



Inspire Coding Conventions for ActionScript 3 v1.1

艺思哲 AS3 代码规范 v1.1

Prepared by: Jack Li

Oct 16, 2008

ALL RIGHTS RESERVED, INSPIRE © 2008
版权所有: 艺思哲软件(上海)有限公司 © 2008

修改历史 Change History

Date	Author	Changes
Oct 16, 2008	Jack Li	Minor changes. Released as V1.1.
Sep 17, 2008	Jack Li	Add 6.4 Updating Flex SDK regularly
August 14, 2008	Jack Li	Add 6.1.3 unregister listeners
August 2, 2008	Jack Li	V1.0 Initial writing.

[visit riashanghai.com](http://visit.riashanghai.com)

目录 Table of Contents

1	简介 Introduction	1
1.1	规范 Convention.....	1
1.2	重要建议 Recommendation importance	1
1.3	与传统规范的差别 Major differences with traditional code conventions.....	2
1.4	与 Adobe 代码规范的主要差别 Major difference with Adobe conventions.	2
2	命名规则 Naming Conventions.....	4
2.1	命名规则概要 General naming guidelines.....	4
2.2	包命名 Package names.....	5
2.3	类命名 Class names.....	5
2.4	接口命名 Interface names.....	7
2.5	变量/属性命名 Variable/property names.....	7
2.6	函数命名 Function names.....	11
2.7	事件命名 Event names	12
2.8	命名空间的命名 Namespace names	13
3	文件格式 File Format.....	14
3.1	文件命名 File name.....	14
3.2	文件内容 File content.....	14
4	语句 Statements.....	17
4.1	包和导入 Packages and imports.....	17

4.2	声明 Declarations.....	17
4.3	控制流程 Control Flows.....	18
4.4	Literals.....	20
4.5	留白 White spaces.....	21
5	文档与注释 Documentation & Commenting.....	23
5.1	通用规则 General rules.....	23
5.2	ASDoc.....	24
6	ActionScript 最佳实践 ActionScript Best Practices	29
6.1	通用规划 General Programming.....	29
6.2	日志 Logging.....	30
6.3	使用FlexUnit 进行测试Testing with FlexUnit.....	31
6.4	有规律的更新 Flex SDK Updating Flex SDK Regularly.....	33
	附录Appendices	34
	ASDoc 标签参考 ASDoc tag reference	34

1 简介Introduction

本文档提供了艺思哲公司内部使用的 ActionScript 代码编写规范,这些建议都是基于现有标准和最佳实践经验.

This document provides ActionScript coding recommendations for use within Insprise. The recommendations are based on established standards and best practices.

1.1 规范Convention

1.1.1 规范概述This is the recommendation summary

下面列出了一种简易使用本规范的方法:

Below table lists a sample using the recommended way:

Sample:

```
public class Sql Statement {  
    ...  
}
```

在某些情况下,一个供比较的表格列出了不规范书写方法(Bad)和建议的书写方法. In some cases, a comparison table lists bad practice (Bad) and recommended practice (Good):

<i>Bad</i>	<i>Good</i>
<code>if(t > 2006) ...</code>	<code>if(currentYear > 2006) ...</code>

1.2 重要建议Recommendation importance

- **必须(Must):** 必须遵守 A must requirement must be followed;
- **应该(Should):** 强烈要求 is a strong recommendation;
- **可以(Can):** 一般性的建议 is a general guideline

违反: 代码规范的目的是增加代码的可读性,便于程序的维护.所有有利于代码可读性的违反都是被允许的.

Violation: The goal of the code convention is to increase code readability in order to

make code easy to maintain. Any violation is allowed if it enhances code readability.

1.3 与传统规范的差别 Major differences with traditional code conventions

1.3.1 建议使用Tabs缩进. Tabs are recommended to use for indentation

几乎所有的现代 IDEs 对 TABs 都有很好的支持. Almost all modern IDEs provide excellent support for TABs.

1.3.2 每行最大的长度扩大到 128 栏 The max column of a line is extended to 128

Monitors become larger and larger, so there is no reason to keep the old 80 columns limit.

1.4 与Adobe 代码规范的主要差别 Major difference with Adobe conventions

1.4.1 左大括号须放置于组合语句开始的末尾 The opening brace must be at the end of line that begins the compound statement

在艺思哲,我们在 Java/C#/C/PHP 中均使用这种风格,同样,也使用于 ActionScript. At Inspire, we are using this style in Java/C#/C/PHP. Thus ActionScript is no exception.

