入图片元素分为两类，第一类选择本地图片进行插入，第二类通过图片的 base64 数据进行插入。

通过本地图片插入图片

```
var ctl = document.getElementById("myWriterControl");
ctl.DCExecuteCommand("InsertImageElement", true, null)
```

通过 base64 数据插入

```
var ctl = document.getElementById("myWriterControl");
var options =
    [
          //图片
         {
            'image':
             {
                 id: "123",  //图片 id
                 src: redpicbase64data, //图片 base64 数据
                 width: "100",  //图片宽度
                 height: "100",  //图片高度
                 savecontentinfile: false
                 }
              }
        ];
var result = ctl.SetChildElements("容器 ID", options, " afterBegin")
```

注意：双击图片模式会是弹出图片编辑对话框的，可以对图片进行编辑。

不需要图片编辑模式，在插入图片的时候，对图片属性设置 EnableEditImageAdditionShape: false，就禁用了图片编辑功能。

2. 图片常用方法

1. 获取图片对象

```
var ctl = document.getElementById("myWriterControl");
var result=ctl.GetElementById('图片编号')
//通过图片编号获取到图片对象
```

5.3.1.3.11. 批注

1. 什么是批注

就是当上级医生查看下级医生书写的病历时候，觉得什么地方不合理，就可以进行选中插入一个批注进行提醒。或者是某块内容需要重点关注的都可以选中插入批注进行提醒。

2. 如何插入批注

调用代码：

```
 var ctl = document.getElementById("myWriterControl");
let options = {
                author: "李四主任医师",
                AuthorID: "lisi",
                BackColor: "#0000FF",
                BorderColor: "#FF0000",
                ForeColor: "#D9D919",
                Text: "这是内容"
            }
 var result = ctl.DCExecuteCommand('InsertComment', false, options)
```

注意：默认当编辑器处于特殊视图模式下是插入不了批注的，例如：阅读模式、编辑器只读、打印预览模式下。

只有编辑器只读比较特殊，能通过调用文档选项让编辑器处于只读模式下，也能插入批注。

```
var ctl = document.getElementById("myWriterControl");
ctl.DocumentOptions.BehaviorOptions.CommentEditableWhenReadonly=true;
ctl.ApplyDocumentOptions()
```

3. 批注常用方法

1. 获取批注列表

```
var ctl = document.getElementById("myWriterControl");
ctl.getCommentList()
//获取文档中所有的批注列表
```

2. 删除指定批注

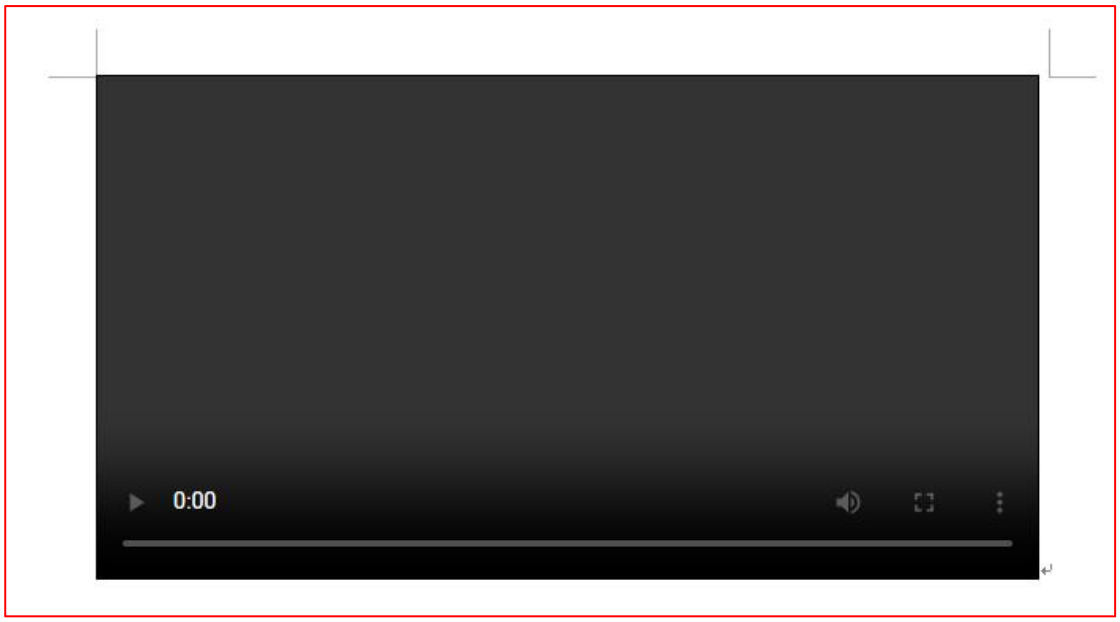
```
var ctl = document.getElementById("myWriterControl");
ctl.DeleteComment(1)
//通过 getCommentList 方法返回的批注列表中没有批注对象下都存在属性 IndexByList，该值就是批注索引，通过传这个值给删除批注的方法就能删除指定批注
```

5.3.1.3.12. 音视频元素

1. 如何创建音视频元素

调用一下代码，在当前光标处插入音视频

```
var ctl = document.getElementById("myWriterControl");
let options =
{
  Width: '1871',
  Height: '1000',
  FileName: 'http://www.dcwriter.cn/static/images/websiteexplain.mp4'
 }
ctl.DCExecuteCommand('insertmediaelement', false, options);
```



以上就是编辑器常用的结构化元素介绍，只有当用户对于编辑器内置的结构化元素有个清晰的认知，对于后面的业务功能开发才能。

5.3.2.根据元素的编号进行单一赋值

上面的结构化元素章节里面有介绍每个元素都可以设置编号的，但是要求元素的编号尽量需要唯一，存在相同 ID 的元素，通过 ID 进行取值赋值只能取到文档中的第一个 ID 的元素。

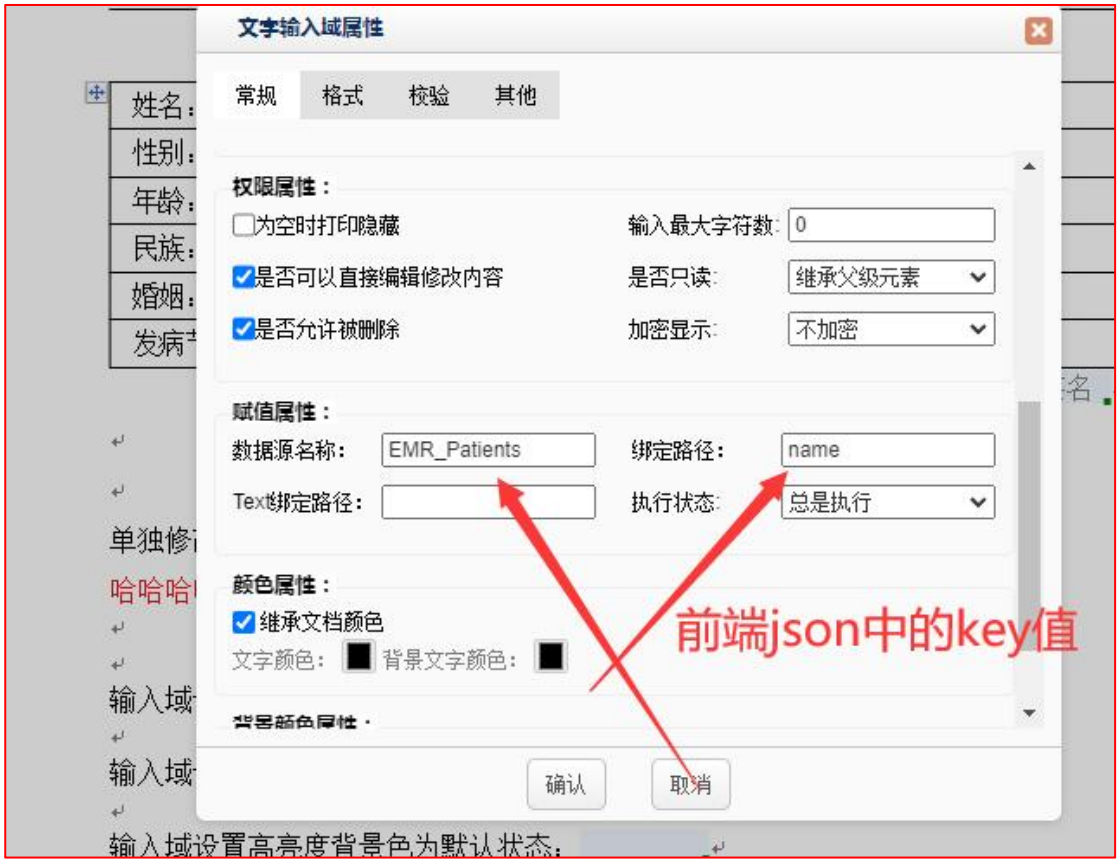
```
var ctl = document.getElementById("myWriterControl");
ctl.SetElementTextByID('元素 ID','元素内容');
```

假如一个文档中存在相同 ID 的元素，比如多个病程记录中都存在患者姓名，那 ID 就重复了，或者文档中页眉存在一个患者姓名，正文存在一个患者姓名。不同的场景编辑器有不同的接口来实现。那就可以通过元素属性下的 NativeHandle 值当做参数传到赋值方法中即可。

注意：通过该接口需要重复调用，也就是需要对几个元素赋值，就需要调用几次这个方法。

5.3.3.数据源绑定进行批量赋值

在结构化元素设计和介绍章节上面有介绍数据源属性和绑定路径属性的作用，这个就是需要用户在制作模板的时候，把结构化元素需要自动赋值的设置好数据源值和绑定路径值。然后通过前端的接口，就可以根据前端生成的绑定数据和结构化元素设置的绑定路径进行一一对应，进行批量的赋值




```
var ctl = document.getElementById("myWriterControl");
var values = {
       name: "张三",
        sex: "女",
        age: "20",
        //住院次数
        indate: "2019-3-27",
         //门诊号
         mingzu: "汉族",
         //一卡通号
         recorddate: "2019-3-28",
          //住院号
          marriage: "3333",
          //住院病区号（病室）
          patientname: "004",
     }
ctl.SetDocumentParameterValue("EMR_Patients", values);
var result = ctl.WriteDataFromDataSourceToDocument();
```

其中代码中创建的 values 就是需要客户生成的 json 数据，是一级结构。里面是键值对的关系，其中 key 值就是代表每一个元素的绑定路径，然后方法 SetDocumentParameterValue 方法第一个参数是元素的数据源值，第二个参数就是 json 数据，然后通过方法 WriteDataFromDataSourceToDocument 把值绑定到文档中。其中

SetDocumentParameterValue 方法可以调用多次，绑定不同的数据源。注意：WriteDataFromDataSourceToDocument 方法只需要最后面调用一次即可。

5.3.4.根据元素的编号进行取值

上面的结构化元素章节里面有介绍每个元素都可以设置编号的，但是要求元素的编号尽量需要唯一，存在相同 ID 的元素，通过 ID 进行取值赋值只能取到文档中的第一个 ID 的元素。

```
var ctl = document.getElementById("myWriterControl");
ctl.GetElementTextByID('元素 ID');
```

假如一个文档中存在相同 ID 的元素，比如多个病程记录中都存在患者姓名，那 ID 就重复出现，或者文档中页眉存在一个患者姓名，正文存在一个患者姓名。不同的场景编辑器有不同的接口来实现。那就可以通过元素属性下的 NativeHandle 值当做参数传到赋值方法中即可。

注意：通过该接口需要重复调用，也就是需要对几个元素取值，就需要调用几次这个方法。

5.3.5.取数据源绑定值

取数据源绑定的内容，当用户修改了一开始绑定的内容，也能通过数据源获取到最新值：

```
var ctl = document.getElementById("myWriterControl");
ctl.WriteDataFromDocumentToDataSource()
var result=ctl.GetDocumentParameterValue("Patient");
```

5.3.6.指定元素引用用户自定义片段

用户生成片段，插入到指定位置流程代码。

5.4.打印

编辑器内置打印功能，通篇打印、指定页打印、套打、区域选择打印。默认打印方式调用的是浏览器打印。

```
var ctl = document.getElementById("myWriterControl");
ctl.PrintDocument();
```

注意：浏览器打印无法控制打印文档的纸张，即文档是 A4 纸张，浏览器打印上也得选择对应的纸张类型进行打印。无法自适应。

注意：通过 canvas 进行绘图打印，需要使用 html 打印也是支持。

注意：调用打印方法不需要弹出浏览器打印界面，可以使用静默打印功能，通过在客户端上开机自启动打印小程序（exe）来进行实现。

5.4.1.单文档打印

1.描述好优缺点

2.代码调用

5.4.1.1.浏览器全文打印（打印预览模式下也能正常调用）

```
优点：此功能可直接通过浏览器打印全部文档，其他视图下也可以打印
代码：var ctl = document.getElementById("myWriterControl");
   var result = ctl.PrintDocument();//打印文档
```

5.4.1.2.浏览器指定页打印

```
优点：此功能可直接打印指定页的内容，无需在通过浏览器的打印窗口进行选择
代码：let CreatePrintOptions =
          {
             PrintRange: "SomePages",
             SpecifyPageIndexs: "1,2",
          };
let ctl = document.getElementById("myWriterControl");
ctl.PrintDocument(CreatePrintOptions);
//通过浏览器直接打印第一页和第二页
```

5.4.1.3.生成可供用户自由打印的 html 数据

```
优点：此功能生成的 html 数据如不使用都昌的打印方法，可以是客户自己通过第三方插件进行打印
代码：let ctl = document.getElementById("myWriterControl");
      ctl.PrintAsHtml(null, function (a) {   //回调函数中返回 html 数据
                  ctl.PrintByHtml(a);//打印 html
            });
```

