

## [推荐](#)

- [架构](#)
  - [分布式](#)
  - [Redis](#)
  - [Mysql](#)
  - [IOS](#)
  - [Android](#)
  - [HTML5](#)
  - [JavaScript](#)
  - [Python](#)
  - [PHP](#)
  - [Git](#)
  - [Docker](#)
  - [Solr](#)
  - [Elasticsearch](#)
- 
- [推荐](#)
  - [IT资讯](#)
  - [热门标签](#)

[取消](#)

## 搜索历史

## 热搜词

- [android](#)
- [apache](#)
- [oracle](#)
- [spring](#)
- [eclipse](#)
- [tomcat](#)
- [php](#)
- [javascript](#)
- [ios](#)
- [c](#)
- [数据库](#)
- [xml](#)
- [web](#)
- [windows](#)
- [linux](#)
- [sql](#)

- [c++](#)
- [server](#)
- [html](#)
- [c#](#)
- [java](#)
- [mysql](#)



itBoth

JSP设计模式基础：View Helper模式——学习如何使用View Helper模式使得Model数据适应表现层的需要

2017-06-12 xplee0576[ItBoth.com](http://ItBoth.com)

推荐： [JSP设计模式基础：View Helper模式——学习如何使用View Helper模式使得Model数据适应表现层的需要 \(3\)](#)

[ 创建菜单在某些情况下，动态的产生菜单或集合链接是很有用的。控制器执行相应的

动作，这些动作轮流产生一系列的链接项。  
使用Helper的View产生格式化的链接来存储在Model

## 前言

这篇从《JSP设计模式基础》（Apress, 2004）摘录的文章，描述了View Helper模式，并且向我们展示了如何创建一些能增加到你自己的工具箱的有用的View Helper(2,300 字， November 1, 2004)。

View Helper模式告诉我们，我们能使用Helpers来使得在一个应用中，Model数据能够适应表现层的需要。典型的，表现层一般都包括一些JSP页面。这些页面由一些用来给用户显示内容的HTML和图片组成。然而，当这些页面需要显示一些存储在Model上的动态信息的时候，这里有一个问题出现了。你希望能够避免在页面上为了

显示那些动态数据而使用嵌入的Java代码，你就得使用一些Helper来帮助你实现上述的功能。要抛弃那些在控制器Servlet里将一些Model数据作为一个属性存储在request里面的想法。在一个页面里，你要获取Model数据，有三种选择。你可以以JSP脚本的形式嵌入Java代码；你也可以使用EL；或者你也可以使用一个Helper帮你取得数据。根据将表现层和商业逻辑分离的原则，使用一些Helper来帮助我们使得数据适应表现层的要求比将表现层代码和Java代码混在一起有意义（参见图1）。

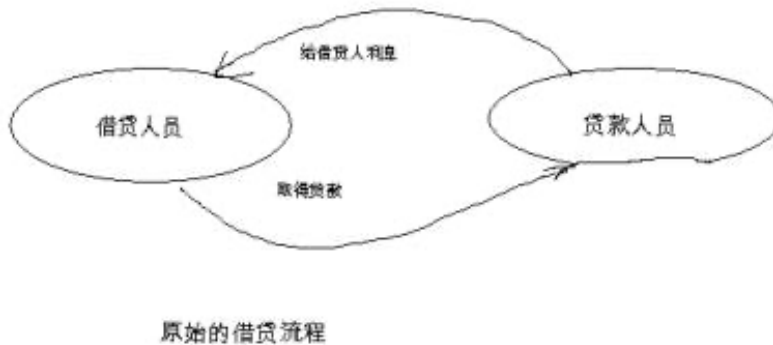


图1

你可以设想，通过使用简单易用的Helper代替Java代码，页面设计人员开发表现层变得简单多了。当然，前提是开发人员发布了一个Helper目录并且详细描述了这些Helper怎么使用；因为这样开发人员才会使用那些Helper。然而，如果在开发人员能够提供使Helper运行起来的Model数据之前，页面设计人员就已经设计了页面，就又有新的问题。解决这个问题一个有用的技巧是在

Helper里设置一些假数据以便在没有Model数据的时候显示；还有一个可以替代的方法是在Model里设置一些能使Helper运行的假数据。无论哪种方法，页面设计人员都不会在等待开发人员的时候闲起来。