<i>Bad</i>	<i>Good</i>
<pre>i f (l oggedI n()) { text.l abel = "Wel come. "; }</pre>	<pre>i f (l oggedI n()) { text.l abel = "Wel come. "; } el se { text.l abel = "Access deni ed! "; }</pre>

```
el se  
{  
    text.Label = "Access denied!";  
}
```

2 命名规则 Naming Conventions

2.1 命名规则概要 General naming guidelines

2.1.1 使用含义丰富的名字 Use meaningful names

<i>Bad</i>	<i>Good</i>
<code>if(t > 2009) ...</code>	<code>if(currentYear > 2009) ...</code>

避免使用过于模糊的名字,如单个字母.

Avoid using general names like a single character.

2.1.2 在缩写中,只将首字母大写 Capitalize only the first letter in an acronym

<i>Bad</i>	<i>Good</i>
<code>function getHTTPHost():String</code>	<code>function getHttpHost():String</code>

Exceptions:

- 如果一个类成员是由缩写开始,则该缩写全部使用小写字母.
When a class member¹ starting with an acronym, use all lowercases: **var httpHost:int;**
- 本约定并不适用到常量.
It does not apply to constant names: static const **DEFAULT_HTTP_HOST = 80;**

¹ Class member: in this document, class members refer to variables/properties and functions, excluding sub-classes.

2.1.3 所有的名字都应该使用英文 All names should be written in English

我们应该使用英语来命名,以方便在国际环境下交流,毕竟英语是当之无愧的国际性的语言.

English, the de facto international language, should be used in order to communicate properly in international context.

2.2 包命名 Package names

2.2.1 包的名字应全部为小写 Package names should be in lower case

Sample: `package com.insprise.app { ... }`

2.3 类命名 Class names

2.3.1 类应该以名词单数形式,首字母大写,大小写混排,方式命名 Class names should be nouns in singular form, written in mixed cases starting with upper case

Sample:

```
public class Sql Statement {  
    ...  
}
```

2.3.2 表示一组事物的类应使用复数形式 Classes representing collections should have names in plural form

Sample:

```
public class Sql Statements {  
    var sql Statements: Array;  
    ...  
}
```

2.3.3 异常类的名字须以Error开头 Names of exception classes should be prefixed with Error

对一个已知类型的变量来说, 其名称以类型开头要比以类型结尾更容易识别。
It's much easier to find a particular class extending a known type if its name starts with the type name rather than its name ends with the type name.

Sample:

```
public class ErrorSql extends Error {  
    ...  
}
```

2.3.4 接口的默认实现类可以以Default开头 Default implementations of interfaces can be prefixed with *Default*

Sample:

```
public class DefaultSql Parser implements ISql Parser {  
    ...  
}
```

2.4 接口命名Interface names

2.4.1 接口的名字应为以字母”I”开头的名词或形容词

Interface names should be nouns or adjectives prefixed with letter ‘I’

Samples:

```
public interface I Sql Engine;
public interface I Sortable;
```

2.5 变量/属性命名Variable/property names

2.5.1 私有类的变量名称应为以下划线为前缀,小写字母开头的大小写混排.Private class variable names

should be in mixed cases starting with lower case prefixed with ‘_’

Samples:

```
private var _currentYear: int;
private var _eventTitle: String;
```

注意:函数中局部变量的名字不必在前面加_;

Note: names of local variables in functions do not need the prefix ‘_’.

2.5.2 属性名应以小写字母开头,大小写字母混排. Property

names should be in mixed cases starting with lower case

Samples:

```
public var username: String; // property: public var
public function get password(): String; // property: getter/setter
public function set password(password_: String): void;
```

2.5.3 参数名可以增加一个后缀_,以便与变量或属性区分 Parameter name can be suffixed with '_' to differentiate it with variable/property name

Samples:

```
public function set password(password_:String):void {  
    this._password = password_; // _password is class var.  
}
```

或者,也可以一个通用的名字,如'value'或'val'.

Alternatively, use a generic name 'value' or 'val'.