5.4.1.4.生成可供用户自由打印的 Base64-PDF 数据

```
优点：此功能生成的 pdf 数据如不使用都昌的打印方法，可以是客户自己通过第三方插件进行打印
代码：var ctl = document.getElementById("myWriterControl");
          ctl.SaveDocumentToPdfBase64String(function (base64String) {
                console.log(base64String, '==============base64String');
            });/通过接口的回调函数获取 html 数据
```

5.4.2.多文档合并打印

5.4.2.1.多文档共用一个页眉页脚且各个文档拼接在一起打印

5.4.2.1.1.生成多文档下 Html 数据进行打印

```
    var arr = [strtest, strBindFileXml];
              var obj = {
                    "files": arr,//数组对象，存的是 xml 文档
                    "format": "xml",//xml 文档格式
                    "base64": "false",//是否是 base64 字符串
                    "megedoc": "false",//是否合并
                     "modefile": strtest,//病程合并模式下提供页眉页脚的主文档 XML 字符串
                      "splitmode": "none"//各个文件合并时中间的分隔模式，"none"不分隔，"pagebreak"换新页，"linebreak"，用换行符分隔，"line"，用水平线分隔
```

```
                }
                var ctl = document.getElementById("myWriterControl");
```

5.4.2.1.2.生成多文档下 Base64-pdf 数据进行打印

```
通过 splitmode 属性传入 none 使文档不进行换页
function GetPDFByFiles() {
            var ctl = document.getElementById("myWriterControl");
            var arr1 = new Array();
            arr1.push(strtest);
            var obj = {
                "files": arr1,//xml 文档数组
                "base64": "false",//是否是 base64
                "megedoc": "false",//是否合并文档
                "splitmode": "none",//"none"不分隔，"pagebreak"换新页，"linebreak"，用
换行符分隔，"line"，用水平线分隔
                "resulttype": "Base64String"//返回值是 base64pdf 字符串
            };
            //ctl.GetPDFByFiles(obj);
            var base64pdf = ctl.GetPDFByFiles(obj);

        }
```

5.4.2.2.多文档共用一个页眉页脚且各个文档独立打印

5.4.2.2.1.生成多文档下 Html 数据进行打印

```
let ctl = document.getElementById("myWriterControl");
          var files = new Array();
          files.push(lasttablestrxml);
          files.push(userxmlstr)
          var options = {
             "files": files,
             "format": "xml",
             "base64": false,
          }
 var globalsavestring = ctl.getHtmlByFiles(options);
//globalsavestring 返回值就是生成可打印的 html 数据
```

5.4.2.2.2.生成多文档下 Base64-pdf 数据进行打印

```
var options={
  "files": arr,
  "base64": "false",
```

```
  "megedoc": "false"//是否合并
  "modefile": "xmlstring"//病程合并模式下提供页眉页脚的主文档XML字符串
  "splitmode": "pagebreak"//各个文件合并时中间的分隔模式, "none"不分隔, "pagebreak"换新页, "linebreak", 用换行符分隔, "line", 用水平线分隔
  "filename": ""//导出PDF的文件名
  "resulttype": "Base64String"//返回类型: "DownloadFile"表示直接下载文件;"Base64String"表示返回pdf二进制的BASE64字符串
   "forceblacktextcolor": "false"//设置导出的PDF强制使用黑色字体
  }
let res=ctl.GetPDFByFiles(options);
```

5.4.2.3.多文档使用各自页眉页脚进行打印

5.4.2.3.1.生成多文档下 Html 数据进行打印

```
var ctl = document.WriterControl;
  var options = {
                        Files: [
                              [xmlcontent],
                              [xmlcontent],
                              [xmlcontent]
                        ]
                };
                ctl.GetPrintPreviewHTML2(options, function (htmlstr) {
                      ctl.PrintByHtml(htmlstr);
                });
```

5.4.2.3.2.生成多文档下 Base64-pdf 数据进行打印

```
function GetPDFByFiles() {
            let ctl = document.getElementById("myWriterControl");
            var files = new Array();
            files.push(userxmlstr);
            files.push(lasttablestrxml);
            var options = {
                "files": files,
                "base64": "false",
                "resulttype": "Base64String",//参数不存在或者为空字符串时候, 默认为Base64String 值
             };
            globalsavestring = ctl.GetPDFByFiles(options);
            console.log("GetPDFByFiles 方法返回值: ", globalsavestring);
        }
```

注意: 多文档下页码的显示问题。

5.4.3.静默打印

优点：不弹出浏览器打印界面，直接通过默认打印机进行打印

5.4.4.续打

1.描述续打业务

2. 通过代码设置续打位置

业务：客户需要从从上一次文档打印的位置继续打印文档。

5.4.5.区域选择打印

1.描述区域选择打印业务

2. 开启功能

业务：客户需要打印文档某一页中部分内容 代码：var ctl = document.getElementById("myWriterControl"); ctl.SetBoundsSelectionViewMode(true);//开启区域选择打印功能，参数必填 true 开启，fales 关闭

四．更多功能开发

4.1.病程记录

病程记录在住院系统是一个重要场景，由此编辑器内置一个特殊元素来进行处理，因为病程记录分为首次病程记录、日常病程、主治医师首次查房记录、副主任医师查房记录、交班记录、转科记录、阶段小结、抢救记录等等。

编辑器能把病人下的各种病程记录合并加载到一起进行显示，且每份病程记录还是独立的病历文档。

4.1.1.多病程记录插入

代码如下：

```
var ctl = document.getElementById("myWriterControl");
        ctl.DocumentLoad = function () {
            ctl.ClearDocumentBody();
            var file = new Array();
            file.push(SUB1);//首次病程文档
            file.push(SUB2);// 日常病程
            file.push(SUB3);//主任医师查房记录
            file.push(SUB4);// 副主任医师查房记录
            file.push(SUB5);//转出记录
            //构造两个选项分别对应两个子文档
            var options = new Array();
            //每份病程记录对应的相关属性控制、详细的病程属性参考接口文档
            var attr1 = {
                "ID": "subdoc1",
                "Attributes":{"医疗机构":"医院一"}
            };
            var attr2 = {
                "ID": "subdoc2",
                "Attributes":{"医疗机构":"医院二"}
            };
            var attr3 = {
                "ID": "subdoc3",
                "Attributes":{"医疗机构":"医院二"}
            };
            var attr4 = {
                "ID": "subdoc4",
                "Attributes":{"医疗机构":"医院二"}
            };
            var attr5 = {
                "ID": "subdoc5",
                "Attributes":{"医疗机构":"医院三"}//通过自定义属性，然后来实现页脚处显示
不同的医疗机构
            };
            options.push(attr1);
            options.push(attr2);
            options.push(attr3);
            options.push(attr4);
            options.push(attr5);
            var options = {
                Files: file,
                Options: options,
                Usebase64: "false",
            };
            let result = ctl.AppendSubDocuments(options);
```

```
                ctl.RefreshInnerView(true)
            }
ctl.LoadDocumentFromString(SUBMAINXML)
```

注意：通过接口插入病程记录是不会带上页眉页脚的，所以需要通过加载文档的接口先把病程所需的文档优先加载出来，然后通过接口清空正文，调用接口进行插入病程记录。

4.1.2.追加病程记录

医生需要在指定日期下病程记录下追加病程记录时，可以调用接口实现：

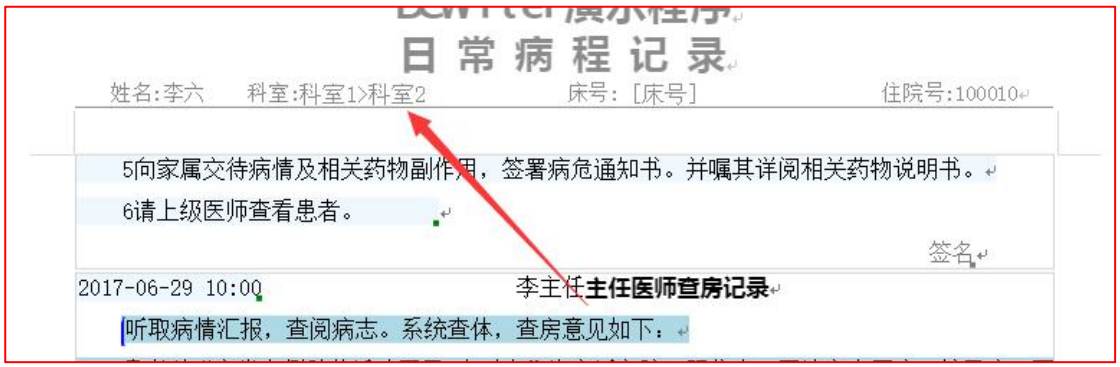
```
var ctl = document.getElementById("myWriterControl");
            var file = new Array();
            file.push(SUB2);//  日常病程
            var options = new Array();
            //每份病程记录对应的相关属性控制、详细的病程属性参考接口文档
            var attr1 = {
                "ID": "subdoc6",
            };
            options.push(attr1);
            var options = {
                Files: file,
                Options: options,
                Usebase64: "false",
            };
 let result = ctl.InsertSubDocuments(options);
ctl.RefreshInnerView(true)
```

注意：该接口是通过当前光标处病程下插入新的病程记录。

4.1.3.病程转科

4.1.3.1.什么是病程转科

转科、转床及转院制度是指在医疗服务中，将病人从一个科室、病床或医院转移到另一个科室、病床或医院的一项制度，在病程文档中就体现在页眉处，各页可以单独显示科室信息。



4.1.3.2.如何实现病程转科

实现病程转科功能需要满足两点：

第一点：优先加载的病程记录所需要的页眉页脚文档中需要存在一个标签文本元素，转科记录就是通过这个元素来进行承载的。

标签文本元素需要设置如下几个属性：

其中模式设置成 Normal、属性名可以自定义，但是需要跟第二个条件相匹配、连接文本就是两个科室中间的连接字符，可以自定义。当页眉中存在这个元素时，第一点就算完成了。

第二点，就是前端在组成病程所需的 Options 对象下各个病程记录对应的 options 值中需要设置每个病程记录的自定义属性，为键值对，key 值代表标签文本元素设置的属性名，Value 值代表科室名称，编辑器内部自动实现转科功能。

```
var attr5 = {
   "ID": "subdoc5",
   "Attributes":{"科室":"内科"}//通过自定义属性，然后来实现页脚处显示不同的医疗机构
};
options.push(attr5);
var options = {
   Files: file,
   Options: options,
   Usebase64: "false",
};
```

当文档中一页存在多个病程记录时，页眉上标签文本元素就是出现某科室--某科室字样，当文档中只存在一个病程记录，就之后显示该病程所处的科室字样，满足上述显示效果表示病程转科功能设计完毕。

4.1.4.病程保存

病程记录支持全篇病程保存、也支持单份病程记录进行保存。可通过病程记录下 Modified 属性值来判断病程记录是否被修改，没有修改的病程记录不进行保存。

```
var ctl = document.getElementById("myWriterControl");
var sublist=ctl. GetCourseRecords();
subdlist.forEach(element => {
    var sub=ctl.ElementProperties(element);
     var modified=sub.Modified;
     if(modified==true){
     var options =
     {
         FileFormat: 'xml',
         SubDocID: sub.NativeHandle,
      };
var subxml = ctl.SaveSubDocumentToString(options);
    }
 });
```

4.1.5.病程打印

病程打印比较特殊，医生可能会把当前病人下的病程优先打印出来，医生后续还是会追加病程记录，这时候就得保存第一次病程最后一次打印位置，当病程续打，根据最后一次打印位置把原来打印的病程记录先进行遮罩，从追加的地方在进行打印。

记录病程最后一次打印位置代码

```
var ctl = document.getElementById("myWriterControl");
        var printresult = ctl.GetPrintResult();//获取文档正文最后一次打印位置。
        resultpostion = printresult.BodyHeight;//获取需要设置续打位置的值
```

病程续打位置设置代码:

```
var ctl = document.getElementById("myWriterControl");
 ctl.SetJumpPrintMode(true);//开启续打模式
 ctl.JumpPrintPosition = resultpostion;//使用上一次的打印位置
```

4.1.6.单个病程记录签名

4.1.7.病程记录只读

4.1.8.病程记录强制分页

1.先解释为啥病程记录需要强制分页

2.如何实现

4.1.9.删除指定病程（痕迹）

4.1.10.病程边框设置

4.1.11.获取所有病程记录

4.2.护理记录与评估量表

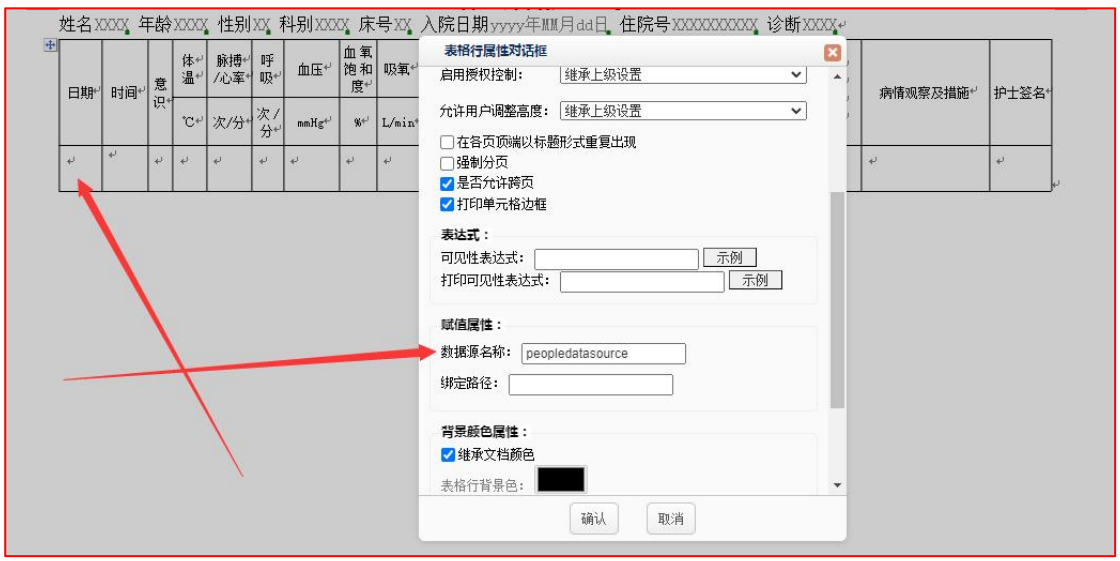
4.2.1.护理记录单

病人护理数据通过护士在硬件设备上进行收集，可通过编辑器把对应的数据进行可视化界面展示和打印。

4.2.2.1.护理记录单-数据自动赋值

护理记录单的数据赋值需要满足两点，第一点就是护理记录单模板需要设置了数据源信息：

护理记录模板只需要两块，第一块标题行，把要在另一页开头需要展示的表格行下 HeaderStyle 属性设置成 true，另一块就是数据行，数据行属性对话框上单独设置数据源值（需要符合标识符规范）



