文档编码	
密级	
文档版本	
拟制人	DNA
日期	

久其研发与应用平台
（DNA）
界面框架
基础入门

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

更新说明

日期	版本	更新说明	作者	备注

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

1 概述

1.1 文档适用范围

该文档主要提供给应用 D&A 平台的用户界面层开发人员使用，并假设读者对 D&A 平台有一定基本的了解。

1.2 名词解释

名词	解释

1.3 标记符号

符号	说明	示例
■ 背景+斜体	子系统、模块、功能名称	*首页>常用>我的工作台*
■ 半角括号	窗体可视标签	请在[标题]区域输入工作流标题
■ 加粗+下划线	按钮或热链接	点击**办理**按钮
■ 双引号	界面提示文字	“任务已经发送给 XXX”
■ 	帮助	 帮助：XXX
■ 	提醒	 提醒：XXX
■ 	警告	 警告：XXX
■ 	重要	 重要：XXX
■ 	技巧	 技巧：XXX

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

DNA3.x 界面开发指南

2 前言

在软件开发过程中，界面编程虽然不是最核心的一环，但却是最为繁琐的一环，在B/S模式中更是如此。

作为DNA框架的重要组成部分，DNA界面框架旨在简化软件的界面开发过程，提升界面的开发效率和代码的可维护性。

DNA界面框架在应用级别主要包括控件库，Portal和虚拟组件，其中控件库和Portal已经基本完善，并支撑了资产管理、VA6等产品的开发和应用。虚拟组件则处于探索阶段，旨在提供一种全新的界面编程模型。

本文档主要对DNA界面框架的控件库部分进行介绍，通过阅读本文档，编程者应可以利用控件库提供的各种控件和相关接口进行界面开发，掌握DNA界面开发的基本原理和技巧。另外，在本文档的最后，还提供了控件库的完全参考、相关API以及一些常见问题，供编程者在开发过程中查询和参考。除此之外，我们还会通过其他的形式提供一些样例程序和范例代码。

需要注意的是，本文档主要侧重于介绍DNA界面编程的基础概念和控件特性，对于使用DNA Developer的界面设计器进行快速界面设计仅进行了少量介绍，我们认为，掌握本文档的各种概念和知识是进行DNA界面编程的根基，只有具备了这些根基，才能更好的利用工具，提高工作效率和质量。

3 快速入门

3.1 编程环境

在开始第一个界面程序之前，我们需要先了解DNA界面编程环境的一些基础概念。

首先，DNA界面编程环境是一个基于Java语言的虚拟界面环境，整个界面是一个以控件为基础的对象模型树，通过事件进行驱动。

多数情况下，编程者只需要针对这个虚拟界面环境进行编程，所有对象都在一个Java虚拟机内部，数据交换是非常简单的事情。然后，对于某些交互性较强的应用，为了提供更好的用户体验，则需要理解客户端和服务器端的概念，并且使用javascript语言进行客户端编程。这在某些程度上又和传统的BS编程环境（例如JSP）有些类似，不同的是，JSP中的服务器端编程和客户端编程是混杂在一起的，数据的交换非常随意，编写的代码没有规范，维护性较差。而DNA界面编程环境中，大多数情况是不需要关心客户端概念的，对于少量的客户端编程，其编程环境也是规范的，开发人员只能通过指定的编程方式和数据交换接口进行开发，代码可维护性高。

另外，DNA界面框架还提供了基于XML的Form文档模型来描述界面，主要用于可视化的界面设计工具使用。有兴趣的编程者也可以自己尝试使用。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

注：DNA界面的开发模式和SWT比较相似，熟悉SWT的编程者会很容易开始进行相关开发工作。

3.2 依赖插件

DNA界面框架基于微内核技术，各个层次的作用划分比较清楚，所以插件的数量也比较多，在进行DNA界面编程之前，首先应该清楚这些插件的作用和应该依赖那些插件。

插件名称	作用	必须
Com.jiuqi.dna.ui.common	包括公共常量定义接口JWT，	是
Com.jiuqi.dna.ui.wt	包括所有基础的对象接口、控件接口	是
Com.jiuqi.dna.ui.lightweight	轻量级控件库接口	可选
Com.jiuqi.dna.ui.thr.ole	Ole控件包装器接口	可选
Com.jiuqi.dna.ui.thr.ole.flash	一些扩展的Flash控件	可选
Com.jiuqi.dna.ui.custom	一些常用扩展的自定义控件 Combo工具包也在这个插件里面	可选
Com.jiuqi.dna.ui.portal	Portal插件	可选
Com.jiuqi.dna.ui.launch.web	包括Servlet定义的接口	可选

注：

1、 对于某些可选插件，例如Ole、Custom，LightWeight等，由于也很常用，一般也都会选择

2、 如果使用DNA Developer进行开发，则在建立界面工程后，会自动增加对其中部分插件的依赖。

3.3 第一个界面程序

3.3.1 界面入口

任何程序都需要一个入口来开始，即 main 函数。DNA 界面编程中的 main 函数就是 com.jiuqi.dna.ui.wt.UIEntry 接口的 createUI 方法。

```
/**
 * 创建UI界面应用
 * @param args  访问参数值
 * @param shell 界面应用的Shell控件
 */
public void createUI(String[] args, Shell shell);
```

其中 shell 是界面的根元素，需要在 shell 下面继续创建其他控件；args 参数则是访问链接的入口名称后面的部分，例如：

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

http://127.0.0.1/sample/arg1/arg2，则入口参数就是包括 arg1、arg2 的数组

实现 UIEntry 接口后，还需要在 dna.xml 中进行相关的注册才能被相关的终端访问。
完整的注册过程如下：

1、 注册 entry

```
<dna>
    <publish>
        <ui-entrys>
            <uientry name="sample"
                class="com.jiuqi.dna.ui.sample.MainEntry" />
        </ui-entrys>
    </publish>
</dna>
```

2、 注册 Servlet

```
<dna>
    <servlets>
        <servlet space="dna/ui" path="/sample/*"
    class="com.jiuqi.dna.ui.launch.http.SystemEntryServlet" />
    </servlets>
</dna>
```

注意：

1、 注册 entry 时，整个 ui-entrys 标签必须在 publish 标签下，而注册 servlet 时，整个 servlets 标签必须在 dna 根标签下
2、 注册 servlet 时，class 必须是 *SystemEntryServlet*
3、 如果想使用空的入口名称，即只输入地址和端口就可以访问，则 servlet 的 path 需要设置为“//*”

另外，在实际应用中，直接使用 UIEntry 的机会不会太多，Portal 层会提供基于 ViewPart 的入口，编程者只关心应用界面的部分，而不是整体。

3.3.2 Welcome 程序

不管怎样，我们都可以开始第一个界面程序了，不是 HelloWorld，是 Welcome。
下面是代码清单。

WelcomeEntry.java

```
package com.jiuqi.dna.ui.sample;

import com.jiuqi.dna.ui.common.constants.JWT;
import com.jiuqi.dna.ui.wt.UIEntry;
import com.jiuqi.dna.ui.wt.events.ActionEvent;
import com.jiuqi.dna.ui.wt.events.ActionListener;
import com.jiuqi.dna.ui.wt.layouts.RowLayout;
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
import com.jiuqi.dna.ui.wt.widgets.Button;
import com.jiuqi.dna.ui.wt.widgets.Label;
import com.jiuqi.dna.ui.wt.widgets.MessageDialog;
import com.jiuqi.dna.ui.wt.widgets.Shell;
import com.jiuqi.dna.ui.wt.widgets.Text;

public class WelcomeEntry implements UIEntry {

    public void createUI(String[] args, Shell shell) {
        shell.setLayout(new RowLayout());
        Label label = new Label(shell);
        label.setText("请输入你的姓名：");
        final Text text = new Text(shell, JWT.SINGLE);
        Button button = new Button(shell, JWT.PUSH);
        button.setText("确定");
        button.setImage(FileImageDescriptor.createImageDescriptor(
                "com.jiuqi.dna.ui.sample",
                "images/ok.gif"));
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String name = text.getText();
                if (name == null || name.trim().length() == 0) {
                    MessageDialog.alert("提示", "请输入姓名！");
                } else {
                    MessageDialog.alert("欢迎", "欢迎您，" + text.getText());
                }
            }
        });
    }
}
```

dna.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dna>
    <publish>
        <ui-entrys>
            <ui-entry name="welcome"
                class="com.jiuqi.dna.ui.sample.WelcomeEntry" />
        </ui-entrys>
    </publish>
    <servlets>
        <servlet space="dna/ui" path="/welcome/*"
            class="com.jiuqi.dna.ui.launch.http.SystemEntryServlet" />
    </servlets>
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
</dna>
```

通过浏览器访问 http://127.0.0.1/welcome 执行结果如下：

不输入姓名或者输入姓名，点击确定，都会弹出对话框：

编程者可以按照上面的过程自行完成程序的编写和运行，不管有何疑问，毕竟第一个界面程序就这样完成了。

接下来我们会对这个程序中涉及到的一些概念进行说明，为 DNA 界面开发打下坚实的基础。

3.4 控件的创建和销毁

在 welcome 程序中，首先创建了 3 个控件：一个标签、一个单行文本输入框、一个普

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

通的按钮。

可以看到，创建一个控件至少需要一个参数，即父容器。Welcome 例子中就是使用 shell 作为控件的父容器来创建控件的。

控件的销毁，在例子中没有，但是很简单，即调用控件的 dispose 方法即可。

注意：DNA 界面开发中，创建控件和销毁控件和 SWT 的模式类似，在控件构造时即完成控件创建，并没有使用类似 Swing 的 add 和 remove 的模式。

3.5 控件风格

在创建控件时，编程者会发现有时还需要另外一个参数，即style风格参数。

风格参数决定了控件的基本形式和特性，在控件创建后一般不能更改。

风格参数有两个作用，一是决定控件的基本表现形式，在DNA界面控件库中，很多控件，例如单行文本输入框、多行文本输入框、密码输入框，都是Text控件，普通按钮、单选按钮、多选按钮，都是Button控件，具体由使用者在创建控件时通过风格确定，如单行文本输入框对应JWT.SINGLE风格，多行文本输入框对应JWT.MULTI风格，普通按钮对应JWT.PUSH风格，单选按钮对应JWT.RADIO风格等等。

但是请注意，使用DNA Developer的可视化界面设计器时，控件的表现形式的风格已经被不同的控件来对应，开发者只需直接拖动相应的具体控件即可。

风格参数的另外的一个作用是决定控件的某些特性，例如Table、List、Tree等控件的选择模式风格，JWT.SINGLE对应单选模式，JWT.MULTI对应多选模式。

关于风格，还有如下的一些概念需要理解：

风格的组合：一个控件通常具有多种不同类型的风格，可以通过或运算符组合，例如JWT.CHECK|JWT.MULTI。

默认风格：创建控件时如果不指定风格值，则使用控件的默认风格值JWT.NONE，例如Button控件默认是JWT.PUSH类型。

风格值复用：所有的风格值都在JWT类中进行定义，不同的控件会共用相应的风格值，表达含义不同，例如JWT.MULTI用在Text控件上标识多行文本输入框，而用在Table上标识多选模式。关于风格值对不同控件的作用，请参考附录。

3.6 控件属性

控件属性是控件的基本特性，每个控件都有若干个属性供编程者设置，以改变控件的状态、行为等。每个属性通常都有 set 方法和 get（或者 is）方法。

例如文本输入框控件的文本属性，setText 方法是设置编辑的文本，getText 是得到编辑的文本。

控件的基本属性包括位置、大小、前景色、背景色等等，具体每个控件详细的属性说明请参考附录。

需要注意的是，系统会为每个用户缓存其会话对应的界面信息，只要对于的界面未被销毁，其包含的控件对象都会在内存中保存，而影响控件内存占用的因素则主要是控件属性，所以在编程过程中要尽量做到属性对象的重用和共用。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

3.7 控件事件

DNA 界面是事件驱动的，编程者只需要在控件上增加相应的事件监听器即可。例如在前面的例子中：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {......}
});
```