2.5.4 常量的名字必须全部为大写字母 Constants must have names with all letters upper-cased

Sample:

```
Math.PI; var UPPER_LIMIT:int;
```

同类别的一组常量,名字前应加一个相同的前缀

A collection of constants in the same category should be prefixed by a common name:

Sample:

```
public static const SEARCH_GOOGLE:int = 1;  
public static const SEARCH_YAHOO:int = 2;
```

2.5.5 命名一组对象时,应使用复数形式 Use the plural form for names representing a collection of objects

Sample:

```
private var _users:Array;  
public var books:ArrayCollection;
```

2.5.6 布尔型变量不应使用否定性名字 Negated boolean variable names should be avoided

<i>Bad</i>	<i>Good</i>
<pre>var isNotFound: Boolean; var isNotEnough: Boolean;</pre>	<pre>var isFound: Boolean; var isEnough: Boolean;</pre>

2.5.7 在嵌套循环中,使用有意义的名字来命名循环控制变量 Use meaningful names for counters in nested loops

嵌套循环中常出现 Bug,通过使用有意义的名字来命名循环控制变量,不仅可以减少 Bug,还可以增加代码的可读性.

Bugs occur frequently in nest loops. Using meaningful names for counters in nested loops reduces bugs and enhances readability.

<i>Bad</i>	<i>Good</i>
<pre>for(var i:int = 0; i < getRows(); i++) { for(var j:int = 0; j < getCols(); j++){ ... } }</pre>	<pre>for(var row:int = 0; row < getRows(); row++) { for(var col:int = 0; col < getCols(); col++){ ... } }</pre>

2.5.8 使用合适的前缀来命名类型为UI组件的变量 Prefix names of variables referencing UI components with proper abbreviations of the UI type

对一个已知类型的变量来说, 其名称以类型开头要比以类型结尾更容易识别.

It's much easier to find a particular variable of a known type if its name starts with the type name rather than its name ends with the type name.

<i>UI Component</i>	<i>Abbreviation</i>	<i>Example</i>
Button	button	buttonOk
ButtonBar	buttonBar	buttonBarControls
CheckBox	check	checkI sMa l e
ComboxBox	combo	comboCountri es
DataGri d	dataGri d	dataGri dEmpl oyees
DateFi el d	dateFi el d	dateFi el dStart
Label	l abel	l abel Ti t l e
Li nkBar	l i nkBar	l i nkBarFavori tes
Li st	l i st	l i stPersons
TextArea	text	textContent
TextI nput	text	textTi t l e
Tree	tree	treeFol ders
VBox	vbox	vboxCenter
Form	form	formPersonal I nfo
FormI tem	fi	fi Surname
Gri d	gri d	gri dLeft
Gri dI tem	gi	gi Top
Panel	panel	panel Mai n
State	state	stateShowSi gnI n
TabNavi gator	tabNav	tabNavPages

2.6 函数命名 Function names

2.6.1 函数名称应以小写字母开头,大小写字母混合Function names should be in mixed cases starting with lower case

Samples:

```
functi on getCurrentYear(): i nt
```

2.6.2 用动词命名函数 Use verbs to name functions

Samples:

```
functi on startDatabase(): voi d;  
functi on getDatabaseStatus(): voi d;
```

2.6.3 函数名字可以忽略类或对象名称,以避免重复Function names can omit class/object name to avoid duplication

函数中已经暗含类或对象的名称,因此在命名时没有必要包含他们.

The name of class/object is implicit thus there is no need to include it when naming class members.

<i>Bad</i>	<i>Good</i>
<pre>publ i c class Font { functi on getFontFami l y: Stri ng(); }</pre>	<pre>publ i c class Font { functi on getFami l y: Stri ng(); }</pre>

2.6.4 单例类应该通过一个名为getInstance()的静态函数返回他们唯一的值Singleton classes should return their

sole instance through a static function named *getInstance*

Samples:

```
public class Toolkit {
    private static const toolkit: Toolkit = new Toolkit();
    public static function getInstance(): Toolkit { return toolkit; }
}
```

2.6.5 事件处理函数可以命名为onEventType Event handler function can be named as *onEventType*

Samples:

```
private function onClick(event: MouseEvent): void
private function onInitialize(event: Event): void
```

2.7 事件命名 Event names

对一个已知类型的变量来说, 其名称以类型开头要比以类型结尾更容易识别. It's much easier to find a particular class extending a known type if its name starts with the type name rather than its name ends with the type name.