对数据行中全部的单元格在设置绑定路径（需要符合标识符规范，绑定路径对应的值就是单元格上需要显示的内容，最好跟数据库中存储对应内容的字段相关联）。配置模板完毕就需要生成对应的数据进行绑定。

第二点生成前端数据绑定表格

```
var ctl = document.getElementById("myWriterControl");
        if (ctl != null) {
            var arr = new Array();
            var obj1 = {
                date: "12-18",
                time: "16:56",
                BQGCJL:"遵医嘱予告病重，患儿有阵发性犬吠样咳嗽、鼻塞、流涕，可见吸气性三凹征，可闻及少许喉鸣音予拍背。注意观察患儿咳嗽的性质、持续时间及生命体征变化"
            };
            var obj2 = {
                date: "07-17",
                time: "09:47",
                BQGCJL: "患儿有阵发性犬吠样咳嗽，有气促气喘，可见吸气性三凹征，血气分析：氧分压(PO2)64.1 mmHg，遵医嘱予低流量给氧、血氧饱和度监测"
            };
            arr.push(obj1);
            arr.push(obj2);
            ctl.SetDocumentParameterValue("NurseRecordDS", arr);
            ctl.WriteDataFromDataSourceToDocument();
```

第一步定义数组，然后根据每个表格行数据生成对应的每条 json（键值对、键为对应单元格的绑定路径，值为单元格显示的内容），分别追加到数组中，最后调用绑定方法 SetDocumentParameterValue，第一个参数为表格行设置的数据源值，第二个参数为定义的数组值。通过方法 WriteDataFromDataSourceToDocument 把值绑定到文档上。

4.2.2.1.护理记录单-数据保存

调用方法就可以把绑定的内容（获取界面单元格内容、即使已经修改过绑定的数据）通过数组进行返回，调用代码为

```
var ctl = document.getElementById("myWriterControl");
        ctl.WriteDataFromDocumentToDataSource();
        console.log(ctl.GetDocumentParameterValue("NurseRecordDS"));
```

第一步先把文档中绑定数据添加到内存数据源列表中，然后传数据源模式获取该指定数据源值的内容。

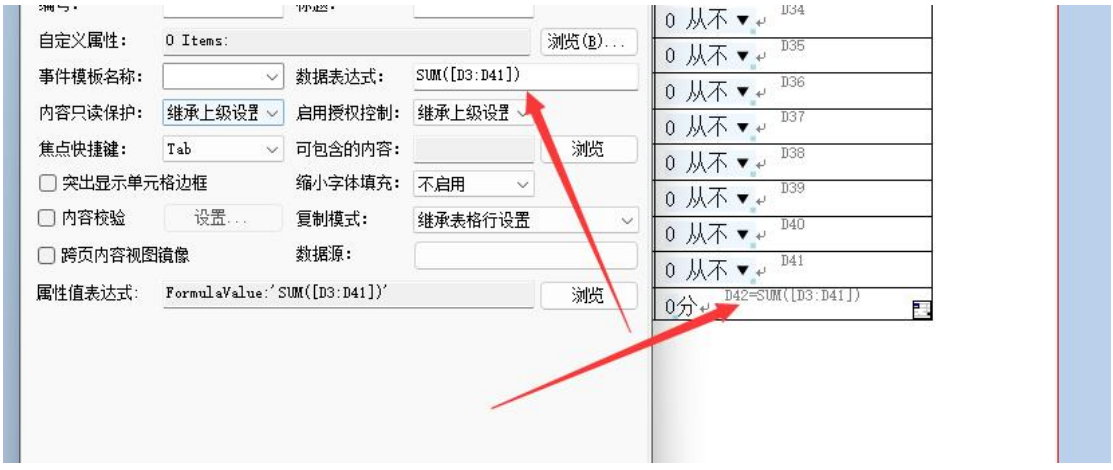
4.2.2.护理评估单

护理评估单，当了解完计算规则，就可以通过函数来进行计算，然后通过不同函数相互组合就能满足量表功能。当护理评估单模板维护完毕，护士只需要引用不同评估单即可满足自动计算需求。

4.2.2.1.常用函数示例

1.求和函数（SUM）

当需要对表格列中数值进行总分计算，可以在总分单元格中单元格属性对话框找到数值表达式填写：SUM([C3::C40])，其中 C3、C40 表示单元格的背景编号（执行命令 ShowBackgroundCellID 就能显示）该表达式就表示从第三列第三个单元格和到第四十个单元格之间的总和



2.根据区间值不同获取不同文本（LOOKUP）

比如根据总分范围不同得到不同的功能状态

≤8 分重度损伤，预后差；

9-11 分中度损伤；

≥12 分轻度损伤；数值越低，预示病情越重。

可以在需要显示状态的输入域属性对话框中数值表达式填写：LOOKUP([总分输入域 ID],0,'重度损伤',8,'中度损伤',11,'轻度损伤')

语言反应	☐4分：言语错乱，定向障碍 ☑3分：说话能被理解，但毫无意义 ☐2分：能发声，但不能被理解 ☐1分：不发生	3
A5 总分：	9	B5
功能状态：	中度损伤 ▼	B6

注：GCS量表总分范围3-15分：

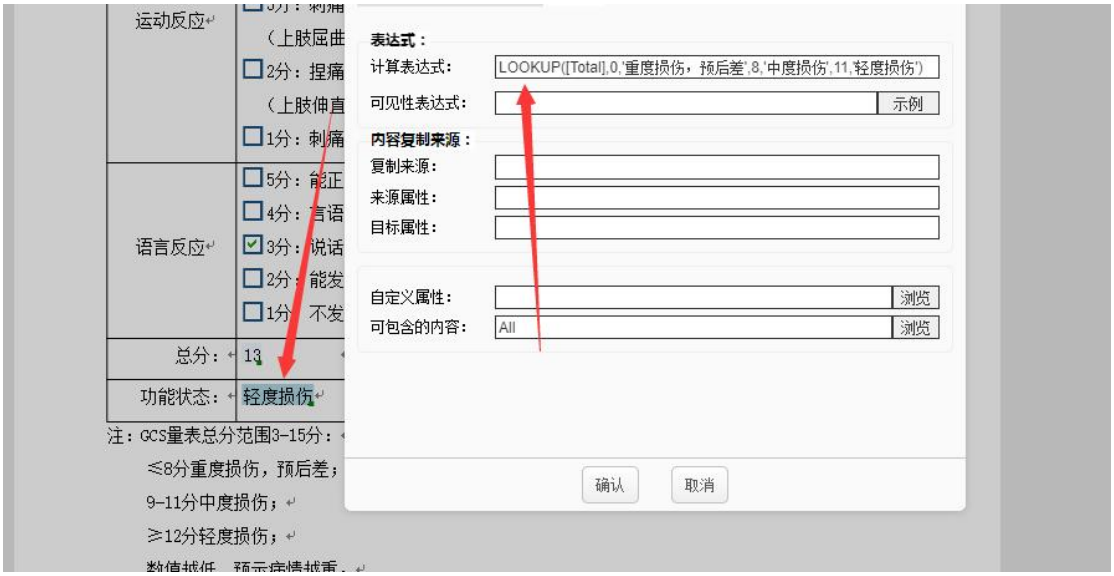
≤8分重度损伤，预后差；

9-11分中度损伤；

≥12分轻度损伤；

≤8分提示昏迷；≥9分提示无昏迷；

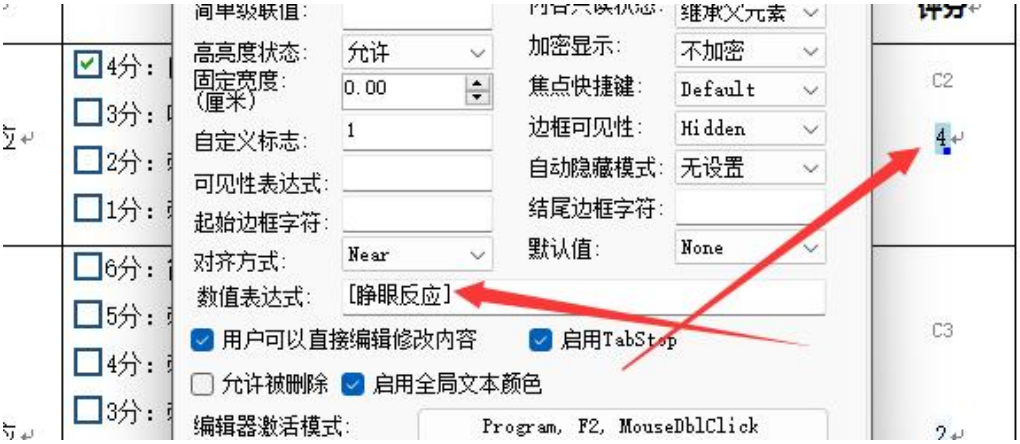
数值越低，预示病情越重。



3.单选框组求和

根据客户勾选的一组单选框中的某个选项，自动在输入域显示对应的勾选项的数值。在输入域属性对话框数值表达式填入：[单选框组 Name 值]

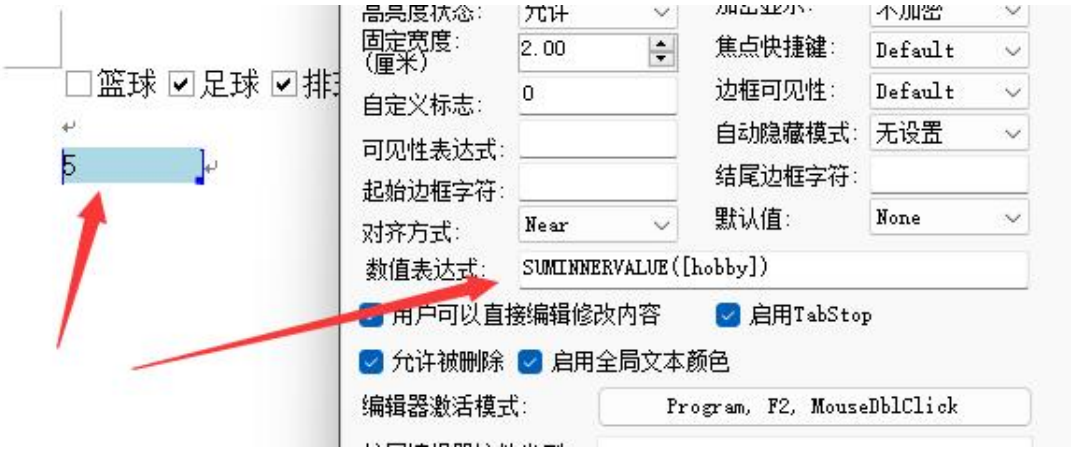
项目	状态	评分
A2 睁眼反应	☑4分：自动睁眼 ☐3分：听到言语命令时患者睁眼 ☐2分：刺痛时睁眼 ☐1分：刺痛时睁眼	C2 4



注意:输入域于输入域之间也能这样进行求和计算:[field1]+[field2]--就是对应把 ID 为 field1 输入域和 ID 为 field2 输入域值相加。

4.复选框组求和（SUMINNERVALUE）

根据客户勾选的一组复选框中的多个选项，根据勾选的选项得出总分，在输入域属性对话框数值表达式填入：SUNINNERVALUE([复选框组 Name 值])

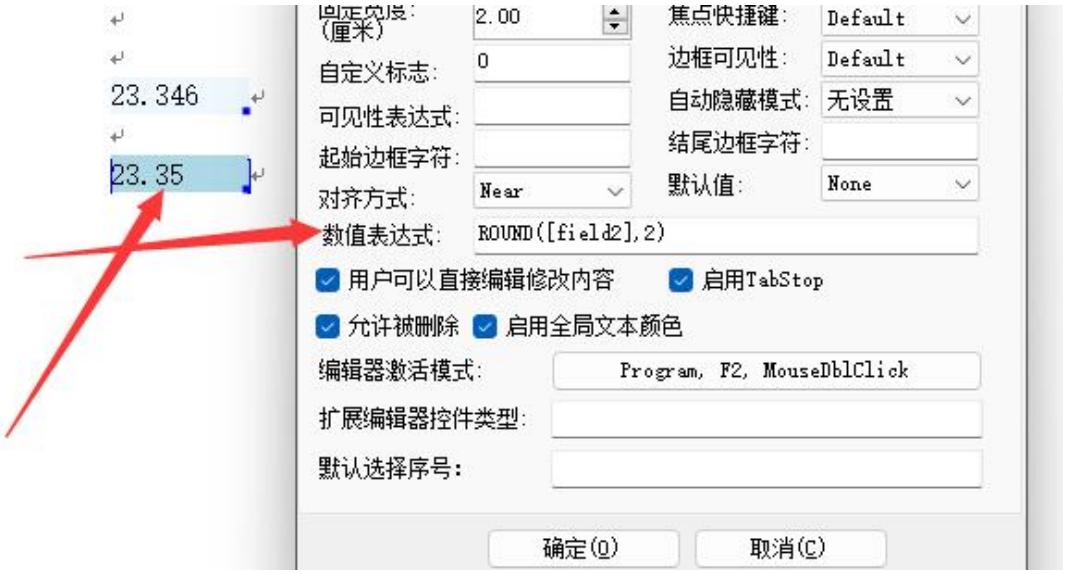


5.四舍五入(ROUND)

