**Java新知—Java学习中遇到的知识**

目录

[关于Eclipse IDE的使用技巧 2](#_Toc377901360)

[1.安装新软件 2](#_Toc377901361)

[2.配置Tomcat服务器 2](#_Toc377901362)

[3创建Ant的build.xml 3](#_Toc377901363)

[Java基础（Java SE:面向对象、多线程、IO、网络、数据库、常用类和API） 10](#_Toc377901364)

[0. 关于配置环境变量 10](#_Toc377901365)

[2. 利用javadoc生成代码文档(2013.12.26) 10](#_Toc377901366)

[2.目录操作：File类、FilenameFilter接口(2013.12.26) 12](#_Toc377901367)

[3.关于编码风格 12](#_Toc377901368)

[4. 在2中讲到的正则表达式 13](#_Toc377901369)

[5.文件操作 13](#_Toc377901370)

[6.数据库操作 16](#_Toc377901371)

[Java Web(JDBC、DAO、Servlet、表单、Jsp、JSTL、Tomcat配置) 21](#_Toc377901372)

[1.JSP 21](#_Toc377901373)

[2.Servlet 21](#_Toc377901374)

[3.JDBC 46](#_Toc377901375)

[4.用户自定义标签 46](#_Toc377901376)

[5.EL与JSTL 46](#_Toc377901377)

[SSH三大核心框架(Structs、Spring、Hibernate) 46](#_Toc377901378)

[1.Structs2 46](#_Toc377901379)

[2.Spring 57](#_Toc377901380)

[3.Hibernate 154](#_Toc377901381)

[EJB（Enterprise JavaBean，WebService、Entity Bean等） 179](#_Toc377901382)

[常见问题及解答 179](#_Toc377901383)

[关于Eclipse IDE 179](#_Toc377901384)

[关于Tamcat 179](#_Toc377901385)

[关于Java 179](#_Toc377901386)

[关于SSH框架的整合 180](#_Toc377901387)

## 关于Eclipse IDE的使用技巧

### 1.安装新软件

1.Help->install new software

2.可以选择用插件地址在线安装(需要地址)，也可以选择下载好之后再安装（add-archive）

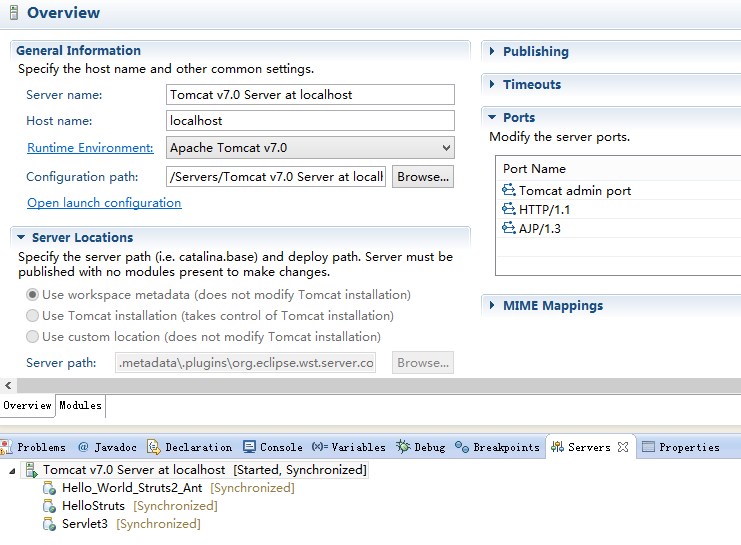
### 2.配置Tomcat服务器

#### 2.1简述

在Eclipse中开发web程序，特别是对于刚接触Eclipse的新手来说，配置Tomcat服务器是最恼火的一件事。不管怎么弄，总会弄出好多问题出来。常见的问题有：如何添加服务器，添加之后又如何配置（要配置啥子？反正我是不知道），如何调试自己的web程序。

#### 2.2将调试输出目录调到Tomcat的webapp目录下Eclipse、Tomcat都可以浏览网站

Windows->show View->other->Servers然后在控制台栏上多了一个Servers选项卡，如果没有服务器的话就会提示你新建一个服务器(无非就是选一下Tomcat的版本和目录之类的，不多说)，有了服务器后，双击它，会显示服务器详细信息，要怎么配置就怎么配置。



#### 2.3 Tomcat环境变量配置

1.配置JDK：添加JAVA\_HOME变量，添加CATALINA\_HOME变量，添加CLASSPATH变量

2.具体如下（目录根据个人的配置有所不一样）：

JAVA\_HOME:D:\Program Files\Java\jdk1.7.0\_60

CATALINA\_HOME:E:\Java\tomcats\apache-tomcat-7.0.47

CLASSPATH:%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\tools.jar;%JAVA\_HOME%\jre\ext\postgresql-9.3-1100.jdbc41.jar;

Path:%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;%CATALINA\_HOME%\bin;%CATALINA\_HOME%\lib;

#### 2.4Tomcat启动失败问题

在Server中配置，然后一般设置编译目录到tomcat的webapp下，注意一旦出现什么错误，注意web.xml及类文件是否有错，因为基本是他们的错。Tomcat启动失败通常是由于项目出错,提示错误有,什么什么child组件启动失败，然后某某项目无法启动导致，就是这个操蛋的项目有问题，不要怀疑Tomcat，就算要怀疑，也只能去怀疑配置环境的问题

### 3创建Ant的build.xml

#### 3.1Ant简介(来自百度百科)

**简介**

Apache Ant,是一个将[软件](http://baike.baidu.com/view/37.htm)编译、测试、部署等步骤联系在一起加以自动化的一个工具，大多用于Java环境中的软件开发。由Apache[软件](http://baike.baidu.com/view/37.htm)基金会所提供。[1]

用户群：大多数的Java设计都被用于管理大量信息流，例如[纽约州](http://baike.baidu.com/view/190078.htm)就使用Apache Ant去管理[美国](http://baike.baidu.com/view/2398.htm)最大的青年计划，每天可以实时更新超过25万学生的记录。

只要使用过Linux系统的读者，应该知道 make这个命令。当编译[Linux内核](http://baike.baidu.com/view/573460.htm)及一些[软件](http://baike.baidu.com/view/37.htm)的[源程序](http://baike.baidu.com/view/546605.htm)时，经常要用这个命令。Make命令其实就 是一个[项目管理工具](http://baike.baidu.com/view/3169783.htm)，而Ant所实现功能与此类似。像make，gnumake和nmake这些编译工具都有 一定的缺陷，但是Ant却克服了这些工具的缺陷。最初Ant开发者在开发跨平台的应用时，同样也 是基于这些缺陷对Ant做了更好的设计。

**2****优点**

Ant是Apache[软件](http://baike.baidu.com/view/37.htm)基金会JAKARTA目录中的一个子项目，它有以下的优点。跨平台性。Ant是纯[Java语言](http://baike.baidu.com/view/229611.htm)编写的，所以具有很好的跨平台性。操作简单。Ant是由一个内置任务和可选任务组成的。Ant运行时需要一个XML文件(构建文件)。 Ant通过调用target树，就可以执行各种task。每个task实现了特定接口对象。由于Ant构建文件 时XML格式的文件，所以很容易维护和书写，而且结构很清晰。Ant可以集成到开发环境中。由于Ant的跨平台性和操作简单的特点，它很容易集成到一些开发环 境中去。

**3****开发**

Ant的构建文件

当开始一个新的项目时，首先应该编写Ant构建文件。构建文件定义了构建过程，并被团队开发 中每个人使用。Ant构建文件默认名为build.xml，也可以取其他的名字。只不过在运行的时候 把这个命名当作参数传给Ant。构建文件可以放在任何的位置。一般做法是放在项目顶层目录中。 这样可以保持项目的简洁和清晰。下面是一个典型的项目层次结构。

(1) src存放文件。

(2) class存放编译后的文件。

(3) lib存放第三方JAR包。

(4) dist存放打包，发布以后的代码。

Ant构建文件是XML文件。每个构建文件定义一个唯一的项目(Project元素)。每个项目下可以定 义很多目标(target元素)，这些目标之间可以有依赖关系。当执行这类目标时，需要执行他们所 依赖的目标。每个目标中可以定义多个任务，目标中还定义了所要执行的任务序列。Ant在构建目标时必须调 用所定义的任务。任务定义了Ant实际执行的命令。Ant中的任务可以为3类。

（1） 核心任务。核心任务是Ant自带的任务。

（2） 可选任务。可选任务实来自第三方的任务，因此需要一个附加的JAR文件。

（3） 用户自定义的任务。用户自定义的任务实用户自己开发的任务。

**4****标签**

1**.<project>标签**

每个构建文件对应一个项目。<project>标签时构建文件的根标签。它可以有多个内在属性， 就如代码中所示，其各个属性的含义分别如下。

(1) default表示默认的运行目标，即指定默认的target（即任务）。这个属性是必须的。

(2) basedir表示项目的基准目录。

(3) name表示项目名。

(4) description表示项目的描述。

每个构建文件都对应于一个项目，但是大型项目经常包含大量的子项目，每一个子项目都可以有

自己的构建文件。

2**.<target>标签**

一个项目标签下可以有一个或多个target标签。一个target标签可以依赖其他的target标签。例 如，有一个target用于编译程序，另一个target用于生成[可执行文件](http://baike.baidu.com/view/159830.htm)。在生成[可执行文件](http://baike.baidu.com/view/159830.htm)之前必须先编译该文件，因此可执行文件的target依赖于编译程序的target。Target的所有属性如下。

(1).name表示标明，这个属性是必须的。

(2).depends表示依赖的目标。

(3)if表示仅当属性设置时才执行。

(4)unless表示当属性没有设置时才执行。

(5)description表示项目的描述。

Ant的depends属性指定了target的执行顺序。Ant会依照depends属性中target出现顺序依次执行 每个target。在执行之前，首先需要执行它所依赖的target。程序中的名为run的target的 depends属性compile，而名为compile的target的depends属性是prepare，所以这几个target执 行的顺序是prepare->compile->run。一个target只能被执行一次，即使有多个target依赖于它。如果没有if或unless属性target总 会被执行。

**3.<mkdir>标签**

该标签用于创建一个目录，它有一个属性dir用来指定所创建的目录名，其代码如下：

<mkdir dir=”${class.root}”/>

通过以上代码就创建了一个目录，这个目录已经被前面的property标签所指定。

4**<jar>标签**

该标签用来生成一个JAR文件，其属性如下。

(1) destfile表示JAR文件名。

(2) basedir表示被归档的文件名。

(3) includes表示被归档的文件模式。

(4) exchudes表示被排除的文件模式。

5**．<javac标签>**

该标签用于编译一个或一组java文件，其属性如下。

(1).srcdir表示源程序的目录。

(2).destdir表示[class文件](http://baike.baidu.com/view/3922213.htm)的输出目录。

(3).include表示被编译的文件的模式。

(4).excludes表示被排除的文件的模式。

(5).classpath表示所使用的类[路径](http://baike.baidu.com/view/59642.htm)。

(6).debug表示包含的调试信息。

(7).optimize表示是否使用优化。

(8).verbose 表示提供详细的输出信息。

(9).fileonerror表示当碰到错误就自动停止。

6**．<java>标签**

该标签用来执行编译生成的.[class文件](http://baike.baidu.com/view/3922213.htm)，其属性如下。

(1).classname 表示将执行的类名。

(2).jar表示包含该类的JAR文件名。

(3).classpath所表示用到的类路径。

(4).fork表示在一个新的[虚拟机](http://baike.baidu.com/view/1132.htm)中运行该类。

(5).failonerror表示当出现错误时自动停止。

(6).output 表示输出文件。

(7).append表示追加或者覆盖默认文件。

7**.<delete>标签**

该标签用于删除一个文件或一组文件，去属性如下。

(1)/file表示要删除的文件。

(2).dir表示要删除的目录。

(3).includeEmptyDirs 表示指定是否要删除空目录，默认值是删除。

(4).failonerror 表示指定当碰到错误是否停止，默认值是自动停止。

(5).verbose表示指定是否列出所删除的文件，默认值为不列出。

8**.<copy>标签**

该标签用于文件或文件集的拷贝，其属性如下。

(1).file 表示[源文件](http://baike.baidu.com/view/385166.htm)。

(2).tofile 表示目标文件。

(3).todir 表示目标目录。

(4).overwrite 表示指定是否覆盖目标文件，默认值是不覆盖。

(5).includeEmptyDirs 表示制定是否拷贝空目录，默认值为拷贝。

(6).failonerror 表示指定如目标没有发现是否自动停止，默认值是停止。

(7).verbose 表示制定是否显示详细信息，默认值不显示。

9**.<exec>执行文件**：

<execexecutable="${base.dir}/email.bat" >

</exec>

**5****数据类型**

在构建文件中为了标识文件或[文件组](http://baike.baidu.com/view/701355.htm)，经常需要使用[数据类型](http://baike.baidu.com/view/675645.htm)。数据类型包含在 org.apache.tool.ant.types包中。下面就简单介绍构建文件中一些常用的数据类型。

1. argument 类型

由Ant构建文件调用的程序，可以通过<arg>元素向其传递[命令行参数](http://baike.baidu.com/view/1646320.htm)，如apply,exec和java任 务均可接受嵌套<arg>元素，可以为各自的过程调用指定参数。以下是<arg>的所有属性。

(1).values 是一个命令参数。如果参数种有空格，但又想将它作为单独一个值，则使用此属性 。

(2).file表示一个参数的文件名。在构建文件中，此文件名相对于当前的工作目录。

(3).line表示用空格分隔的多个[参数列表](http://baike.baidu.com/view/8423750.htm)。

(4).path表示路径。

2.ervironment 类型

由Ant构建文件调用的[外部命令](http://baike.baidu.com/view/639618.htm)或程序，<env>元素制定了哪些[环境变量](http://baike.baidu.com/view/95930.htm)要传递给正在执行的系 统命令，<env>元素可以接受以下属性。

(1).file表示[环境变量](http://baike.baidu.com/view/95930.htm)值得文件名。此文件名要被转换位一个[绝对路径](http://baike.baidu.com/view/25377.htm)。

(2).path表示[环境变量](http://baike.baidu.com/view/95930.htm)的路径。Ant会将它转换为一个本地约定。

(3).value 表示[环境变量](http://baike.baidu.com/view/95930.htm)的一个直接变量。

(4).key 表示[环境变量](http://baike.baidu.com/view/95930.htm)名。

注意 file path 或value只能取一个。

3.filelist类型

Filelist 是一个支持命名的文件列表的数据类型，包含在一个filelist类型中的文件不一定是

存在的文件。以下是其所有的属性。

(1).dir是用于计算绝对文件名的目录。

(2).files 是用逗号分隔的文件名列表。

(3).refid 是对某处定义的一个<filelist>的引用。

注意 dir 和files 都是必要的，除非指定了refid(这种情况下，dir和files都不允许使用)。

4.fileset类型

Fileset [数据类型](http://baike.baidu.com/view/675645.htm)定义了一组文件，并通常表示为<fileset>元素。不过，许多ant任务构建成了 隐式的fileset,这说明他们支持所有的fileset属性和嵌套元素。以下为fileset 的属性列表。

(1).dir表示fileset 的基目录。

(2).casesensitive的值如果为false，那么匹配文件名时，fileset不是区分大小写的，其默认 值为true.