如果需要移除事件监听器，则需要保留监听器对象的实例，以便移除时使用，如：

```
ActionListener listener = new ActionListener() {
    public void actionPerformed(ActionEvent e) {......}
};
button.addActionListener(listener);
......
button.removeActionListener(listener);
```

注：如果是使用DNA Developer的可视化设计工具进行开发，当选择了监听某个事件之后，将自动增加一个事件监听器，开发者只需要实现相关方法即可。

一般情况下，编写事件监听器时，编程者需要理解在处理事件代码的过程中，用户界面是默认是处于阻塞状态的，这就是事件处理是同步模式。如果事件处理时间太长，则用户体验会很不好，这时应该启动异步任务，将长时操作交给后台处理，及时结束 UI 线程并返回用户界面。

不同的控件有不同的事件监听器，例如 MouseClickListener 是所有控件具有的，ActionListener 则是例如 Button、Text 等控件具有的。每个事件监听器至少有一个（一般都是一个，很少情况有两个）接口方法需要实现，接口方法包括一个事件对象作为参数，例如 ActionEvent，从事件对象中可以得到若干和事件相关的属性，例如事件源、是否按下 Ctrl 等键等等，详细的参数请参考附录。

除了一般的最常用的服务器端同步事件模式外，DNA 界面框架还提供异步事件和客户端事件，用于解决频繁访问服务器的应用场景，后面会做专门的说明，这里就不详细介绍了。

3.8 控件布局

我们知道，控件的 Width 和 Height 决定了控件的大小，控件的 X 和 Y 属性决定了控件在其父容器中位置，通过编程设置控件的这些属性就可以控制控件的大小和位置。但是由于用户界面分辨率多样性、用户界面伸缩性等原因，通过直接指定控件的位置和大小来编写用户界面，适用性很差。

和 SWT 类似，DNA 界面编程中也提供布局机制来更好的控制控件的位置和大小。

布局机制相当于是通过一定的模式来根据实际情况（分辨率、父区域大小等）动态计算控件的位置和大小，而不是事先通过编程指定。

一般而言，要使用布局机制需要进行两个方面的工作：

1、 为父容器设置布局算法（Layout 接口的子类），告诉父容器如何对其子进行布局
2、 为子控件设置对应的布局数据（LayoutData 接口的子类），告诉布局算法在布局技术时对该控件有何要求。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

需要注意的是一旦为容器设置了布局算法，则对其控件设置大小和位置就不再有效了。

布局算法只能由系统提供，还不能自行编写。目前提供了 5 种布局算法：FillLayout、RowLayout、GridLayout、BorderLayout、TableLayout。其中最常用的是前三种，基本满足大部分的应用界面。我们会在后面做详细的介绍。

3.9 控件样式

DNA 界面框架中提供一套完整的机制来控制控件的样式，下面先介绍有关的基本概念。

基本样式：每个控件包括前景色、背景色、背景图片、字体、光标、边框等 6 个基本样式。

控件行为状态：控件包括两个基本的行为状态 Hover（光标悬浮）和 Active（获取焦点）。控件在不同的行为状态下可以具有和普通状态下不同的基本样式。通过设置不同的样式值，可以快速的实现光标悬浮和获取焦点时控件的特殊的样式效果。

控件模式状态：一般情况下控件处于正常状态，当控件被禁用时即处于禁用状态，其他还可以根据具体控件来自定义只读、错误等扩展模式状态。可以为控件的不同模式状态设置不同的基本样式来快速实现控件处于不同的模式状态时的特殊样式效果。

注意：目前只有当前控件处于正常模式状态下时，控件的行为状态样式才会生效。

控件基类（Control）中包括了若干方法来设置和获取这些样式属性。

基本样式的设置方法包括：

```
public void setForeground(Color color);
public void setBackground(Color color);
public void setBackimage(ImageDescriptor image);
public void setFont(Font font);
public void setCursor(Cursor cursor);
public void setBorder(CBorder border);
```

光标悬浮时的样式的设置方法包括：

```
public void setHoverForeground(Color color);
public void setHoverBackground(Color color);
public void setHoverBackimage(ImageDescriptor image);
public void setHoverFont(Font font);
public void setHoverCursor(Cursor cursor);
public void setHoverBorder(CBorder border);
```

控件获取焦点时的样式的设置方法包括：

```
public void setActiveForeground(Color color);
public void setActiveBackground(Color color);
public void setActiveBackimage(ImageDescriptor image);
public void setActiveFont(Font font);
public void setActiveCursor(Cursor cursor);
public void setActiveBorder(CBorder border);
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

由于模式状态不是固定的，就没有提供直接的方法来设置不同模式状态下的样式的方法，但是可以通过样式对象 CssStyles 来设置。CssStyles 对象包括了基本样式、行为状态样式和模式状态样式，其详细接口请参考附录。

除直接通过代码设置控件的各种样式外，还可以通过文件样式单或者数据样式单（皮肤）来控制控件的各种样式，后面会做专门的介绍，这里不再详细说明。

3.10 图片

在 DNA 界面框架中，图片通过 ImageDescriptor 对象描述，所有涉及到图片的接口都接受该对象作为参数。

在实际应用中，一般有两种类型的图片，一种是和源代码一起提供的文件图片，一种动态生成的图片（通过存储在数据库中的数据或者其他图片处理算法生成）。创建这两种图片分别需要使用 FileImageDescriptor 和 DataImageDescriptor，将在后面详述。

编码实践：创建文件类型的图片时，需要传递插件 ID 和图片路径参数，但是显然前面 welcome 程序中的样例代码不值得提倡，应该由同一的类似 ResourceManager 类来对这些图片进行管理，插件 ID 也应该以常量定义的方式提供

4 基础机制

4.1 基础对象

4.1.1 JWT

com.jiuqi.dna.ui.common.constants.JWT 类可能是 DNA 界面编程中最常用的一个类，这里面几乎包括了所有和界面编程相关的常量定义，常用的可以分为以下几类（完整说明请参考 API）。

4.1.1.1控件风格名

控件风格名通常没有任何前缀，例如 JWT.WRAP、JWT.MULTI 等，具体的控件风格含义可以参考每个控件的说明。

4.1.1.2系统常用样式名

系统默认定义了若干个常用的样式名，可以直接把这些样式名设置到控件中以达到特定效果。所有系统样式名均以 CSS_SYSTEM_开头。目前有如下系统样式名可以使用：

CSS_SYSTEM_DEFAULT_BORDER：使用默认边框样式

CSS_SYSTEM_NONE_BORDER：使用无边框样式

CSS_SYSTEM_STYLED_BORDER：使用风格化边框样式

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

CSS_SYSTEM_STYLED_NOBACKGROUND：使用无背景样式

4.1.1.3字体修饰定义

目前定义了五种字体修饰，可以在构造 Font 对象时使用，如下所示：

FONT_STYLE_PLAIN：无修饰
FONT_STYLE_BOLD：黑体
FONT_STYLE_ITALIC：斜体
FONT_STYLE_UNDERLINE：下划线
FONT_STYLE_INLINE：删除线

4.1.1.4边框线风格定义

目前定义了五种边框，可以在构造 CBorder 对象时使用，如下所示：

BORDER_TYPE_PLAIN：无边框
BORDER_TYPE_IMAGE：图片边框
BORDER_STYLE_SOLID：实线边框
BORDER_STYLE_DOTTED：虚线边框
BORDER_STYLE_DASHED：点划线边框

4.1.1.5光标类型定义

系统定义了很多光标类型，用来构造 Cursor 对象，所有这些常量都以 CURSOR_为前缀。

4.1.1.6KeyCode 定义

在界面编程中，经常要根据键值判断是哪个键被按下了，系统定义了常用的 KeyCode，所有 KeyCode 常量均以 KEYCODE_为前缀，例如 F1 键的 Code 是 KEYCODE_F1。

4.1.1.7客户端事件名称定义

和控件的服务器端事件不同，客户端事件没有强类型的定义，都是通过字符串来标识，这些字符串被定义成常量，都以 CLENT_EVENT_为前缀，例如 CLENT_EVENT_ACTION。

4.1.2 基础样式相关

在界面框架的样式体系中，包括颜色、字体、边框、光标、图片、图片边框等样式对象，这里对每个对象进行详细的介绍。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

4.1.2.1Color

Color 对象用来描述颜色，可以用一个整型值来创建一个 Color 对象。

```
/**根据指定颜色值创建颜色对象
 * @param color 颜色值
 */
public Color(int color);
```

该值需要是一个 0 到 65535 之间的一个数值，实际应用最好使用十六进制值，例如 0xFF0000，相应的数据位以 RGB 形式对应。

另外，Color 中定义了若干常用的颜色对象，例如 COLOR_BLACK，COLOR_RED 等等，可以直接使用。

4.1.2.2Font

Font 对象用来描述字体。创建一个 Font 对象需要大小、名称和风格等参数：

```
/**创建指定的字体对象
 * @param size 大小 px
 * @param name 名称
 * @param style 风格，可选风格值可从JWT中查询
 */
public Font(int size, String name, int style);
```

字体大小的单位是pt，系统默认是9pt；
字体名称是字符串类型，默认是“宋体”；
字体风格的值为JWT中定义的常量，在前面已经有描述。

4.1.2.3CBorder

CBorder 对象用来描述控件的边框。创建一个 CBorder 需要粗细、风格和颜色值几个参数：

```
/**创建边框
 * @param thick 粗细，像素值
 * @param style 风格，可选风格在JWT中有常量定义
 * @param color 颜色值
 */
public CBorder(int thick, int style, int color) ;
```

边框粗细的单位是像素，系统默认是1像素；
边框风格的值在JWT中定义，可以是实线、虚线和点划线，默认是实线；

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

边框颜色值是整型值，默认是0（黑色）。

目前暂时不支持对边框的具体一边做独立的设置。

4.1.2.4Cursor

Cursor 对象用来描述光标。创建一个光标对象需要一个整型值，JWT 中有所有的光标常量定义：

```
/**创建光标值，光标值数值可从JWT中查询
 * @param cursor 光标值
 */
public Cursor(int cursor);
```

4.1.2.5ImageDescriptor

在前面的快速入门中我们提到，DNA 界面框架中的所有图片都是由 ImageDescriptor 对象来描述的，即图片描述符。ImageDescriptor 对象实例主要包括图片的唯一标识和图片的数据，以及图片的尺寸、类型等。ImageDescriptor 的类型根据图片产生的方式可以分为文件图片和数据图片，无论是哪种类型的 ImageDescriptor，都会占据系统内存，所以需要注意管理和销毁（调用 dispose 方法）。

一般情况，由于系统中文件类型图片数量是有限的，所以不需要太关注相应 ImageDescriptor 资源的销毁；而数据类型的图片就要特别注意，特别是类似数据图表的图片，一般都是临时生成，就要及时销毁，否则会造成系统内存泄漏。

创建一个文件类型的图片描述符需要使用 FileImageDescriptor 对象的静态方法，并提供两个参数（插件 ID 和图片在插件中的相对路径）：