当用户在计算乘积或者余数的时候，有时候会出现小数点的情况，这个时候就可以通过在输入域属性对话框中填入：ROUND([分值输入域 ID],2)，表示把分值输入域保留二位小数点且进行四舍五入

23.346

23.35


23.346
23.35
(厘米)
2.00
自定义标志:
0
可见性表达式:
起始边框字符:
对齐方式:
Near
数值表达式:
ROUND([field2],2)
焦点快捷键:
Default
边框可见性:
Default
自动隐藏模式:
无设置
结尾边框字符:
默认值:
None
用户可以直接编辑修改内容
启用TabStop
允许被删除
启用全局文本颜色
编辑器激活模式:
Program, F2, MouseDblClick
扩展编辑器控件类型:
默认选择序号:
确定(O)
取消(C)

4.3.右键菜单

五代编辑器是通过在右键菜单事件中用户来自定义右键菜单的样式,右键菜单里面的功能就调用编辑器下的命令方法来进行实现，需要示例，请联系都昌技术人员。

4.4.权限管控

文档渲染到编辑器上可以同步开启用户权限功能，高权限的用户输入的内容，低权限用户无法编辑，低权限用户输入的内容高权限用户可以修改，且用户键盘输入的内容都将带有痕迹。而且是单个字符的权限管控。

```
var ctl = document.getElementById("myWriterControl");
        var options={
           ID:"zhangsan",//必须设置
           Name:"张三",//必须设置
```

```
            PermissionLevel:2,//必须设置
            ClientName:"111.11.11.11",//非必填
            Description:"住院医生"//非必填
        }
var result= ctl.UserLoginByUserLoginInfo(options,true);
```

注意：登录接口的调用一定得在文档加载接口的下面调用，用户权限功能才能起效果。当登录成功的时候编辑器默认处于留痕视图模式下，用户输入、删除的内容都会带上痕迹，也可以在登录接口后面调用清洁视图的接口。

注意：当编辑器处于清洁视图模式下的时候，获取文档中的所有痕迹信息列表，通过方法 GetDocumentUserTrackInfos 获取的痕迹列表不会带上删除痕迹。方法中传参数 true，才能获取文档中所有的痕迹信息列表。

4.5.页面设置

4.5.1.设置页面设置

```
var ctl = document.getElementById("myWriterControl");
            var opts = {
                LeftMargin: 30,
                RightMargin: 30,
            };
            ctl.ChangeDocumentSettings(opts);
            ctl.RefreshDocument();
//修改文档左右边距
```

4.5.2.获取页面设置

```
var ctl = document.getElementById("myWriterControl");
    ctl.GetDocumentPageSettings();
//获取页面设置
```

4.6.水印

4.6.1.插入水印

```
var ctl = document.getElementById('myWriterControl');
                let watermarkInfo = {
                    "Type": "Text",
                    "DensityForRepeat": 1,
                    "Repeat": true,
                    "Alpha": 32,
                    "ColorValue": "#EA161E",
                    "Angle": 80,
                    "font": "微软雅黑, 22, style=Bold",
                    "Text": "哈哈哈，我是水印",
};
                ctl.SetDocumentWatermark(watermarkInfo);
```

```
//插入文字水印
```

4.6.1.书写显示水印打印隐藏水印

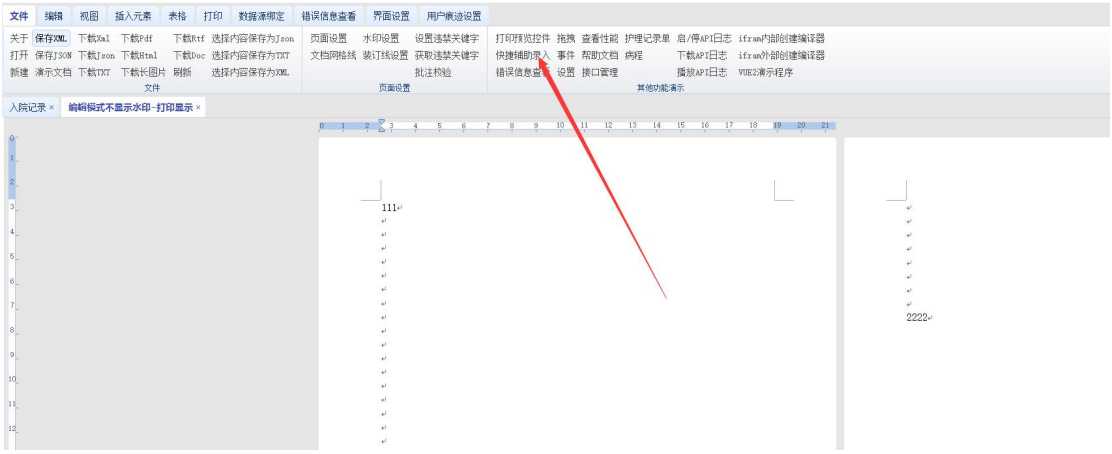
```
var ctl = document.getElementById("myWriterControl");
            var opts = {
               PageIndexsForPrintBackgroundImage:-1,
            };
            ctl.ChangeDocumentSettings(opts);
            ctl.RefreshDocument();
//PageIndexsForPrintBackgroundImage 属性值设置成负数，表示不打印水印
```

4.6.2.书写隐藏水印打印显示水印

```
var ctl = document.getElementById("myWriterControl");
            var opts = {
               PageIndexsForShowBackgroundImage:-1,
            };
            ctl.ChangeDocumentSettings(opts);
            ctl.RefreshDocument();
//PageIndexsForShowBackgroundImage 属性值设置成负数，表示所有页都不显示
```

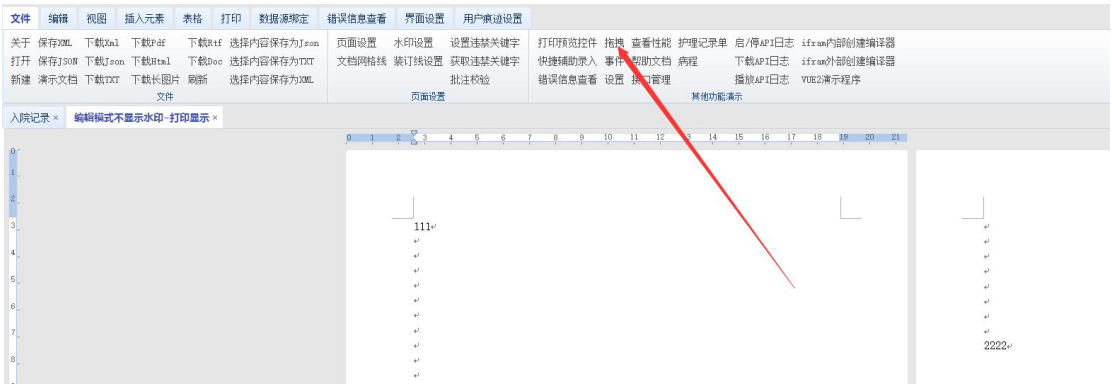
4.7.快捷赋值录入

请参考 https://www.dcwriter.cn:8765



4.8.外部拖拽数据编辑器生成对应内容

请参考 https://www.dcwriter.cn:8765



4.9.留痕权限控制

4.9.1.开启痕迹功能

```
let ctl = document.getElementById("myWriterControl");
        var options = {
            ID: "zhangsan",//必须设置
            Name: "张三",//必须设置
            PermissionLevel: 2,//必须设置
            ClientName: "111.11.11.11",//非必填
            Description: "住院医生"//非必填
        };
        var result = ctl.UserLoginByUserLoginInfo(options, true);
//注意接口的执行需要放在加载文档接口的后面才行，且调用完毕，编辑器自动切换到复杂视图
```

4.9.2.关闭痕迹功能

```
let ctl = document.getElementById("myWriterControl");
ctl .DocumentOptions.SecurityOptions.EnablePermission = false;
ctl .DocumentOptions.SecurityOptions.EnableLogicDelete = false;
ctl .DocumentOptions.SecurityOptions.ShowLogicDeletedContent = false;
ctl l.DocumentOptions.SecurityOptions.ShowPermissionMark = false;
ctl .DocumentOptions.SecurityOptions.ShowPermissionTip = false;
ctl .ApplyDocumentOptions();
```

4.9.3.获取痕迹列表信息

```
var ctl = document.getElementById("myWriterControl");
    ctl.GetDocumentUserTrackInfos(true);
//方法传 true，表示在清洁视图模式下也能获取到逻辑删除的痕迹，但是需要痕迹列表中定位到文档中对应的痕迹处，请开启复杂视图。
```

4.9.4.提交用户痕迹

```
var ctl = document.getElementById("myWriterControl");
ctl.DCExecuteCommand('AdministratorViewMode',false,true);
```

```
    ctl.CommitDocumentUserTrace()
ctl.DCExecuteCommand('AdministratorViewMode',false,false)
//注意：提交用户痕迹是把文档中的痕迹信息全部提交，也就是文档中不会在存在痕迹信息了，该功能使用需要处于管理员模式下才有效果
```

4.10.前端事件

4.10.1.编辑器加载完成事件

```
function WriterControl_OnLoad(rootElement) {
            console.log("编辑器初始化加载完成事件")
            rootElement.LoadDocumentFromString(strDemoFileXml);
        }
```

4.10.2.文档加载完成事件

```
var ctl = document.getElementById("myWriterControl");
ctl.DocumentLoad=function(a,b){
      console.log("文档加载完成事件")
   }
  ctl.LoadDocumentFromString(strDemoFileXml);
//注意该事件的定义必须在调用加载方法的前面才有效果
```

4.10.3.编辑器内容改变事件

```
var ctl = document.getElementById("myWriterControl")
        ctl.DocumentContentChanged = function (a, args) {
            console.log("编辑器内容改变事件")
        }
```

4.10.4.文档元素内容改变事件

```
var ctl = document.getElementById("myWriterControl")
        ctl.EventContentChanged = function (a, args) {
            console.log("元素内容改变事件")
        }
```

4.10.5.输入域动态下拉事件

```
var ctl = document.getElementById("myWriterControl")
 ctl.QueryListItems = function (ele, eventObject) {
            eventObject.AddResultItemByTextValue("汉族", "liuyi");
            eventObject.AddResultItemByTextValue("回族", "chener");
            eventObject.AddResultItemByTextValue("壮族", "zhangsan");
            eventObject.AddResultItemByTextValue("满族", "lisi");
            eventObject.AddResultItemByTextValue("苗族", "wangwu");
            eventObject.AddResultItemByTextValue("维吾尔族", "zhaoliu");
            eventObject.AddResultItemByTextValue("土家族", "sunqi");
            eventObject.AddResultItemByTextValue("彝族", "zhouba");
```

```
            eventObject.AddResultItemByTextValue("藏族", "wujiu");
            eventObject.AddResultItemByTextValue("蒙古族", "zhengshi");
            eventObject.AddResultItemByTextValue("布依族", "zhengshi");
        }
//注意输入域一定得设置动态下拉属性才行，且输入域需要设置激活模式
```

4.10.6.右键菜单事件

```
//联系南京都昌提供右键菜单的示例
```

4.10.7.键盘点击事件

```
var ctl = document.getElementById("myWriterControl")
ctl.ondocumentkeydown = function (e) {
   var result = rootElement.GetElementProperties(rootElement.CurrentElement())
  if (result && result.TypeName && result.TypeName == "XTextInputFieldElement") {
            //tab 键聚焦上一个输入域
            if (e && e.code === 'Enter') {
                e.handle = false
                e.preventDefault();
                console.log('当前输入域:', result)
                rootElement.FocusPreviousInput()
            }
        }
    }
//实现纯键盘操作，通过 tab 键向下切换，然后通过 enter 键向上切换输入域焦点，前提
编辑器处于严格表单模式
```

4.10.8.元素鼠标点击事件

```
var ctl = document.getElementById("myWriterControl")
ctl.EventElementMouseClick = function (eventSender, args) {
            console.log("元素鼠标单击")
            args.Handled = true;
        };
//注意当单击业务结束之后后面需要调用 Handled 属性，要不然会触发多次该事件
```

4.10.9.元素鼠标双击事件

```
var ctl = document.getElementById("myWriterControl")
ctl.EventElementMouseDblClick = function (eventSender, args) {
            console.log("元素鼠标双击")
            args.Handled = true;
        };
//注意当单击业务结束之后后面需要调用 Handled 属性，要不然会触发多次该事件
```

4.10.10.按钮点击事件

```
var ctl = document.getElementById("myWriterControl")
```

```
ctl.EventButtonClick=function(a,b){
        console.log("按钮点击事件")
      }
```

4.10.11.输入域下拉项选择前事件

```
var ctl = document.getElementById("myWriterControl")
ctl.EventBeforeFieldContentEdit=function(object,b){
            console.log(object,"object",b);
            console.log("下拉项选择前事件")
       }
```

4.10.12.输入域下拉项选择后事件

```
var ctl = document.getElementById("myWriterControl")
ctl.EventAfterFieldContentEdit=function(object,b){
            console.log(object,"object",b);
            console.log("下拉项选择后事件")
       }
```

4.10.13.粘贴前事件

```
var ctl = document.getElementById("myWriterControl")
 ctl.EventBeforePaste = function (copydata) {
              console.log("  粘贴前事件 EventBeforePaste    " , copydata)
              //return false;//拦截粘贴
          }
```