(3).defaultexcludes 用来确定是否使用默认的排除模式，默认为true。

(4).excludes 是用逗号分隔的需要派出的文件模式列表。

(5).excludesfile 表示每行包含一个排除模式的文件的文件名。

(6).includes 是用逗号分隔的，需要包含的文件模式列表。

(7).includesfile 表示每行包括一个包含模式的文件名。

5.patternset 类型

Fileset 是对文件的分组，而patternset是对模式的分组，他们是紧密相关的概念。 <patternset>支持4个属性：includesexcludex includexfile 和excludesfile,与fileset相 同。Patternset 还允许以下嵌套元素：include,exclude,includefile 和excludesfile.

6.filterset 类型

Filterset定义了一组过滤器，这些过滤器将在文件移动或复制时完成文件的[文本替换](http://baike.baidu.com/view/893270.htm)。

主要属性如下：

(1).begintoken 表示嵌套过滤器所搜索的记号，这是标识其开始的字符串。

(2).endtoken表示嵌套过滤器所搜索的记号这是标识其结束的字符串。

(3).id是过滤器的唯一标志符。

(4).refid是对构建文件中某处定义一个过滤器的引用。

7.Path类型

Path元素用来表示一个类路径，不过它还可以用于表示其他的路径。在用作揖个属性时，路经中 的各项用分号或冒号隔开。在构建的时候，此[分隔符](http://baike.baidu.com/view/1268377.htm)将代替当前平台中所有的路径分隔符，其拥 有的属性如下。

(1).location 表示一个文件或目录。Ant在内部将此扩展为一个绝对路径。

(2).refid 是对当前构建文件中某处定义的一个path的引用。

(3).path表示一个文件或路径名列表。

8.mapper类型

Mapper类型定义了一组输入文件和一组输出文件间的关系，其属性如下。

(1).classname 表示实现mapper类的类名。当内置mapper不满足要求时，用于创建定制mapper.

(2).classpath表示查找一个定制mapper时所用的类型路径。

(3).classpathref是对某处定义的一个类路径的引用。

(4).from属性的含义取决于所用的mapper.

(5).to属性的含义取决于所用的mapper.

(6).type属性的取值为identity，flattenglob merge regexp 其中之一，它定义了要是用的

内置mapper的类型。

**6****运行**

安装好Ant并且配置好路径之后，在命令行中切换到构建文件的目录，输入Ant命令就可以运行 Ant.若没有指定任何参数，Ant会在[当前目录](http://baike.baidu.com/view/396351.htm)下查询build.xml文件。如果找到了就用该文件作为 构建文件。如果使用了–find 选项，Ant 就会在上级目录中找构建文件，直至到达文件系统的根目录。如果构建文件的名字不是build.xml，则Ant运行的时候就可以使用–buildfile file ,这里file 指定了要使用的构建文件的名称。

**Android**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **开发** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | ▪ [Android 软件开发工具包](http://baike.baidu.com/searchword/?word=Android%20%E8%BD%AF%E4%BB%B6%E5%BC%80%E5%8F%91%E5%B7%A5%E5%85%B7%E5%8C%85&pic=1&sug=1&enc=utf8) | ▪ **Apache Ant** | ▪ [Dalvik](http://baike.baidu.com/view/1551869.htm) | ▪ [Developer Challenge](http://baike.baidu.com/searchword/?word=Developer%20Challenge&pic=1&sug=1&enc=utf8) | | ▪ [Eclipse](http://baike.baidu.com/view/23576.htm) | ▪ [Google Code](http://baike.baidu.com/view/2252816.htm) | ▪ [Google App Inventor](http://baike.baidu.com/searchword/?word=Google%20App%20Inventor&pic=1&sug=1&enc=utf8) | ▪ [Google I/O](http://baike.baidu.com/view/8871201.htm) | | ▪ [Bionic](http://baike.baidu.com/view/3153584.htm) | ▪ [APK](http://baike.baidu.com/view/1351499.htm) | ▪ [Roboto](http://baike.baidu.com/searchword/?word=Roboto&pic=1&sug=1&enc=utf8) | ▪ [快速启动](http://baike.baidu.com/view/3824539.htm) | |  |  |  |  | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **使用者接口** | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | ▪ [HTC Sense](http://baike.baidu.com/view/2992159.htm) | ▪ [Motorola Motoblur](http://baike.baidu.com/searchword/?word=Motorola%20Motoblur&pic=1&sug=1&enc=utf8) | ▪ [Samsung TouchWiz](http://baike.baidu.com/searchword/?word=Samsung%20TouchWiz&pic=1&sug=1&enc=utf8) | ▪ [Sony UXP Experience](http://baike.baidu.com/searchword/?word=Sony%20UXP%20Experience&pic=1&sug=1&enc=utf8) | |  |  |  |  | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **应用程式** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | ▪ [地球](http://baike.baidu.com/view/2489.htm) | ▪ [Goggles](http://baike.baidu.com/searchword/?word=Goggles&pic=1&sug=1&enc=utf8) | ▪ [Gmail](http://baike.baidu.com/view/2418.htm) | ▪ [地图](http://baike.baidu.com/view/30267.htm) | | ▪ [星空](http://baike.baidu.com/view/146296.htm) | ▪ [Talk](http://baike.baidu.com/view/112883.htm) | ▪ [翻译](http://baike.baidu.com/view/9107.htm) | ▪ [Voice](http://baike.baidu.com/view/1291602.htm) | | ▪ [YouTube](http://baike.baidu.com/view/357961.htm) |  |  |  | |  |  |  |  | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **服务** | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | ▪ [Google Play](http://baike.baidu.com/view/8058603.htm) | ▪ [Google搜索](http://baike.baidu.com/searchword/?word=Google%E6%90%9C%E7%B4%A2&pic=1&sug=1&enc=utf8) | ▪ [Android云端推送（C2DM）](http://baike.baidu.com/searchword/?word=Android%E4%BA%91%E7%AB%AF%E6%8E%A8%E9%80%81%EF%BC%88C2DM%EF%BC%89&pic=1&sug=1&enc=utf8) |  | |  |  |  |  | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Google Nexus** | |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | ▪ [Nexus One](http://baike.baidu.com/view/3136941.htm) | ▪ [Nexus S](http://baike.baidu.com/view/4726989.htm) | ▪ [Galaxy Nexus](http://baike.baidu.com/view/6445916.htm) | ▪ [Nexus 7](http://baike.baidu.com/view/8866161.htm) | | ▪ [Nexus Q](http://baike.baidu.com/view/8875928.htm) |  |  |  | |  |  |  |  | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **第三方固件** | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | ▪ [CyanogenMod](http://baike.baidu.com/view/5079569.htm) | ▪ [MIUI](http://baike.baidu.com/view/4691942.htm) | ▪ [InsertCoin](http://baike.baidu.com/searchword/?word=InsertCoin&pic=1&sug=1&enc=utf8) |  | |  |  |  |  | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **衍生平台** | |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | ▪ [MIUI](http://baike.baidu.com/view/4691942.htm) | ▪ [OPhone](http://baike.baidu.com/view/2410021.htm) | ▪ [百度易平台](http://baike.baidu.com/view/6406266.htm) | ▪ [Replicant](http://baike.baidu.com/searchword/?word=Replicant&pic=1&sug=1&enc=utf8) | | ▪ [Le OS](http://baike.baidu.com/searchword/?word=Le%20OS&pic=1&sug=1&enc=utf8) | ▪ [阿里云OS](http://baike.baidu.com/view/6201244.htm) | ▪ [Flyme OS](http://baike.baidu.com/view/8085291.htm) |  | |  |  |  |  | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **列表** | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | ▪ [Android相关主题](http://baike.baidu.com/searchword/?word=Android%E7%9B%B8%E5%85%B3%E4%B8%BB%E9%A2%98&pic=1&sug=1&enc=utf8) | ▪ [Android装置列表](http://baike.baidu.com/searchword/?word=Android%E8%A3%85%E7%BD%AE%E5%88%97%E8%A1%A8&pic=1&sug=1&enc=utf8) | ▪ [Android版本历史](http://baike.baidu.com/searchword/?word=Android%E7%89%88%E6%9C%AC%E5%8E%86%E5%8F%B2&pic=1&sug=1&enc=utf8) | ▪ [Android开源应用](http://baike.baidu.com/searchword/?word=Android%E5%BC%80%E6%BA%90%E5%BA%94%E7%94%A8&pic=1&sug=1&enc=utf8) | |  |  |  |  | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **其他** | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  | | --- | --- | --- | --- | | ▪ [Root机](http://baike.baidu.com/searchword/?word=Root%E6%9C%BA&pic=1&sug=1&enc=utf8) |  |  |  | |  |  |  |  | | |

参考资料

#### 3.2创建build.xml

Eclipse 自动生成 Ant的Build.xml 配置文件,生成的方法很隐蔽

选择你要生成Build.xml文件的项目,右键. Export-> General -> Ant Buildfiles .

点Next,再点Finish.

生成完毕.

希望使用的可以试试了。总算不用再傻傻的自己编写build.xml了。

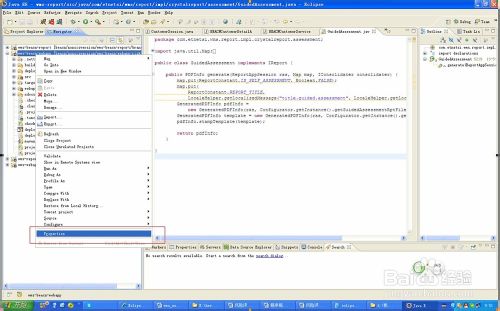
#### 3.3 eclipse打包war文件

最近在做数据迁移的工作， 自己的任务是将应用程序迁移到新的测试服务器上，然而在我们开发现场使用的Web服务器是Tomcat，而客户现场测试环境使用的Web服务器是Weblogic。怎样将我们开发现场的应用部署到客户现场的Web服务器上，我产生了疑惑。最好发现其实很简单，只要将应用程序使用Eclipse打包成war文件，在开发现场上的Tomcat服务器上部署成功，将这个war包部署到客户现场的Weblogic服务器，也能部署成功！下面是eclipse下打包war文件的详细过程。

**方法/步骤**

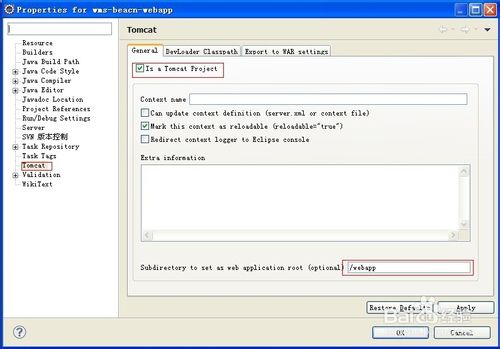
1. 1

在项目上点右键->properties

[[](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=1)步骤阅读](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=1)

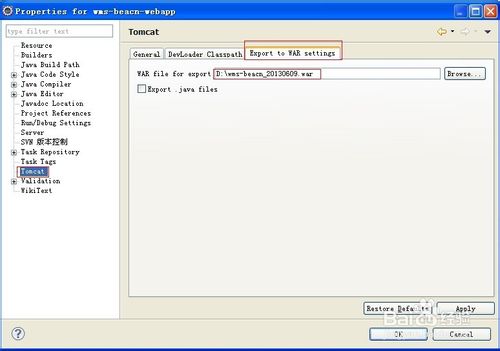
1. 2

在'Tomcat'下 'General' 选项卡，将 'Is a Tomcat Project' 勾选上，选择启动路径为webapp

[[](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=2)步骤阅读](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=2)

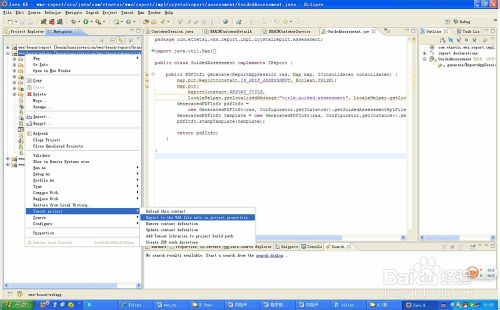
1. 3

在'Tomcat'下 'export to war settings' 选项卡，输入要导出的war文件路径和文件名，确定,返回项目

[[](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=3)步骤阅读](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=3)

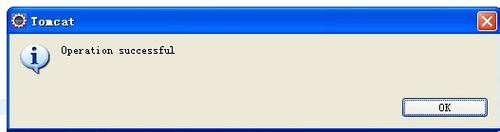
1. 4

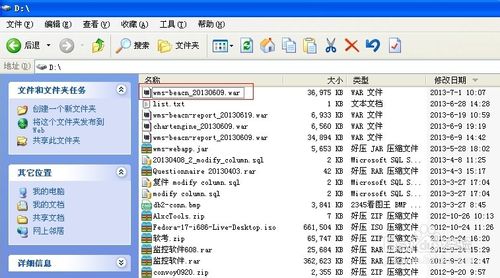
在项目上点右键->tomcat project->Export to the war file sets in project properties生成WAR包

[[](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=4)步骤阅读](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=4)

1. 5

打包成功，在指定的目录下看到生成的war文件！

[[](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=5)步骤阅读](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=5)

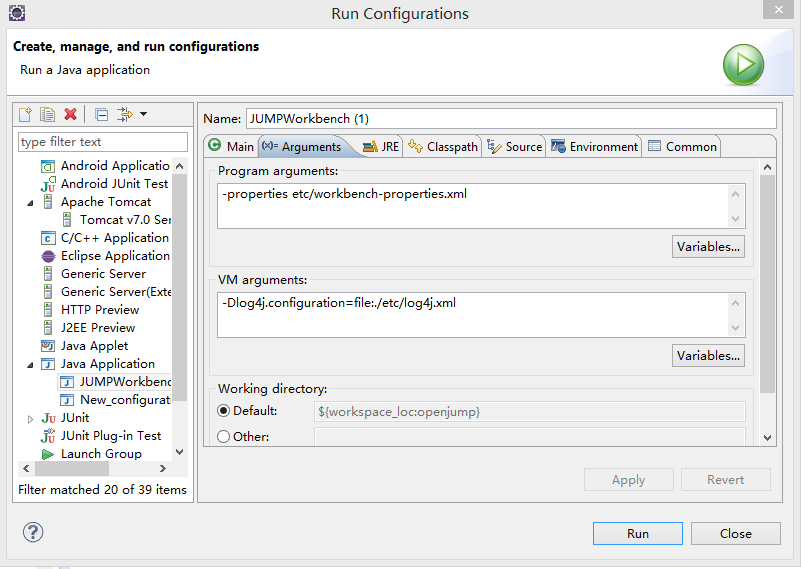
[[](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=6)步骤阅读](http://jingyan.baidu.com/album/54b6b9c026455d2d583b478c.html?picindex=6)

END

#### 3.4新建一个Ant Builder

右键项目，选择属性，Builder，右边框框中如果没看到有Ant Builder则单击右边new按钮新建一个，跟着不知走可以了。

### 4.关于Program Argument即main中的args



在里面添加即可添加args，注意args的格式：[-name,p1,p2,…,-name2,p1,p2,…,…]

### 5.Eclipse安装jdk

window -> preference -> java -> install jres -> add -> standard vm -> 设置好相应的jdk即可。如我的默认安装位置为d盘，选择jdk目录即可：D:\Program Files\Java\jdk1.7.0\_60