2.7.1 时间的名称要以"Event"开始 The name of event class should starting with "Event"

Samples:

```
public class EventRelationshipResolved extends Event ...
```

2.7.2 事件的类型名字要以"event"开始 The type name of an event should starting with "event"

Samples:

```
public class EventRelationshipResolver extends Event ...
```

2.8 命名空间的命名 Namespace names

2.8.1 命名空间的名字要以小写字母开头,并以'_'作为分隔符
Namespace name should start with lower letter and
separate word using '_'

Samples:

```
namespace Inspri se_basi s_i nternal ;
```

3 文件格式 File Format

3.1 文件命名 File name

3.1.1 ActionScript源文件必须以.as为扩展名,MXML文件以.xml为扩展名 ActionScript source files must have the extension .as and MXML source files must have the extension .xml.

ActionScript 源文件的后缀必须为.as,MXML 文件的名字必须以.xml 为后缀.
The names of ActionScript source files must have ".as" as suffixes, and the names of the MXML files must have ".xml" as suffixes.

3.2 文件内容 File content

3.2.1 所有的文本文件的字符编码必须为UTF-8

The encoding for all text files must be UTF-8.

所有的文件应该国际化.UTF-8 可以避免在编码转换时出错.
All files should be internationalization ready. UTF-8 saves troubles of transforming text files from various encodings.



To use UTF-8 encoding in Flex Builder projects: Properties -> Resource -> Text file encoding -> Other -> UTF-8 -> [Apply]

3.2.2 一个AS源文件不能超过 2000 行

An AS source file should have at most 2000 lines

代码过长往往很难读懂,而且显示了拙劣的设计,因此 AS 文件的总行数尽量控制在 2000 行以内。

The total number of lines in an AS source file should be 2000 or less. An AS source file with too many lines is hard to read, and often it indicates poor design.

3.2.3 每行最多 128 栏/字符 Max 128 columns/chars in a single line

128 行是代码编写和打印的平衡点.

128 is a balance catering both needs for code writing and code printing.

3.2.4 缩进应使用 TABs 而不是空格 Indentations should be written as TABs instead of spaces

缩进可以使得代码清晰易读,正如多数的 IDE 工具,对使用 TAB 缩进提供了很好的支持,即便不全是如此,我们还是要求使用 TAB 缩进.

Indentation lays out the code flow clearly. As most of IDEs, if not all, provide well support for TABs, it is recommended to use TABs.

3.2.5 左大括号要放置在行末 The opening brace must be at the end of line that begins the compound statement

左大括号必须防止在组合语句开始行的末尾,而不是另起一行.

You must put the opening brace in the same line as the compound statement instead of putting it in a new line by itself.

<i>Bad</i>	<i>Good</i>
<pre>i f (l oggedI n()) { text.l abel = "Wel come. "; } el se { text.l abel = "Access deni ed! "; }</pre>	<pre>i f (l oggedI n()) { text.l abel = "Wel come. "; } el se { text.l abel = "Access deni ed! "; }</pre>

3.2.6 使用缩进以表明嵌套层次. Use indentations to indicated nest levels

Samples:

```
public class Toolkit {  
  
    public static function getTotal (nums:Array):int {  
        var total:int = 0;  
        for(var i:int = 0; i < nums.length; i++) {  
            total += nums[i];  
        }  
        return total;  
    }  
}
```

4 语句 Statements

4.1 包和导入 Packages and imports

4.1.1 只导入需要的实体,避免使用通配符 AS types should be in proper packages instead of the default package

类和接口应该放在合适的包中.

Classes and interfaces should be put in proper packages.

4.1.2 只导入需要的实体,避免使用通配符 Import only the necessary entities, avoid using wildcards

<i>Bad</i>	<i>Good</i>
<code>import mx. Logging. *;</code>	<code>import mx. Logging. Log;</code>
<code>import mx. containers. *;</code>	<code>import mx. containers. VBox;</code>

4.2 声明 Declarations

4.2.1 类/接口声明 Class/interface declarations

A class/interface declaration should be organized in the following orders:

1. 文档/注释 documentation/comments;
2. 类/接口语句 class/interface statement;
3. 静态变量顺序:static variables in the order: public, protected, (default), private;
4. 实例变量顺序:instance variables in the order: public, protected, (default), private;
5. 构造函数 constructors;
6. 函数 functions;

例外:实体的逻辑块在以上规则之上

Exception: The **logical group** of entities precedes the above rule.