```
/**
 * 创建文件型图片描述符
 * @param pluginId
 *            文件所在插件id
 * @param imageFilePath
 *            文件在插件工程中的路径
 * @exception JWTException
 *                图片路径不正确时将会抛出异常
 * @return 图片描述对象
 */
public static FileImageDescriptor
    createImageDescriptor(String pluginId, String imageFilePath);
```

创建一个数据类型的图片描述符需要使用 DataImageDescriptor 的几个不同的静态工厂方法，这些方法的参数主要是图片 ID、数据和所属控件几个参数的不同组合形式。下面对这几个参数的作用进行说明：

- 数据：数据的提供主要有三种不同的方式，即 byte[]、ImageData 和

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

ImageDataProvider。其中 byte[]直接将所有数据生成一张图片，ImageData 对象可以指定其中 byte[]中的某一部分生成一张图片，而 ImageDataProvider 则需要实现一个通过 context 生成一个 ImageData 的方法来提供图片数据。

- 图片 ID：图片 ID 是可选的，主要用来唯一标识图片，如果系统中已经有相应 ID 的图片，则不会再创建新的图片。图片 ID 应该是有意义，例如如果对应的图片是由数据库的一条记录来生成的，那一般可以用相应记录的 ID 来作为图片 ID。绝对不要随机生成 ID 来作为图片 ID。
- 宿主控件：一般情况，如果图片是临时生成的，就无法赋予其一个有意义的 ID，这时为便于销毁，通常在创建图片描述符时传入一个宿主控件参数，这样当宿主控件销毁时会自动销毁对应的图片资源。

典型的方法例如：

```
/**
 * 根据有意义的id和图片数据来创建数据图片
 *
 * @param imageId
 *            图片描述id
 * @param data
 *            数据
 * @return 图片描述对象
 */
public static ImageDescriptor
        createImageDescriptor(String imageId,byte[] data);

/**
 * 创建一个临时数据图片并绑定到宿主控件上，当宿主控件销毁是图片资源自动销毁
 *
 * @param data
 *            图片数据
 * @param owner
 *            图片宿主控件
 * @return 图片描述对象
 */
public static ImageDescriptor
        createImageDescriptor(byte[] data,Widget owner);
```

4.1.2.6ImageBorder

ImageBorder 对象是一个特殊的资源对象，用来描述特殊结构的图片边框，主要用于 ImageBorderComposite 和 StyledPanel 两个控件，作为参数使用。

一个 ImageBorder 对象包括 8 个 ImageDescriptor 对象，即用 8 张图片来描述一个图片边框，参数通过一个 ImageDescriptor 数组传入。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
/**创建图片边框
 * @param images 图片数组
 */
public ImageBorder(ImageDescriptor[] images) ;
```

8 张图片分别对应左上角图片、上边背景、右上角图片、右边背景、右下角图片、下边背景、左下角图片、左边背景。

另外，ImageBorder 提供了几个系统定义的边框可以使用：

- ImageBorder.COMMON_IMAGE_BORDER
- ImageBorder.DEFAULT_IMAGE_BORDER
- ImageBorder.STYLED_WINDOW_IMAGE_BORDER
- ImageBorder.STYLED_PANEL_IMAGE_BORDER

4.1.3 尺寸相关

4.1.3.1Point

Point 对象用来描述一个点的位置或者一个矩形的尺寸，包括 x 和 y 两个属性。当描述点位置时，x 和 y 分别对应点的 x 坐标和 y 坐标；当描述矩形的尺寸时，分别对应矩形的长和宽。

4.1.3.2Rectangle

Rectangle 对象用来同时描述位置和尺寸，包括 x、y、width、height 四个属性，分别对应 x 坐标、y 坐标、长、宽。

Rectangle 对象中还提供了若干个方法来帮助进行一些和位置尺寸相关的处理，例如合并区域，具体请参考 API，这里不做详述。

4.1.4 其他

4.1.4.1GraphicsContext

GraphicsContext 是一个工具类，主要提供一些系统字体、文字尺寸计算相关的方法，目前提供了两个方法：

```
/**计算指定字体下指定字符串的尺寸
 * 只计算单行文本，参数字符串中"\n\r"等换行符无效
 * @param font 字体
 * @param text 文本
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
     * @return Point 大小
     */
    public static Point computeSize(Font font, String text);


    /**将字体对象转换为JDK的AWT中字体类型的对象
     * @param font 字体对象
     * @return AWT字体对象
     */
    public static java.awt.Font convertJWTFont2AWTFont(Font font);
```

4.1.4.2PageConfigure

PageConfigure 对象主要用于 Display 提供的打印功能中的页面设置，该对象的构造如下所示：

```
    /**
     *
     * @param pageWidth
     *          页面宽度（毫米）
     * @param pageHeight
     *          页面高度（毫米）
     * @param leftMargin
     *          页面左边距（毫米）
     * @param topMargin
     *          页面上边距（毫米）
     * @param rightMargin
     *          页面右边距（毫米）
     * @param bottomMargin
     *          页面下边距（毫米）
     */
    public PageConfigure(int pageWidth, int pageHeight, int leftMargin,
            int topMargin, int rightMargin, int bottomMargin) ;
```

关于Display提供的打印功能后面会做简单介绍。

4.2 Display 对象

Display 对象是 DNA 界面框架编程中一个非常重要的和用户会话相关的对象，可以通过任何控件的 getDisplay 方法来获取到，也可以在 UI 线程中通过其 Display.getCurrent 静态方法得到，在非 UI 线程则无法得到（关于 UI 线程和非 UI 线程的概念后面会做详细介绍）。

直观上来看 Display 对应一个用户会话屏幕，提供以下基本的方法，以便得到各种基础信息：

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

- 得到用户实际屏幕的尺寸（getScreenWidth 和 getScreenHeight 方法）
- 得到屏幕的分辨率（getDPI 方法）
- 得到当前会话的界面入口名（getEntryName 方法）
- 得到当前会话的本地化信息（getLocale 方法）
- 得到主窗口 Shell 对象（getMainShell 方法）

除此之外，Display 对象还提供其他多种机制和方法，下面分别做详细介绍。

4.2.1 下载

如果需要将数据下载到客户端，或者在浏览器中打开对应的编辑器来查看内容，则可以使用 Display 中的 exportFile 方法。

Display 中一个共有三个 exportFile 方法：

```
/**
 * 导出指定数据
 * @param fileName      输出客户端的文件名称
 * @param contentType   输出内容的类型，如文本文件为：text/plain
 * @param contentLength  内容长度
 * @param exporter      导出器
 */
public void exportFile(String fileName, String contentType,
       int contentLength, Exporter exporter);


/**
 * 导出指定数据
 * @param fileName      输出客户端的文件名称
 * @param contentType   输出内容的类型，如文本文件为：text/plain
 * @param contentLength  内容长度
 * @param exporter      导出器
 */
public void exportFile(String fileName, String contentType,
       int contentLength, ExporterWithContext exporter);


/**
 * 导出指定数据
 * @param fileName      输出客户端的文件名称
 * @param contentType   输出内容的类型，如文本文件为：text/plain
 * @param contentLength  内容长度
 * @param exporter      导出器
 * @param openWindow    是否在一个新窗口中直接打开
 */
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199　　传真：010-58561196
网址：http://www.jiuqi.com.cn　E-mail：jiuqi@jiuqi.com.cn

```
public void exportFile(String fileName, String contentType,
        int contentLength, ExporterWithContext exporter,
        boolean openWindow);
```

上述方法中的文件名、文件类型、文件长度等参数的含义比较明确，而导出器接口有两种，分别是 Exporter 和 ExportWithContext：

```
/**
 * 普通导出器，将待导出的数据写到客户端的输出流中
 */
public static interface Exporter {
    public void run(OutputStream outputStream) throws IOException;
}

/**
 * 带Context的导出器，将待导出的数据写到客户端的输出流中，提供Context对象
 */
public static interface ExporterWithContext {
    public void run(Context context, OutputStream outputStream)
            throws IOException;
}
```

编程者需要根据需要选择实现相应的接口，将需要下载的文件内容写入到输出流中。

另外，如果 openWindow 参数为 true，则意味浏览器不会弹出保存对话框，而是在一个新窗口中直接打开下载内容（仅对 IE 浏览器生效）。

4.2.2 打印

Display 提供了一个基本的打印网页的功能，即调用 Web 浏览器的打印方法来打印所输出的内容，方法如下所示：

```
/**
 * 打印数据
 *
 * @param pageConfigure
 *          页面设置
 * @param printers
 *          打印内容输出器（多页，每页独立输出）
 */
public void print(
        PageConfigure pageConfigure, PrinterWithContext[] printers);
```

打印需要两个参数，一个是页面配置对象，用于描述页面尺寸和边距信息等，前面已经

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

有所介绍。另外一个参数是打印器接口，是一个数组对象，每个对应一个打印页：

```
/**
 * 打印器，将待打印的内容写到writer中，提供Context对象
 */
public static interface PrinterWithContext {
    public void run(Context context, PrintWriter printWriter)
            throws IOException;
}
```

编程者需要实现该接口来向 PrintWriter 对象中输出所需要打印的内容，通常是 HTML 文档。

4.2.3 状态监听定时器和状态查看监听器

Display 中提供 createStateMonitorTimer 和 removeStateMonitorTimer 两个方法来创建和移除状态监听定时器，通过该定时器来要求 web 浏览器按照指定间隔来查看服务器端的特定状态，并触发状态查看监听器。

关于状态监听定时器和状态查看监听器的细节在后面的 UI 线程和非 UI 线程相应机制中会做详细的介绍。

4.2.4 执行界面操作任务

Display 提供 syncExec 方法来执行界面操作任务，关于该方法的细节在后面的 UI 线程和非 UI 线程相应机制中会做详细的介绍。

4.2.5 存取会话数据

由于 Display 本身就是和用户会话相关的，所以可以利用 Display 来存取和会话相关的数据。

Display 提供了下面几个方法来存取会话数据：

```
/**
 * 在Display中缓存一个数据
 * @param key    键
 * @param value  值
 */
public void setData(Object key, Object value);

/**
 * 从Display中获取已缓存的数据
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199　　传真：010-58561196
网址：http://www.jiuqi.com.cn　E-mail：jiuqi@jiuqi.com.cn

```
     * @param key 键
     * @return 值
     */
    public Object getData(Object key);

    /**
     * 从Display中删除已缓存的数据
     *
     * @param key 键
     * @return 值
     */
    public Object removeData(Object key);
```

4.2.6 存取 Cookie 数据

通过 Display 还可以在用户浏览器环境中存取 Cookie 数据，相关方法如下：

```
    /**
     * 设置Cookie，服务器向客户端同步时将重置客户端的Cookie。 更改Cookie内容后必须调用此方法以保证Cookie数据被重新发送到客户端。
     * @param cookie 数据结构
     */
    public void setCookie(Cookie cookie);

    /**
     * 获取Cookie，客户端首次同步服务器端时将客户端保存的Cookie传送至服务器端，构造服务器端Cookie结构
     * @return Cookie
     */
    public Cookie getCookie();

    /**
     * 清除Cookie，服务器向客户端同步时将清除客户端的Cookie
     */
    public void clearCookie();
```

需要注意的是，相关方法只对整个 Cookie 对象进行存取，所以一般而言在操作 Cookie 之前需要先调用 getCookie 方法得到 Cookie 对象，存取完毕后再调用 setCookie 方法设置回去。

4.2.7 控件事件处理前后的监听

Display 提供 UserRequestListener 监听器来辅助完成用户界面事件的处理前和处理后的

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

处理。下面是该监听器的增加和移除的方法：

```
/**
 * 加入用户请求处理监听器
 * @param listener  用户请求处理监听器
 */
public void addUserRequestListener(UserRequestListener l);

/**
 * 移除用户请求处理监听器
 * @param listener  用户请求处理监听器
 */
public void removeUserRequestListener(UserRequestListener l);
```