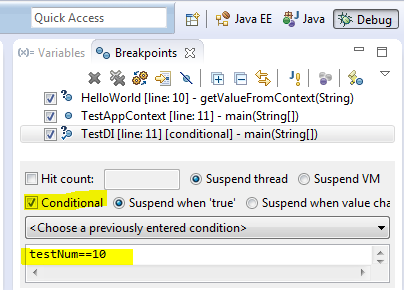
### 调试技巧

在开始之前，推荐大家去看看[Eclipse shortcuts](http://javapapers.com/core-java/eclipse-shortcuts/)这篇文章，它将会给你带来很多方便。在本文中使用的是Eclipse Juno版（Eclipse 4.2），在开始前给大家提3点建议！

1. 不要使用[System.out.println](http://javapapers.com/core-java/system-out-println/)作为调试工具
2. 把所有涉及到的组件日志级别激活并使用
3. 使用日志分析器来读取日志

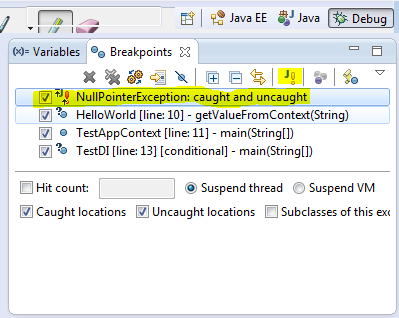
#### **1.条件断点**

如果你不知道如何添加断点，只需点击左边面板（行号前面）断点即被创建。在调试界面中，“断点”视图会把所有被创建的断点列出来。我们可以给它加一个布尔条件，也就是说，该断点会被激活并且如果布尔条件为真，就会执行该断点，否则将会跳过往下执行。



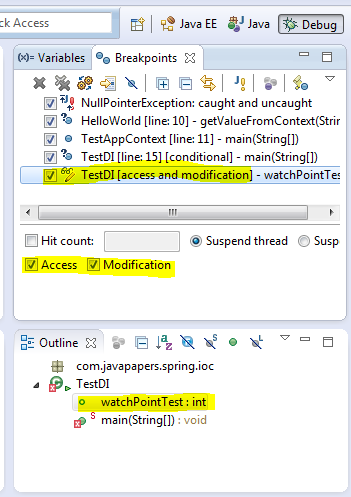
#### **2.异常断点**

在断点视图中，有一个J!标记按钮！我们可以使用该按钮来添加一个Java异常断点。例如，我们想让程序在遇到空指针异常（NullPointerException）时，仍然能继续调试，那么我们可以使用该按钮来添加一个异常断点！



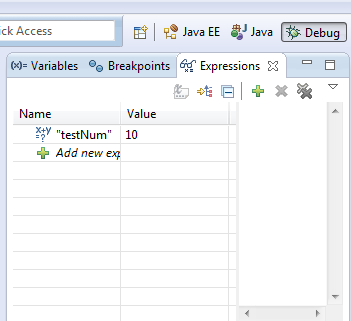
#### **3.监视点**

这是一个非常好的功能，当选定的属性访问或修改程序时，程序会停止执行并允许进行调试。在Outline视图中选择一个类变量并从上下文菜单中选择切换监视点，属性监视点将会被创建，在断点（Breakpoints）视图中会把所有监视点用列表的形式显示出来。



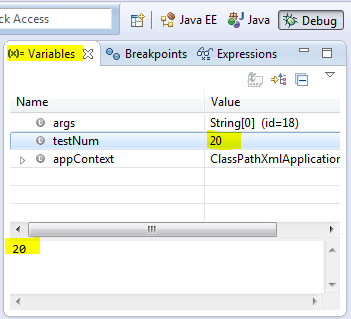
#### **4.评估/检查**

按Ctrl+Shift+D或者Ctrl+Shift+I来显示选定变量或者表达式的值。我们也可以给一个变量或表达式添加永久观察点，当程序在调试时，这些观察点就会在表达式视图（Expression view）中显示出来。



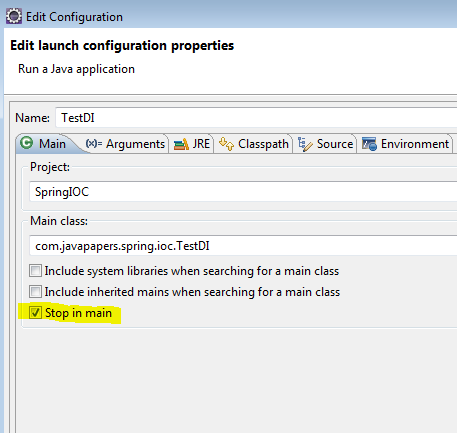
**5.修改变量值**

在调试过程中，我们可以修改变量值。先选好一个变量然后进入变量视图（Variables view），根据变量类型在其对应的Value列里输入值即可。



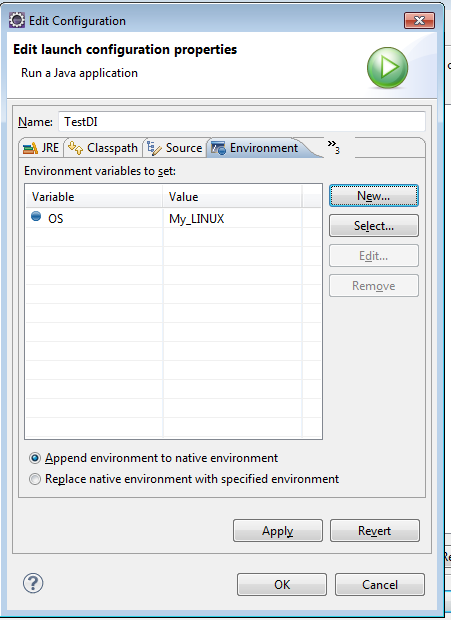
#### **6.在Main函数里面停止执行**

在运行/调试设置中，编辑配置对话框中有“Main”这个选项卡，我们可以勾选“Stop in main”这个复选框。如果选中，那么在调试一个基于main方法的Java程序时，程序会在main方法第一行位置便停止执行。



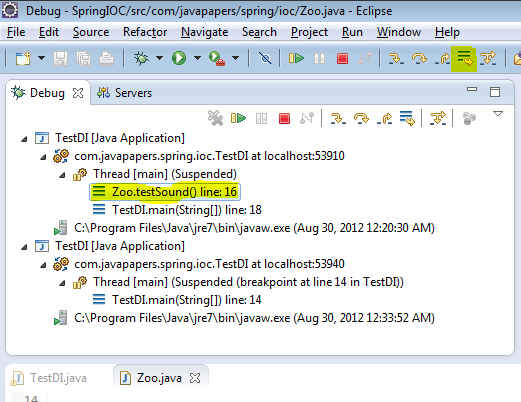
#### **7.环境变量**

并不是在系统属性中添加环境变量，我们可以在编辑配置对话框中很方便地进行添加。



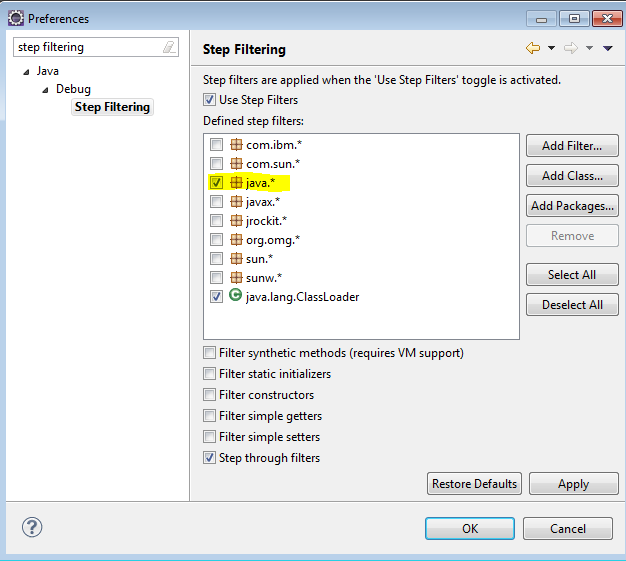
#### **8.Drop to Frame**

这也是我最喜欢的一个功能。调试期间，可以重新跳到调用堆栈框架的开始处执行，并且变量值也会回到最初。根据回档调整堆栈的深度，这个功能的主要用途是所有变量状态可以快速回到方法开始执行时候的样子，然后你可以重新进行一遍一遍执行，这样就可以在你关注的地方进行多次调试，但是在执行过程中也会产生一些副作用，比如插入到数据库里面的数据是无法删除的！



#### **9.分布过滤**

当我们进入（F5）方法的时候，我们还可以访问其外部库（比如java.\*），我们可能不需要这个库，就可以在Perference选项卡页面添加一个过滤器来排除这个包。



#### **10.进入、跳出和返回**

我把这个放在最后一点，在调试过程中，这些是必须要了解（最好掌握）的东西：

F5——进入：移动到下一个步骤，如果当前行有一个方法调用，该控件将会跳转到被调用方法的第一行执行。

F6——跳出：移动到下一行。如果在当前行有方法调用，那么会直接移动到下一行执行。不会进入被调用方法体里面。

F7——返回：从当前方法中跳出，继续往下执行。

F8——移动到下一个断点处执行。



原文链接：[javapapers](http://javapapers.com/core-java/top-10-java-debugging-tips-with-eclipse/)

### 配置

#### 1修改编码格式

如果要使新建立工程、java文件直接使UTF-8则需要做以下工作：   
1、windows->Preferences...打开"首选项"对话框，左侧导航树，导航到general->Workspace，右侧 Text file encoding，选择Other，改变为UTF-8，以后新建立工程其属性对话框中的Text file encoding即为UTF-8。

2、windows->Preferences...打开"首选项"对话框，左侧导航树，导航到general->Content Types，右侧Context Types树，点开Text，选择Java Source File，在下面的Default encoding输入框中输入UTF-8，点Update，则设置Java文件编码为UTF-8。其他java应用开发相关的文件 如：properties、XML等已经由Eclipse缺省指定，分别为ISO8859-1，UTF-8，如开发中确需改变编码格式则可以在此指定。

3、经过上述两步，新建java文件即为UTF-8编码，Eclipse编译、运行、调试都没问题，但是做RCP应用的Product输出时、或者 插件输出时，则总是出错，要么不能编译通过(输出时要重新compile)、要么输出的插件运行时中文显示乱码。此时需要再RCP应用、或插件 Plugin工程的build.properties中增加一行，javacDefaultEncoding.. = UTF-8。让输出时编译知道java源文件时UTF-8编码。这个设置需要保证所有的java源文件时UTF-8编码格式，如果不全是，可以参考 Eclipse帮中(Plug-in Development Environment Guide > Reference > Feature and Plug-in Build configuration)，建议全部java源文件是UTF-8编码。

如果插件开发、RCP应用开发原来基于其他编码，如GB18030，想转换为UTF-8，则首先，做以上工作；然后通过查找编码转换工具，如基于 iconv的批量转换工具，将原编码转换为UTF-8编码，注意只转换java源文件，其他类型文件可能已经是比较合适的编码了；将原工程属性中的 Text file encoding，从原编码改为UTF-8即可。

Java基础（Java SE:面向对象、多线程、IO、网络、数据库、常用类和API）

### 关于配置环境变量

1. 安装JDK即可，然后添加JAVA\_HOME：C:\Program Files\Java\jdk1.7.0\_25（JDK路径），注意，前后不能有“;”；添加classpath：%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar;；添加Path路径： %JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;

### 利用javadoc生成代码文档(2013.12.26)

#### 1.1简介

代码文档时编码的一个重要部分，很多时候并不是把代码写好了就可以了，还要写文档才行。如果代码和文档分开（代码写在一块，文档写在其他地方），文档维护起来就非常困难。利用javadoc就可以很好地解决代码文档问题，Java提供了一种特殊的**注释语法**来标记文档，和一个**工具**来提取那些注释(哪有那么复杂，不就是定义了一些规则，然后就按照规则解析文档并生成一个Html而已)。

#### 1.2注释语法(按照这种语法写的注释就可以生成文档)

格式：/\*\* 1.注释写在星号的右边

\* 2.内容可以直接是文本，也可以是@xxx标签(自己@一下就看到了)更可以是HTML格式的文本，会被解析哦。

\* 3.类、属性、方法可以进行文档注释，在头顶加上这么一段就可以了（public和protected，private需要用-private标记才行）

\*4.需要生成文档就javadoc -subpackages mypackage –d ../../docs -docencoding GBK -charset GBK(看不懂就试试就可以了，实在不行可以看看1.3又长又臭的介绍)

\*/

#### 1.3 使用javadoc输出源文件中的java文档（带有javadoc命令的参数）

在程序设计中总会有很多的工作花在写开发文档上，其中程序代码的注释又是比重很大的一块。我们不会拿着源代码四处散播，也没有人愿意看长长的复杂的代码。于是就让我们看浏览简单清晰的API文档吧。在JAVA中提供了一个工具javadoc来帮助生成文档，在java源文件中类、属性、方法的注释都可以生成。

　　在命令行中输入javadoc -help 将会看到下面的帮助：

用法：javadoc [选项] [软件包名称] [源文件] [@file]  
-overview <文件>          读取 HTML 文件的概述文档  
-public                   仅显示公共类和成员  
-protected                显示受保护/公共类和成员（默认）  
-package                  显示软件包/受保护/公共类和成员  
-private                  显示所有类和成员  
-help                     显示命令行选项并退出  
-doclet <类>              通过替代 doclet 生成输出  
-docletpath <路径>        指定查找 doclet 类文件的位置  
-sourcepath <路径列表>    指定查找源文件的位置  
-classpath <路径列表>     指定查找用户类文件的位置  
-exclude <软件包列表>     指定要排除的软件包的列表  
-subpackages <子软件包列表> 指定要递归装入的子软件包  
-breakiterator            使用 BreakIterator 计算第 1 句  
-bootclasspath <路径列表> 覆盖引导类加载器所装入的  
     类文件的位置  
-source <版本>            提供与指定版本的源兼容性  
-extdirs <目录列表>       覆盖安装的扩展目录的位置  
-verbose                  输出有关 Javadoc 正在执行的操作的消息  
-locale <名称>            要使用的语言环境，例如 en\_US 或 en\_US\_WIN  
-encoding <名称>          源文件编码名称  
-quiet                    不显示状态消息  
-J<标志>                  直接将 <标志> 传递给运行时系统