使用Helper有如下优点：

- 。表现层的组件是标准化的，为应用提供了统一的look和feel
- 。Java代码从页面设计人员那里被抽离，使得他们有了易用的Helper来访问Model
- 。如果Model还不存在的话，你可以创建一些Helper来显示一些假数据。这样，不管应用程序的准备如何，页面设计人员都可以进行他们的设计工作。
- 。Helper作为业务数据和表现层的中介，它可以把两者清晰的分离开来。

## 实现JavaBeans Helper的策略

在为JSP页面开发Helper的时候，你有两种选择。你可以使用JavaBeans或者定制标签，具体选择哪一种取决于你在Helper中所要处理的数据。一般来说，JavaBeans适合你处理单个的数据，而定制标签却更适合使用在那些处理一系列数据的场合。然而，需要着重指出的是，你可以用任何一种方法来处理两种类型的数据。

## 实现View Helper模式的策略

你可以在一个JSP页面里用JavaBeans来实现Helper。当处理和格式化单一的文本数据时，JavaBeans模式的Helper非常简单易用。那些内置的JSP标签会让你非常简单和直观的使用JavaBeans。对JavaBeans的使用包括简单的声明，后面就可以引用该给定的标签了，就像下面那样：

```
<%-- Declare bean --%>
<jsp:useBean id="myBean"
class="jspBook.util.myBean"/>
<%-- Get first name from bean --%>
```



Hello <jsp:getProperty name="myBean"  
property="firstName"/>,  
welcome to Acme Products' online store!

JavaBeans能做的事可不仅仅是简单的将数据项从Model里取出来，它还能格式化制定的数据项、进行计算或产生大块的数据项。如你所想，它非常适合使用内嵌的JSP标签来获取数据项。但是如果你使用JavaBeans太多，那么你的JSP页面将因为太多的Java代码而变得混乱不堪，不管你使用多少EL。在这种情况下，你应该把所有的附加行为封装到一个制定标签里面去。

## 实现定制标签Helper的策略

为了对付复杂Model的转换，定制标签能够嵌入Java代码，操作好几个有关数据的算子，只提供简单的标签给页面设计者使用。为了使用定制标签，你必须写一个继承了TagSupport或BodyTagSupport的类。你可以在标签库描述符里声明你的类，如下所示：

## Listing 1. An Example TLD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
web-jsptaglibrary_2_0.xsd"
version="2.0" >

<tlib-version>1.0</tlib-version>
<jsp-version>2.0</jsp-version>
<short-name>myTags</short-name>
<description>
    Tag library to support the examples in Chapter 8
</description>
<tag>
    <name>myTag</name>
    <tag-class>jspbook.ch08.myTag</tag-class>
```

```
<body-content>JSP</body-content>  
<attribute>  
  <name>myAttribute</name>  
  <required>yes</required>  
</attribute>  
</tag>  
</taglib>
```

通过在JSP页面里首先使用taglib指示符声明以后，这个定制标签就能够在页面里被引用，如下所示：