需要注意及时移除所注册的监听器。

该监听器的接口说明如下：

```
/**
 * 用户请求过程处理监听器
*/
public interface UserRequestListener extends WTEventListener {

    /**
     * 处理之前触发该方法
     * @param e 用户请求过程事件
     */
    public void beforeProcess(UserRequestEvent e);

    /**
     * 处理之后触发该方法
     * @param  用户请求过程事件
     */
    public void postProcess(UserRequestEvent e);
}
```

4.2.8 应用和取消样式单

Display 提供了在当前用户会话**应用**和**取消应用**系统样式单的方法，详细相关机制请参考后面关于控件样式和样式单相关的说明。

4.2.9 设置和得到当前焦点控件

Display 中提供设置和得到当前焦点控件的方法：

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199　　传真：010-58561196
网址：http://www.jiuqi.com.cn　E-mail：jiuqi@jiuqi.com.cn

```
/**
 * 设置当前焦点控件
 * @param widget  控件
 */
public void setFocusedWidget(Control widget);

/**
 * 获取当前焦点控件
 * @return 控件
 */
public Control getFocusedWidget();
```

4.2.10 重置（reset）和刷新（Refresh）

Display 提供 reset 和 refresh 方法来处理界面的整体刷新，不同的是前者将重置整个会话（所有会话数据丢失），而后者则只是重新加载主页面。所以 reset 方法通常用于类似注销功能，而 refresh 则通常在类似切换主题时使用。

4.2.11 打开 URL 和打开入口

通过 Display 可以在新的 Web 浏览器窗口中打开一个指定的 URL，默认可以不指定窗口名，这样每次都会打开新的窗口；如果指定窗口名，则总会在一个窗口中打开 URL。

```
/**
 * 加入需要打开的URL列表
 * @param url URL
 * @param windowName 打开URL的窗口名
 */
public void openUrl(String url, String windowName);
```

通过 Display 可以在新的 Web 浏览器中打开当前服务器上指定的界面入口，并指定相应参数，如果 singleton 参数为 true，则每次都会重用一个浏览器窗口，否则每次都会打开新窗口。

```
/**
 * 加入需要打开的入口列表
 * @param args
 * @param singleton
 */
public void openEntry(
        String entryName, String[] args, boolean singleton);
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199　　传真：010-58561196
网址：http://www.jiuqi.com.cn　E-mail：jiuqi@jiuqi.com.cn

4.2.12 浏览器级关闭提示

在很多和数据编辑有关的场景，我们需要在用户离开当前编辑器时提醒用户是否需要保存数据。如果是在DNA框架内部，可以通过监听相应控件事件进行处理，但是如果是用户直接关闭Web浏览器，则需要额外的处理方式。

由于Web浏览器的限制，目前在用户直接关闭浏览器，或者转向其他URL时，只能对用户给予提示，并提供“是”和“否”两个选项，用户选择是则关闭或者离开浏览器页面，选择否则取消。

Display中提供了相应的方法来使得编程者可以要求界面框架在Web浏览器关闭或者离开时给予相应提示，方法如下：

```
/**
 * 标记关闭浏览器的提示
 * @param owner 宿主部件，关闭提示与该部件相关联，部件销毁时自动响应清除标记
 * @param description 提示信息
 */
public void markCloseNotify(Widget owner, String description);
```

owner参数主要用来方便取消这个标记，description参数用来指定提示消息。

另外，编程者也可以根据需要取消之前的标记，只需要传入之前关联的宿主控件即可：

```
/**
 * 移除关闭浏览器的提示
 * @param owner  宿主部件
 */
public void removeCloseNotify(Widget owner);
```

4.2.13 默认异常处理

类似JSP的ErrorPage，编程者可以使用Display对象为系统设置一个默认的错误处理器，接口方法为：

```
/**
 * 设置全局异常处理器，该处理器将捕捉事件处理程序中产生的未处理的异常
 * @param exceptionHandler   异常处理器
*/
public void setExceptionHandler(ExceptionHandler exceptionHandler);
```

ExceptionHandler接口如下所示：

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
public interface ExceptionHandler {

    /**
     * 界面异常处理方法
     * @param t 异常
     */
    public void processException(Throwable t);

}
```

如果没有为系统设置默认异常处理，当发生未被处理的异常时，最终当前用户会话会被终止。

4.3 UI 线程和工作线程

和所有的界面编程环境一样，UI 线程被定义为主线程，原则上只能在 UI 线程中才能对控件进行操作。UI 线程由用户动作发起，并且通过控件事件响应，在整个 UI 线程处理过程中，用户界面将会挂起，直到事件处理结束。

对于传统的桌面界面编程，在 UI 线程处理时，将无法处理界面事件，时间稍长就会出现著名的“界面无响应”错误。在 DNA 界面环境中，不会出现这种错误，但是界面同样会被锁定（状态栏显示“**正在与服务端交互…**”），用户无法操作界面，时间太长体验也是不好的。所以无论如何，**不要在 UI 线程中进行长时间的业务处理操作，应该尽快让 UI 事件响应完成**。

比较好的方式是启动一个异步线程（工作线程）来完成长时间的工作，一般可使用 Context 的 asyncHandle 或者 asyncGet 方法来启动工作线程。

当然，通常在启动异步工作线程后，还需要在完成处理后再去更新界面，或者在处理过程中不断报告进度。这时需要注意，前面提到，在非 UI 线程中是不能直接操作控件（会报运行时异常），具体点来说，就是无法在后台直接把数据“推”到界面（浏览器），只能通过间接的手段，在 DNA 界面框架中提供了以下的机制来解决这个问题：

4.3.1 服务器“推”机制

4.3.1.1界面操作任务

界面操作任务要解决的问题是在非 UI 线程中不能直接操作界面控件的问题，既然不能直接操作，那就间接操作，即执行一个界面操作任务，这个是通过 Display 接口的 syncExec 方法来完成的，该方法说明如下：

```
/**
 * 服务器端推（非UI线程中调用） 任务将进入队列，在下次UI请求中被执行。
 *
 * @param runnable
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
     *            任务
     */
    public void syncExec(Runnable task);
```

即可以在 Runnable 接口的 run 方法中进行控件的操作，并将 Runnable 对象通过 Display 的 syncExec 进行执行。如果是 UI 线程中调用 syncExec，则 Runnable 的 run 方法会被马上调用，**如果是在非 UI 线程中调用 syncExec，则 Runnable 的 run 方法会在下一次 UI 线程的最后被调用**。

4.3.1.2状态查看定时器

前面说到，非 UI 线程中通过 display.syncExec 可以加入一个执行任务，并在该任务里面进行界面操作，而相应的任务将在下一次 UI 线程的**最后**被执行。

但是指望用户操作界面来触发一次 UI 线程，然后才把工作线程对界面的修改反应到界面，这显然是不可以接受的。所以在 DNA 界面框架中提供了一个状态查看定时器的机制来解决这个问题。

该机制的基本原理就是通过**创建一个状态查看定时器来要求浏览器端定时的到服务器来查看是否有需要被执行的界面操作任务，如果有，则自动的发起一次 UI 线程，保证相应的界面执行任务能被及时的执行并反应到用户界面中**。

Display 对象中提供了创建和移除状态查看定时器的方法，如下所示：

```
/**
 * 创建指定时间间隔的定时器 <b>注意：单位为秒</b>
 * @param interval  定时秒数
 * @return 定时器对象
 */
public Object createStateMonitorTimer(int interval);

/**
 * 移除定时器
 * @param timer 定时器
 */
public void removeStateMonitorTimer(Object timer);
```

需要注意以下几点：

1、 定时器的定时单位是秒，实际应用可以根据异步的界面操作需要多久被反应到界面来确定这个时间。一般来说，查询进度相关的可以在 1 秒，邮件通知可以在 30 秒。

2、 需要注意，系统允许有不同的模块创建各自的状态查看定时器，然后使用最小的那个时间来进行统一的操作，而不会独立为每个定时器单独进行操作。

3、 **创建状态查看定时器后，如果相应程序关闭，应该将相应定时器移除**

4.3.1.3状态查看监听器

大多数时候，我们在工作线程中没有办法获取到界面控件引用，而且**在工作线程中操作**

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199　　传真：010-58561196
网址：http://www.jiuqi.com.cn　E-mail：jiuqi@jiuqi.com.cn

界面控件也不符合系统的架构模式。所以，不应该在工作线程（特别是在 Service 中）来执行界面操作任务，而应该在状态查看监听器中进行。

状态查看监听器就是上面提到的由状态查看定时器来定时触发浏览器向服务器端查询是否有界面执行任务时产生的事件监听器。

该事件监听器可以通过 Display 的以下方法来增加和移除。

```
/**
 * 添加状态检查监听器，当客户端发起一个状态服务时状态检查监听器将被调用
 * @param listener 状态检查监听器
 */
public void addStateMonitorListener(StateMonitorListener l) ;

/**
 * 移除状态检查监听器
 * @param listener  状态检查监听器
 */
public void removeStateMonitorListener(StateMonitorListener l);
```

StateMonitorListener 接口说明如下：

```
/**
 * 状态监控监听器
*/
public interface StateMonitorListener extends WTEventListener {
    /**
     * 指定间隔时间触发此方法
     * @param e 状态监控事件
     */
    public void stateMonitorCheck(StateMonitorEvent e);
}
```

使用状态查看监听器需要注意以下几点：

1、 StateMonitorListener 的 stateMonitorCheck 方法执行过程是**非 UI 线程**的，所以在该方法内部也应该使用 display.syncExec 来执行界面操作任务

2、 需要注意在合适的时机移除所注册监听器

4.3.2 应用场景

UI 线程、工作线程和服务器端“推”机制的应用场景主要有：

1、 数据查询

数据查询通常会需要较长时间，所以必须启动异步线程来执行。一般而言，状态查看定时器的定时时间可以根据相应查询能反映的进度的情况和整个查询需要花费的总时间来合理设置。例如如果一个查询总共估计花费 1 秒以内，这时把定时器设置为 2 秒，则会造成查询缓慢的假象。而如果一个查询要花费很长时间，定时器设置 为 5 秒，

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

则会造成进度报告不够及时。

2、 长时间操作

用户对于长时间操作一般有两种需求，一种用户需要等待操作结束，但是应该提供处理进度。另外一种情况用户并不想等待操作结束，而是在提交请求后会做其他操作，由其自行的查询处理进度。

对于第一种情况，就需要利用和数据查询同样的模式来处理，第二种情况，应该提供统一查询系统工作线程工作情况的功能。

3、 消息通知：

系统服务层会提供消息通道，以便类似聊天、预警、邮件通知等发生的消息可以通畅的在服务层传输。而消息最终需要利用服务器“推”机制来通知到用户界面。

ProgressPage 和 ProgressBar 组件对上述机制进行了一定的封装，可以较好的完成查询和长时间操作的进度报告，其使用机制后面会做详细介绍。

4.4 控件事件机制

我们在快速入门中提到，DNA 界面是事件驱动的，编程者只需要在控件上增加相应的事件监听器即可。我们可以根据事件监听器的处理位置是在服务端或者客户端来分为服端事件和客户端事件。

4.4.1 服务端事件

服务端事件即对相应事件的处理是通过服务端的 java 代码来完成的，例如对 Button 的点击动作，我们可以在 Button 上增加 ActionListener 监听器的实现来处理。