通过标准 doclet 提供：  
-d <目录>                         输出文件的目标目录  
-use                              创建类和软件包用法页面  
-version                          包含 @version 段  
-author                           包含 @author 段  
-docfilessubdirs                  递归复制文档文件子目录  
-splitindex                       将索引分为每个字母对应一个文件  
-windowtitle <文本>               文档的浏览器窗口标题  
-doctitle <html 代码>             包含概述页面的标题  
-header <html 代码>               包含每个页面的页眉文本  
-footer <html 代码>               包含每个页面的页脚文本  
-bottom <html 代码>               包含每个页面的底部文本  
-link <url>                       创建指向位于 <url> 的 javadoc 输出的链接  
-linkoffline <url> <url2>         利用位于 <url2> 的软件包列表链接至位于 <url> 的文档  
-excludedocfilessubdir <名称 1>:..排除带有给定名称的所有文档文件子目录。  
-group <名称> <p1>:<p2>..         在概述页面中，将指定的软件包分组  
-nocomment                        抑止描述和标记，只生成声明。  
-nodeprecated                     不包含 @deprecated 信息  
-noqualifier <名称 1>:<名称 2>:...从输出中排除限定符的列表。  
-nosince                          不包含 @since 信息  
-notimestamp                      不包含隐藏时间戳  
-nodeprecatedlist                 不生成已过时的列表  
-notree                           不生成类分层结构  
-noindex                          不生成索引  
-nohelp                           不生成帮助链接  
-nonavbar                         不生成导航栏  
-serialwarn                       生成有关 @serial 标记的警告  
-tag <名称>:<位置>:<标题>         指定单个变量自定义标记  
-taglet                           要注册的 Taglet 的全限定名称  
-tagletpath                       Taglet 的路径  
-charset <字符集>                 用于跨平台查看生成的文档的字符集。  
-helpfile <文件>                  包含帮助链接所链接到的文件  
-linksource                       以 HTML 格式生成源  
-sourcetab <制表符长度>           指定源中每个制表符占据的空格数  
-keywords                         使软件包、类和成员信息附带 HTML 元标记  
-stylesheetfile <路径>            用于更改生成文档的样式的文件  
-docencoding <名称>               输出编码名称  
　　于是，安装帮助的说明自己来生成代码吧。下面的例子生成了指定包以及子包的所有文档，而且指定了输出编码与html文件编码。

javadoc -subpackages mypackage -d ../../docs -docencoding GBK -charset GBK

 　　其中的各个选项不用我再说了，查看上面的帮助就一目了然了。将API文档压缩打包，或者上传到网上，分享自己的java类库，让更多人能够很容易的学习与掌握更多的java类库。

### 2.目录操作：File类、FilenameFilter接口(2013.12.26)

#### 2.1 File类(别误以为是文件，叫FilePath更准确)

File既可以是一个特定文件的名称，也可以是一个目录下的一组文件的名称，用list()方法获得，返回的是一个数组，注意是数组。

File类与FilenameFilter接口，调用File类的list()函数时，可以传递一个FilenameFilter对象，用于筛选符合某个正则表达式的文件。

Class DirFilter implements FilenameFilter{

Private Pattern pattern;

Public DirFilter(String regex){pattern = Pattern.compile(regex);}

//必须，该接口只有这么一个方法，没有其他任何属性或方法。

Public boolean accept(File dir,String name){return pattern.matcher(name).matches();}

}

调用：String[] array = file.list(new DirFilter(“正则表达式”));//得到的是符合某个正则表达式的文件或目录**名(包括路径，如：D:\dir\dir\text.java)**

### 3.关于编码风格

#### 3.1 Java的编码风格

**命名风格：**Java编程语言编码约定中，代码风格是这样规定的：类名的首字母要大写，如果由多个字母组成，那么把他们并在一起，每个单词首字母大写；方法、字段、对象引用名等风格一样，只是首字母小写。（既然Java是这样的，那就入乡随俗，摈弃C#的方法名首字母大写）

**关于大括号：**一下仅是个人观点，Java中大括号第一个通常是和紧跟代码后面的，写惯C#的人肯定很不习惯，所以，还是用回C#的风格，大括号另起一行。

### 4. 在2中讲到的正则表达式

### 5.文件操作

#### 5.1 简述

Java类库中的I/O类分成输入（Inputstream）和输出（OutputStream）两部分，很少使用单一的类，多数使用装饰者模式，即需要多几个类来包装。

#### 5.2 InputStream类型

|  |  |  |
| --- | --- | --- |
| 类 | 功能 | 使用方法 |
| ByteArrayInputStream |  |  |
| StringBufferInnputStream |  |  |
| FileInputStream | 用于从**文件**中读取信息 |  |
| PipedInputStream | 管道化 |  |
| SequenceInputStream | 将多个InputStream转化为一个 |  |
| FilterInputStream | 抽象类，作为**装饰者**的接口，装饰者为其他InputStream类提供有用功能 |  |

#### 5.3 OutputStream类型

|  |  |  |
| --- | --- | --- |
| 类 | 功能 | 使用方法 |
| ByteArrayOutputStream | 在内存中创建缓冲区，所有送往“流”的数据都要放置在次缓冲区 |  |
| FileOutputStream | 用于将信息写至文件 |  |
| PipedOutputStream |  |  |
| FilterOutputStream | 抽象类，作为**装饰者**的接口，装饰者为其他OutputStream类提供有用功能。 |  |

#### 5.4 装饰者接口及类：FilterInputStream、FilterOutputStream

##### 5.4.1 描述

分别继承了InputStream和OutputStream。

##### 5.4.2 FilterInputStream类型

|  |  |  |
| --- | --- | --- |
| 类 | 功能 | 构造器参数 |
| DataInputStream | 与DataOutputStream搭配使用，因此可以按照可移植方式从流读取基本数据类型 | InputStream |
| BufferInputStream | 代表使用缓冲区，可防止每次读写时都得进行实际写操作（**为什么呢**） |  |
| LineNuberInputStream | 跟踪输入流的行号；可调用getLineNumber()和setLineNumber(int) |  |
| PushbackInputStream | 具有“能弹出一个字节的缓冲区”，因此可以将读到的最后的一个字符回退 |  |

##### 5.4.3 FilterOutputStream类型

|  |  |  |
| --- | --- | --- |
| 类 | 功能 | 构造器参数 |
| DataOutputStream | 与DataInputStream搭配使用，因此可以按照可移植方式从流读取基本数据类型 | OutputStream |
| PrintStream | 用于生产格式化输出。其中DataOutputStrea处理数据存储，PrintStream处理显示 |  |
| BufferedOutputStream | 使用它可以避免每次发送数据时都要进行实际的写操作。代表使用缓冲区，调用flush()清空缓冲区 |  |

#### 5.5 Reader和Writer

##### 5.5.1 简介

InputStream和OutputStream在面向字节形式的I/O中提供极有价值的功能，Reader和Writer则提供兼容Unicode和面向字符的I/O功能。几乎所有的原始I/O流类都有对应的Reader和Writer类来提供天然的Unicode操作。

##### 5.5.2 与InputStream/OutputStream对比

|  |  |  |  |
| --- | --- | --- | --- |
| In/OutStream | 对应Reader/Writer | In/OutStream | 对应Reader/Writer |
| InputStream | Reader | ByteArrayInputStream | CharArrayReader |
| OutputStream | Writer | ByteArrayOutputStream | CharArrayWriter |
| FileInputStream | FileReader | PipedInputStream | PipedReader |
| FileOutputStream | FileWriter | PipedOutputStream | PipedWriter |
| 无对应类 | StringWriter |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 过滤器Filter | Reader/Writer |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### 5.6 自我独立的类：RandomAccessFile

#### 5.7 利用FileInputStream、FileInputReader和BufferedReader缓冲输入文件

##### 5.7.1 解决中文乱码问题

BufferedReader br=new BufferedReader(new InputStreamReader(new FileInputStream(fileName),"UTF-8"));

br.readLine();

完整代码：

**public** **static** String read(String filename) **throws** IOException

{

FileInputStream fileInputStream = **new** FileInputStream(filename);

InputStreamReader isd = **new** InputStreamReader(fileInputStream,"UTF-8");// 指定编码形式：UTF-8

BufferedReader bufferedReader = **new** BufferedReader(isd);

String s;

StringBuilder sb = **new** StringBuilder();

**while**((s = bufferedReader.readLine()) != **null**)

{

sb.append(s + "\n");

}

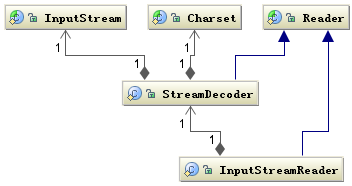
bufferedReader.close();

**return** sb.toString();

}

Java的I/O类处理如图：

Reader 类是 Java 的 I/O 中读字符的父类，而 InputStream 类是读字节的父类，InputStreamReader 类就是关联字节到字符的桥梁，它负责在 I/O 过程中处理读取字节到字符的转换，而具体字节到字符的解码实现它由 StreamDecoder 去实现，在 StreamDecoder 解码过程中必须由用户指定 Charset 编码格式。值得注意的是如果你没有指定 Charset，将使用本地环境中的默认字符集，例如在中文环境中将使用 GBK 编码。



                                    Java的I/O类处理图

**总结**：Java读取数据流的时候，一定要指定数据流的编码方式，否则将使用本地环境中的默认字符集。

### 6.数据库操作

#### 6.1JDBC连接MySQL

##### 6.1.1JDBC简介

JDBC是Sun公司制定的一个可以用Java语言连接数据库的技术。

**一、JDBC基础知识**

        JDBC（Java Data Base Connectivity,java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。JDBC为数据库开发人员提供了一个标准的API，据此可以构建 更高级的工具和接口，使数据库开发人员能够用纯 Java API 编写数据库应用程序，并且可跨平台运行，并且不受数据库供应商的限制。

**1、跨平台运行：**这是继承了Java语言的“一次编译，到处运行”的特点；

**2、不受数据库供应商的限制：**巧妙在于JDBC设有两种接口，一个是面向应用程序层，其作用是使得开发人员通过 SQL调用数据库和处理结果，而不需要考虑数据库的提供商；另一个是驱动程序层，处理与具体驱动程序的交互，JDBC驱动程序可以利用JDBC API创建Java程序和数据源之间的桥梁。应用程序只需要编写一次，便可以移到各种驱动程序上运行。Sun提供了一个驱动管理器，数据库供应商——如 MySQL、Oracle，提供的驱动程序满足驱动管理器的要求就可以被识别，就可以正常工作。所以JDBC不受数据库供应商的限制。

        JDBC API可以作为连接Java应用程序与各种关系数据库的纽带，在带来方便的同时也有负面影响，以下是JDBC的优、缺点。**优点如下：**

* **操作便捷：**JDBC使得开发人员不需要再使用复杂的驱动器调用命令和函数；
* **可移植性强：**JDBC支持不同的关系数据库，所以可以使同一个应用程序支持多个数据库的访问，只要加载相应的驱动程序即可；
* **通用性好：**JDBC-ODBC桥接驱动器将JDBC函数换成ODBC；
* **面向对象：**可以将常用的JDBC数据库连接封装成一个类，在使用的时候直接调用即可。

**缺点如下：**

* 访问数据记录的速度受到一定程度的影响；
* 更改数据源困难：JDBC可支持多种数据库，各种数据库之间的操作必有不同，这就给更改数据源带来了很大的麻烦

##### 6.1.2JdbC连接数据库的流程和原理

**1、**在开发环境中加载指定数据库的驱动程序。例如，接下来的实验中，使用的数据库是MySQL，所以需要去下载MySQL支持JDBC的驱动程序(最新的是：mysql-connector-java-5.1.18-bin.jar)；而开发环境是MyEclipse，将下载得到的驱动程序加载进开发环境中(具体示例的时候会讲解如何加载)。

**2、**在Java程序中加载驱动程序。在Java程序中，可以通过 “Class.forName(“指定数据库的驱动程序”)” 方式来加载添加到开发环境中的驱动程序，例如加载MySQL的数据驱动程序的代码为：  Class.forName(“com.mysql.jdbc.Driver”)

**3、**创建数据连接对象：通过DriverManager类创建数据库连接对象Connection。DriverManager类作用于Java程序和JDBC驱动程序之间，用于检查所加载的驱动程序是否可以建立连接，然后通过它的getConnection方法，根据数据库的URL、用户名和密码，创建一个JDBC Connection 对象。如：Connection connection =  DriverManager.geiConnection(“连接数据库的URL", "用户名", "密码”)。其中，URL=协议名+IP地址(域名)+端口+数据库名称；用户名和密码是指登录数据库时所使用的用户名和密码。具体示例创建MySQL的数据库连接代码如下：

              Connection connectMySQL  =  DriverManager.geiConnection(“jdbc:mysql://localhost:3306/myuser","root" ,"root" );

**4、**创建Statement对象：Statement 类的主要是用于执行静态 SQL 语句并返回它所生成结果的对象。通过Connection 对象的 createStatement()方法可以创建一个Statement对象。例如：Statement statament = connection.createStatement(); 具体示例创建Statement对象代码如下：

             Statement statamentMySQL =connectMySQL.createStatement();

**5、**调用Statement对象的相关方法执行相对应的 SQL 语句：通过execuUpdate()方法用来数据的更新，包括插入和删除等操作，例如向staff表中插入一条数据的代码：

       statement.excuteUpdate( "INSERT INTO staff(name, age, sex,address, depart, worklen,wage)" + " VALUES ('Tom1', 321, 'M', 'china','Personnel','3','3000' ) ") ;

通过调用Statement对象的executeQuery()方法进行数据的查询，而查询结果会得到 ResulSet对象，ResulSet表示执行查询数据库后返回的数据的集合，ResulSet对象具有可以指向当前数据行的指针。通过该对象的 next()方法，使得指针指向下一行，然后将数据以列号或者字段名取出。如果当next()方法返回null，则表示下一行中没有数据存在。使用示例代 码如下：

       ResultSet resultSel = statement.executeQuery( "select \* from staff" );

**6、**关闭数据库连接：使用完数据库或者不需要访问数据库时，通过Connection的close() 方法及时关闭数据连接。

##### 6.1.3连接实例

1.安装MySQL数据库，自己安装，这里不多啰嗦。

2.(命令行程序)创建数据，创建数据表

Create database JavaDB;

create table student

(

sno char(12) PRIMARY KEY,

sname char(10) NOT NULL,

sdeptno char(20) NOT NULL,

sid char(18) NOT NULL

);

INSERT INTO student(sno,sname,sdeptno,sid)

VALUES('201005090212','胡盛海','地物','440921199008043819');

1. 下载MySQL支持JDBC的驱动程序（如果已经有了，可跳过这一步）。前往MySQL[官网](http://www.mysql.com/products/connector/)(<http://www.mysql.com/products/connector/>)下载驱动程序,，MySQL针对不同的平台提供了不同的连接器，我们需要的是DBC Driver for MySQL (Connector/J)，如下图所示，点击 Download 跟着网站的引导进行下载。打开下载得到的压缩包（mysql-connector-java-5.1.18.zip），将其中的Java包（mysql- connector-java-5.1.18-bin.jar），复制到MySQL目录下(仅是为了方便才放到这里，随便放哪里都可以，到时候将其添加到程序中即可)，以备加载驱动程序时使用。
2. 在Eclipse中创建Java Project，添加mysql- connector-java-5.1.18-bin.jar包，在资源浏览器中，右键JRE System Library或者Referenced Libraries，子菜单选择Build Path，Configure Build Path，弹出窗体选择右边的Add External JARS，然后找到mysql- connector-java-5.1.18-bin.jar包，添加即可。
3. 代码如下（创建类MySqlDB，然后将代码粘贴过去即可，当然，数据库、表、用户密码什么的要是自己的）：

package databaseHelper;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class MySqlDB

{

// 创建静态全局变量

static Connection conn;

static Statement st;// 创建用于执行静态sql语句的Statement对象

private static String forName = "com.mysql.jdbc.Driver";

private static String uri = "jdbc:mysql://localhost:3306/JavaDB";

private static String user = "root";

private static String pwd = "8625809";

public static Connection getConnection()

{

Connection con = null; //创建用于连接数据库的Connection对象

try

{

Class.forName(forName);// 加载Mysql数据驱动

con = DriverManager.getConnection(uri, user, pwd);// 创建数据连接

}

catch (Exception e)