4.2.2 变量的声明要在代码块的开头,而不是在用到它们的地方 Declarations of variables should be made at the beginning of the block, not where they are used

Sample:

```
public function method():void {  
    var value:int = 0;  
    ...  
    for(...) {  
        value += num;  
    }  
}
```

例外:for 循环

Exception: for loops: for(var i:int = 0; i < size; i++)

4.3 控制流程 Control Flows

4.3.1 For,while,if语句格式如下 for, while, if statements should be in this format:

<pre>for (init; condition; update) { ... }</pre>	<pre>if (condition) { ... } else if (condition) { ... } else { ... }</pre>
<pre>while (condition) { ... }</pre>	

4.3.2 循环/条件语句必须以braces结束 The body of a loop/condition statement must be enclosed with braces

<i>Bad</i>	<i>Good</i>
<pre>for(i nt i=0; i <size; i++) val ++;</pre>	<pre>for(i nt i = 0; i < size; i++) { val ++; }</pre>
<pre>i f(i > 0) val ++;</pre>	<pre>i f(i > 0) { val ++; }</pre>

优点: 增强可读性, 避免在后续升级时产生 bug

Benefits: enhances readability; avoids introducing bugs during maintenance update.

4.3.3 使用临时变量以避免复合条件语句 Use temporary variables to avoid complex condition statements

<i>Bad</i>	<i>Good</i>
<pre>i f(screenX > 800 && screenY > 640 && dpi > 72 && i mage.i sReady()) { di spl ay(); }</pre>	<pre>var screenOK: Boolean = screenX > 800 && screenY > 640 && dpi > 72; i f(screenOK && i mage.i sReady()) { di spl ay(); }</pre>

4.3.4 Switches 语句应该套用以下格式, 并且每个分支必须注释清楚 Switches should be properly formatted as below and any fall through must be commented clearly

```
swtich (condi ti on) {
case A:
    ...
    break;
```

```
case B:
    ...
    /* falls through */

default:
    ...
    Break;
}
```

4.3.5 以层次显示嵌套语句的结束括号 Label closing braces in nested controls with many levels

Sample:

```
for(var i:int = 0; i < val; i++) {
    for(var col:int = 0; col < COL; col++) {
        for(var row:int = 0; row < ROW; row++) {
            ...
        } // end for each col
    } // end for each row
} // end for each int in [0, val)
```

4.4 Literals

4.4.1 Magic numbers should not be used

<i>Bad</i>	<i>Good</i>
<pre>if(value > 99) { log("Value is too big!"); }</pre>	<pre>var static const LIMIT:int = 99; if(value > LIMIT) { log("Value is too big!"); }</pre>

4.4.2 浮点数应该含有小数点 Floating points literals should be written with decimal points

<i>Bad</i>	<i>Good</i>
<code>var val : Number = 72;</code>	<code>var val : Number = 72.0;</code>

4.5 留白 White spaces

恰当的使用空格可以有效提高代码的可读性. Proper use of white spaces enhances code readability significantly.

4.5.1 使用空格的通用规则 General rules for using space characters

- 操作符,冒号的前后都应有一个空格.
Operators, colons should be surrounded by single space chars;
- 逗号,分号和 AS 的保留字之后须有一个空格.
Commas, semicolons and AS reserved words should be followed with a space char;

<i>Bad</i>	<i>Good</i>
<code>val = (a+b-c);</code>	<code>val = (a + b - c);</code>
<code>case 99:</code>	<code>case 99 :</code>
<code>method(a, b);</code>	<code>method(a, b);</code>
<code>for(var i : int=0; i<9; i++)</code>	<code>for (var i : int = 0; i < 9; i++)</code>
<code>if(i > 9)</code>	<code>if (i > 9)</code>

4.5.2 逻辑单元应该以空行分割 Logical units should be separated by blank lines

Sample:

```
// draw borders
```

```
g.setColor(...);  
g.fill(...);  
  
// draw image  
g.draw(...);
```

5 文档与注释 Documentation & Commenting

5.1 通用规则 General rules

5.1.1 利用自动注释代码来减少注释 Use self-documenting code to minimize comments

在允许的情况下，使用自动代码而不是注释

Whenever possible, use self-documenting code instead of comments.