```
<%@ taglib uri="/helpers" prefix="helpers" %>  
  
<helpers:myTag myAttribute="some value">  
  Body text...  
</helpers:myTag>
```

我更倾向于使用定制标签作为View Helper的实现方式。因为它们集中存储在应用服务器的时候，它们给了开发人员更多的访问Servlet上下文

的权限并且提供了更多的性能优点。另一个我倾向于使用定制标签的原因是它更利于非Java开发人员直观的使用，它们的格式更像标准的HTML标签，这些HTML标签对于我们大多数的人来说再熟悉不过。最后，一旦这些定制标签经过了你的开发和测试，你就可以在你的整个项目的所有JSP页面使用它们。一旦这些定制标签被设计得更加通用，那么你可以将它们打包起来在所有的项目中使用到它们。

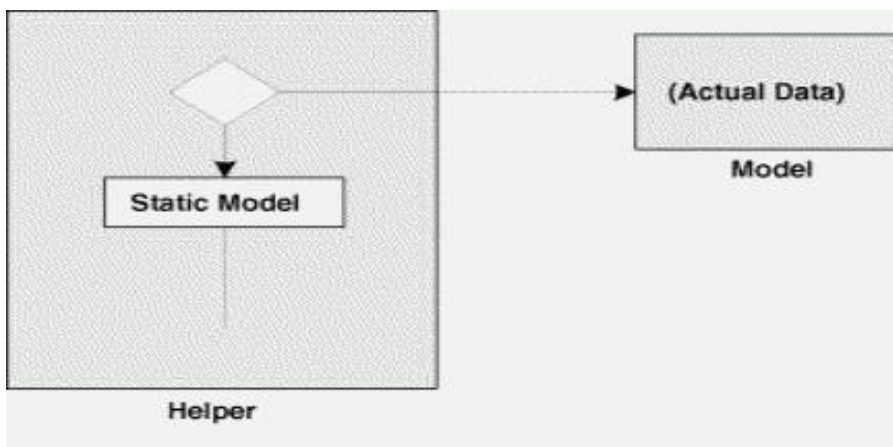
## 实现Model分离的策略

不管是使用定制标签或者JavaBeans，提供独立的Helper是非常有用的，这些独立的Helper能够在没有Model数据存在的时候提供一系列的假数据来代替Model数据。这使得页面开发人员可以独立于开发团队而完成它们的任务。为了实现这种策略，Helper需要去检测Model的存在，以便使用一个真实的Model数据或者使用一个静态的假

Model数据（如下图）。

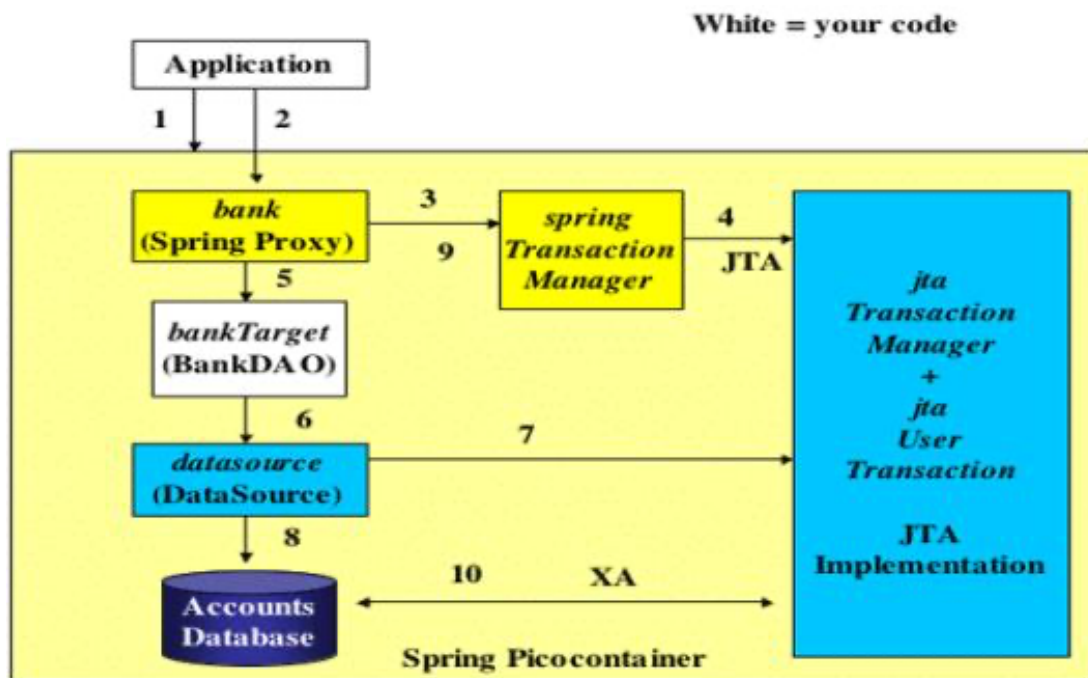
推荐：[JSP设计模式基础：View Helper模式——学习如何使用View Helper模式使得Model数据适应表现层的需要（2）](#)

[ 应用View Helper模式下面的Helper可能对你在某些方面特别有用，但是最起码，他们要告诉你怎么在你的应用中使用View Helper模式。下面是一个定制标签的实现并且被声明在he



按照这个思路，我们需要创建一个静态的Model

来精确复制真实的Model。不是所有的时候都能让这两者保持同步。一个在某些时候更完美的替代方法是，让开发人员创造一些假数据到Model里头，以便页面开发人员能够当作真实Model已经存在一样；同时也确保了他们自己工作的那个Model永远是正确的（见图三）。



## 应用View Helper模式

下面的Helper可能对你在某些方面特别有用，但是最起码，他们要告诉你怎么在你的应用中使用View Helper模式。下面是一个定制标签的实现并

且被声明在helpers.tld文件里。这个文件在web.xml 文件里做为一个条目和/helpers标签uri相关联。如下所示：