{

System.out.println("数据库连接失败" + e.getMessage());

}

return con; //返回所建立的数据库连接

}

/\* 插入数据记录，并输出插入的数据记录数\*/

public static void insert(String sqlstr)

{

conn = getConnection(); // 首先要获取连接，即连接到数据库

try

{

//String sql = sqlstr;// 插入数据的sql语句

st = (Statement) conn.createStatement(); // 创建用于执行静态sql语句的Statement对象

int count = st.executeUpdate(sqlstr); // 执行插入操作的sql语句，并返回插入数据的个数

System.out.println("向student表中插入 " + count + " 条数据"); //输出插入操作的处理结果

conn.close(); //关闭数据库连接

}

catch (SQLException e)

{

System.out.println("插入数据失败" + e.getMessage());

}

}

/\* 更新符合要求的记录，并返回更新的记录数目\*/

public static void update(String sqlstr)

{

conn = getConnection(); //同样先要获取连接，即连接到数据库

try

{

//String sql = "update student set sid='440921199008043819' where sname = '胡盛海'";// 更新数据的sql语句

st = (Statement) conn.createStatement(); //创建用于执行静态sql语句的Statement对象，st属局部变量

int count = st.executeUpdate(sqlstr);// 执行更新操作的sql语句，返回更新数据的个数

System.out.println("student表中更新 " + count + " 条数据"); //输出更新操作的处理结果

conn.close(); //关闭数据库连接

}

catch (SQLException e)

{

System.out.println("更新数据失败");

}

}

/\* 查询数据库，输出符合要求的记录的情况\*/

public static void query(String sqlstr)

{

conn = getConnection(); //同样先要获取连接，即连接到数据库

try

{

//String sql = sqlstr; // 查询数据的sql语句

st = (Statement) conn.createStatement(); //创建用于执行静态sql语句的Statement对象，st属局部变量

ResultSet rs = st.executeQuery(sqlstr); //执行sql查询语句，返回查询数据的结果集

System.out.println("最后的查询结果为：");

while (rs.next())

{

String name = rs.getString("sname");

String sno = rs.getString("sno");

String sid = rs.getString("sid");

String depart = rs.getString("sdeptno");

//输出查到的记录的各个字段的值

System.out.println(name + " " + " " + sno + " " + sid + " " + depart + " " );

}

conn.close(); //关闭数据库连接

}

catch (SQLException e)

{

System.out.println("查询数据失败");

}

}

/\* 删除符合要求的记录，输出情况\*/

public static void delete(String sqlstr)

{

conn = getConnection(); //同样先要获取连接，即连接到数据库

try

{

//String sql = "delete from student where sname = '唐长江'";// 删除数据的sql语句

st = (Statement) conn.createStatement(); //创建用于执行静态sql语句的Statement对象，st属局部变量

int count = st.executeUpdate(sqlstr);// 执行sql删除语句，返回删除数据的数量

System.out.println("student表中删除 " + count + " 条数据\n"); //输出删除操作的处理结果

conn.close(); //关闭数据库连接

}

catch (SQLException e)

{

System.out.println("删除数据失败");

}

}

/\* 获取数据库连接的函数\*/

public static void main(String[] args)

{

insert("INSERT INTO student(sno,sname,sdeptno,sid)" + " VALUES('201005090214','唐长江','地物','440921199008043817')"); //插入添加记录

query("select \* from student"); //查询记录并显示

update("update student set sid='440921199008043819' where sname = '胡盛海'"); //更新记录数据

delete("delete from student where sname = '唐长江'"); //删除记录

query("select \* from student"); //查询记录并显示

}

}

## Java Web(JDBC、DAO、Servlet、表单、Jsp、JSTL、Tomcat配置)

### 1.JSP

#### 1.1HTML+CSS

#### 1.2JSP内置对象

#### 1.3JSP内置对象

#### 1.4JSP指令

#### 1.5JavaBean

### 2.Servlet

#### 问题及解答

##### web.xml

主要是Servlet的配置，注意Servlet类的@WebServlet("/MyUniqueServlet")这个标签，再注意一下servlet-class（相当于import）和url-pattern(相当于文件路径)两个标签的含义即可，给一个完整例子：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app>

<servelt>

<servlet-name>MyUniqueServlet</servlet-name>

<servlet-class>com.hello915.javaweb.servlets.MyUniqueServlet</servlet-class>

</servelt>

<servlet-mapping>

<servlet-name>MyUniqueServlet</servlet-name>

<url-pattern>/com/hello915/javaweb/servlets/MyUniqueServlet</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

##### JDBC包问题

需要用到数据库如postgresql数据库，就要用到postgresql-9.3-1100.jdbc41.jar这么一个包，如果没有将该jar包放对位置并且配置好环境路径就会一直提示找不到合适的jdbc什么的。

解决：将该jar包放在jre的lib\ext目录下，并在环境变量classPath中添加该包（关于环境变量的配置请看前面EclipseIDE使用技巧相关章节）。至于其他引用的包，好像没有这个要求，如GSON直接放在lib目录下再添加引用即可(jdbc包不引用会怎么样呢???会不会没问题)

##### 通过Servlet插入数据库出现中文乱码问题

问题分析：

* 1. 确认Eclipse编辑器使用的是UTF-8编码格式
  2. 通过URLEncoder编码URLEncoder.*encode*("张灿烂","utf-8")再发送到Servlet
  3. Servlet通过request.getParameter("name")获取参数，打印出来是乱码
  4. 获得参数后通过JDBC插入数据库
  5. 在pgAdmin里查看时乱码（确认建库使用UTF-8编码格式）
  6. request.setCharacterEncoding("UTF-8");后正常，但再写回response时又乱码

问题解答：（参考<http://blog.csdn.net/xiazdong/article/details/7217022>）

1. 先设置request.setCharacterEncoding("UTF-8");
2. 再设置response.setContentType("text/html;charset=UTF-8");
3. response.setCharacterEncoding("UTF-8");
4. 总结，就是字符的编码问题，弄清楚中间的关系就可以了。
5. 添加一条：以上代码只有在Servlet的post方法里设置才有效，传递到其他函数再设置则无效

#### 2.1Eclipse下第一个成功Servlet程序样例步骤

##### 2.1.1新建一个动态网页程序(File->New->Dynamic Web Project)

##### 2.1.2创建Servlet类

在Java Resources文件夹中新建一个Servlet类(可以放在包里，配置时记得路径即可)

创建过程中可以勾选一些必要的要实现的方法，如doGet，doPost以及要实现的接口HttpServlet等。然后在doGet或者doPost方法中编写要实现的功能，如输出一个Hello World，也可以写一个Xml文档(实习公司中的项目就是这么干的)，然后这些内容就会返回给调用者。

##### 2.1.3在WebContent中的Web-INF文件夹添加web.xml配置文件

格式如下(注意：HelloWorld Servlet是在default包中的，如果在其他包请指明路径)：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app>

<servelt>

<servlet-name>HelloWorld</servlet-name>

<servlet-class>HelloWorld</servlet-class>

</servelt>

<servlet-mapping>

<servlet-name>HelloWorld</servlet-name>

<url-pattern>/HelloWorld</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>DoGetDemo.jsp</welcome-file>

</welcome-file-list>

</web-app>

##### 2.1.4创建测试页面

新建一个Jsp页面，然后在表单中指定action=”要调用的servlet的URI”，method=”get/post”，然后再添加一个input类型为submit，这样运行时(注意web.xml中要指定启动页面)单击提交就会调用该Servlet。

#### 2.2调用Servlet的方法

##### 2.2.1通过JSP页面调用Servlet(表单的action和method属性)

<form action=”Servlet地址” method=”get/post”></form>

##### 2.2.2通过连接调用Servlet

<a href=”地址?parameter1=value1”></a>

#### 2.3处理get和post

doGet()处理get请求，doPost()处理post请求，注意get和post请求处理起来是没什么区别的，但是，get请求在调用时能看到参数(地址+参数)，且参数数目有限制，好似是512字节还是多少来的，而post请求是看不到参数的，且参数长度无限制，是安全的。应该根据实际情况而选择哪种请求。

#### 2.4Servlet操作文件(读取写入和文件操作时一样的)

#### 2.5Servlet下载文件

#### 2.6上传文件

commons-io下载地址：http://commons.apache.org/io/download\_io.cgi

common-fileupload组件是apache的一个开源项目之一，可以从<http://jakarta.apache.org/commons/fileupload/>下载。

该组件简单易用，可实现一次上传一个或多个文件，并可限制文件大小。  
下载后解压zip包，将commons-fileupload.jar,和commons-io里面后缀为jar复制到你的项目的webapp\WEB-INF\lib\下，如果目录不存在请自建目录。

这个项目是用来上传文件，文件路径为workspace\项目名称\build\weboutput\file\项目下，如果没有该文件夹请创建一个。否则会发生找不到路径的情况  
import java.io.File;  
import java.io.IOException;  
import java.io.PrintWriter;  
import java.util.Iterator;  
  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
  
import org.apache.commons.fileupload.FileItem;  
import org.apache.commons.fileupload.FileItemFactory;  
import org.apache.commons.fileupload.disk.DiskFileItemFactory;  
import org.apache.commons.fileupload.servlet.ServletFileUpload;  
  
/\*\*  
 \* Servlet implementation class FileUpload  
 \*/  
public class FileUpload extends HttpServlet {  
    private static final long serialVersionUID = 1L;  
         
    /\*\*  
     \* @see HttpServlet#HttpServlet()  
     \*/  
    public FileUpload() {  
        super();  
        // TODO Auto-generated constructor stub  
    }  
  
    /\*\*  
     \* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  
     \*/  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        // TODO Auto-generated method stub  
    }  
  
    /\*\*  
     \* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  
     \*/  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        // TODO Auto-generated method stub  
        //判断提交过来的表单是否为文件上传菜单   
        boolean isMultipart= ServletFileUpload.isMultipartContent(request);  
        if(isMultipart){  
            //构造一个文件上传处理对象  
            FileItemFactory factory = new DiskFileItemFactory();  
            ServletFileUpload upload = new ServletFileUpload(factory);  
              
            Iterator    items;  
            try{  
                //解析表单中提交的所有文件内容  
                items=upload.parseRequest(request).iterator();  
                while(items.hasNext()){  
                    FileItem item = (FileItem) items.next();  
                    if(!item.isFormField()){  
                        //取出上传文件的文件名称  
                        String name = item.getName();  
                        //取得上传文件以后的存储路径  
                        String fileName=name.substring(name.lastIndexOf('\\')+1,name.length());  
                        //上传文件以后的存储路径  
                        String path= request.getRealPath("file")+File.separatorChar+fileName;  
                          
                        //上传文件  
                        File uploaderFile = new File(path);  
                        item.write(uploaderFile);  
                        //打印上传成功信息  
                        response.setContentType("text/html");  
                        response.setCharacterEncoding("GB2312");  
                        PrintWriter out = response.getWriter();  
                          
                        out.print("<font size='2'>上传文件为:"+name+"<br>保存的地址为"+path+ "</font>");  
                          
                          
                    }  
                }  
            }catch(Exception e){  
                e.printStackTrace();  
            }  
              
              
        }   
          
    }

}

#### 2.7获取请求信息头部内容

#### 2.8获取请求信息

#### 2.9获取参数信息

#### 2.10JSP页面与Servlet以及Servlet与JSP页面间传递参数

##### 2.10.1问题描述

很多时候并不是简简单单，调用一下Servlet然后Servlet负责往回写写东西如：Hello World，就满足要求了，调用Servlet往往是要查询数据库等等，然后将结果传递到另一个页面（也包括本页面），这就涉及到页面与Servlet间的参数传递。

##### 2.10.2网络解答

[**11. jsp与servlet之间页面跳转及参数传递实例**](http://blog.csdn.net/ssy_shandong/article/details/9328985)

分类： [Java Web](http://blog.csdn.net/ssy_shandong/article/category/1503925) 2013-07-17 16:24 637人阅读 [评论](http://blog.csdn.net/ssy_shandong/article/details/9328985#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/ssy_shandong/article/details/9328985#report)

[jsp servlet传递参数](http://www.csdn.net/tag/jsp%20servlet%e4%bc%a0%e9%80%92%e5%8f%82%e6%95%b0)

目录[(?)[+]](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. [从一个jsp页面跳转到另一个jsp页面时的参数传递](http://blog.csdn.net/ssy_shandong/article/details/9328985#t0)
2. [jsp页面传递参数给servlet](http://blog.csdn.net/ssy_shandong/article/details/9328985#t1)
3. [从servlet到jsp页面中传递参数](http://blog.csdn.net/ssy_shandong/article/details/9328985#t2)
4. [从一个servlet到另一个servlet时的参数传递](http://blog.csdn.net/ssy_shandong/article/details/9328985#t3)

**1.从一个jsp页面跳转到另一个jsp页面时的参数传递**

    (1)使用request对象获取客户端提交的信息

    login.jsp页面代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <form action="getinfo.jsp">
7. <input type="text" name="name">
8. <input type="submit" value="Enter" name="submit">
9. </form>
10. </body>
11. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<form action="getinfo.jsp">

<input type="text" name="name">

<input type="submit" value="Enter" name="submit">

</form>

</body>

</html>

    getinfo.jsp页面代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <p>获取文本框提交的信息：
7. <%
8. String textContent = request.getParameter("name");
9. %>
10. <br/>
11. <%=textContent%>
13. </p>获取按钮的名字：
14. <%
15. String buttonName = request.getParameter("submit");
16. %>
17. <br/>
18. <%=buttonName%>
20. </body>
21. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<p>获取文本框提交的信息：

<%

String textContent = request.getParameter("name");

%>

<br/>

<%=textContent%>

</p>获取按钮的名字：

<%

String buttonName = request.getParameter("submit");

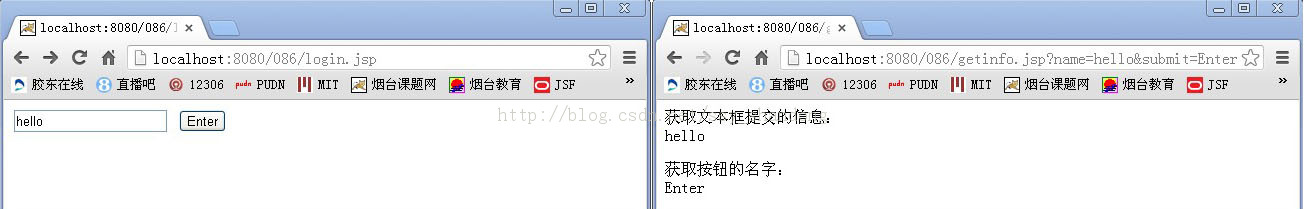
%>

<br/>

<%=buttonName%>

</body>

</html>



    (2)使用session对象获取从客户端提交的信息

    session1.jsp页面代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <form action="session2.jsp" method="post">
7. 请输入您的姓名：
8. <input type="text" name="thename">
9. <input type="submit" value="submit">
10. </form>
11. </body>
12. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<form action="session2.jsp" method="post">

请输入您的姓名：

<input type="text" name="thename">

<input type="submit" value="submit">

</form>

</body>

</html>

    session2.jsp通过request对象提取session1.jsp表单中的thename值,并将其保存到session对象中，session2.jsp代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <%
7. String name = request.getParameter("thename");
8. session.setAttribute("thename",name);
9. %>
10. 您的姓名是：<%=name%>
11. <p>
12. <form action="session3.jsp?pwd=123456" method="post">
13. 您喜欢吃什么？
14. <input type="text" name="food"/>
15. <p>
16. <input type="submit" value="submit">
17. </form>
18. </body>
19. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<%