由于是在服务端进行处理，编程者可以很方便的进行编码，轻松的获取控件状态，调用业务逻辑，更新界面。这也是 DNA 界面框架的一大优势所在。

对于服务端事件，我们还需要有以下的认识：

1、 服务端事件通常由用户操作引起，但是也可以由事件监听器实现中的 java 代码调用引起，例如设置了文本输入框的值也会触发文本输入框的文本变化事件

2、 如果没有必要对事件进行处理，就不要注册相应的处理器。界面引擎只能判断相应事件是否有监听器，而无法判断监听器实现是否为空，而一旦判断相应事件有监听器，则当用户动作触发相应事件时，就会通知到服务端，产生调用。

3、 应该尽量避免无谓的服务端事件监听，类似保存、删除按钮这类控件的点击行为由服务器端处理是很正常的，但是对表格行的选择事件也提交到服务端处理，就很难编写出高效的界面程序。特别是那些对用户操作响应要求很高的核心功能来说，就要特别注意了。

4、 很多频繁触发的事件并没有对应的服务端事件，例如键盘动作、光标移动等，但是可以使用客户端事件来很处理。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

4.4.1.1异步事件

服务端事件默认都是同步的，即在用户动作触发到事件响应返回的这段时间内，界面是阻塞的，这点我们在之前也已经多次说明了。但是通过在实现事件监听器时，同时声明实现 AsyncListener 接口，就可以改变这种策略，从而使得在用户动作触发到事件响应返回的这段时间内，用户仍然可以操作界面。

AsyncListener 是一个标识接口，没有任何方法需要实现。

```
/**
 * 异步监听器标识接口，实现此接口的监听器为异步事件监听器
*/
public interface AsyncListener;
```

我们要理解，即使声明了事件是异步的，即在服务器处理前一个事件时，仍然可以操作界面，也并不意味着可以有多个事件被同时处理，界面引擎只是将新产生的事件放入队列，等前面的事件处理完毕后再顺序处理。对应 UI 线程而言，永远都是单线程的。

所以在使用了异步事件后，可能出现这种情况：用户点击按钮，产生一个服务端事件并由服务端代码处理，这时因为声明了该事件为异步事件，用户可以继续在另外一个输入框中输入数据；但是如果第一个事件的监听器中的代码正好是将相应的输入框禁用或者销毁了，这就意味着后面的输入动作是无效的。这种情况为保证界面数据一致性，界面引擎会自动纠正这些问题，将输入的值清除，但是很显然，这样对用户来说未必友好。

所以，异步事件在某些场合的使用会带来更好的界面体验，但是也存在导致用户操作失效的问题，编程者在使用时需要注意。

4.4.2 客户端事件

和服务端事件对比来看，客户端事件则是使用 javascript 脚本函数来处理相应用户操作的事件。由于 javascript 代码在浏览器端执行，所以相比服务端事件而言，具有较好的用户体验，其缺点也是显而易见的，即编码较为繁琐，除此之外，客户端事件的处理还有以下特点：

1、 在客户端事件监听器的处理脚本中，只能对控件进行读写操作，不能创建和销毁控件。
2、 在客户端事件监听器的处理脚本中，无法直接操作到服务端的数据，所以需要预先将所需数据准备好。
3、 客户端事件监听器的处理脚本都是静态的，必须以 js 文件的形式提供，无法像在 jsp 中那样动态输出 js 脚本，这在一定程度是是对脚本编写的规范，避免出现难以维护的代码。

总的来说，在必要的地方使用客户端事件，可以提供更好的界面体验，特别是对于那些核心功能而言。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

4.4.2.1事件监听器

客户端事件监听器的实现对应一个全局的 javascript 函数，并放在一个 js 文件中（该为需要通过 dna.xml 中注册），例如：

```
Sample.testEventHandler = function(event, widget) {
    ……
};
```

其中 event 和 widget 是两个固定的参数，event 是事件对象，从中可以得到事件相关信息，widget 是对应的控件对象。

客户端事件监听器的注册需要在服务器端完成，利用 Widget 的相关方法：

```
/**
 * 注册指定类型的客户端事件脚本，当指定类型的客户端事件被触发时，执行脚本代码
 * @param eventType  客户端事件类型
 * @param name  全局函数名称
 * @return String 脚本的注册id，删除该脚本时用
 */
public String addClientEventHandler(String eventType, String name);
```

其中 evenetType 参数是客户端事件类型，其取值在 JWT 中有常量定义（请参考前面有关 JW 类的说明），而 name 参数则是全局的 javascript 函数名，例如 Sample.testEventHandler。

事件监听器的注销需要使用注册时返回的 ID：

```
/**
 *注销客户端消息脚本
 * @param id  注册时产生的脚本id
 */
public void removeClientMessageHandler(String id);
```

需要特别注意，在早期的版本中，注册事件监听器时并不是直接通过全局函数名，而是通过另外一个不需要注册到 dna.xml 中的 js 文件和其中的特定函数名来完成的。通常需要另外两个参数：

1、 JsFileDescriptor：指定的 js 文件描述符，该 js 文件和全局的 js 文件不同，不需要注册到 dna.xml 中

2、 FunctionName：指定文件中的相应函数名

这种机制已经被废除了，请直接定义全局函数，并注册全局函数名。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199　　传真：010-58561196
网址：http://www.jiuqi.com.cn　E-mail：jiuqi@jiuqi.com.cn

4.4.2.2脚本文件注册

前面所说的 javascript 函数都需要放到一个 js 文件中，并通过 dna.xml 进行注册，以便被界面框架识别，成为全局函数。

注册方式例如：

```
<ui-client-scripts>
  <script name="xxxx" path="/xx/xx.js" />
</ui-client-scripts>
```

其中 name 为注册标识，保证唯一即可。Path 为指定 js 的位置。

4.4.3 一些特殊事件说明

大多数的控件事件都比较好理解，例如 Button 的 ActionListener，List 的 Selection 事件等，直接通过 API 或者 java doc 就可以知道如何使用。

这里主要对一些比较特殊的事件进行一些额外的说明。

4.4.3.1WindowClosing 事件

WindowClosing 事件针对 Window（窗体）和 TabFolder（页签）。

我们以 Window 为例来说明，如果编程者没有在 Window 上注册 WindowListener 监听器，则在用户点击窗口关闭按钮时，窗口将直接被关闭。但是如果编程者在 Window 上注册了 WindowListener 监听器，则在用户点击窗口关闭按钮时，窗口将不会被关闭，而是触发监听器的 windowClosing 方法，由该方法的实现来进行处理，如果需要关闭窗口，编程者需要自行调用窗口的 close 或者 dispose 方法。

4.4.3.2SelectionChanging 事件

SelectionChanging 事件针对 List、Table、Tree、TabFolder 等控件。

如果没有为这些控件增加相应的 SelectionChangingListener 监听器，则当用户选择相应项目时，将直接完成选择操作。但是如果编程者为这些控件增加了 SelectionChangingListener 监听器，则用户的选择行为将不会被完成，而是会触发监听器的 selectionChanging 方法，并告知编程者用户将要选择的项目，由编程者来进行处理。

4.4.3.3单击和双击事件

单击和双击事件本没有什么特殊之处，主要是因为历史原因，即早期的界面框架只提供了鼠标事件，相应的监听器中包括了单击和双击两个方法，这样不管客户端是否单击还是双

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

击都会触发服务器端处理。

由于界面框架在处理客户端单击行为时，总是要等一个间隔以便确认用户不是在进行双击，所以对单击的响应反而变慢了。为了避免这种问题，后来将鼠标事件分为了单击和双击两个事件，编程者应该尽可能的使用具体的事件监听器。

即如果确实应该同时监听单击和双击两个事件，则可以使用鼠标事件监听器，但是如果仅需要监听单击或者双击，就应该使用具体的单击事件监听器或双击事件监听器。

4.5 界面消息机制

4.5.1 概述

事件机制是基于单一控件行为的基础处理机制，DNA 界面是由控件事件来驱动的。但是我们在实际编程中会发现，很多时候我们不仅要关心某个控件的行为，而且还要关注其他界面视图的行为，这些界面未必是当前界面编写者所编写。如果要通过传统的方式来实现，就必定造成两个界面的代码之间形成耦合，有时甚至无法实现。

界面消息机制为更大的界面元素（Page）之间建立了一个数据通道，使得不同的界面之间可以使用这个通道传递数据，被传输的数据就是消息。消息机制的本质是提供一套机制，使得数据体（消息）可以从界面元素树一个 Page 开始，向上（冒泡）或者向下（广播）传输到其他 Page。

4.5.2 相关接口说明

4.5.2.1Situation 对象

Situation 是一个和界面元素相关的上下文对象，继承至 Context。整个界面有一个根 Situation，其余的 Situation 对象都是与 Page 对象一一对应的，即一个 Page 对象对应一个 Situation 对象，Page 的树形结构也反映到 Situation 的层次上，从而形成了一颗 Situation 树。关于 Situation 树后面还会做详细的说明。

Situation 继承 Context，Context 的生命周期是请求级的，而 Situation 的生命周期则和其所属的 Page 页面一致，当 Page 对象销毁时，Situation 也将被销毁。另外需要注意，Situation 只能在 UI 主线程中被使用。

Situation 对象通常可以通过 page 的 getContext 方法获取。

除了 Context 具有的接口方法外，Situation 还具有发送消息、注册消息监听器的相关接口。与消息相关的主要接口如下所示：

```
/**
 * 获得父级情景
 */
public Situation getParent();

/**
 * 获得根级情景
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
     */
    public Situation getRoot();

    /**
     * 向自己以及所有下级发送消息，消息处理完成之后才返回。
     *
     * @param message
     *            消息对象
     * @return 返回消息处理过程中的一些信息
     */
    public <TMessage> MessageResult<TMessage>
                broadcastMessage(TMessage message);

    /**
     * 向自己以及各上级依次发送消息，消息处理完成之后才返回。
     *
     * @param message
     *            消息对象
     * @return 返回消息处理过程中的一些信息
     */
    public <TMessage> MessageResult<TMessage>
                bubbleMessage(TMessage message);

    /**
     * 声明监听某消息
     *
     * @param <TMessage>
     *            消息类型
     * @param messageClass
     *            消息的类
     * @param listener
     *            监听器
     * @param directions
     *            监听消息的方向，不指定代表监听各种方向
     */
    public <TMessage> MessageListenerRegHandle<TMessage>
           regMessageListener(
               Class<TMessage> messageClass,
               MessageListener<? super TMessage> listener);
```

通过 Situation 接口可以以广播（broadcast）和冒泡（bubble）的方式来发送具体消息对象，但目前只能使用同步的方式来发送消息，异步方式在界面层暂时没有实现。

广播将把消息发送给当前 situation 以及其所有子层次的各个 situation，冒泡将把消息发

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

送给当前 situation 以及其父层次的各个 situation。

通过 regMessageListener 方法注册的消息监听器需要实现 MessageListener 接口，请参考下面的介绍。注册监听器后会返回一个 MessageListenerRegHandle 接口，通过该接口可以进行注销监听器等操作，也请参考下面的介绍。

4.5.2.2MessageListener

在 Situation 注册消息监听器需要实现 MessageListener 接口，并实现 onMessage 方法：

```
/**
 * 消息监听器
*/
public interface MessageListener<TMessage> {
    public void onMessage(Situation context, TMessage message,
            MessageTransmitter<TMessage> transmitter);
}
```