5.1.2 保持代码和文档同步 Keep code and documentation in synchronization

为了明显的反应变化,代码更改之后,文档也必须随之更新.

When code is updated, documentation must be updated to reflect the changes.

5.1.3 注释所有的实体-包,类型,以及所有的成员 Document all entities – packages, types, and all members

注释所有的实体---不管是什么类型,或是否可见

Document all the entities – regardless of types (packages, classes, interfaces, variables, properties, and functions) and visibility (private, protected, default, public).

5.2 ASDoc

5.2.1 包注释 Document a package

当调用 asdoc 工具时你才能注释一个包.只有两种方法:要么通过-包选项或 xml 配置中的<package>标记

You can only document a package when invoking asdoc tool. There are two ways – either through `-package` option or `<package>` tags in XML configuration.

Sample:

```
asdoc -doc-sources my_dir -output myDoc -package com.insprise.common
"Contains common library classes" -package com.insprise.test
"Contains test classes" ...
```

```
asdoc -load-config+=insprise.xml
<file-config>
  <packages>
    <package>
      <string> com.insprise.common</string>
      <string>Contains common library classes.</string>
    </package>
  </packages>
</file-config>
```

5.2.2 注释一个AS类/接口 Document an AS class/interface

Sample:

```
/*
 * $ID ... $
 * License info
 */

package com.insprise.app
{

import ...;

/**
 * This class represents a manager. A manager can manage one or
 * more employees. ← Summary paragraph
 *
 * <p>other details ... </p>
 */
```

```
* @see com.insprise.app.Employee
*/
public class Manager extends Employee { ... }
```

5.2.3 注释私有变量 Document a private variable

Sample:

```
/** The id of this employee's direct manager. */
private _managerId: String;
```

5.2.4 注释属性 Document a property

一个公共变量就是属性: 说明属性并列出来默认值.

For a public variable as a property: explain the property and list the default value.

Sample:

```
/**
 * The surname of the employee.
 * @default null
 */
public surname: String;
```

一对 **getter/setter** 函数也是属性: 注释一个的同时使用@private 隐藏另外一个.

For a pair of getter/setter functions as a property: document either one and hide the other one using @private.

Sample:

```
/**
 * The age of the employee.
 */
public function get age():int { ... }

/**
 * Sets the age.
 * @throws Error if the given age is invalid.
 * @private
 */
public function set age(age_:int):void { ... }
```

5.2.5 注释函数 Document a function

Sample:

```
/**
 * Calculates salary of the given month for this employee.
 * <p>Salary is calculated in this way ... </p>
 * @example How to calculate January's salary:
 * <listing version="3.0">// obtain salary for this month
 * calculateSalary(new Date().getMonth());
 * </listing>
 * @param month the month, valid range [0, 11]
 * @return the salary for the given month
 * @throws Error If the input parameter month is invalid
 *
 * @see Date#getMonth()
 */
public function calculateSalary(month: int): Number { ... }
```

注释的标记应按照以下顺序 The order of block tags should be:

1. @param
2. @return
3. @throws
4. @see

5.2.6 使用@inheritDoc从被重写的函数继承注释 Inherit comment from the overridden function using @inheritDoc

Sample:

```
/**
 * Calculate salary for this manager.
 * @inheritDoc
 */
override public function calculateSalary(month: int): Number { ... }
```

@inheritDoc 可以从父类或接口中复制被重写函数的主要 ASDoc 注释, @param 和 @return 注释,不会复制其他的标记.如果需要一个没有被重写的函数中复制住址,使用@copy,如:@copy Date#getMonth();

@inheritDoc will copy main ASDoc comment, @param and @return content from the overridden function from super classes or interfaces. All other tags are not copied. If you need to copy comments from a non-overriding function, use **@copy**, e.g., @copy Date#getMonth()

5.2.7 注释事件 Document an event

你需要去注释一个事件的类型.对一个传播事件的类来说,如果有事件类型,则传递这个常量;对一个事件类来说,则传递该类型的字面名称.

You need to document the type of the event. For a class dispatching an event, you pass the constant if any as the event type; for an event class, you pass the literal type name.