String name = request.getParameter("thename");

session.setAttribute("thename",name);

%>

您的姓名是：<%=name%>

<p>

<form action="session3.jsp?pwd=123456" method="post">

您喜欢吃什么？

<input type="text" name="food"/>

<p>

<input type="submit" value="submit">

</form>

</body>

</html>

    session3.jsp的主要任务是显示结果，它从session对象提取thename的值并显示它，以证明在多个页面中可以通过session对象存储和维持信息，session3.jsp代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <pre class="java" name="code"><%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <%
7. String food = request.getParameter("food");
8. String name = (String)session.getAttribute("thename");
9. String pwd = request.getParameter("pwd");
10. %>
11. 您的姓名是：<%=name%>
12. <p>
13. 您喜欢吃：<%=food%>
14. <p>
15. 您的密码是：<%=pwd%>
16. </body>
17. </html></pre>
18. <pre></pre>
19. <pre></pre>
20. <pre></pre>
21. <pre></pre>
22. <pre></pre>
23. <pre></pre>
24. <pre></pre>

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <%
7. String food = request.getParameter("food");
8. String name = (String)session.getAttribute("thename");
9. String pwd = request.getParameter("pwd");
10. %>
11. 您的姓名是：<%=name%>
12. <p>
13. 您喜欢吃：<%=food%>
14. <p>
15. 您的密码是：<%=pwd%>
16. </body>
17. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<%

String food = request.getParameter("food");

String name = (String)session.getAttribute("thename");

String pwd = request.getParameter("pwd");

%>

您的姓名是：<%=name%>

<p>

您喜欢吃：<%=food%>

<p>

您的密码是：<%=pwd%>

</body>

</html>

    程序运行结果如图2所示：



    (3)利用隐藏域传递数据

    使用隐藏空间记录用户提交日期的hiddenindex.jsp代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <form action="hidden.jsp" method="post" name="form1">
7. <!-- 隐藏字段，系统时间，并转换为毫秒数 -->
8. <input type="hidden" name="time" value="<%=new java.util.Date()%>">
9. <p>
10. 请输入您的姓名：<input type="text" name="name">
11. </p>
12. <input type="submit" name="submit" value="submit">
13. </form>
14. </body>
15. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<form action="hidden.jsp" method="post" name="form1">

<!-- 隐藏字段，系统时间，并转换为毫秒数 -->

<input type="hidden" name="time" value="<%=new java.util.Date()%>">

<p>

请输入您的姓名：<input type="text" name="name">

</p>

<input type="submit" name="submit" value="submit">

</form>

</body>

</html>

     显示隐藏空间信息的hidden.jsp代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <%
7. String name = request.getParameter("name");
8. String date = request.getParameter("time");
9. %>
10. <p>您的姓名是：
11. <%=name %>
12. </p>
13. <p>提交时的系统时间为：
14. <%=date %>
15. </p>
16. </body>
17. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<%

String name = request.getParameter("name");

String date = request.getParameter("time");

%>

<p>您的姓名是：

<%=name %>

</p>

<p>提交时的系统时间为：

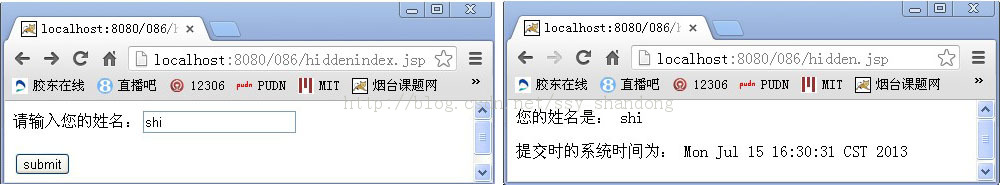
<%=date %>

</p>

</body>

</html>

    程序运行结果如下：



    使用隐藏域的一个缺点是用户可通过查看网页源程序读到隐藏域的值。

    (4)通过超链接传递数据

    建立链接到其他分页的超级链接page.jsp代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <%
5. int PageCount = 15;
6. %>
7. <html>
8. <body>
9. <table>
10. <tr>
11. <td>
12. <a href=showpage.jsp?ToPage=<%=1%>>到第1页</a>
13. </td>
14. <td>
15. <a href=showpage.jsp?ToPage=<%=PageCount%>>到最后一页</a>
16. </td>
17. </tr>
18. </table>
19. </body>
20. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%

int PageCount = 15;

%>

<html>

<body>

<table>

<tr>

<td>

<a href=showpage.jsp?ToPage=<%=1%>>到第1页</a>

</td>

<td>

<a href=showpage.jsp?ToPage=<%=PageCount%>>到最后一页</a>

</td>

</tr>

</table>

</body>

</html>

    接收显示分页showpage.jsp代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <%
7. String toPage = request.getParameter("ToPage");
8. out.println("第" + toPage + "页");
9. %>
10. </body>
11. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<%

String toPage = request.getParameter("ToPage");

out.println("第" + toPage + "页");

%>

</body>

</html>

    运行结果如下：



**2.jsp页面传递参数给servlet**

    (1)使用request对象接收参数

    新建名为ForwardServlet的Servlet类，代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. package com.shi.servlet;
3. import java.io.IOException;
5. import javax.servlet.ServletException;
6. import javax.servlet.http.HttpServlet;
7. import javax.servlet.http.HttpServletRequest;
8. import javax.servlet.http.HttpServletResponse;
10. /\*\*
11. \* 本类用来获得表单请求信息，并根据请求信息转发页面
12. \*/
13. public class ForwardServlet extends HttpServlet
14. {
15. private static final long serialVersionUID = 1L;
17. /\*\*
18. \* 构造函数
19. \*/
20. public ForwardServlet()
21. {
22. super();
23. }
25. /\*\*
26. \* 初始化
27. \*/
28. public void init() throws ServletException
29. {}
31. /\*\*
32. \* doGet()方法
33. \*/
34. public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
35. {
36. doPost(request, response);
37. }
39. /\*\*
40. \* doPost()方法
41. \*/
42. public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
43. {
44. String name = request.getParameter("name");
45. String pwd = request.getParameter("pwd");
46. if((name!=null && !name.equals("")) && (pwd!=null && !pwd.equals("")))
47. {
48. if(name.equals("mr")&&pwd.equals("123"))
49. {
50. request.getRequestDispatcher("success.jsp").forward(request, response);
51. }
52. else
53. {
54. request.getRequestDispatcher("error.jsp").forward(request, response);
55. }
56. }
57. }
59. /\*\*
60. \* 销毁
61. \*/
62. public void destroy()
63. {
64. super.destroy();
65. }
66. }

package com.shi.servlet;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* 本类用来获得表单请求信息，并根据请求信息转发页面

\*/

public class ForwardServlet extends HttpServlet

{

private static final long serialVersionUID = 1L;

/\*\*

\* 构造函数

\*/

public ForwardServlet()

{

super();

}

/\*\*

\* 初始化

\*/

public void init() throws ServletException

{}

/\*\*

\* doGet()方法

\*/

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

doPost(request, response);

}

/\*\*

\* doPost()方法

\*/

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

String name = request.getParameter("name");

String pwd = request.getParameter("pwd");

if((name!=null && !name.equals("")) && (pwd!=null && !pwd.equals("")))

{

if(name.equals("mr")&&pwd.equals("123"))

{

request.getRequestDispatcher("success.jsp").forward(request, response);

}

else

{

request.getRequestDispatcher("error.jsp").forward(request, response);

}

}

}

/\*\*

\* 销毁

\*/

public void destroy()

{

super.destroy();

}

}

    在web.xml中配置ForwardServlet类，代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <?xml version="1.0" encoding="GB2312"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5. http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">
7. <!-- 配置servlet -->
8. <servlet>
9. <servlet-name>ForwardServlet</servlet-name>
10. <servlet-class>com.shi.servlet.ForwardServlet</servlet-class>
11. </servlet>
12. <servlet-mapping>
13. <servlet-name>ForwardServlet</servlet-name>
14. <url-pattern>/forward</url-pattern>
15. </servlet-mapping>
17. <welcome-file-list>
18. <welcome-file>index.jsp</welcome-file>
19. </welcome-file-list>
20. </web-app>

<?xml version="1.0" encoding="GB2312"?>

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">

<!-- 配置servlet -->

<servlet>

<servlet-name>ForwardServlet</servlet-name>

<servlet-class>com.shi.servlet.ForwardServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ForwardServlet</servlet-name>

<url-pattern>/forward</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

    新建index.jsp，其中<form>中的action属性值为Servlet类在web.xml中配置的url，提交方式为post，代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <form action="forward" method="post">
7. <table align="center">
8. <tr>
9. <td>用户名：</td>
10. <td>
11. <input type="text" name="name"/>
12. </td>
13. </tr>
14. <tr>
15. <td>密码：</td>
16. <td>
17. <input type="password" name="pwd"/>
18. </td>
19. </tr>
20. <tr>
21. <td colspan="2">
22. <input type="submit" value="登录"/>
23. </td>
24. </tr>
25. </table>
26. </form>
27. </body>
28. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<form action="forward" method="post">

<table align="center">

<tr>

<td>用户名：</td>

<td>

<input type="text" name="name"/>

</td>

</tr>

<tr>

<td>密码：</td>

<td>

<input type="password" name="pwd"/>

</td>

</tr>

<tr>

<td colspan="2">

<input type="submit" value="登录"/>

</td>

</tr>

</table>

</form>

</body>

</html>

    登录成功页面success.jsp，代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. 欢迎<%=request.getParameter("name") %>，登录成功！
7. </body>
8. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

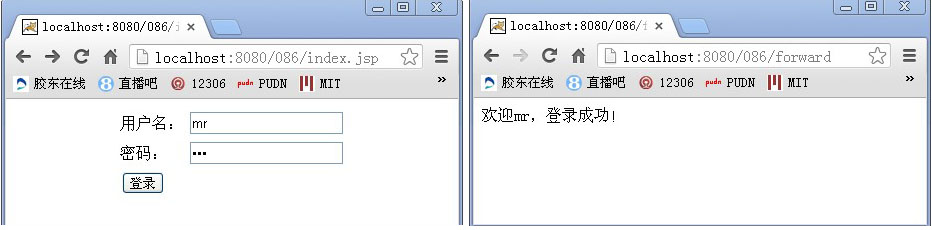
<body>

欢迎<%=request.getParameter("name") %>，登录成功！

</body>

</html>

    程序的运行结果图5所示：



    (2)使用session对象，url，form表单传递参数

    在jsp2servlet.jsp中，分别使用request对象，session对象，url，form表单设置需要传递的参数，代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <%
7. session.setAttribute("testSession","Hello session");
8. request.setAttribute("testRequest","Hello request");
9. %>
10. <a href="http://localhost:8080/086/jsp2servlet?hrefAction=toServlet">单击此处</a>
11. <form action="jsp2servlet?action=toServlet" method="post" name="form1">
12. <input type="text" name="username"/>
13. <input type="submit" value="submit"/>
14. </form>
15. </body>
16. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<%

session.setAttribute("testSession","Hello session");

request.setAttribute("testRequest","Hello request");

%>

<a href="http://localhost:8080/086/jsp2servlet?hrefAction=toServlet">单击此处</a>

<form action="jsp2servlet?action=toServlet" method="post" name="form1">

<input type="text" name="username"/>

<input type="submit" value="submit"/>

</form>

</body>

</html>

    在Jsp2Servlet中接收jsp页面中传递的参数并将其显示，代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. package com.shi.servlet;
3. import java.io.IOException;
4. import java.io.PrintWriter;
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
11. public class Jsp2Servlet extends HttpServlet
12. {
13. /\*\*
14. \* 自动生成
15. \*/
16. private static final long serialVersionUID = 1L;
18. /\*\*
19. \* 构造函数
20. \*/
21. public Jsp2Servlet()
22. {
23. super();
24. }
26. /\*\*
27. \* 初始化
28. \*/
29. public void init()
30. {}
32. /\*\*
33. \* doGet()
34. \*/
35. public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
36. {
37. doPost(request, response);
38. }
40. /\*\*
41. \* doPost()
42. \*/
43. public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
44. {
45. String strSession = (String)request.getSession().getAttribute("testSession");
46. String strRequest = (String)request.getAttribute("testRequest");
47. String strHref = request.getParameter("hrefAction");
48. String strForm = request.getParameter("action");
49. String username = request.getParameter("username");
51. PrintWriter out = response.getWriter();
52. out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">");
53. out.println("<html>");
54. out.println("<body>");
55. out.println("保存到session中的内容： " + strSession + "<br>");
56. out.println("保存到request中的内容：" + strRequest + "<br>");
57. out.println("超链接中传递的参数值为：" + strHref + "<br>");
58. out.println("form表单中传递的action值为：" + strForm + "<br>");
59. out.println("form表单内input标签中username的值： " + username + "<br>");
60. out.println("</body>");
61. out.println("</html>");
62. }
64. /\*\*
65. \* 销毁
66. \*/
67. public void destroy()
68. {
69. super.destroy();
70. }
71. }

package com.shi.servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class Jsp2Servlet extends HttpServlet

{

/\*\*

\* 自动生成

\*/

private static final long serialVersionUID = 1L;

/\*\*

\* 构造函数

\*/

public Jsp2Servlet()

{

super();

}

/\*\*

\* 初始化

\*/

public void init()

{}

/\*\*

\* doGet()

\*/

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

doPost(request, response);

}

/\*\*

\* doPost()

\*/

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

String strSession = (String)request.getSession().getAttribute("testSession");

String strRequest = (String)request.getAttribute("testRequest");

String strHref = request.getParameter("hrefAction");

String strForm = request.getParameter("action");

String username = request.getParameter("username");

PrintWriter out = response.getWriter();

out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">");

out.println("<html>");

out.println("<body>");

out.println("保存到session中的内容： " + strSession + "<br>");

out.println("保存到request中的内容：" + strRequest + "<br>");

out.println("超链接中传递的参数值为：" + strHref + "<br>");

out.println("form表单中传递的action值为：" + strForm + "<br>");

out.println("form表单内input标签中username的值： " + username + "<br>");

out.println("</body>");

out.println("</html>");

}

/\*\*

\* 销毁

\*/

public void destroy()

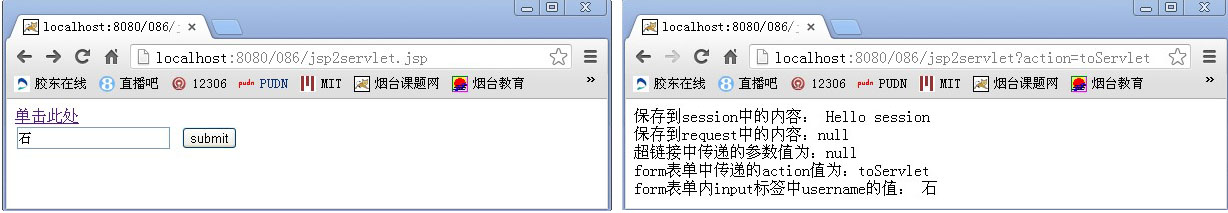
{

super.destroy();