4.10.14.插入病程回调事件

```
var ctl = document.getElementById("myWriterControl")
ctl.EventAfterInsertSubDocuments = function (event) {
        console.log("病程插入成功的回调函数:EventAfterInsertSubDocuments")
    }
//注意调用 AppendSubDocuments 方法或者调用 InsertSubDocuments 方法都会触发该事件
```

4.10.15.鼠标拖拽事件

```
//需要从编辑器外部拖拽内容到编辑器上，请联系都昌技术人员询问示例 demo
```

4.10.16.打印前事件

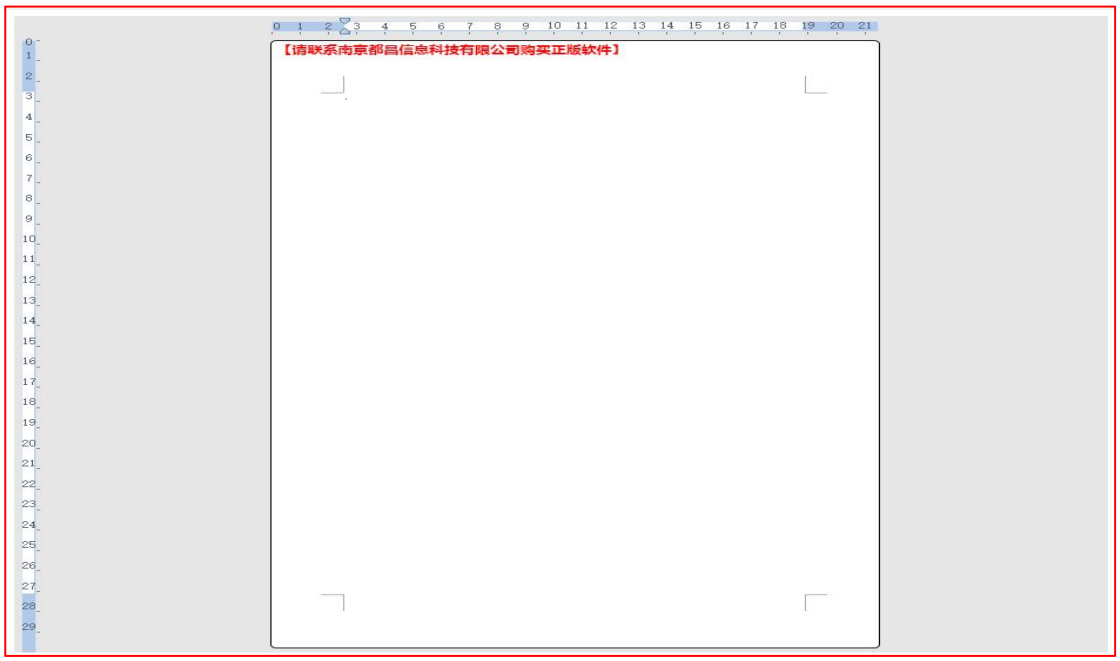
```
var ctl = document.getElementById("myWriterControl")
ctl.EventAfterPrintPreview = function (e) {
        console.log("111")
    }
```

4.11.获取全文结构化元素列表

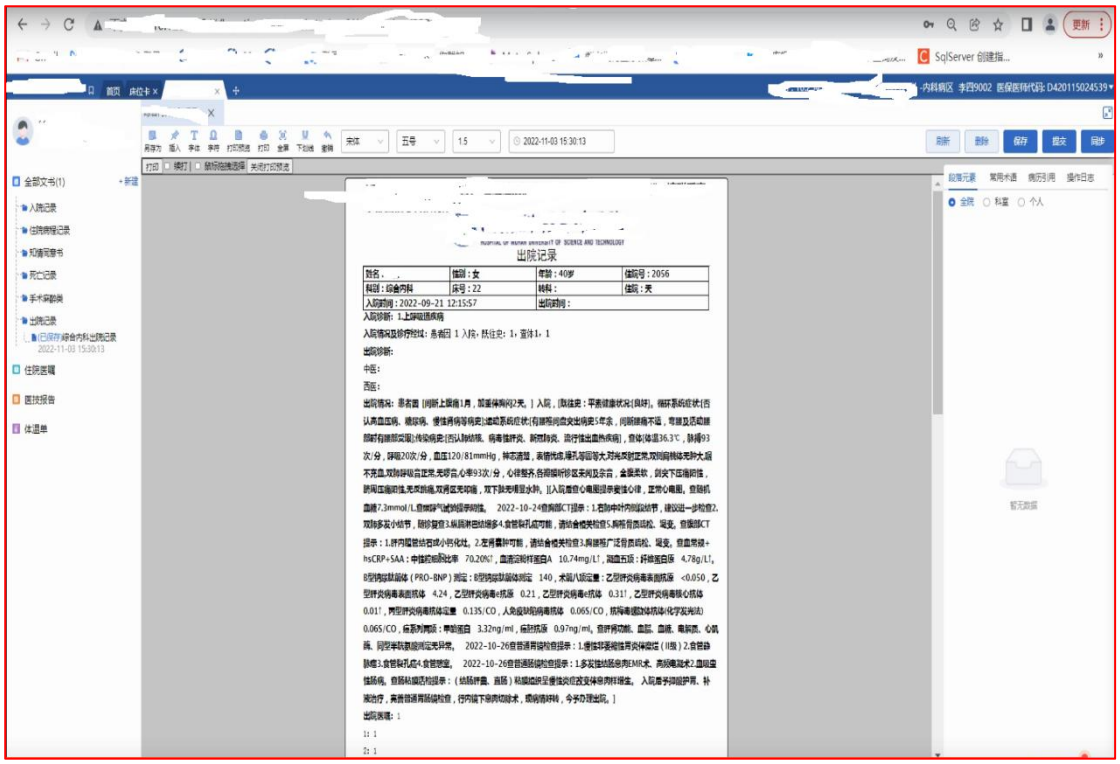
4.12.风格设计-工具栏功能

都昌提供给客户的五代编辑器类似于通过 word 新建了一个空白文档，然后需要客户自己设计空白文档外面的 UI 和工具栏、加载书写病历文档、保存病历文档，右键菜单、元素的赋值、取值等都需要用户结合五代接口文档调用内置好的 API 进行开发。

提供给客户效果图展示：



电子病历系统上对编辑器二次开发案例效果图：

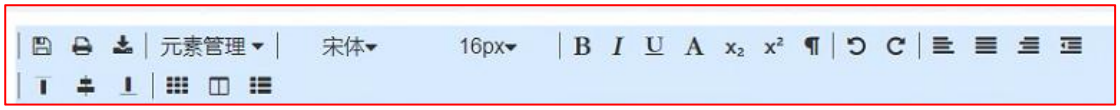


从上图可以看出，中间界面就是编辑器控件，然后编辑器上方就是工具栏，这块需要通过编辑器内置的 API 来实现相应的功能，然后左边就是病人的病历文书节点，然后医生通过点击对应的节点，从业务层获取到病历的数据，然后通过调用接口让编辑器渲染出来。

4.13.工具栏的设计

工具栏的 UI 风格设计得用户自己设计，功能需要通过编辑器内置的命令来实现。

4.13.1.用户设计的工具栏风格



4.13.2.工具栏功能设计

注意：方法 DCExecuteCommand 方法存在三个参数。
第一个参数：命令功能名称。

第二个参数：是否需要通过弹窗来设置功能参数，true:弹出、fasle：不弹出
第三个参数：当传 null 的时候，表示第一次设置功能，第二次取消。或者传 true 设置功能，传 false 取消功能

4.13.2.1.粗体

选中需要设置粗体的内容，然后调用下面的代码：

```
var ctl = document.getElementById("myWriterControl");
ctl.DCExecuteCommand(‘Bold’,false,null)
```

4.13.2.2.字体

选中需要设置字体的内容，然后调用下面的代码：

```
var ctl = document.getElementById("myWriterControl");
ctl.DCExecuteCommand(‘FontName’,false,’楷体’)
```

第三个参数，可以在设计字体功能的时候，设置成下拉项，然后把选择的下拉项提供给第三个参数里面。

4.13.2.3.字号

选中需要设置字号的内容，然后调用下面的代码：

```
var ctl = document.getElementById("myWriterControl");
ctl.DCExecuteCommand(‘FontSize’,false,’24’)
```

第三个参数，可以在设计字号功能的时候，设置成下拉项，然后把选择的下拉项提供给第三个参数里面。

4.13.2.4 剪切

选中需要剪切掉的内容，然后调用下面的代码：

```
var ctl = document.getElementById("myWriterControl");
ctl.DCExecuteCommand(‘Cut’,false,null)
```

4.13.2.5.复制

选中需要复制的内容，然后调用下面的代码：

```
var ctl = document.getElementById("myWriterControl");
ctl.DCExecuteCommand(‘Copy’,false,null)
```

4.13.6.粘贴

在需要粘贴的地方先通过鼠标点击，然后调用下面的代码：

```
var  ctl  =  document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'Paste' ,false,null)
```

4.13.2.7.上标

选中需要设置上标的内容，然后调用下面的代码：

```
var  ctl  =  document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'Superscript' ,false,null)
```

4.13.2.8.下标

选中需要设置下标的内容，然后调用下面的代码：

```
var  ctl  =  document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'Subscript' ,false,null)
```

4.13.2.9.段落

选中需要设置段落样式的内容，然后调用下面的代码：

```
var  ctl  =  document.getElementById("myWriterControl");
ctl.DCExecuteCommand( 'paragraphformat' ,true,null)
```

4.13.2.10.格式刷

先选中需要对应样式的内容，然后调用下面的代码：

```
var ctl = document.getElementById("myWriterControl");
    ctl.DCExecuteCommand( 'FormatBrush' ,false,null)
```

然后鼠标样式会变成一个格式刷的图标，选中需要同步样式的内容就行。

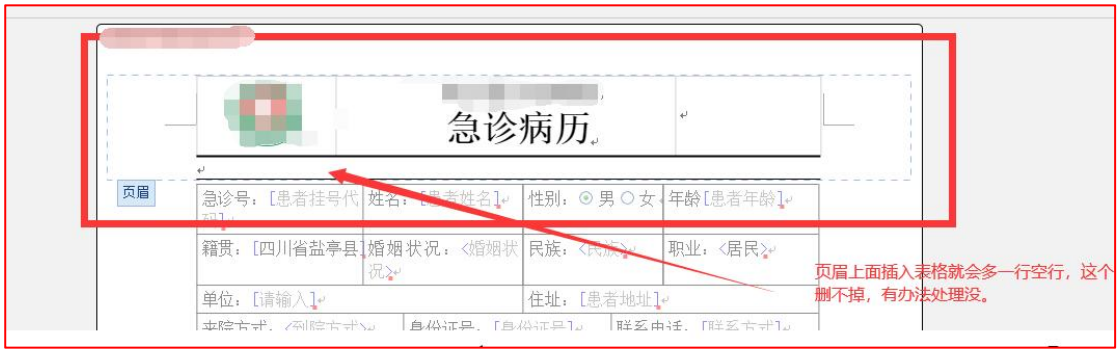
上面只介绍常用的工具栏功能，需要额外扩展的话，请参考文档链接：https://docs.qq.com/doc/DVGVGYlNSQkNxWEls 处 9.前端命令。

五.开发常见问题类

5.1.文档选项类

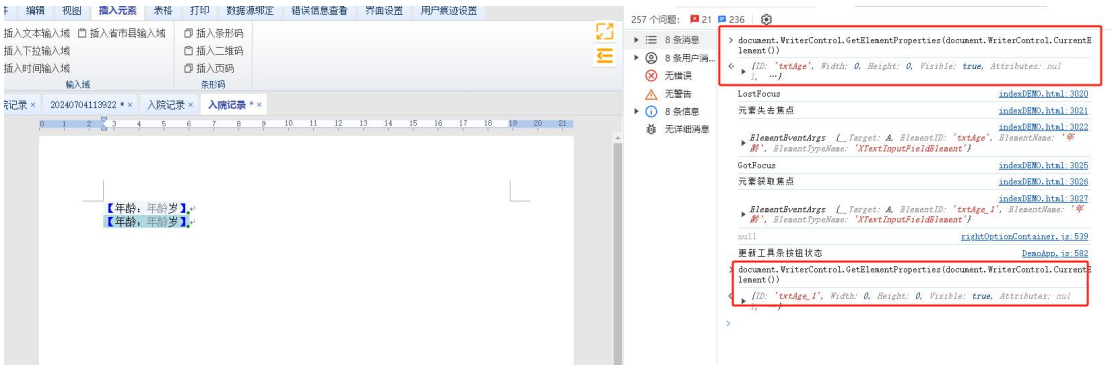
5.1.1.插入表格到编辑器中，表格后面出现一个空白行，或者病程记录与病程记录中间出现一个空白行。

编辑器界面表现形式为：



答：是由于编辑器下文档选项：DocumentOptions.BehaviorOptions.ParagraphFlagFollowTableOrSection 默认值为 false，导致的。该文档选项推荐用户在编辑器初始化的时候设置成 true。表格或者病程记录后面就不会出现一个空白行。

5.1.2.输入域 id 重复自动校正 id 避免重复

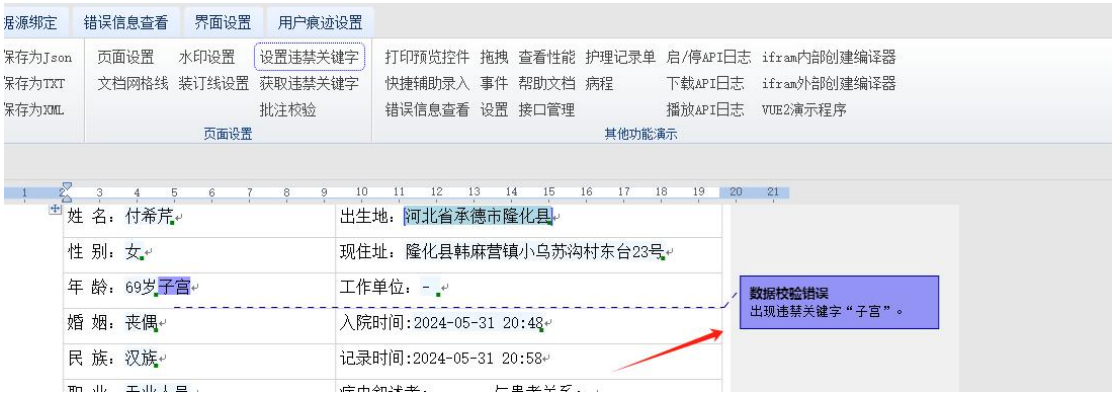


答：DocumentOptions.BehaviorOptions.ValidateIDRepeatMode="AutoFix"。ID 重复性校验模式。可选择值为：None 不做任何校验，DetectOnly 只进行检查不校验，AutoFix 自动修正 ID 号，ThrowException 抛出异常。