Sample:

```
/** MyUI.as (the class dispatching the event) ↴
 * Dispatched when the user clicks the left button
 * @eventType flash.events.MouseEvent.CLICK
 */
[Event(name="click", type="flash.events.MouseEvent")]

/** flash.events.MouseEvent (the event type class) ↴
 * Defines the value of the type property of a click
 * event object. ...
 * @eventType click
 */
public static const CLICK:String = "click"
```

5.2.8 使用@private来排除一个一个类/接口/属性/函数从 ASDoc输出 Exclude a class/interface/property/function from ASDoc output using @private

默认情况下,ASDoc 会收集所有的公共类,接口,函数和属性.但是出于保密或暂不公开等原因,有时候需要将一个公共类避免被 ASDoc 输出.在这种情况下,我们使用 @private 标记.

By default, ASDoc generates documents for all public classes, interfaces, functions and properties. For reasons like confidentiality and brevity, it might be necessary to exclude

a public class/interface/property/function from the ASDoc output. In such cases, you use @private tag.

Sample:

```
/**
 * @private
 */
public class FrameworkInternal ...

/**
 * @private
 * Performs internal sanity check, automatically called by system.
 */
public function sanityCheckInternal():void ...
```

5.2.9 使用行尾注释来解释局部变量 Document local variables with end-line comments

Sample:

```
var start:int; // the start index of the sub-string
var end:int;   // the end index of the sub-string
```

5.2.10 使用单行注释来详细解释 Using one-line comment to explain in detail

Sample:

```
var bankAccount:BankAccount = getBankAccount();
if(isValidDate(bankAccount)) {

    // First, the bank information is retrieved, then money
    // is transferred using XXX standard.
    // XXX standard can be found at http://...
    ...
}
```

6 ActionScript 最佳实践 ActionScript Best Practices

6.1 通用规划 General Programming

6.1.1 务必开启严格模式 Strict mode must be enabled

在 FlexBuilder 或编译器命令行中指定 '-strict' 模式.

Specify '-strict' option in FlexBuilder or to the compiler command line.

6.1.2 慎用 ArrayCollection 的排序/筛选 Use sort/filters of ArrayCollection with extra caution

ArrayCollection 的排序/筛选常导致错误,强烈的建议你不要使用.如果必须使用,务必注释清楚.

ArrayCollection's sort/filters could create a lot of trouble. We strongly recommend you do not use them at all. If you must use them, do document clearly.

6.1.3 在不需要继续使用时, 务必解除监听函数对事件的监听 You must make sure a registered listener always gets unregistered when it is no longer in use

当注册一个监听函数时, 必须保证这个监听函数在不需继续使用时会被解除.

When you register an event listener, you must make sure that listener will always get unregistered when it is no longer in use.

6.2 日志 Logging

6.2.1 使用mx.logging来代替底层的logging实现. Use mx.logging instead of low level logging implementations

mx.logging 包提供了简单而又高层次的接口.

The mx.logging package provides simple high level API.

6.2.2 为每个类获得一个ILogger实例 Obtain a ILogger instance for each class

为每个类获得一个 ILogger 实例, 不要为所有类的日志使用一个单独的 log 实例, 因为这样会降低可配置性和可控制性.

Obtain a ILogger instance for each class. Do not use a single log instance for logging in all classes, which reduces the configurability and managability.

Sample:

```
import mx.logging.ILogger;
import mx.logging.Log;

public class FtpClient {

    private static var log:ILogger =
        Log.getLogger("com.insprise.app.FtpClient");
    ...
}
```

6.2.3 选择恰当的日志层次 Select proper logging levels

选择恰当的日志层次:Select proper logging levels:fatal-导致程序提早终止的重大错误;error-运行时错误或不可预料的状态;warn-不正确的使用接口, 不符合规则/未预料到的动作等;info-运行时的时间;debug-系统运行过程中的详细信息.

Select proper logging levels: **fatal** – severe errors that cause premature termination; **error** – runtime errors or unexpected conditions; **warn** – poor use of API, undesirable/unexpected actions, etc.; **info** – interesting runtime events; **debug** – detailed information on the flow through the system.

Sample:

```
public function backup():void {
    var host:String = "ftp3.insprise.com";
    Log.info("Connecting to " + host);

    try {
        connect(host);
        Log.info("Connected.");
    } catch (e:IOError) {
        Log.fatal("Failed to connect to " + host, e);
        return;
    }
    ...
}
```

6.2.4 使用条件日志来减少系统开销 Use conditional logging to reduce overheads

日志会增加运行时的开销,但是用条件日志可以减少开销。

Logging does introduces runtime overheads. By using conditional logging, one can reduce the overheads.