其中 TMessage 为泛型，需要指定为具体的消息对象类型。

4.5.2.3MessageListenerRegHandle

在 Situation 上注册消息监听器后，会返回 MessageListenerRegHandle 接口，通过该接口可以进行消息监听器的注销等操作，下面仅说明几个重要方法：

```
/**
 * 注销该监听器
 */
public void unRegister();

/**
 * 设置有效性，默认为true
 *
 * @param value
 *            监听器是否生效
 */
public void setEnabled(boolean value);


/**
 * 设置是否监听注册消息类型的子类型，默认为false
 */
public void setListenSubMessage(boolean value);
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
/**
 * 设置是否监听冒泡（向上级发送）消息，默认为true
 */
public void setListenBubble(boolean value);

/**
 * 设置是否监听广播（向下级发送）消息，默认为true
 */
public void setListenBroadcast(boolean value);
```

需要注意，系统默认不监听指定消息对象的子类型（即只监听具体类型，不监听子类），系统默认监听从各个方向发来的消息；可以使用这个接口的相关方法来更改这些行为。

4.5.2.4MessageTransmitter

在 MessageListener 的 onMessage 方法中，除了可以得到消息对象外，还可以得到一个 MessageTransmitter 接口，该接口主要具有以下方法：

```
/**
 * 获得消息的发送者的情景
 */
public Situation getSender();

/**
 * 获得当前消息的传递方向
 */
public MessageDirection getDirection();

/**
 * 获得监听器注册句柄，用以改变监听设置
 */
public MessageListenerRegHandle<TMessage> getRegHandle();

/**
 * 获得当前情景与消息的发送情景间的距离(与发送者之间)
 */
public int getDistance();

/**
 * 获取消息允许传播到的最远距离（与发送者之间）
 */
public int getMaxDistance();
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
/**
 * 设置消息允许传播到的最远距离（与发送者之间）
 */
public void setMaxDistance(int value);

/**
 * 中止当前消息的继续传递（将传播最远距离限制设为<0）
 */
public void terminate();
```

其中主要使用 terminate 方法来终止当前消息的继续传递。

4.5.2.5消息对象

和 Context 中的任务处理器不同，系统并没有为消息对象设置任何的基接口，也就是说任何对象都可以作为消息进行传递，监听者只要使用相应的对象类型来替换接口中的泛型类型即可。

但是需要注意，系统默认只会通知到那些和具体类型一致的监听器，如果需要同时能监听到指定类型的子类型，则需要通过 `MessageListenerRegHandle` 接口来更改这个策略。

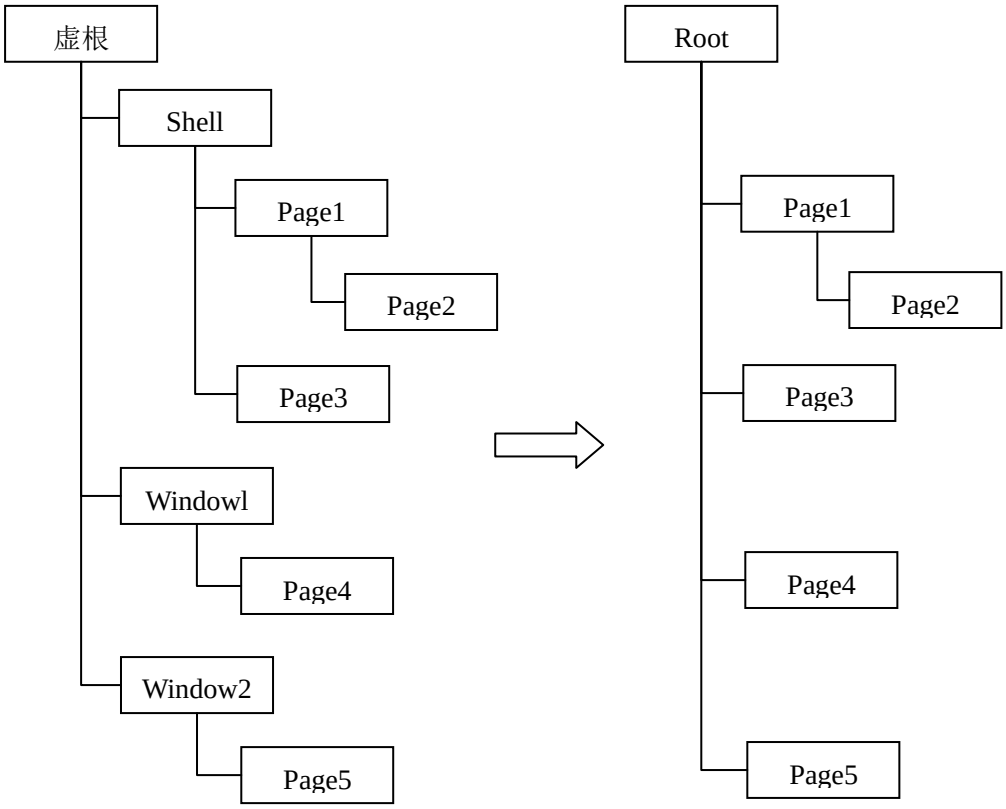
4.5.3 Situation 层次结构

前面介绍 Situation 时已经对其层次结构有了基本了解。这里我们将做深入的说明。

我们知道，对于一个界面会话而言，所有控件从顶层的虚根开始，构成了一棵界面控件结构树，具备了消息传递的载体基础，但是由于界面控件结构树的层次和控件数量太多，很多容器控件用于组合和布局界面，并不具有特定的功能特性，基于原始的控件结构树进行消息的传递既无实际意义，还存在效率问题。

所以在 DNA 界面框架中，消息的传递依赖的树形结构，并不是原始的控件树，而是前面介绍的 Page 树，即将控件树根据 Page 进行重新组织，形成一棵新的 Page 树，由于每个 Page 对应一个 Situation 对象，这棵树通常称为 Situation 树，每个 Situation 节点代表了具有特定范围功能的界面容器。消息在 Page 之间的传递既具有实际的意义，也提高了传递的效率。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

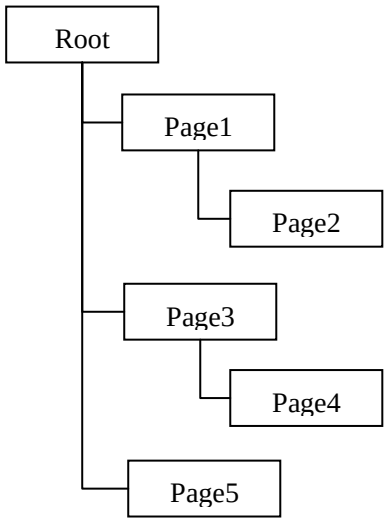


控件树（忽略非 Page 节点）　　Situation 层次树

从上图可以看到，左边是一个控件树的结构，从虚根节点开始，窗体和 Shell 是同级，右边是都应的 Situation 层次树。

除正常根据控件树的 Page 层次结构来确定 Situation 的层次结构外，有一种特殊的机制可以对 Situation 层次进行调整：在创建一个 Window 时，如果指定了宿主控件，则可以改变窗体内部 Page 的父 Situation，即窗体的直接子 Page 的父 Situation 不再是根 Situation，而是宿主控件所属的 Page 的 Situation。

例如上图中，如果指定了 Window1 的宿主控件是 Page3 的一个子控件，则新的 Situation 层次树如下：



公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199　　传真：010-58561196
网址：http://www.jiuqi.com.cn　E-mail：jiuqi@jiuqi.com.cn

4.5.4 消息机制的作用

消息机制最重要的作用就是可以降低界面之间的耦合度。

消息机制将消息的发送者和接收者分离开来，发送者只需要在合适的时机在自身的 Situation 对象上以一定的方向（向上冒泡或向下广播）将消息发送出去，无须关心谁会处理这个消息，也无须知道这个消息被处理多少次；而消息的接收者在其 Situation 对象上注册消息监听器，接收指定类型的消息。当该类型的消息到达时，接收者负责对其处理，它无须关心消息的发送者，也无须关心消息是如何传递给它的。

消息的发送者和接收者只对消息本身（数据）感兴趣，而消息的发送者和接收者可视为不同的功能界面（Page），因此消息机制使得在不同界面之间传递数据变得非常自然和简单。当需要在不同界面之间传递数据时消息机制就可以大显身手了。

4.5.5 消息的流转过程

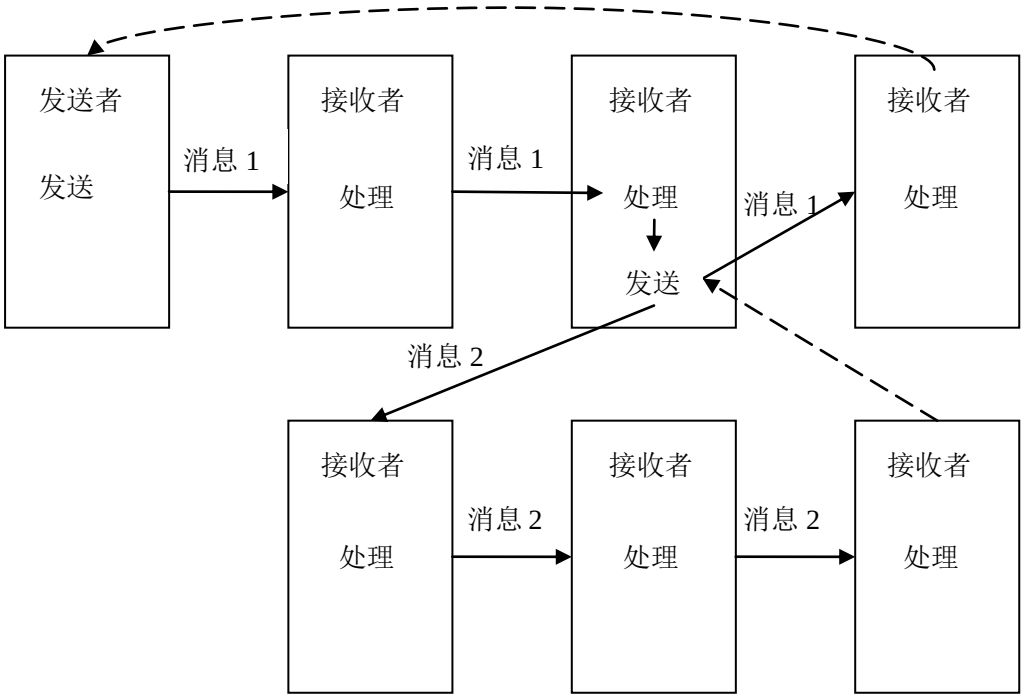
消息由般都通过其自身界面的 situation 对象进行发送，一般有两个时机：

- 界面初始化时
- 用户触发动作事件时

发送者在合适的时机将消息发送出去，消息沿 Situation 层次结构树进行传播，在传播路径上遇到对这个消息感兴趣的接收者时消息被传递给该接收者进行处理，若该接收者没有强制中止消息的传播过程，则其处理完毕后消息继续传播，直到所有对这个消息感兴趣的接收者都处理完这个消息或者消息在某次处理过程中被强行终止，消息从被发送到被处理完成或终止就完成了一次流转过程。

我们可以将消息被发送出去后沿 Situation 层次结构树进行传播的路径视为一条消息链路，对消息感兴趣的每个接收者可视为这条链路上的一个一个的节点，则可将消息的流转过程示意如下：

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn



消息流转过程

需要注意的是，消息的流转过程是同步的。即消息被发送者调用 broadcaseMessage 或 bubbleMessage 将消息发送出去后，直到消息被处理完毕或终止相应的发送消息的方法执行才会结束。界面在加载时可以利用消息机制的这一机制去查询初始化数据。