注意：因为文书中很多操作都是通过结构化元素的编号来进行处理的，当 ID 重复之后，大多数接口的调用都只能获取到相同 ID 元素列表中的第一个，所以需要最大可能的让文档中

不存在相同 ID 的元素

5.1.3.设置文档违禁字，进行文档校验



答：DocumentOptions.BehaviorOptions.ExcludeKeywords=”围巾，帽子”；
实现说明：全文违禁关键字列表，可包含多个违禁关键字，各个关键字之间用半角逗号分开。需要配合批注校验的方式可以做到图片中的效果

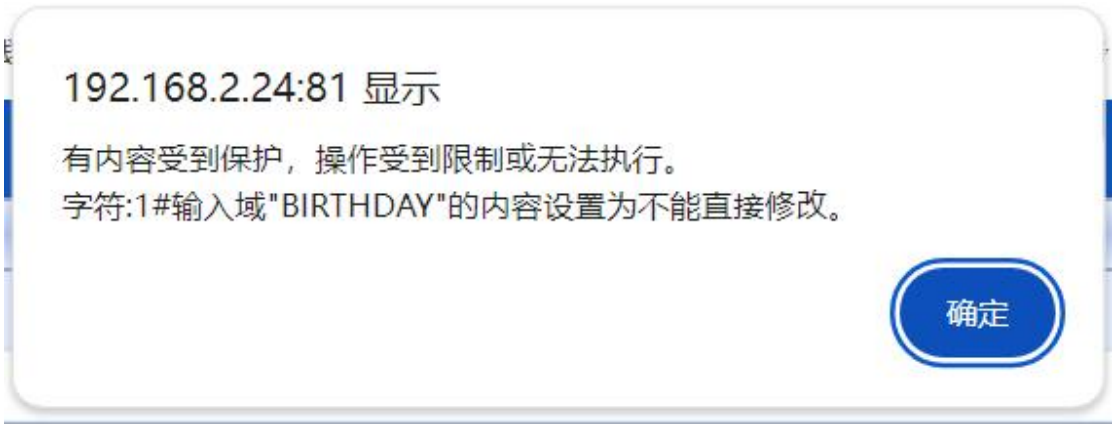
5.1.4.设置只读输入域的背景色

展示效果：

答：DocumentOptions.ViewOptions.ReadonlyFieldBackColor='red',也支持颜色格式为 RGB 格式。

5.1.5.遇到删除不可删除或内容保护的元素出现弹窗提示

展示效果：



答;DocumentOptions.BehaviorOptions.PromptProtectedContent="None" ，默认为 Details，也可以通过事件来自定义弹窗样式，不使用编辑器内置弹窗。

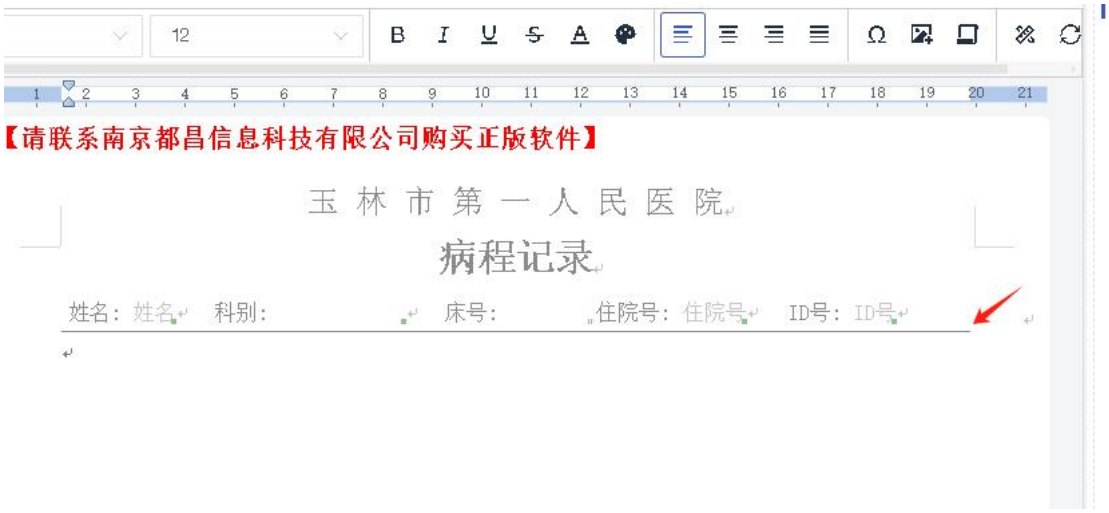
5.1.6.设置展示文档的编辑状态属性(输入域的小绿点如何取消展示)

展示效果:

答: DocumentOptions.ViewOptions.ShowInputFieldStateTag="true"默认为 true

5.1.7.取消页眉里面的水平横线

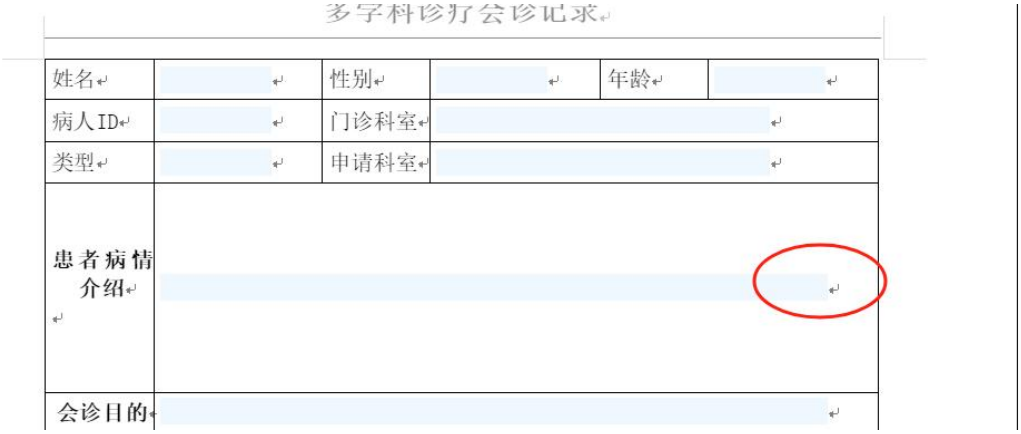
展示效果:



答：DocumentOptions.ViewOptions.ShowHeaderBottomLine = fasle;

5.1.8.文书中段落结束标记符号进行隐藏

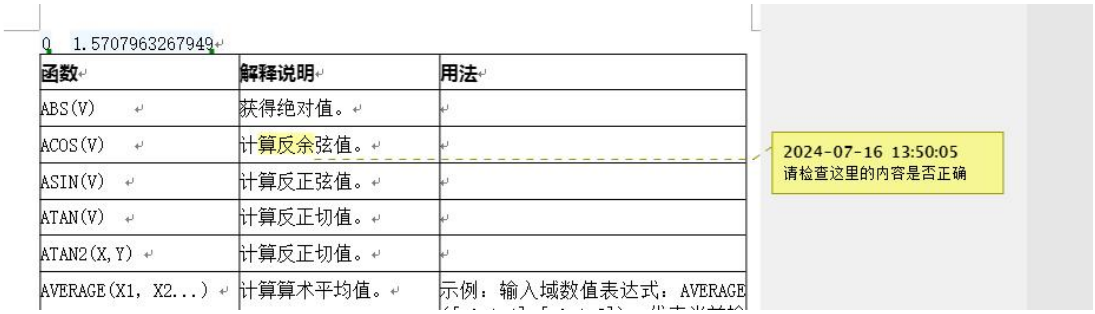
展示效果：



答：DocumentOptions.ViewOptions.ShowParagraphFlag = "false"

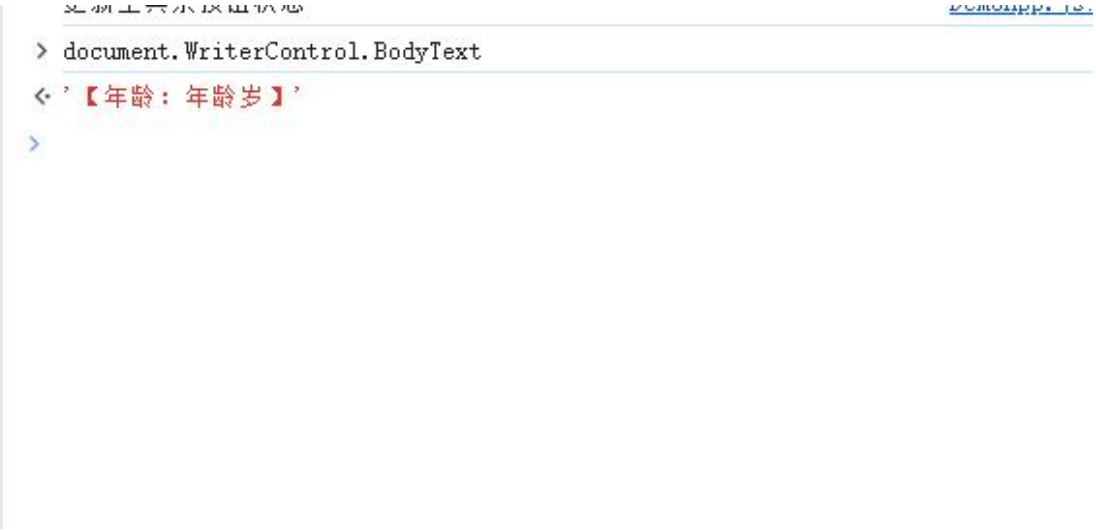
5.1.9.病历只读时需要插入批注

当编辑器设置属性 Readonly 设置只读时，默认是无法对选中的内容添加批注需要设置文档选项 DocumentOptions.BehaviorOptions.CommentEditableWhenReadonly=true;这个时候在只读情况下依旧能添加批注。



5.1.10.获取病历文本内容不包含输入域边框

ctl.BodyText 返回的正文内容中默认会带上输入域的边框文本的。



可以通过设置文档选项 DocumentOptions.BehaviorOptions.OutputFieldBorderTextToContentText=false;不输出输入域边框，或者直接通过调用方法 ctl.SaveDocumentToString('text')获取正文内容也不会带上输入域边框文本。

5.1.11 保存文档获取压缩 xml 数据内容

默认调用编辑器保存方法返回的 xml 是带有缩进控制内容的，阅读起来方便，但是文档会稍微大一点。

当需要获取压缩格式的 xml 数据，可以设置文档选项 DocumentOptions.BehaviorOptions.OutputFormatedXMLSource=false;就会获取到去掉了格式的 xml 数据.

5.1.12.从编辑器外部复制内容粘贴到编辑器时需要粘贴内容样式为当前光标处样式

当用户在 word 中复制一段带有样式的内容粘贴到编辑器时，默认是复制的内容是什么格式粘贴就是什么格式的，但是这样可能就无法跟编辑器加载文档的内容格式保持一致，就需要设置文档选项 DocumentOptions.BehaviorOptions.AutoClearTextFormatWhenPasteOrInsertContent=true;让粘贴的内容格式为当前光标处的样式。

5.1.13.删除自己输入的内容不带上痕迹

当执行了用户登录接口，这个时候用户的任何操作都是会留有痕迹的，当文档进行保存并重新加载之后，还是由当前用户进行登录，这个时候删除自己输入的内容时候带上逻辑删除的样式，当需要用户删除自己输入的内容不产生逻辑删除的样式，就需要设置文档选项 DocumentOptions.SecurityOptions.RealDeleteOwnerContent=false;这样用户输入只能纯在

创建痕迹，删除的自己输入的内容为物理删除不带有任何痕迹。

5.1.14.打印发下文档不是所见即所得

出现该问题需要从两方面进行排查：
第一个方面就是输入域的边框在打印的时候是否进行了占位，当一行里面存在多个输入域的时候，打印的时候输入域边框没有占位就会导致下一行的内容往上面移动导致不是所见即所得。设置文档选项 DocumentOptions.ViewOptions.FieldBorderPrintVisibility="Hidden"；输入域边框隐藏但是占位，然后再打印看下是否为所见即所得
第二个方面就是输入域为空时，输入域背景文本是否隐藏且占位，打印的时候排查看下空输入域的背景文本是否占位即可，当没有占位设置文档选项 DocumentOptions.ViewOptions.PreserveBackgroundTextWhenPrint=true；打印的时候空输入域背景文本隐藏且占位。

5.2.命令类

5.2.1.插入纯文本输入域（设置了数据源绑定、自定义属性、文本长度校验，数值表达式、可见性表达式属性）

插入代码：

```
var ctl=document.getElementById("myWriterControl")
var options=
{
ID: "field1",//输入域元素的编号
Name: "field1",//输入域元素的名称
InnerEditStyle: "Text",//输入域元素类型—Text 表示为纯文本输入域
ValueBinding:{"Datasource":"xx","BindingPath":"11"},//输入域数据源绑定值
Attributes:{"科室":"精神科"},//输入域自定义属性
ValidateStyle:{"Required":true, "ValueType":"Text", "MinLength":1, "MaxLength":20,
"CheckMaxValue":true, "CheckMinValue":true},//输入域校验属性，设置了必填和文本长度校验
VisibleExpression: "[field2]+[field3]",//输入域数值表达式
```

```
ValueExpression: "[sex]='女'",//输入域可见性表达式，当另外一个 ID 为 sex 的输入域值不为女，该输入域就会隐藏
}
ctl.DCExecuteCommand("InsertInputField",false,options)
```