}

}

    程序运行结果如图6所示：



    servlet无法获取request.setAttribute()中的内容，因为在jsp2servlet.jsp中提交时，生成的一个新的request，因此无法取到jsp页面中request.setAttribute()中设置的值。

**3.从servlet到jsp页面中传递参数**

    从servlet到jsp页面传递参数，可以使用URL，request对象，session对象，新建Servlet2Jsp.java，代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. package com.shi.servlet;
3. import java.io.IOException;
5. import javax.servlet.ServletException;
6. import javax.servlet.http.HttpServlet;
7. import javax.servlet.http.HttpServletRequest;
8. import javax.servlet.http.HttpServletResponse;
10. public class Servlet2Jsp extends HttpServlet
11. {
12. /\*\*
13. \* 自动生成
14. \*/
15. private static final long serialVersionUID = 1L;
17. /\*\*
18. \* 构造函数
19. \*/
20. public Servlet2Jsp()
21. {
22. super();
23. }
25. /\*\*
26. \* 初始化
27. \*/
28. public void init()
29. {}
31. /\*\*
32. \* doGet()
33. \*/
34. public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
35. {
36. doPost(request, response);
37. }
39. /\*\*
40. \* doPost()
41. \*/
42. public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
43. {
44. String url = "result.jsp?first=传递第一个参数&second=传递第二个参数";
45. String strRequest = "request传值";
46. String strSession = "session传值";
47. request.setAttribute("strRequest", strRequest);
48. request.getSession().setAttribute("strSession", strSession);
50. /\*\*
51. \* 客户端跳转：效率低
52. \* session范围属性，url中的参数会传递下去，request范围属性不传递
53. \*/
54. //response.sendRedirect(url);
56. /\*\*
57. \* 服务器端跳转：常用，效率高
58. \* request范围属性，session范围属性，url中的参数会传递
59. \*/
60. request.getRequestDispatcher(url).forward(request, response);
61. }
63. /\*\*
64. \* 销毁
65. \*/
66. public void destroy()
67. {
68. super.destroy();
69. }
70. }

package com.shi.servlet;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class Servlet2Jsp extends HttpServlet

{

/\*\*

\* 自动生成

\*/

private static final long serialVersionUID = 1L;

/\*\*

\* 构造函数

\*/

public Servlet2Jsp()

{

super();

}

/\*\*

\* 初始化

\*/

public void init()

{}

/\*\*

\* doGet()

\*/

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

doPost(request, response);

}

/\*\*

\* doPost()

\*/

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

String url = "result.jsp?first=传递第一个参数&second=传递第二个参数";

String strRequest = "request传值";

String strSession = "session传值";

request.setAttribute("strRequest", strRequest);

request.getSession().setAttribute("strSession", strSession);

/\*\*

\* 客户端跳转：效率低

\* session范围属性，url中的参数会传递下去，request范围属性不传递

\*/

//response.sendRedirect(url);

/\*\*

\* 服务器端跳转：常用，效率高

\* request范围属性，session范围属性，url中的参数会传递

\*/

request.getRequestDispatcher(url).forward(request, response);

}

/\*\*

\* 销毁

\*/

public void destroy()

{

super.destroy();

}

}

    在web.xml中配置该servlet后，新建参数接收页result.jsp，代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <%
7. String strURL = request.getParameter("first");
8. String strRequest = (String)request.getAttribute("strRequest");
9. String strSession = (String)request.getSession().getAttribute("strSession");
11. %>
12. <p>
13. URL中取得的属性值为：<%=strURL%>
14. </p>
15. <p>
16. request中取得的属性值为：<%=strRequest%>
17. </p>
18. <p>
19. session中取得的属性值为：<%=strSession%>
20. </p>
21. </body>
22. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<%

String strURL = request.getParameter("first");

String strRequest = (String)request.getAttribute("strRequest");

String strSession = (String)request.getSession().getAttribute("strSession");

%>

<p>

URL中取得的属性值为：<%=strURL%>

</p>

<p>

request中取得的属性值为：<%=strRequest%>

</p>

<p>

session中取得的属性值为：<%=strSession%>

</p>

</body>

</html>

    程序的运行结果如图7所示：



**4.从一个servlet到另一个servlet时的参数传递**

    (1)通过超链接和表单传递数据

    通过超链接后面给定要传递参数的值，但长度有限制，通过超链接和表单传递数据，使用request.getParamter()方法，代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. package com.shi.servlet;
3. import java.io.IOException;
4. import java.io.PrintWriter;
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
11. public class ServletA extends HttpServlet
12. {
13. /\*\*
14. \* 自动生成
15. \*/
16. private static final long serialVersionUID = 1L;
18. /\*\*
19. \* 构造函数
20. \*/
21. public ServletA()
22. {
23. super();
24. }
26. /\*\*
27. \* 初始化
28. \*/
29. public void init()
30. {}
32. /\*\*
33. \* doGet()
34. \*/
35. public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
36. {
37. doPost(request, response);
38. }
40. /\*\*
41. \* doPost()
42. \*/
43. public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
44. {
45. response.setContentType("text/html");
46. PrintWriter out = response.getWriter();
47. String name = "hello, servlet";
48. out.println("<html>");
49. out.println("<body>");
50. /\*\*
51. \* 超链接
52. \*/
53. out.println("<a href='ServletB?name="+name+"'>this is a test</a>");
54. /\*\*
55. \* 表单
56. \*/
57. out.println("<form action='ServletB' method='post'>");
58. out.println("用户名：");
59. out.println("<input type='text' name='username'>");
60. out.println("<input type='submit' vlue='提交'>");
61. out.println("</form>");
62. out.println("</body>");
63. out.println("</html>");
64. out.flush();
66. }
68. /\*\*
69. \* 销毁
70. \*/
71. public void destroy()
72. {
73. super.destroy();
74. }
75. }

package com.shi.servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class ServletA extends HttpServlet

{

/\*\*

\* 自动生成

\*/

private static final long serialVersionUID = 1L;

/\*\*

\* 构造函数

\*/

public ServletA()

{

super();

}

/\*\*

\* 初始化

\*/

public void init()

{}

/\*\*

\* doGet()

\*/

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

doPost(request, response);

}

/\*\*

\* doPost()

\*/

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String name = "hello, servlet";

out.println("<html>");

out.println("<body>");

/\*\*

\* 超链接

\*/

out.println("<a href='ServletB?name="+name+"'>this is a test</a>");

/\*\*

\* 表单

\*/

out.println("<form action='ServletB' method='post'>");

out.println("用户名：");

out.println("<input type='text' name='username'>");

out.println("<input type='submit' vlue='提交'>");

out.println("</form>");

out.println("</body>");

out.println("</html>");

out.flush();

}

/\*\*

\* 销毁

\*/

public void destroy()

{

super.destroy();

}

}

    接收servlet中传递参数的代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. package com.shi.servlet;
3. import java.io.IOException;
4. import java.io.PrintWriter;
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
11. public class ServletB extends HttpServlet
12. {
13. /\*\*
14. \* 自动生成
15. \*/
16. private static final long serialVersionUID = 1L;
18. /\*\*
19. \* 构造函数
20. \*/
21. public ServletB()
22. {
23. super();
24. }
26. /\*\*
27. \* 初始化
28. \*/
29. public void init()
30. {}
32. /\*\*
33. \* doGet()
34. \*/
35. public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
36. {
37. doPost(request, response);
38. }
40. /\*\*
41. \* doPost()
42. \*/
43. public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
44. {
45. String name = request.getParameter("name");
46. String username = request.getParameter("username");

49. response.setContentType("text/html;charset=gb2312");
50. PrintWriter out = response.getWriter();
51. out.println("<html>");
52. out.println("<body>");
53. out.println("超链接传递的参数值为：" + name);
54. out.println("form表单中提交的用户名为：" + username);
55. out.println("</body>");
56. out.println("</html>");
57. out.flush();
58. }
60. /\*\*
61. \* 销毁
62. \*/
63. public void destroy()
64. {
65. super.destroy();
66. }
67. }

package com.shi.servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class ServletB extends HttpServlet

{

/\*\*

\* 自动生成

\*/

private static final long serialVersionUID = 1L;

/\*\*

\* 构造函数

\*/

public ServletB()

{

super();

}

/\*\*

\* 初始化

\*/

public void init()

{}

/\*\*

\* doGet()

\*/

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

doPost(request, response);

}

/\*\*

\* doPost()

\*/

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

String name = request.getParameter("name");

String username = request.getParameter("username");

response.setContentType("text/html;charset=gb2312");

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<body>");

out.println("超链接传递的参数值为：" + name);

out.println("form表单中提交的用户名为：" + username);

out.println("</body>");

out.println("</html>");

out.flush();

}

/\*\*

\* 销毁

\*/

public void destroy()

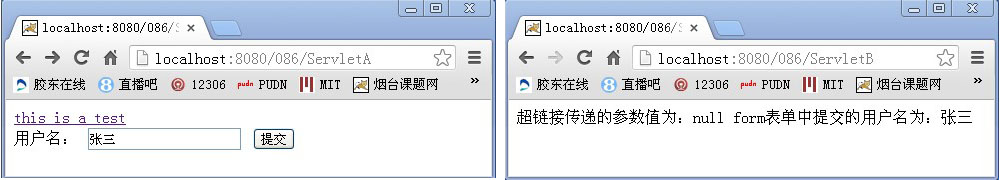
{

super.destroy();

}

}

    程序运行结果如图8所示：



    (2)使用setAttribute()和getAttribute()方法传递参数

    设置参数的ServletA代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. package com.shi.servlet;
3. import java.io.IOException;
4. import java.io.PrintWriter;
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
11. public class ServletA extends HttpServlet
12. {
13. /\*\*
14. \* 自动生成
15. \*/
16. private static final long serialVersionUID = 1L;
18. /\*\*
19. \* 构造函数
20. \*/
21. public ServletA()
22. {
23. super();
24. }
26. /\*\*
27. \* 初始化
28. \*/
29. public void init()
30. {}
32. /\*\*
33. \* doGet()
34. \*/
35. public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
36. {
37. doPost(request, response);
38. }
40. /\*\*
41. \* doPost()
42. \*/
43. public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
44. {
45. /\*\*
46. \* setAttribute()和getAttribute()
47. \*/
48. String data = "This is a test, we need more test for this";
49. request.setAttribute("strAtt", data);
50. request.getRequestDispatcher("/ServletB").forward(request, response);
51. }
53. /\*\*
54. \* 销毁
55. \*/
56. public void destroy()
57. {
58. super.destroy();
59. }
60. }

package com.shi.servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class ServletA extends HttpServlet

{

/\*\*

\* 自动生成

\*/

private static final long serialVersionUID = 1L;

/\*\*

\* 构造函数

\*/

public ServletA()

{

super();

}

/\*\*

\* 初始化

\*/

public void init()

{}

/\*\*

\* doGet()

\*/

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

doPost(request, response);

}

/\*\*

\* doPost()

\*/

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

/\*\*

\* setAttribute()和getAttribute()

\*/

String data = "This is a test, we need more test for this";

request.setAttribute("strAtt", data);

request.getRequestDispatcher("/ServletB").forward(request, response);

}

/\*\*

\* 销毁

\*/

public void destroy()

{

super.destroy();

}

}

    获取参数的ServletB代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. package com.shi.servlet;
3. import java.io.IOException;
4. import java.io.PrintWriter;
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
11. public class ServletB extends HttpServlet
12. {
13. /\*\*
14. \* 自动生成
15. \*/
16. private static final long serialVersionUID = 1L;
18. /\*\*
19. \* 构造函数
20. \*/
21. public ServletB()
22. {
23. super();
24. }
26. /\*\*
27. \* 初始化
28. \*/
29. public void init()
30. {}
32. /\*\*
33. \* doGet()
34. \*/
35. public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
36. {
37. doPost(request, response);
38. }
40. /\*\*
41. \* doPost()
42. \*/
43. public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
44. {
45. String strAtt = (String)request.getAttribute("strAtt");
47. response.setContentType("text/html;charset=gb2312");
48. PrintWriter out = response.getWriter();
49. out.println("<html>");
50. out.println("<body>");
51. out.println("getAttribute()得到的数据值为：" + strAtt);
52. out.println("</body>");
53. out.println("</html>");
54. out.flush();
55. }
57. /\*\*
58. \* 销毁
59. \*/
60. public void destroy()
61. {
62. super.destroy();
63. }
64. }

package com.shi.servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class ServletB extends HttpServlet

{

/\*\*

\* 自动生成

\*/

private static final long serialVersionUID = 1L;

/\*\*

\* 构造函数

\*/

public ServletB()

{

super();

}

/\*\*

\* 初始化

\*/

public void init()

{}

/\*\*

\* doGet()

\*/

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

doPost(request, response);

}

/\*\*

\* doPost()

\*/

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

String strAtt = (String)request.getAttribute("strAtt");

response.setContentType("text/html;charset=gb2312");

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<body>");

out.println("getAttribute()得到的数据值为：" + strAtt);

out.println("</body>");

out.println("</html>");

out.flush();

}

/\*\*

\* 销毁

\*/

public void destroy()

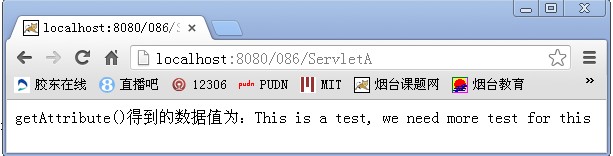
{

super.destroy();

}

}

    程序的运行结果如图9所示：



**5.JSP页面跳转方法**

***(1)RequestDispatcher.forward()***

    在服务器端起作用，当使用forward()时，Servlet engine传递HTTP请求从当前的Servlet or JSP到另外一个Servlet，JSP或普通HTML文件，即你的form提交至a.jsp,在a.jsp用到了forward()重定向到b.jsp，此时form提交的所有信息在b.jsp都可以获得，参数自动传递。但forward()无法重定向至有frame的jsp文件，可以重定向至有frame的html文件，同时forward()无法再后面带参数传递，比如servlet?name=tom，这样不行，可以程序内通过response.setAttribute("name",name)来传至下一个页面。

    重定向后浏览器地址栏URL不变。

    通常在Servlet中使用，不在jsp中使用。

***(2)response.sendRedirect()***

    在用户的浏览器端工作，sengRedirect()可以带参数传递，比如servlet?name=tom传至下个页面，同时它可以重定向至不同的主机上，sendRedirect()可以重定向至有frame的jsp文件。

    重定向后在浏览器地址栏上会出现重定向页面的URL。

**上述两种方式代码见图7。**

    由于response是jsp页面中的隐含对象，故在jsp页面中可以用response.sendRedirect()直接实现重定位。

    注意：

    ①.使用response.sendRedirect()时，前面不能有HTML输出

    这并不是绝对的，不能有HTML输出其实是指不能有HTML被送到了浏览器。事实上现在的Server都有cache机制，一般在8K(JSP SERVER)，这意味着，除非你关闭了cache，或者你用out.flush()强制刷新，那么在使用sendRedirect之前，有少量的HTML输出也是允许的。

    ②.response.sendRedirect()之后，应该紧跟一句return

    我们已经知道response.sendRedirect()是通过浏览器来做转向的，所以只有在页面处理完成后，才会有实际的动作。既然你已经要做转向了，那么后面的输出还有什么意义呢？而且有可能会因为后面的输出导致转向失败。