Sample:

```
private function executeFTPCommand(command:String):void {
    if(Log.isDebugEnabled ()) {
        Log.debug("Executing " + command);
    }
    ...
}
```

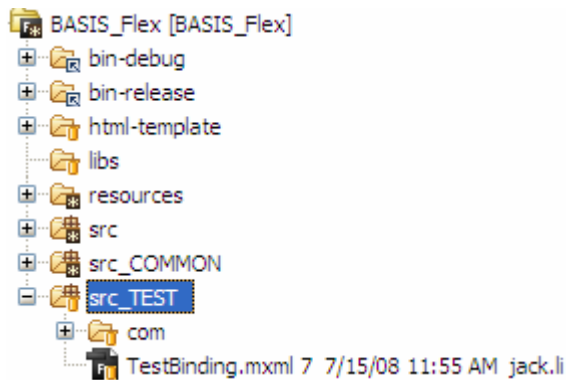
6.3 使用FlexUnit 进行测试 Testing with FlexUnit

除 FlexUnit 之外,还有很多可用的测试框架,但在艺思哲,我们只用 FlexUnit .

Besides FlexUnit, there are many other testing 'frameworks' available. However, at Insprise, we select FlexUnit.

6.3.1 将测试代码放入单独的目录中 Use a separate source tree for testing code

Place testing code in a separate source tree from the production code. For example, if the source folder for the production code is named as 'src', then the source folder for the testing code can be named as 'src_test' (as shown in the picture below).



6.3.2 测试类的包可以命名为<产品类的包>.test The testing class package can be named <production class package>.test

例如,如果产品类的包名为 `com.insprise.basis.unidb`, 那么测试类的包可以叫做 `com.insprise.basis.unidb.test`.

For example, if the production class package is `com.insprise.basis.unidb`, then the testing class package can be named as `com.insprise.basis.unidb.test`.

6.3.3 测试类可以命名为Test<产品类名> The testing class name can be named as Test<production class name>

举例来说,如果作品类名为 `FtpClient`,那么测试类可以叫做 `TestFtpClient`.

For instance, if the production class name is `FtpClient`, then the testing class can be named `TestFtpClient`.

6.3.4 测试方法应命名为test<首字母大写的作品方法名字>_<情节>

The testing method should be named as test<production method name with first letter capitalized>_<scenario>

如果只有一个情景,那么_<情景>部分可以忽略.

If there is only one scenario, then _<scenario> part will be omitted from the name.

Sample:

```
public function testConnect_withSsl():void { ... }
```

6.4 有规律的更新 Flex SDK Updating Flex SDK Regularly

为了修复漏洞, Adobe会经常的更新Flex SDK. 我们建议你更新到最新版本. 对于 Flex 3 来说, 访问[此处](#)以获得最新版本的SDK.

下载并解压缩到 C:\Program Files\Adobe\Flex Builder 3\sdk, 在FlexBuilder中选择 Windows -> Preferences -> Flex -> Installed Flex SDKs, 加入该SDK.

Adobe updates Flex SDK frequently for bug fixes. It is recommended that you should update your Flex SDK to its latest milestone release. For Flex 3, visit <http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+3> for latest builds.

You download and unzip a build to C:\Program Files\Adobe\Flex Builder 3\sdk and add it in Flex Builder by select the Windows -> Preferences -> Flex -> Installed Flex SDKs.

附录 Appendices

ASDoc 标签参考 ASDoc tag reference

Tag	Explanation/Sample
@copy	@copy method/property/variable (refer to target of @see)
@default	@default value (without quotation)
@eventType	@eventType typeName (without quotation)
@example	@example exampleText <listing version="3.0"> ... </listing>
@inheritDoc	@inheritDoc
@internal	@internal text (ASDoc will not output such text)
@param	@param paramName description
@private	@private (ASDoc will not output elements marked private)
@return	@return description
@see	@see reference. Reference types: @see "a text string" @see http://insprise.com/ @see Array (top level class) @see global#int() (top level function) @see Array#length (property, omit class name if in the same class) @see Array#pop() (function omit class name if in the same class)
@throws	@throws errorClass decription