```
<taglib>  
  <taglib-uri>/helpers</taglib-uri>  
  <taglib-location>/WEB-INF/tlds/helpers.tld</taglib-location>  
</taglib>
```

## 格式化文本

在下面的章节里，我将以一个用来格式化日期和货币的View Helper开始。虽然这个需求在Model里实现可能比较简单，但是你有很多原因使得你在View里实现它们，而不是在Model里。例如，你可能使用不同的格式来显示它们，或者可能内容因为要被不同的设置访问而必须提供不同的访问方式。

你可以在定制标签体内封装很多格式用来格式化标签所取得的数据，然后你给标签一个属性来取

得使用者所需要的格式类型，根据这个属性选择相应的格式并将其输出出来。如下所示，你可以在标签库表述符里描述你的标签：

Listing 2. helpers.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
web-jsptaglibrary_2_0.xsd"
version="2.0" >
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>helperTags</short-name>
  <description>
    Tag library to support the examples in Chapter 8
  </description>
```



```
<tag>
  <name>FormatTag</name>
  <tag-class>jspbook.ch08.FormatTag</tag-class>
  <body-content>JSP</body-content>
  <attribute>
    <name>format</name>
    <required>yes</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>

</taglib>
```

这个标签的Model能够无所不包，在这个例子中，你将创建一个静态的JavaBeans包含两个String类型的属性，一个用来保存Date，一个用来保存货币。你将通过使用标准的JSP setProperty标签在页面里设置这些值。为了达到这样的目的，你的JavaBeans必须为这两个属性提供方法入门。如下所示，是用来产生JavaBeans的Java代码：

## Listing 3. FormattingModel.java

```
package jspbook.ch08.beans;

import java.io.Serializable;

public class FormattingModel implements
    Serializable {

    private String dateValue;
    private String currencyValue;
    public FormattingModel () {}

    /* Accessor Methods */
    public void setDateValue (String _date)
    {
        this.dateValue = _date;
    }

    public String getDateValue ()
```

```
{  
    return this.dateValue;  
}  
public void setCurrencyValue (String _currency)  
{  
    this.currencyValue = _currency;  
}  
  
public String getCurrencyValue ()  
{  
    return this.currencyValue;  
}  
}
```

标签是一个简单的body标签，由BodyTagSupport类继承而来。所有的格式化代码都在formatValue（）方法里。在doAfterBody（）里，一旦获取了数据，这个方法就会被调用。调用formatValue（）方法的结果被写回了页面标签所

在的位置。你可以使用java.text包的类来格式化日期或货币，而且，你可以使用SimpleDateFormat 和 DecimalFormat类。

标签处理者也提供了Locale 对象，通过相应的set方法，能够完成对内容的国际化。因为这个标签的职责是格式化日期和货币，那么它必然在不同的地区有不同的格式化要求。看看下面的代码，特别是要注意formatValue () 方法：