4.5.6 消息机制的使用原则

消息机制是对事件机制的一种补充和增强，通过良好的设计，可以开发出低耦合、模块化的界面程序。消息机制使得界面编写变得更加灵活，也使得界面组合变得更加灵活。

但是同时消息机制也是把双刃剑，滥用消息机制会使得程序代码的可读性和可维护性变差。所以一定要记住，不要滥用消息机制，消息机制的使用应该遵循 “解耦”的原则：即只有不同功能的界面之间需要进行通信并要避免耦合时，才使用消息。

在一个耦合性紧密的界面内部时，尽量使用直接方法调用，使用消息只会使得整个程序难于理解。

4.5.7 注意事项

- 使用 Situation 时要格外关注其生命周期。如果在程序的其它地方有保存 Situation 对象的引用，那么当该 Situation 从属的 Page 被销毁时就不能再使用这个 Situation 了。
- 要特别注意对消息监听器的注册位置，一般而言，不要轻易在非本界面视图的其他 Situation 上注册监听器（例如父 Situation 或者根 Situation），如果确实有必要这样做，则一定要注意在合适的时机注销监听器（通常是在页面销毁时）。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

- 一定要明确通过控件的 getContext 方法得到的 Situation 是哪个层次的 Situation，因为并不是每个控件层次都有对应 Situation，如果是确定相应的 Situation 是和当前界面视图相关的，则不需要关心相应监听器的注销（Page 销毁时监听器自然就无效了），否则就要和前面所说的那样，自行处理监听器的注销。

4.5.8 有关客户端消息机制

前面所有的说明都是针对服务器的控件模型的，然后和控件的客户端事件一样，DNA 界面框架也提供了类似的客户端消息机制。

4.5.8.1消息发送接口

在客户端脚本中，可以通过 widget 来直接发送消息。

广播发送：

```
broadcastMessage(messageType,messageObject);
```

冒泡发送：

```
bubbleMessage(messageType,messageObject);
```

其中 messageType 为字符串，用来表示消息类型，messageObject 是任意 javascript 对象，用来作为消息数据。

4.5.8.2消息监听器

客户端消息监听器的注册注销机制和客户端事件的注册注销机制类似，都在服务器端 Widget 上提供接口：

```
/**
 * 注册指定类型的消息的处理脚本，当控件接收到该类型的消息时，执行脚本代码
 *
 * @param messageType
 *            消息类型
 * @param name
 *            全局函数名称
 * @return String 脚本的注册id，删除该脚本时用
 */
public String addClientMessageHandler(
                    String messageType, String name);

/**
 * 注销客户端事件脚本
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
     *
     * @param id
     *            注册时产生的脚本id
     */
    public void removeClientEventHandler(String id);
```

消息监听器的注册和注销都需要在服务器端进行，在注册客户端消息监听器时，需要两个参数，一个消息名，用于说明要监听哪个名称消息，消息名由其他界面的客户端脚本在发出消息时指定；另外一个参数是消息处理其的客户端脚本全局函数名，和客户端事件一样的规则。下面是一个消息监听器的例子（和客户端事件监听器一样，消息监听器的实现需要使用 javascript 函数）：

```
Sample.testMsgHandler = function(event, widget) {
    ……
};
```

其中 event 提供 getMessage 方法可以得到消息发送者传递的数据对象。

相应的脚本代码的 js 文件同样需要中被注册到 dna.xml 文件中才能被识别。

4.6 界面布局机制

在快速入门中我们已经对布局机制已经有了初步的介绍，这里我们将详细介绍和布局相关的接口方法、对象，以及各种布局算法。

4.6.1 相关控件接口

使用布局机制主要涉及到三个控件接口：

1、 容器控件（Composite）的布局方式设置接口：即告知容器使用何种算法来布局其子控件

```
/**
 * 设置容器的布局对象
 * @param layout  布局算法对象
 */
public void setLayout(Layout layout);
```

2、 普通控件（Control）的布局数据设置接口：即告知父容器在计算子控件布局时，针对具体子控件应该做何特殊处理

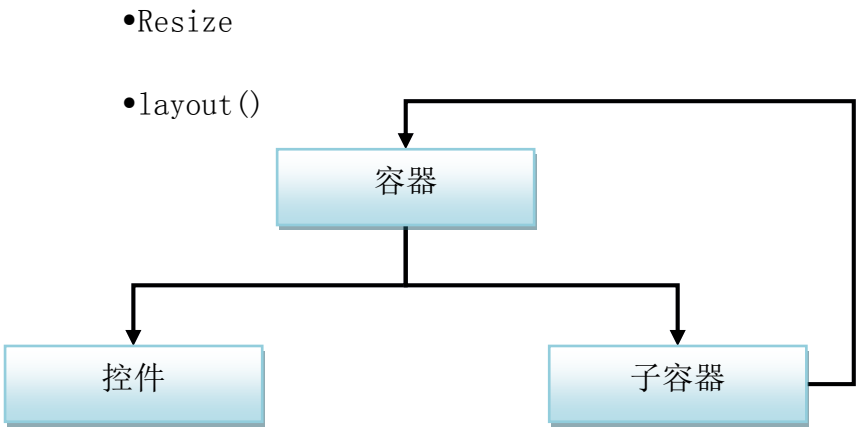
```
/**
 * 设置布局数据
 * @param layoutData 布局数据
 */
public void setLayoutData(Object layoutData);
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

3、 容器控件（Composite）的布局调用接口：对容器调用 layout 方法。

```
/**
 * 通知系统对该容器进行布局运算
 */
public void layout();
```

4.6.2 布局流程和规则



上图描述了布局流程，容器触发布局算法时仅有两种方式：大小变化和显式的调用 layout 方法，触发布局算法时容器设置子控件位置和大小，如果子控件是容器控件，根据布局机制，子容器同样会触发布局，从而实现递归布局。

布局机制遵循以下的规则：

- 容器大小变化时自动布局
- 父容器布局，导致子容器大小变化，子容器自动布局
- 增加或者删除容器的子控件，必须对容器显式调用 layout 方法才能使得容器内部的控件布局发生变化
- 对于 Shell、Window、ComboPanel 等肯定有大小的容器，初始化时不需要显式调用 layout 方法，系统会自动触发布局。
- 一旦使用布局，则针对被布局控件直接设置具体的位置和大小将不起任何作用。
- 对于父容器的布局类型，子控件必须设置匹配的布局数据，否则将导致系统错误。

4.6.3 布局算法简介

目前 DNA 界面框架提供 5 种布局：FillLayout、GridLayout、RowLayout、BorderLayout、TableLayout。除 FillLayout 外，其他每个布局都有对应的布局数据，父容器设置的布局，子控件只能使用对应的布局数据。下面会对每个布局做简单的介绍，详细的布局参数请参考附录。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

4.6.3.1FillLayout

FillLayout（填充布局）是一种简单的布局，为容器设置该布局后，如果仅有一个子控件，则该子控件将填充整个父容器。如果存在多个子控件，则这些子控件将按照行方向或者列方向平分父容器区域。

FillLayout 没有对应的布局数据来对子控件进行特殊的控制。

4.6.3.2RowLayout

RowLayout（行布局或者列布局）也是一种简单布局，为容器设置该布局后，子控件将按照行方向或者列表方向排列。

可以为被布局的子控件设置布局数据对象 RowData 来对其进行特殊的布局控制，例如希望明确子控件的大小（主要用于子容器，其他控件多数时候不需要进行设置，系统会自动计算最佳大小）。

4.6.3.3GridLayout

GridLayout（网格布局）是一种相对复杂的布局，它将父容器分成一个表格， 每个子控件占据一个单元格的空间， 每个子控件按添加到父控件的顺序排列到网格中。

可以为 GridLayout 指定列数，也可以为被布局的子控件设置布局数据对象 GridData 来对其进行特殊的布局控制，例如要求指定该控件的尺寸（例如指定高度），或者要求其填充某个方向（例如行方向填充）等。

4.6.3.4BorderLayout

BorderLayout（锚点布局）是一种较为简单的布局，它将父容器分为东西南北中五个角锚点，将子控件分配到这五个锚点上。 其中南北区域水平填充，高度被压缩为最小高度；东西区域垂直填充，宽度为最小宽度；剩余部分为中间区域，缺省部分被中间区域抢占。

可以为布局的子控件设置布局数据对象 BorderData 来要求将其放置在具体的位置上。

4.6.3.5TableLayout

TableLayout（表格布局）是一个比较复杂的布局，类似 HTML 中的表方式布局，比较之前的 GridLayout 要复杂的很多，可以对表格行列，宽度比例等做很精确和灵活的控制。

可以为布局的子控件设置布局数据对象 TableData 来对其进行更精细的控制。

4.7 控件拖拽机制

这里主要介绍 DNA 界面框架中的拖拽机制，了解拖拽机制的原理和用法，以帮助开发

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

人员设计出丰富的界面效果，进一步提高用户体验。

整个拖拽机制需要关注几个要素：拖拽源、拖拽目标、拖拽数据、拖拽监听和拖拽事件。

4.7.1 拖拽源

拖拽源也就是“拖的是什么”。简单的理解就是可以进行拖放的控件，只有光标进入可以作为拖拽源的控件区域，才可以开始进行拖的操作。在 DNA 界面框架中，只要为其设置了拖拽数据提供器，任何一个控件都可以成为拖拽源。

4.7.2 拖拽目标

拖拽目标也就是“放到哪里”。简单的理解也就是可以接受源的目标控件。只有光标进入到目标控件的区域时，才可以进行放的操作。在 DNA 界面框架中，只要为其设置了拖拽事件监听器，任何一个控件都可以成为拖拽源。

4.7.3 拖拽数据

拖拽数据是拖拽源和拖拽目标之间联系的桥梁，拖拽源必须提供拖拽数据，拖拽目标必须知道自己可以接受怎样的拖拽数据。在 DNA 界面框架中，拖拽数据通过拖拽源的拖拽数据提供器和拖拽目标的拖拽事件监听器的泛型参数来指定。

4.7.4 拖拽监听和拖拽事件

在前面的几节中，我们已经了解了什么是拖拽源，拖拽目标，拖拽数据。当一个可以被接收的拖拽源进入拖拽目标并放下时，会触发拖拽目标的拖拽事件监听。

下面我们来详细探讨一下拖拽监听器和拖拽事件。

拖拽监听器和拖拽事件都定义在 com.jiuqi.dna.ui.wt.events 包下面，其中 DragDropListener 的定义如下：

```
public interface DragDropListener<TDragData>
                                extends WTEventListener {
    /**
     * 拖动产生
     * @param event 拖拽事件
     */
    public  void widgetDropped(DragDropEvent<TDragData> event);
}
```

DragDropListener 定义了拖拽目标可以接受的拖拽数据类型已经拖拽事件发生时候所要执行的方法。除此之外，拖拽事件所有的信息都包含在拖拽事件 DragDropEvent 中。DragDropEvent 的定义如下（省略掉构造函数）：

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
/**
 * 拖拽事件
 */
public class DragDropEvent<TDragData> extends WTEventObject {

    /**源控件*/
    public Control dragSource;

    /**数据*/
    public TDragData dragData;

    /**目标信息*/
    public String dropInfo;

    /**源位置*/
    public Point dragPosition;

    /**目标位置*/
    public Point dropPosition;
}
```

可以看出 DragDropEvent 中提供了很丰富的拖拽信息，包括拖拽源控件，拖拽数据，拖拽源和拖拽目标的位置信息（指鼠标在拖拽源开始拖动和在拖拽目标放下时候的坐标）以及目标信息。这是拖拽目标为拖拽事件提供的额外信息，只有少数几个控件为拖拽事件提供了 dropInfo 属性。