注意：自定义属性中的内容格式为{“Key”:”Value”}格式

5.2.2.插入静态下拉输入域

插入代码

```
var options = {
                "ToolTip": "请双击",
                "BackgroundText": "下拉域",
                "ID": "txtList",
                "SpecifyWidth": "300",
                "InnerEditStyle": "DropDownList",
                "ListItems": [
                    {
                        Text: "男",
                        Value: "0"
                    },
                    {
                        Text: "女",
                        Value: "1"
                    },
                    {
                        Text: "未知",
                        Value: "3"
                    },
                    {
                        Text: "未知 1",
                        Value: "4"
                    },
                    {
                        Text: "未知 2",
                        Value: "5"
                    },
                ]
            };
ctl.DCExecuteCommand("InsertInputField",false,options)
```

5.2.3.插入表格插入代码

插入代码

```
function InsertTable() {
                var ctl = document.getElementById("myWriterControl");
                var parameter = {
```

```
                    'RowCount': 4,//行数
                    'ColumnCount': 4,//列数
                    'TableID': 'img_box'//表格 id
                };
                let result = ctl.DCExecuteCommand('InsertTable', true, parameter);// 插入
表格            }
```

5.2.4.删除表格

```
function DeleteTable() {
            var myWriterControl = document.getElementById("myWriterControl");
            myWriterControl.DCExecuteCommand("Table_DeleteTable", false, null);
        }
```

5.2.5.删除表格行

```
function DeleteTableRow() {
             var myWriterControl = document.getElementById("myWriterControl");
            myWriterControl.DCExecuteCommand("Table_DeleteRow", false, null);
        }
```

5.2.6.删除表格列

```
function DeleteTableColumn() {
            var myWriterControl = document.getElementById("myWriterControl");
            myWriterControl.DCExecuteCommand("Table_DeleteColumn", false, null);
        }
```

5.2.7.向上插入行

```
function InsertRowUp() {
             var myWriterControl = document.getElementById("myWriterControl");
            myWriterControl.DCExecuteCommand("Table_InsertRowUp", false, null);
        }
```

5.2.8.向下插入行

```
function InsertRowDown() {
            var myWriterControl = document.getElementById("myWriterControl");
            myWriterControl.DCExecuteCommand("Table_InsertRowDown", false, null);
        }
```

5.2.9.向左插入列

```
function InsertColumnLeft() {
            var ctl = document.getElementById("myWriterControl");
            ctl.ExecuteCommand("Table_InsertColumnLeft", false, null);
        }
```

5.2.10.向右插入列

```
function InsertColumnRight() {
        var ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand("Table_InsertColumnRight", false, null);
    }
```

5.2.11.合并单元格

```
function MegeCell() {
        var ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand("Table_MergeCell", false, null);
    }
```

5.3.12.拆分单元格

```
function SplitCell() {
        var ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand("Table_SplitCellExt", false, '2,3');
    }
```

5.3.13.复制

```
function Copy() {
        let ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand('Copy', false, null);
    }
```

5.3.14.粘贴

```
function Paste() {
        let ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand('Paste', false, null);
    }
```

5.3.15.剪切

```
  function Cut() {
        let ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand('Cut', false, null);
    }
```

5.3.16.纯文本复制

```
function CopyAsText() {
        let ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand('CopyAsText', false, null);
    }
```

5.3.17.纯文本粘贴

```
function PasteAsText() {
        let ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand('PasteAsText', false, null);
    }
```

5.3.18.撤销

```
function Undo() {
        let ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand('Undo', false, null);
    }
```

5.3.19.重做

```
function Redo() {
        let ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand('Redo', false, null);
    }
```

5.3.20.打开客户端文件

```
function FileOpen() {
        let ctl = document.getElementById("myWriterControl");
        ctl.ExecuteCommand('FileOpen', true, null);
    }
```

5.3.21.显示痕迹模式

```
function ComplexViewMode() {
        let ctl = document.getElementById("myWriterControl");
        ctl.DCExecuteCommand('ComplexViewMode', false, null)
    }
```

5.3.22.隐藏痕迹模式

```
function CleanViewMode() {
        let ctl = document.getElementById("myWriterControl");
            ctl.DCExecuteCommand('CleanViewMode', false, null)
    }
```

5.3.编辑器属性类

5.3.1.控制五代编辑器书写界面旁边的背景色

编辑器默认界面效果:

解决方案：在编辑器 div 中设置 style=" background-color: aqua;"即可

设置属性界面效果：

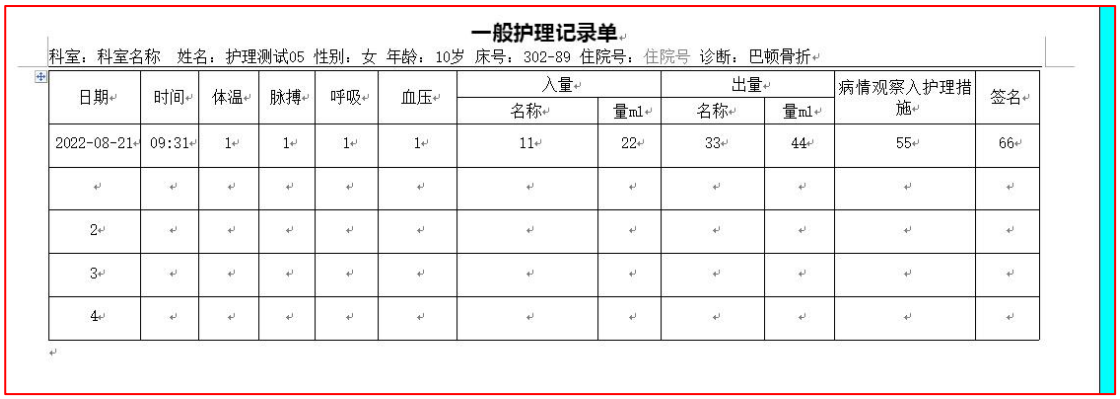
用户需要适配系统 UI，可以自行配置。根据要求自由选择。

```
<!-- <input type="button" value="2.插入图片" onclick="InsertImage()"> -->
<div id="myWriterControl" dctype="WriterControlForWASM"
    style="height: 1200px; background-color: aqua; border-bottom: 0px solid
    RuleVisible="true" AllowDrop="true" AllowDragContent="true" RuleBackColor="
    RegisterCode="0540923FA8D05DAEF96ED5E6B0A730F0CE317F9F0B96BBA3901FD7013D372
    PageCssString="box-shadow:0px 0px 0px grey" OnLoad="ss(this)">
    正在加载...
</div>
<!-- 以下是调用四代的服务地址 -->
```

5.3.2.控制页眉内容和正文内容清晰度显示一样效果。

编辑器默认界面效果：

解决方案：在编辑器 div 设置 GrayingDisabledHeaderFooter 属性值为 false 即可。
设置属性界面效果：



用户需要显示效果跟 word 页眉正文效果，那就不推荐设置。使用默认值。

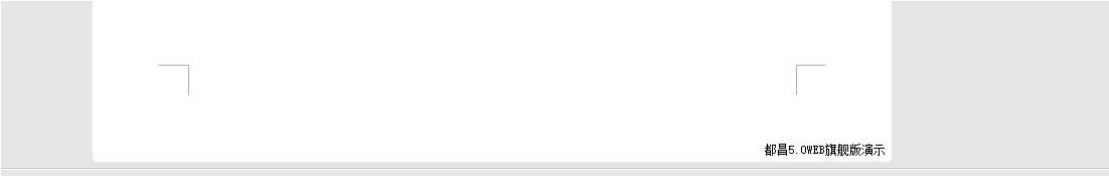
```
<input type="button" value="关闭打印预览" onclick="ClosePrintPreview()">
<div id="myWriterControl" dctype="WriterControlForWASM"
   style="height: 1200px; border-bottom: 0px solid ■black; background-attachment: fixed; background-size:
   RuleVisible="true" AllowDrop="true" AllowDragContent="true" RuleBackColor="rgb(155, 187, 227)" CaretCss
   RegisterCode="05227BCA4C274825E3C664A0F333D0DCF4E46CCF28CB25DE1377F018E92A2C7AFF5A8FBAA3A0B3E3B3989D420
   PageCssString="box-shadow:0px 0px 0px grey" OnLoad="ss(this)"
   ViewOptions.IgnoreFieldBorderWhenPrint="true" EnabledLogAPI="true" GrayingDisabledHeaderFooter="false"
   正在加载...
</div>
<!-- 以下是调用四代的服务地址 -->
```

5.3.3.编辑器前端注册

当前端编辑器 div 中没有设置属性 RegisterCode 值，那编辑器就是处于未注册的状态

当在编辑器 div 中添加 RegisterCore 值，那编辑器就处于注册状态

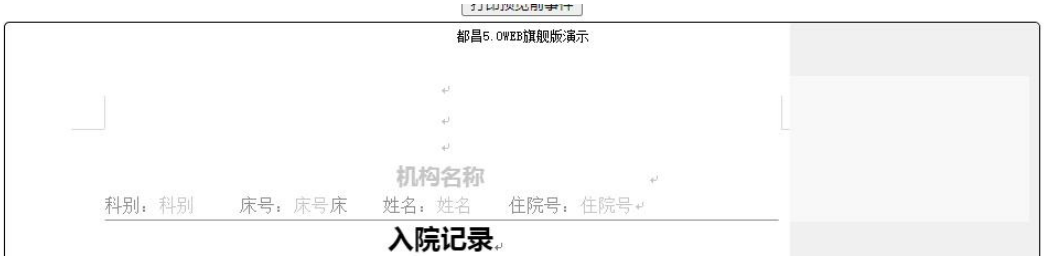
```
<div id="myWriterControl" dctype="WriterControlForWASM"
   style="height: 1200px; border-bottom: 0px solid ■black; background-attachment: fixed; back
   RuleVisible="true" AllowDrop="true" AllowDragContent="true" RuleBackColor="rgb(155, 187, 22
   RegisterCode="04023D6AB58E2F0368E2CB1293E20AEC7E45710588485AFAE01F92A6136F90A67BBF0984254D4
   PageCssString="box-shadow:0px 0px 0px grey" onLoad="this.LoadDocumentFromString(xmlstr,'xml
   正在加载...
</div>
```



5.3.4.编辑器界面注册机构显示位置

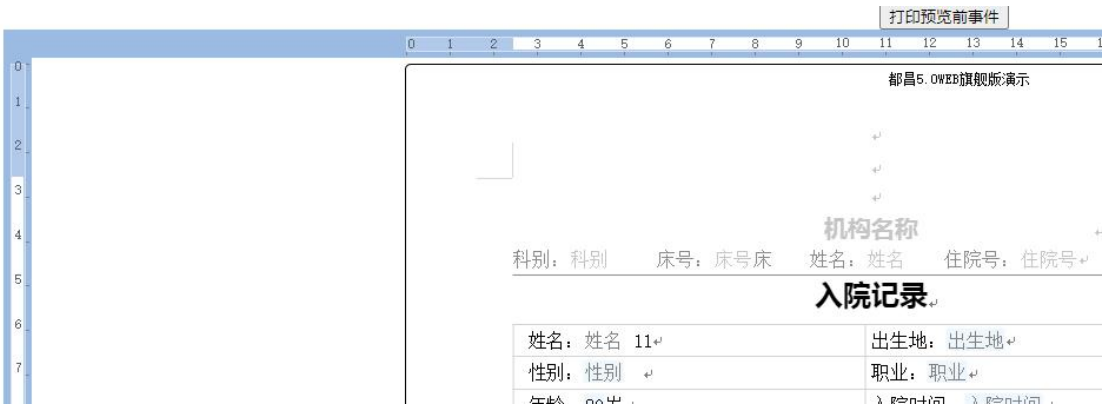
当编辑器注册成功的时候，会在界面左上角显示出当前项目名称或者机构名称，该字样是不允许隐藏，删除的，只能调整显示位置，默认位置就在左上角。

当在编辑器 div 上添加属性 PageTitlePosition 就可以修改显示位置，该属性是个枚举值分别为：BottomCenter--下居中对齐、BottomLeft;--下居左对齐、BottomRight;---下居右对齐、TopCenter;---上居中对齐、TopLeft;---上居左对齐、TopRight;---上居右对齐。



5.3.5.是否显示标尺

对编辑器 div 设置属性 RuleVisible 值为 true 即界面上会显示标尺



设置为 false 即隐藏。

5.3.6.设置标尺背景颜色

界面上标尺背景颜色默认为下面效果：

然后可以通过修改编辑器 div 属性 RuleBackColor 值修改标尺的背景颜色值。支持的颜色格式为 RGB。

```
<div id="myWriterControl" dctype="WriterControlForWASM"
   style="height: 1200px; border-bottom: 0px solid ■black; background-attachment: fixed; background-size: cover; backgr
    AllowDrop="true" AllowDragContent="true"  CaretCss="1,Blue" RuleVisible="true" RuleBackColor="rgb(125, 127, 227)"
   RegisterCode="05C25FAE0C4827CF58FEB1CEF69C650B48804B3106C195A8A5DFCA49BEDC62F2E4E9463BA5390D4B316FABBD7C9E58CFD566CB2
   PageCssString="box-shadow:0px 0px 0px grey" onLoad="this.LoadDocumentFromString(xmlstr,'xml')" PageTitlePosition="Top
   正在加载...
</div>
```