Listing 4. FormatTag.java

```
package jspbook.ch08;

import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.JspTagException;

import javax.servlet.jsp.tagext.BodyTagSupport;
import javax.servlet.jsp.tagext.BodyContent;

import java.io.IOException;
```

```
import java.util.Locale;
import java.util.Calendar;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.text.NumberFormat;
import java.text.DecimalFormat;

public class FormatTag extends BodyTagSupport {

    /* Locale object for internationalization of content
    */
    private Locale locale;

    /* Tag Attributes */
    protected String format;

    /* Static Constants */
    private final static String DATE_LONG = "date";
    private final static String NUMERIC_DECIMAL =
"decimal";
```

```
private final static String NUMERIC_ROUNDED  
= "rounded";
```

```
private final static String  
NUMERIC_CURRENCY = "currency";
```

```
public FormatTag() {  
    locale = Locale.getDefault();  
}
```

```
public void setLocale(Locale locale) {  
    this.locale = locale;  
}
```

```
/* Process Tag Body */  
public int doAfterBody() throws JspTagException {  
    try {  
        BodyContent body = getBodyContent();  
        JspWriter out = body.getEnclosingWriter();  
  
        /* Get Input Value */  
        String textValue = body.getString().trim();
```

```
    /* Output Formatted Value */
    out.println(formatValue(textValue));
}
catch (IOException e) {
    throw new JspTagException(e.toString());
}
return SKIP_BODY;
}

/* Process End Tag */
public int doEndTag() throws JspTagException {
    return EVAL_PAGE;
}

private String formatValue (String _input)
{
    String formattedValue = "";
    try {
        if(format.equals(DATE_LONG)) {
            Calendar cal = Calendar.getInstance();
            cal.setTime(DateFormat.getDateInstance(
                DateFormat.SHORT).parse(_input));
        }
    }
}
```

```
SimpleDateFormat df = new
SimpleDateFormat("EEE, MMM d, yyyy");
    formattedValue = df.format(cal.getTime());
} else
if(format.equals(NUMERIC_DECIMAL)) {
    DecimalFormat dcf = (DecimalFormat)
NumberFormat.getInstance(locale);
    dcf.setMinimumFractionDigits(2);
    dcf.setMaximumFractionDigits(2);
    formattedValue =
dcf.format(dcf.parse(_input));
} else
if(format.equals(NUMERIC_ROUNDED)) {
    DecimalFormat dcf = (DecimalFormat)
NumberFormat.getInstance(locale);
    dcf.setMinimumFractionDigits(0);
    dcf.setMaximumFractionDigits(0);
    formattedValue =
dcf.format(dcf.parse(_input));
} else
if(format.equals(NUMERIC_CURRENCY)) {
    float num = Float.parseFloat(_input);
```



```
        DecimalFormat dcf = (DecimalFormat)
        NumberFormat.getCurrencyInstance();
        formattedValue = dcf.format(num);
    }
}
catch (Exception e) {
    System.out.println(e.toString());
}

return formattedValue;
}

/* Attribute Accessor Methods */
public String getFormat ()
{
    return this.format;
}

public void setFormat (String _format)
{
    this.format = _format;
}
```

```
}  
}
```

最后，你将在JSP页面使用该标签，这里实在是没有什么新东西。页面声明一个JavaBeans来作为Model使用，在这个Model里设置值，使用不同的格式来显示这些值。这些格式化的动作通过FormatTag实现，该标签在helpers.tld里给定，并且在JSP页面使用taglib 指示符来声明。注意：你需要通过标签的一个属性来设置格式类型。format 属性就是用来指定一个格式类型的值，而这个值必须依赖于标签里设定的那些常量来确定。如下所示，是JSP代码：

Listing 5. formatHelper.jsp

```
<%-- Declare tag that we'll use as our helper --%>
```

```
<%@ taglib uri="/helpers" prefix="helpers" %>
```

```
<html>
  <head>
    <title>Text Formatting Example</title>
  </head>
  <body>

    <font/>

    <%-- Declare bean that will act as our model --
%>
    <jsp:useBean id="myBean"
class="jspbook.ch08.beans.FormattingModel"/>

    <jsp:setProperty name="myBean"
property="dateValue" value="12/01/01"/>
    <jsp:setProperty name="myBean"
property="currencyValue" value="23500.253"/>

    <%-- Display Formatted Values (using helper) --
%>
    <center>
```