4.7.5 拖拽目标信息

在前面我们看到 DragDropEvent 除了提供拖拽基本信息的属性外，还提供了一个 dropInfo 属性来提供拖拽目标的附加信息。这个属性具体到各个目标控件意义不尽相同，例如 Grid 控件用它来标识拖拽事件发生时候的光标所在的单元格的序号，RowLayoutPanel 则用来表示拖拽事件发生时候的光标所在行的行序号。对于提供附加信息的拖拽目标控件都提供了静态方法 `getDropPosition` 来转化 dropInfo 其需要的信息，但是其返回值则不一定相同，如下表所示：

控件	转化方法	返回值	返回值含义
Grid	`getDropPosition`	Point	目标单元格序号
List	`getDropPosition`	IterableDropPosition	拖拽目标索引
Table	`getDropPosition`	IterableDropPosition	拖拽目标索引
Tree	`getDropPosition`	IterableDropPosition	拖拽目标索引和父节点
XYLayoutPanel	`getDropPosition`	Point	拖拽目标位置
TableLayoutPanel	`getDropPosition`	Point	目标单元格序号
RowLayoutPanel	`getDropPosition`	int	目标行序号

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

4.7.6 拖拽实现

了解关于拖拽的基本机制后，下面简单的介绍如何实现拖拽效果。概要的说，实现拖拽机制需要下面几个步骤：

1、 确定拖拽源控件，为其设置拖拽源提供器，具体方法如下：

```
/**
 * 设置拖拽数据提供器
 * @param dragDataProvider  拖拽数据提供器
 */
public <TDragData> void setDragDataProvider(
        DragDataProvider<TDragData> dragDataProvider);
```

其中需要为泛型 TDragData 确定对象类型。

2、 确定可以接受相应拖拽源的控件，增加拖拽监听器，方法如下：

```
/**
 * 添加拖拽事件监听器
 * @param l  拖拽事件监听器
 */
public <TDragData> void
           addDragDropListener(DragDropListener<TDragData> l);
```

其中 TDragData 泛型需要指定确定类型，并且和拖拽源的泛型类型一致。

3、 实现监听器的 widgetDroped 方法，并注册至拖拽目标

当用户将拖拽源拖动至拖拽目标放下时，将触发拖拽目标控件的 DragDropEvent。通过实现 widgetDisposed 方法对此行为进行处理。

4.8 控件样式和样式单

在 DNA 界面框架中，每个控件都可以通过样式来控制基本的外观，可控制的外观包括前景色、背景色、背景图片、字体、光标、边框。

和 HTML 的样式控制模式类似，可以直接在控件上设置这些样式，也可以通过样式单的方式。

界面引擎的样式体系可以参考下图所示，这里将对主要概念和控件样式的查询流程进行说明，并介绍样式表的编写方式。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199　　传真：010-58561196
网址：http://www.jiuqi.com.cn　E-mail：jiuqi@jiuqi.com.cn

4.8.1 控件基础样式

控件的基础的样式包括背景色、背景图片、前景色、字体、边框、光标等六个，其中背景色和前景色使用 Color 对象，背景图片使用 ImageDescriptor，字体使用 Font 对象，边框使用 CBorder 对象，光标使用 Cursor 对象。

4.8.2 控件状态

控件在不同时刻可以具有不同的状态，控件的状态可以通过程序改变，也可以因为用户的行为而发生变化。与之对应，控件的状态可分为模式状态与行为状态。模型状态只能通过程序设置控件的属性来改变；行为状态即可以通过响应用户的动作而发生变化，也可以通过程序设置引起变化。

模式状态包括：正常、禁用或其他的自定义模式状态，如只读、错误等。控件在某一时刻只能表现出一种模式状态。

当控件处于正常模式状态时可以具有行为状态，行为状态包括光标悬浮和获取焦点。

4.8.3 控件样式属性和 CSSStyles 对象

控件通过 CSSStyles 对象来存储其各种状态下的各种样式值，并且在 Widget 上提供了

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

正常状态下的样式值的设置方法，如果有必要，可以得到 CSSStyles 来设置其他状态的样式值。

下面是 CSSStyles 的方法说明：

```
/**设置不同状态下的字体样式
 * @param stateName 状态名称
 * @param font 字体
 */
public void setFont(String stateName, Font font);


/**设置不同状态下的背景色样式
 * @param stateName 状态名称
 * @param color 颜色
 */
public void setBackground(String stateName, Color color);


/**设置不同状态下的背景图片样式
 * @param stateName 状态名称
 * @param image 图片
 */
public void setBackimage(String stateName, ImageDescriptor image);


/**设置不同状态下的前景色样式
 * @param stateName 状态名称
 * @param color 颜色
 */
public void setForeground(String stateName, Color color);

/**设置不同状态下的光标样式
 * @param stateName 状态名称
 * @param cursor 光标
 */
public void setCursor(String stateName, Cursor cursor);

/**设置不同状态下的图片边框样式
 * @param stateName 状态名称
 * @param border 图片边框
 */
public void setBorder(String stateName, CBorder border);
```

其中状态名称包括 hover（光标悬浮）、active（获取焦点）、disable（禁用）、readonly（只

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

读）等。

4.8.4 样式单

界面引擎提供和 HTML 类似的样式单来控制控件的外观，样式表中包括系统级控件样式定义和自定义样式类。下面是一个样式单的例子：

```
._system_default_border_ {
    border: 1px solid #000000;
}

._system_none_border_ {
    border: 0px solid #FFFFFF;
}

._system_styled_border_ {
    border: 1px solid #888888;
}

._system_styled_nobackground_ {
}

._system_styled_input_ {
    background_readonly : #FFFFFF;
    background_active : #FFFFAA;
}

DateTime {
    extends: _system_control_ _system_styled_border_
_system_styled_input_;
}

Spinner {
    extends: _system_control_ _system_styled_border_
_system_styled_input_;
}

ComboPanel {
    extends: _system_control_ _system_styled_border_;
    background_unselectable : #FFFFFF;
    background_active : #FFFFAA;
}

Text {
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
    extends: _system_control_ _system_styled_border_
_system_styled_input_;
}
……
```

整个格式和 CSS 样式单类似，但是仅限于基础样式的描述。系统控件样式直接使用控件类型名，自定义类名则需要在之前加上“.”作为前缀。

另外，样式类还支持继承，使用 extends 关键字。

4.8.4.1样式单注册

样式单编写完毕后，需要在 dna.xml 中进行注册，例如：

```
<ui-stylesheets>
    <stylesheet name="system"
                title="系统默认样式单" path="system.css" />
</ui-stylesheets>
```

其中 name 必须唯一，是具体应用程序应用样式的标识，编程时需要使用该标识来应用样式单。path 是具体样式文件的路径。

4.8.4.2应用样式单

样式单被注册到系统中后，还需要在应用程序中进行应用才会生效。

Display 对象中包括应用样式单和取消应用样式单的方法：

```
/**
 * 应用样式单
 * @param id 样式单id
 */
public void applyStyleSheet(String id);

/**
 * 取消应用样式单
 * @param id  样式单id
 */
public void unapplyStyleSheet(String id);
```

其中参数 id 为注册样式时的标识。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

4.8.5 了解样式查询流程

编程者可以根据多种方式来设置控件的样式，而最终界面框架会按照下面的流程来确定控件的实际样式：

1、 确定控件当前的状态：包括模式状态和行为状态
2、 确定样式标识
3、 从控件中直接查找样式值，如果查找到则应用，否则下一步
4、 根据控件的样式定义名称查找样式表相应样式值，如果查找到则应用，否则下一步
5、 根据控件的类型名称查找样式表相应样式值，如果查找到则应用，否则下一步
6、 获取控件的父控件，重复 3-5，如果查找到则应用，否则下一步
7、 根据默认值确定样式

4.9 控件主题和主题管理

DNA 界面框架为控件库提供多套不同的控件主题，不同的控件主题在色调甚至控件结构上都会有所不同。实际使用中可以利用以下方法来查询系统中到底有多少控件主题，以及如何切换控件主题等：

```
/**
 * 获取所有主题
 * @return 所有主题
 */
public static String[] getThemes();

/**
 * 获取当前主题
 * @return 当前主题
 */
public static String getCurrentTheme();

/**
 * 获取默认主题
 * @return 默认主题
 */
public static String getDefaultTheme();

/**
 * 改变当前主题
 * @param theme 要设置的主题
 * @return 修改结果 不支持修改主题或者已经是当前主题或者是不存在的主题返回
false，否则返回true
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

```
 */
public static boolean changeTheme(String theme);
```

4.10 客户端脚本的高级应用

4.10.1 脚本编程注意事项

使用 javascript 进行客户端事件的编程确实不是一件惬意的事情，前面已经对客户端事件有所说明，这里我们主要需要关注以下方面：

1、 由于在 dna.xml 中注册的 js 文件最后会被合并成一个全局的脚本，这意味着在编写相应的脚本时，要注意全局变量的使用，并且要注意使用命名空间来进行隔离，避免命名方方面的重复

2、 使用面向对象的 javascript 编程，可以编写出更易于维护的 javascript 代码

4.10.2 服务端数据交换

复杂的客户端脚本编程，可能需要预先准备相应的服务器端数据，界面框架通过 widget 控件的相关方法来传递数据。

服务器端通过 Widget 的 setClientObject 和 getClientObject 来设置和获取客户端对应的数据。数据使用 JSon 格式，这样在客户端就可以直接和 javascript 对象进行转换，方法接口为：

```
/**
 * 设置客户端数据对象
 * @param key
 * @param value
 */
public void setClientObject(String key, JSONObject value);

/**
 * 获取客户端数据对象
 * @param key
 * @return
 */
public JSONObject getClientObject(String key);
```

而在对应的客户端脚本中，利用 widget 的 getServerObject 和 setServerObject 来获取和设置对应的数据，这里直接对应的是脚本对象，接口方法为：

```
setServerObject : function(key,obj);
getServerObject : function(key);
```


公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn

通过Widget的服务端接口和客户端接口，可以非常方便的进行数据交换。

早期的版本中提供的方法是不包括 key 参数，这样很容易造成数据被覆盖的情况，所以提供了新的 带 key 参数的方法，之前的方法就废弃了。

4.10.3 触发服务端事件

有些情况，客户端脚本中是进行相应预处理，最后还是需要提交到服务器端，类似我们在写普通动态网页时调用表单的 submit 方法，在界面框架中在 widget 对象中提供了 notifyAction 方法，用于触发服务器端 Widget 的相关事件。我们可以在服务器端为 Widget 增加 ClientNotifyListener 监听器来监听该事件并进行相关处理，这里不做详述。

4.10.4 客户端定时器

客户端定时器提供了一种在客户端脚本中定时执行任务的方法，类似于 JavaScript 的 setInterval，其方法定义为：

```
/**
 * 定时执行
 * @param {} fun 待执行的方法
 * @param {} delayTime 定时间隔
 * @param {} args 参数
 * @return {} 定时器句柄
 */
setInterval : function(fun, delayTime, args)
```

在客户端的 Display 对象和控件模型上均提供了该方法，调用该方法后将返回一个句柄。必须在适当时机将定时器清除，清除定时器的方法如下：

```
/**
 * 根据定时器句柄清除定时器
 *
 * @param handle
 */
clearInterval : function(handle)
```

清除定时器的方法在客户端的 Display 对象上提供。

公司总部：北京经济技术开发区西环中路 6 号（100176）
技服中心：北京西直门内南小街国英 1 号 13 层（100035）
电话：010-58561199 传真：010-58561196
网址：http://www.jiuqi.com.cn E-mail：jiuqi@jiuqi.com.cn