5.3.7.修改编辑器的背景颜色

编辑器默认背景颜色为透明色

在编辑器 div 上添加属性值 PageBackColorString 为 Color 值。

```
d-attachment: fixed; background-size: cover; background-position: center top; "
isible="true" RuleBackColor="rgb(125, 127, 227)"
:A49BEDC62F2E4E9463BA5390D4B316FABBD7C9E58CFD566CB2CB38318BC16E4162B9B419B1BC7185D37C
entFromString(xmlstr,'xml')" PageTitlePosition="TopCenter" PageBackColorString="red">


></script>
vriter5.js"></script>
```

5.3.8.修改输入焦点样式

编辑器界面默认输入焦点是以下展示效果

当在编辑器 div 上添加属性 CaretCss 值就可以修改输入焦点的样式。该值可以修改输入焦点三个样式 CaretCss="1,Blue,1"，第一个为输入焦点的大小，第二个为输入焦点的颜色、第三个为输入焦点向右偏移几个像素。

5.3.9.设置病历页眉页脚只读

在编辑器界面双击页眉或者页跟 word 类似，是会进入到页眉里面进行编辑的。

当有时候不允许用户修改页眉页脚的时候，就可以设置编辑器 div 属性 HeaderFooterReadonly 值为 true 即可不允许编辑页眉页脚

5.3.10.编辑器界面选中内容进行内部拖拽

默认编辑器是不允许进行内部拖拽的，当编辑器 div 上添加属性值 AllowDragContent 为 true 时，就可以正常选择内容进行拖拽了。

```
17
18  :input type="button" value="打印预览前事件" onclick="LoadPrintPreview()">
19  :div id="myWriterControl" dctype="WriterControlForWASM"
20     style="height: 1200px; border-bottom: 0px solid ■black; background-attachment: fixed; background-s
21      AllowDrop="true" AllowDragContent="true"  CaretCss="2,red" RuleVisible="true" RuleBackColor="rgb(1
22     RegisterCode="05C25FAE0C4827CF58FEB1CEF69C650B48804B3106C195A8A5DFCA49BEDC62F2E4E9463BA5390D4B316FA
23     PageCssString="box-shadow:0px 0px 0px grey" onLoad="this.LoadDocumentFromString(xmlstr,'xml')" Page
24     正在加载...
25  :/div>
26  :!-- 以下是调用四代的服务地址 -->
27  :script src="https://www.dcwriter.cn:8099/Jquery/jquery-1.7.2.min.js"></script>
28  :script src="https://www.dcwriter.cn:8099/ServicePage.aspx?wasmres=dcwriter5.js"></script>
29  :script>
30     function WriterControl_OnLoad(rootElement) {
31        console.log('控件加载完毕')
```

5.3.11.编辑器界面四周页面边界线条长度

默认编辑器跟 word 类似，四周会出现四个页面边界，能清晰看到上下左右页边距的距离，PageMarginLineLength 属性默认为 30，在编辑器 div 上添加属性 DocumentOptions.ViewOptions.PageMarginLineLength=0,就可以不进行显示。

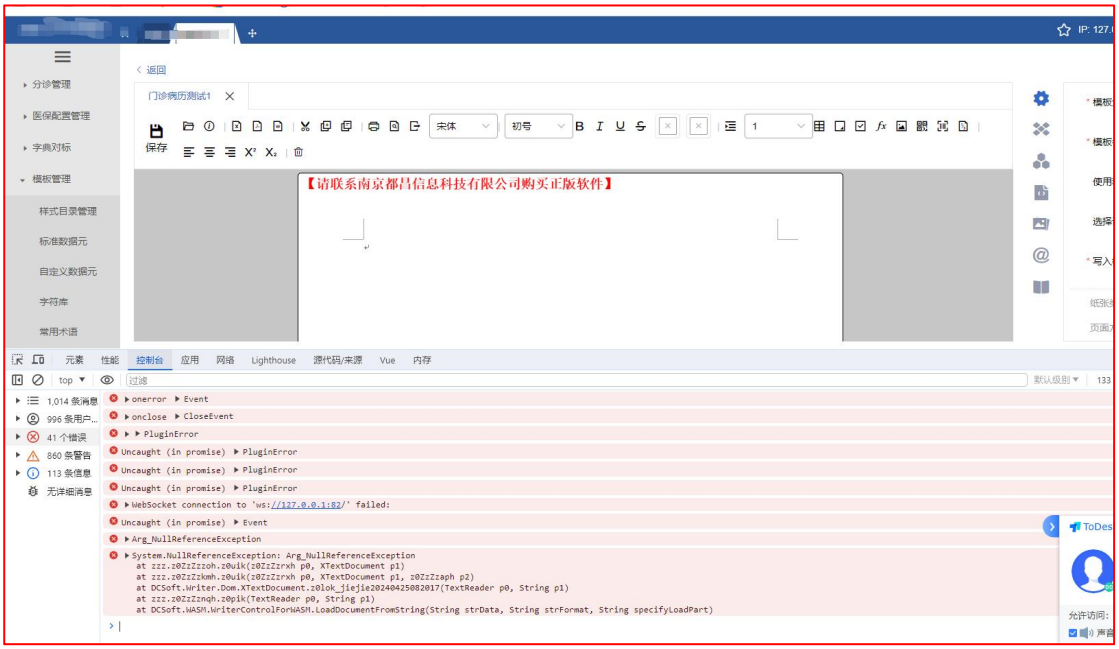
```
<div id="myWriterControl" dctype="WriterControlForWASM"
   DocumentOptions.BehaviorOptions.ParagraphFlagFollowTableOrSection="true"
   DocumentOptions.ViewOptions.PageMarginLineLength=0
   OutputSVGWhitespace = true
   RegisterCode="05B1199EE2D00F38BD1C4EF2DBD24C894B5474A4FDCE8C[illegible]RDF4D8307D2746BF774
   正在加载...
</div>
```

5.4.辅助用户开发类

1.描述方法使用场景
2.如何使用，有无前提

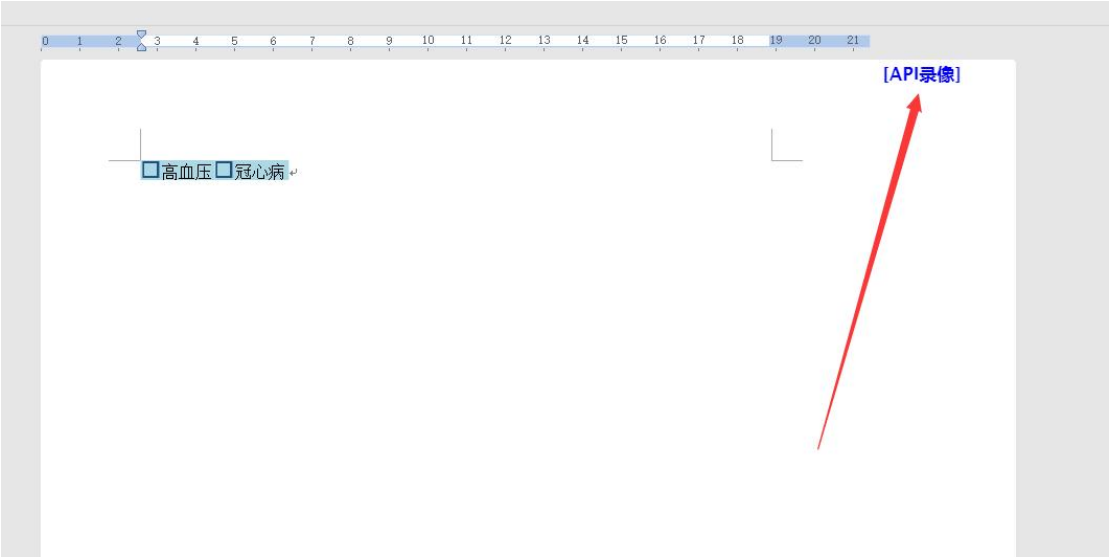
5.4.1.开启编辑器日志功能

当用户浏览器出现报错，并且是偶发才能出现时，例如



或者出现调用对应方法，但是界面上无效，或者是一些异常情况，排查不出问题时，就可以开启编辑器日志功能。

在编辑器 div 上设置属性 EnabledLogAPI 为 true。然后编辑器上显示如下图所示：



表示编辑器日志功能开启完成，当在开发中发现异常时，调用编辑器方法 ctl.DownloadAPIRecordData();会在本地下载一个 JSON 文件，然后把当前界面加载的病历也另存为一份，把两个文档提供给都昌技术人员即可。

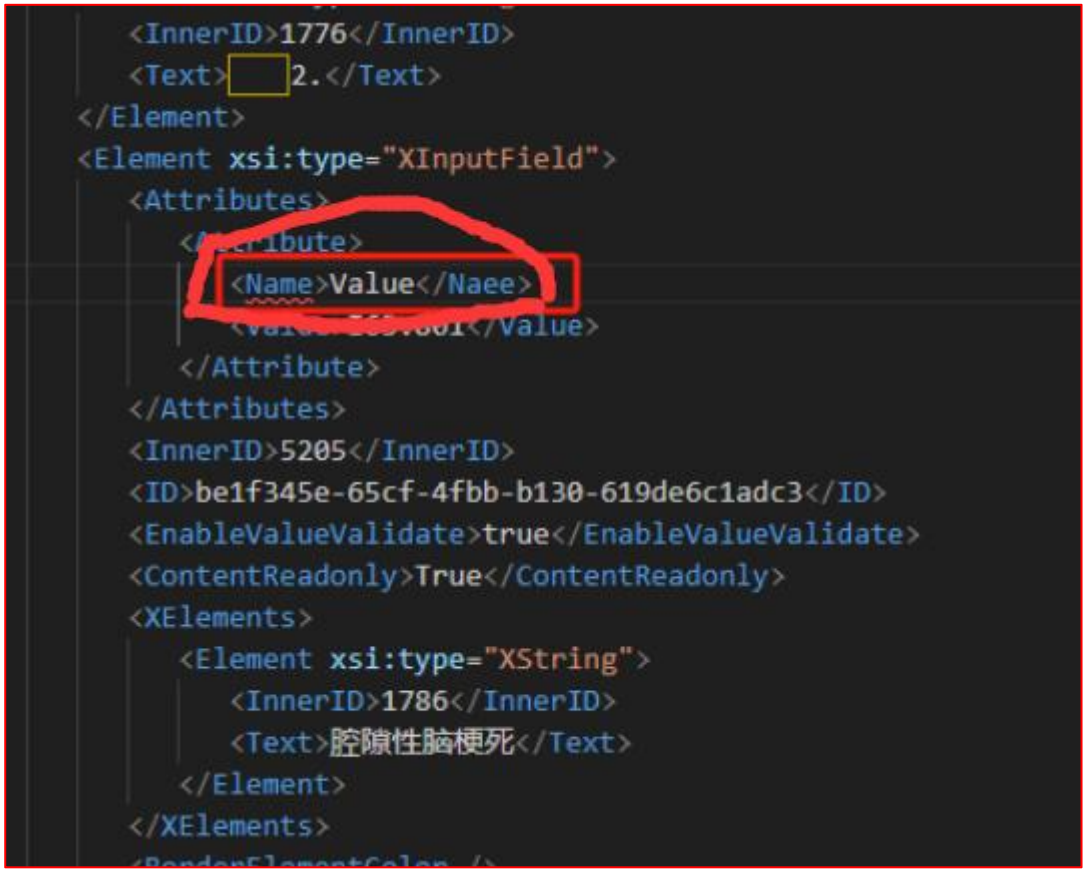
```
<div id="myWriterControl" dctype="WriterControlForWASM"
    style="height: 1200px; border-bottom: 0px solid black; background-
    RuleVisible="true" AllowDrop="true" AllowDragContent="true" RuleBack
    RegisterCode="05227BCA4C274825E3C664A0F333D0DCF4E46CCF28CB25DE1377F0
    PageCssString="box-shadow:0px 0px 0px grey" OnLoad="ss(this)"
    ViewOptions.IgnoreFieldBorderWhenPrint="true" EnabledLogAPI="true" H
    正在加载...
</div>
```

5.5.异常偶发问题类

1.描述应用场景
2.如何避免问题
3.如何定位问题

5.5.1.调用编辑器保存方法，下次加载的时候提示报错。

第一类：节点不匹配



第二类：xml 数据不全

等等问题。

避免方案：把编辑器生成的 xml 数据通过方法 IsValidateXML 进行校验，当返回 false，提示用户在保存一次，或者即使联系都昌技术人员进行排查。

建议：在执行保存方法生成 xml 数据之后，都调用下方法进行校验，这样异常数据就不会流入到数据库中，导致后续操作异常，优先暴露到操作层，比较好定位问题。

```
var ctl=document.getElementById("myWriterControl");
var xmltext=Ctl.SaveDocumentToString('xml')
var result=ctl.IsValidateXML(xmltext);
if(result==false){
  alert("保存文档异常，请重新保存，或者联系开发技术人员进行查看")
}
```

六. 约定

1. DCWriter 在文档中特指五代编辑器。

2. 在跟客户交流中让用户执行某个命令就能解决问题，命令特指五代编辑器内置方法 DCExecuteCommand()

```
//设置加粗
function SetBold(){
    var ctl=document.getElementById("myWriterControl")
    ctl.DCExecuteCommand("Bold",false,true)
}
//设置上标
function SetSuperscript(){
    var ctl=document.getElementById("myWriterControl")
    ctl.DCExecuteCommand("Superscript",false,true)
}
 //设置字体
 function SetFontName(){
    var ctl=document.getElementById("myWriterControl")
    ctl.DCExecuteCommand("FontName",false,'楷体')
}
```

3.在跟客户交流中让用户执行某个文档选项就能解决问题，文档选项特指五代编辑器内置选项 DocumentOptions 属性下的四个文档选项（SecurityOptions、ViewOptions、EditOptions、BehaviorOptions）

```
rootElement.DocumentOptions.BehaviorOptions.ParagraphFlagFollowTableOrSection=true
rootElement.ApplyDocumentOptions()
```