标题

设计器

# 设计器

设计器类似Flash Professional，或者FlashBuilder的MXML设计界面。

设计器功能由UIDesigner插件提供。

## 用法

显然，可视化设计器可以减轻编程人员的负担，让他们从拼界面这种无脑又令人烦躁的工作中解脱出来。MobiMirrage不兼容Flash Professional和MXML，

设计人员可以在设计完成后立即看到新界面在游戏中的效果。

可以为游戏设计多种UI风格，并在运行时实时更换。

您可以为多种屏幕分辨率优化UI，只需要分别载入与自己分辨率适配的描述文件。

# 组件工厂

组件工厂是UIDesigner的核心功能。因为您最终的目的，就是在您的项目中构建组件。

组件工厂套件不仅非常容易置入您自己的项目中，并且在大多数情况下都可以满足您构建组件的需要。

对于那些您项目的独特需求，也可以相当容易的扩展功能。

## 在您的项目中使用组件工厂

首先，将组件工厂的包 potato.designer.plugin.uidesigner.construct 拷贝到您的项目中。

然后，读入您的组件配置文件：（假设它是名为profile的Object对象）

Factory.setData(profile);

如果有多个配置文件，您应该依次读入他们。组件配置文件与IComponentProfile实例并不相同；它通常是一个Object，内部包含数个IComponentProfile实例，以及构建器需要的其他数据。

最后，简单的构建您的组件：（假设您要构建名为” MyComponent”的组件）

var myComponent:\* = Factory.compile(“MyComponent”).component;

## 技术细节

组件工厂包中具有以下类：

Factory 组件工厂。您只需要访问这个类，就可以实现构建组件的功能。

ComponentTree 组件树。这是组件工厂构建过程的返回值。

IComponentProfile 组件描述接口。

IConstructor 构建器接口

BasicConstructor 基础构建器。他是IConstructor的一个示例实现。它可以调用组件的任何方法，以及设置任何属性，只要这些属性和方法的参数能够表示为字符串。

BasicComponentProfile 基础组件描述。它是IComponentProfile的一个示例实现。它是BasicConstructor需要的数据格式。此外，它还是一个动态类。如果您实现了自己的构建器，并希望配合BasicConstructor使用，您可以为此类动态添加您需要的属性。

BasicComponentMemberProfile

基础组件描述的成员变量。

BasicClassTypeProfile 基础类描述。它是BasicConstructor需要的数据格式。

## 扩展组件工厂

组件工厂虽然能满足大部分的需求，但您的项目也难免有自己的专属功能。您可以通过两种方式扩展组件工厂。

扩展类型

如果您只是需要为您的组件定义一种新的参数类型，那么您只需要构建并注册您自己的类型翻译器。

类型翻译器是一个function，它接收一个String作为参数，并返回您的目标值。

比如：

function getMyDate(str:String):Date

{

return new Date(str);

}

然后，将其注册到基础构建器，以便基础构建器可以翻译这个类型。在Guest端以及您自己的项目中执行以下代码：

BasicConstructor.regType(“MyDate”, getMyDate);

这表示您注册了一个名为MyDate的类型。

您还需要在Host端注册您自己的类型，以便设计工具可以识别出您的类型：

TypeTransform. regType(“Date”, “MyDate”);

这表示您注册了一个名为MyDate的类型，对应着Date类。任何需要Date类的参数，可以以MyDate指向的类型翻译器翻译。

实现构建器

当您需要基础构建器无法提供的功能时，您可以实现自己的构建器。实现自己的构建器可以让您完全掌控构建过程；并且您自己的构建器还可以和已有的构建器协同工作。

设想一下，您有一个类MyClass。这个类的实例无法直接new出，而是需要通过静态的creat方法生成。（类似core.display. TextureData）则您可以创建一个构建器MyConstructor：

public class MyConstructor implements IConstructor

{

public static const instance:MyConstructor = new MyConstructor;

public function construct(profile:IComponentProfile, tree:ComponentTree):Boolean

{

var basicProfile:BasicComponentProfile = profile as BasicComponentProfile;

if(basicProfile && “MyClass” == basicProfile.className)

{

tree.component = MyClass.creat();

}

return false;

}

......

}

然后，将构建器注册到构建序列中。由于这个自定义构建器用于创建组件实例，因此它应该先于默认构建器运行。

Factory.constructorList.splice(

Factory.constructorList.indexOf(

DefaultConstructor.instance),

0, MyConstructor.instance);

这是一个简单的示例。您的自定义构建器可能相当复杂，比如拥有自己的数据格式。您还可能需要扩展编辑器。以下几个要点可能有助：

构建序列中靠前的构建器会先运行。并且针对每个组件，可以跳过后面的构建器。

如果您希望与基础构建器协同运行，请依然使用或者扩展基础组件描述。基础构建器需要它才能运行。

构建器可以什么都不做：只需要返回false。