# <h1>Various Date and Currency Formats</h1>

<br/><br/>

<table cellpadding="5">

<tr>

<th>Format</th>

<th>Original Value</th>

<th>Formatted Value</th>

</tr>

<tr>

<td>Long Date</td>

<td>

<jsp:getProperty name="myBean"  
property="dateValue"/>

</td>

<td>

<helpers:FormatTag format="date">

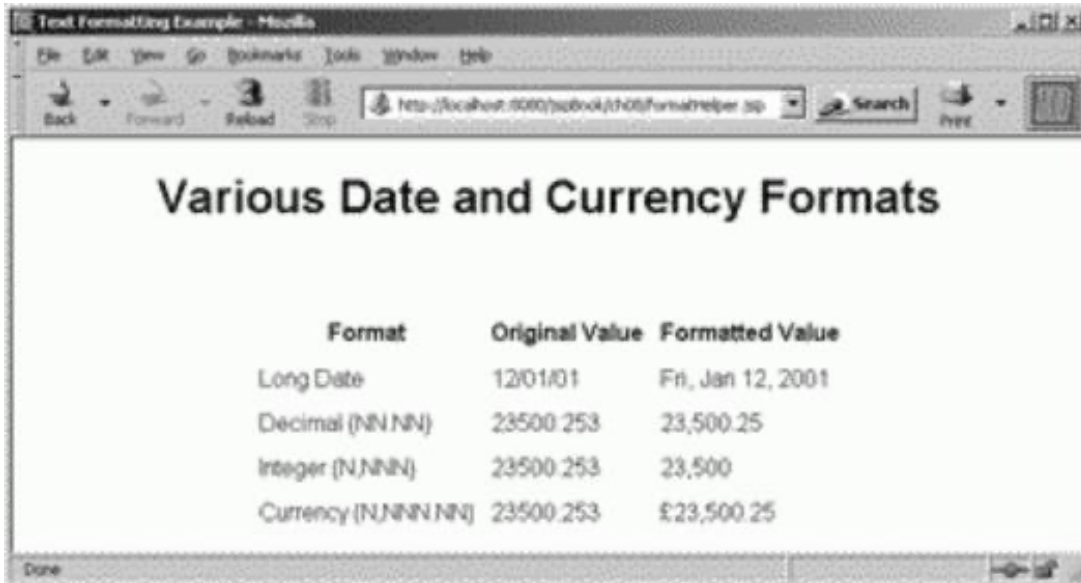
<jsp:getProperty name="myBean"  
property="dateValue"/>

</helpers:FormatTag>

```
</td>
</tr>
<tr>
  <td>Decimal (NN.NN)</td>
  <td>${myBean.currencyValue}</td>
  <td>
    <helpers:FormatTag format="decimal">
      ${myBean.currencyValue}
    </helpers:FormatTag>
  </td>
</tr>
<tr>
  <td>Integer (N,NNN)</td>
  <td>${myBean.currencyValue}</td>
  <td>
    <helpers:FormatTag format="rounded">
      ${myBean.currencyValue}
    </helpers:FormatTag>
  </td>
</tr>
<tr>
  <td>Currency (N,NNN.NN)</td>
```

```
<td>${myBean.currencyValue}</td>
<td>
  <helpers:FormatTag format="currency">
    ${myBean.currencyValue}
  </helpers:FormatTag>
</td>
</tr>
</table>
</center>
</body>
</html>
```

下图是显示结果：



Format	Original Value	Formatted Value
Long Date	12/01/01	Fri, Jan 12, 2001
Decimal (NN.NN)	23500.253	23,500.25
Integer (N.NNN)	23500.253	23,500
Currency (N.NNN.NN)	23500.253	£23,500.25

推荐：[JSP设计模式基础：View Helper模式——学习如何使用View Helper模式使得Model数据适应表现层的需要（1）](#)

[ JSP设计模式基  
础：View Helper模式 ——  
学习如何使用View Helper模式使得Model数  
据适应表现层

主题：

- [设计模式](#)
- [jsp](#)
- [设计](#)
- [bean](#)
- [xml](#)
- [view](#)
- [servlet](#)

分享:

## 相关推荐

- [JSP设计模式基础：View Helper模式——学习如何使用View Helper模式使得Model数据适应表现层的需要 \(2\)](#)
- [JSP设计模式基础：View Helper模式——学习](#)



[如何使用View Helper模式使得Model数据适应表现层的需要 \(1\)](#)

- [JSP设计模式基础：View Helper模式——学习如何使用View Helper模式使得Model数据适应表现层的需要 \(3\)](#)

## 热 门 推 荐

- [JSP设计模式基础：View Helper模式——学习如何使用View Helper模式使得Model数据适应表现层的需要 \(1\)](#)
- [JSP设计模式基础：View Helper模式——学习如何使用View Helper模式使得Model数据适应表现层的需要 \(2\)](#)
- [JSP设计模式基础：View Helper模式——学习如何使用View Helper模式使得Model数据适应表现层的需要 \(3\)](#)
- [JSP设计模式基础：View Helper模式——学习](#)

# 如何使用View Helper模式使得Model数据适应表现层的需要

- 首页
- 关于我们
- 手机版
- 电脑版

©2017 ITBOTH.COM [m.itboth.com](http://m.itboth.com)