    比较：

    ①.Dispatcher.forward()是容器中控制权的转向，在客户端浏览器地址栏中不会出现转向后的地址

    ②.response.sendRedirect()则是完全的跳转，浏览器将会得到跳转的地址，并重新发送请求链接。这样，从浏览器的地址栏中可以看到跳转后的链接地址。

**前者更加高效，在前者可以满足需要时，尽量使用RequestDispatcher.forward()方法。**

    注：在有些情况下，比如，需要跳转到一个其它服务器上的资源，则必须使用HttpServletResponse.sendRedirect()方法。

***(3)<jsp:forward page=""/>***

它的底层部分是由RequestDispatcher来实现的，因此它带有RequestDispatcher.forward()方法的印记。

    如果在之前有很多输出，前面的输出已使缓冲区满，将自动输出至客户端，那么该语句将不起作用，这一点应该特别注意。

    另外要注意：它不能改变浏览器地址，刷新的话会导致重复提交。

    测试代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="gb2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <head>
6. <link rel="stylesheet" type="text/css" href="css/style.css">
7. </head>
8. <body>
9. <jsp:forward page="UpdateServlet">
10. <jsp:param value="list" name="action" />
11. </jsp:forward>
12. </body>
13. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="gb2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<link rel="stylesheet" type="text/css" href="css/style.css">

</head>

<body>

<jsp:forward page="UpdateServlet">

<jsp:param value="list" name="action" />

</jsp:forward>

</body>

</html>

    程序运行结果如图10所示：



***(4)修改HTTP header的Location属性来重定向***

    通过设置直接修改地址栏来实现页面的重定向。

    索引页1.jsp代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <%
7. response.setStatus(HttpServletResponse.SC\_MOVED\_PERMANENTLY);
8. String newLocn = "2.jsp";
9. response.setHeader("Location",newLocn);
10. %>
11. </body>
12. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<%

response.setStatus(HttpServletResponse.SC\_MOVED\_PERMANENTLY);

String newLocn = "2.jsp";

response.setHeader("Location",newLocn);

%>

</body>

</html>

    转向页2.jsp代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. 从1.jsp跳转至此页
7. </body>
8. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

从1.jsp跳转至此页

</body>

</html>

    程序运行结果如图11所示：



***(5)JSP中实现在某页面停留若干秒后，自动重定向到另一页面***

    在jsp文件中，下面的代码：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <meta http-equiv="refresh" content="5; url=target.jsp"

<meta http-equiv="refresh" content="5; url=target.jsp"

    含义是：在5秒后正在浏览的页面将会自动变为target.jsp页。代码中5为刷新的延迟时间，以秒为单位。target.jsp为你想转向的目标页，若为本页则自动刷新本页。

    由上可知，可以通过setHeader来实现某页面停留若干秒后，自动重定向到另一页面。

    索引页5\_1.jsp代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. <%
7. String content = "5; url=5\_2.jsp";
8. response.setHeader("REFRESH",content);
9. %>
10. 5\_1.jsp页面，5秒后转向5\_2.jsp页面。
11. </body>
12. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<body>

<%

String content = "5; url=5\_2.jsp";

response.setHeader("REFRESH",content);

%>

5\_1.jsp页面，5秒后转向5\_2.jsp页面。

</body>

</html>

    转向页5\_2.jsp代码如下：

**[java]** [view plaincopyprint?](http://blog.csdn.net/ssy_shandong/article/details/9328985)

1. <%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>
3. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
4. <html>
5. <body>
6. 从5\_1.jsp转到的页面
7. </body>
8. </html>

<%@ page language="java" import="java.util.\*" pageEncoding="GB2312"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

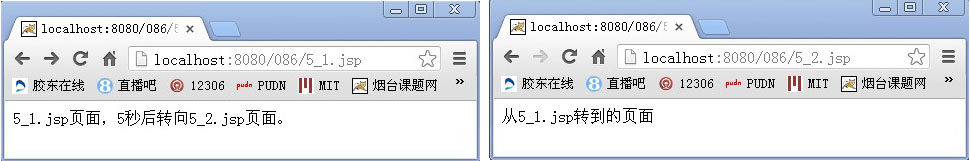
<body>

从5\_1.jsp转到的页面

</body>

</html>

    程序运行结果如下：



    参考资料：

    [1].五种JSP页面跳转方法

##### 2.10.3五种JSP页面跳转方法

**1. RequestDispatcher.forward()**

是在服务器端起作用,当使用forward()时,Servlet engine传递HTTP请求从当前的Servlet or JSP到另外一个Servlet,JSP 或普通HTML文件,也即你的form提交至a.jsp,在a.jsp用到了forward()重定向至b.jsp,此时form提交的所有信息在 b.jsp都可以获得,参数自动传递. 但forward()无法重定向至有frame的jsp文件,可以重定向至有frame的html文件,同时forward()无法在后面带参数传递,比如servlet?name=frank,这样不行,可以程序内通过response.setAttribute("name",name)来传至下一个页面。

重定向后浏览器地址栏URL不变。

例：在servlet中进行重定向

|  |
| --- |
| public void doPost(HttpServletRequest request,HttpServletResponse response)  throws ServletException,IOException  {  　 response.setContentType("text/html; charset=gb2312");  　 ServletContext sc = getServletContext();  　 RequestDispatcher rd = null;  　 rd = sc.getRequestDispatcher("/index.jsp"); //定向的页面  　 rd.forward(request, response);  } |

通常在servlet中使用，不在jsp中使用。

**2. response.sendRedirect()**

是在用户的浏览器端工作,sendRedirect()可以带参数传递,比如servlet?name=frank传至下个页面,同时它可以重定向至不同的主机上,sendRedirect()可以重定向有frame.的jsp文件.

重定向后在浏览器地址栏上会出现重定向页面的URL

例：在servlet中重定向

|  |
| --- |
| public void doPost(HttpServletRequest request,HttpServletResponse response)  throws ServletException,IOException  {  　 response.setContentType("text/html; charset=gb2312");  　 response.sendRedirect("/index.jsp");  } |

由于response是jsp页面中的隐含对象，故在jsp页面中可以用response.sendRedirect()直接实现重定位。

**注意：**

(1) 使用response.sendRedirect时，前面不能有HTML输出；

这并不是绝对的，不能有HTML输出其实是指不能有HTML被送到了浏览器。事实上现在的server都有cache机制，一般在8K（我是说 JSP　SERVER），这就意味着，除非你关闭了cache，或者你使用了out.flush()强制刷新，那么在使用sendRedirect之前，有少量的HTML输出也是允许的。

(2) response.sendRedirect之后，应该紧跟一句return。

我们已经知道response.sendRedirect是通过浏览器来做转向的，所以只有在页面处理完成后，才会有实际的动作。既然你已经要做转向了，那么后的输出还有什么意义呢？而且有可能会因为后面的输出导致转向失败。

**比较：**

(1) Dispatcher.forward()是容器中控制权的转向，在客户端浏览器地址栏中不会显示出转向后的地址；

(2) response.sendRedirect()则是完全的跳转，浏览器将会得到跳转的地址，并重新发送请求链接。这样，从浏览器的地址栏中可以看到跳转后的链接地址。

前者更加高效，在前者可以满足需要时，尽量使用RequestDispatcher.forward()方法。

*注：在有些情况下，比如，需要跳转到一个其它服务器上的资源，则必须使用HttpServletResponse.sendRequest()方法。*

**3. ＜jsp:forward page="" /＞**

它的底层部分是由RequestDispatcher来实现的，因此它带有RequestDispatcher.forward()方法的印记。

如果在之前有很多输出,前面的输出已使缓冲区满,将自动输出到客户端,那么该语句将不起作用,这一点应该特别注意。

另外要注意：它不能改变浏览器地址，刷新的话会导致重复提交

**4. 修改HTTP header的Location属性来重定向**

通过设置直接修改地址栏来实现页面的重定向。

jsp文件代码如下：

|  |
| --- |
| ＜%    response.setStatus(HttpServletResponse.SC\_MOVED\_PERMANENTLY);    String newLocn = "/newpath/jsa.jsp";    response.setHeader("Location",newLocn);  %＞ |

**5. JSP中实现在某页面停留若干秒后,自动重定向到另一页面**

在html文件中，下面的代码：

|  |
| --- |
| ＜meta http-equiv="refresh" content="300; url=target.jsp"＞ |

它的含义：在5分钟之后正在浏览的页面将会自动变为target.html这一页。代码中300为刷新的延迟时间，以秒为单位。targer.html为你想转向的目标页,若为本页则为自动刷新本页。

由上可知，可以通过setHeader来实现某页面停留若干秒后,自动重定向到另一页面。

关键代码：

|  |
| --- |
| String content=stayTime+";URL="+URL;  response.setHeader("REFRESH",content); |

【编辑推荐】

1. [高性能、高弹性JSP和Servlet性能优化](http://developer.51cto.com/art/200901/104932.htm)
2. [Tomcat下JSP经典配置实例](http://developer.51cto.com/art/200902/108709.htm)
3. [JSP开发中Cookie使用完全详解](http://developer.51cto.com/art/200902/109287.htm)

### 3.JDBC

### 4.用户自定义标签

### 5.EL与JSTL

## SSH三大核心框架(Structs、Spring、Hibernate)

### 1.Structs2

#### 1.1MVC框架

##### 1.1.1简述

MVC是一种变成思想，将一个软件分为三层，分别承担不同的业务。Structs2只是MVC的一种实现，关于Web端的。在这之前，在Java Web中有两种实现模式，分别是Model1和Model2：

Model1 = JSP + JavaBean；

Model2 = + JSP + Servlet + JavaBean；

然后有了Structs，再然后有了Structs2，了解下历史然后用Structs2就可以了，后面就只介绍Structs(有点啰嗦了是吧，不过了解下历史也是挺好的)

#### 1.2下载与安装Structs2

##### 1.2.1下载

找个有网的地方，输入Structs2找不到去跳楼。下载下来之后解压出来，可以看到有四个目录以及十来个文本文件：apps，docs，lib，src。

##### 1.2.2应用到程序

估计都是将lib里的jar文件添加到项目，并添加web.xml，struts.xml，struts.properties，struts-default.xml，structs-plugin.xml文件到相应文件夹（往后介绍）即可。

#### 1.3开发Structs2框架程序的基本步骤

##### 1.3.1Struts2工作流程

客户端浏览器发出请求

核心控制器FilterDispatcher接收到请求后根据后面的扩展名来决定是否调用action，以及调用哪个action

在调用action的execute()之前，struts2会调用一系列的拦截器，以提供一些通用的功能，如验证，文件上传等功能。这些拦截器的组合被称为拦截器连。

在调用完拦截器后，Struts2会调用Action的exexute()，在execute()中执行用户的相关操作(有点像Servlet，后面会讲的)

根据execute()的返回值，将处理结果信息返回到浏览器，这些结果可以是HTMl页面，JSP页面，图像等等。

简单如下图：

拦截器1

拦截器2

浏览器

拦截器3

##### 1.3.2开发步骤，以Hello World程序为例

###### 1.3.2.0新建一个Web项目

新建项目时可以选择同时新建一个web.xml的配置文件，如果忘记了要么重新建一个工程，要么，在WebContent->WEB-INF目录下新建一个web.xml，右键项目，选择Export弹出框框选择General在下来选项中选择Ant Buildfile，然后选择项目添加一个build.xml文件。

###### 1.3.2.1配置web.xml（）

配置struts2提供的过滤器，并设置所有的请求都要通过该过滤器（<url-pattern>/\*</url-pattern>）：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"* id=*"WebApp\_ID"* version=*"3.0"*>

<display-name>HelloStruts</display-name>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

</web-app>

###### 1.3.2.2编写JSP页面index.jsp、HelloWorld.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=gb2312"* pageEncoding=*"UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<h1>Hello World! Hello Struts2!</h1>

<p>第一个struts2应用程序：Hello Struts2，单击下面按钮调用Action类</p>

<form action=*"hello"*>

<input type=*"submit"* value=*"调用Action"* />

</form>

</body>

</html>

///////////////

///////HelloWorld.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=gb2312"* pageEncoding=*"UTF-8"*%>

<%@ taglib prefix=*"s"* uri=*"/struts-tags"* %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<h3>下面消息来自于HelloWorldAction Action类</h3>

<h2><s:property value=*"message"* /></h2>

</body>

</html>

###### 1.3.2.3编写Action

Action类是最基本的逻辑处理单元，在MVC模式中分发器分发给不同的Action类来处理请求。

**package** com.actions;

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** HelloWorldAction **extends** ActionSupport

{

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** String message;

**public** **void** setMessage(String msg)

{

**this**.message = msg;

}

**public** String getMessage()

{

**return** message;

}

**public** String execute() **throws** Exception

{

setMessage("Hello ni ge tou!");

**return** *SUCCESS*;

}

}

###### 1.3.2.4配置文件中增加映射(struts2.xml)

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<constant name=*"struts.devMode"* value=*"false"* />

<package name=*"default"* extends=*"struts-default"*>

<action name=*"index"*>

<result>/index.jsp</result>

</action>

<action name=*"hello"* class=*"com.actions.HelloWorldAction"* method=*"execute"*>

<result name=*"success"*>/HelloWorld.jsp</result>

</action>

</package>

</struts>

###### 1.3.2.4构建运行

右键build.xml选择运行Ant Build(没有Ant Build选项？添加一个，Eclipse自带有的)，控制台输出Success后运行项目On Server，即会进入默认页面：index.jsp

#### 1.4Structs2中的配置文件解读

##### 1.4.1web.xml

每一个Web应用都需要一个配置文件web.xml，该文件用来对整个应用程序进行配置。Struts2框架的web.xml文件需要配置一个前端过滤器：StrutsPrepareAndExecuteFilter，用于对Struts2框架进行初始化以及处理所有请求。

The web.xml web application descriptor file represents the core of the Java web application, so it is appropriate that it is also part of the core of the Struts framework. In the web.xml file, Struts defines its FilterDispatcher, the Servlet Filter class that initializes the Struts framework and handles all requests. This filter can contain initialization parameters that affect what, if any, additional configuration files are loaded and how the framework should behave.

In addition to the FilterDispatcher, Struts also provides an ActionContextCleanUp class that handles special cleanup tasks when other filters, such as those used by Sitemesh, need access to an initialized Struts framework.

**Key Initialization Parameters**

* **config** - a comma-delimited list of XML configuration files to load.
* **actionPackages** - a comma-delimited list of Java packages to scan for Actions.
* **configProviders** - a comma-delimited list of Java classes that implement the ConfigurationProvider interface that should be used for building the Configuration.
* **loggerFactory** - The class name of the LoggerFactory implementation.
* **\*** - any other parameters are treated as framework constants.

**Simple Example**

Configuring web.xml for the framework is a matter of adding a filter and filter-mapping.

**FilterDispatcher Example (web.xml)**

<web-app id="WebApp\_9" version="2.4"

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 http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

<init-param>

<param-name>actionPackages</param-name>

<param-value>com.mycompany.myapp.actions</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- ... -->

</web-app>

|  |  |  |
| --- | --- | --- |
|  | **Changed Filter Structure in Struts >= 2.1.3** To split up the the dispatcher phases, FilterDispatcher is deprecated since Struts 2.1.3. If working with older versions, you need to use  ...  <filter>  <filter-name>struts2</filter-name>  <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>  ...  See [SiteMesh Plugin](http://struts.apache.org/release/2.1.x/docs/sitemesh-plugin.html) for an example on when to use seperate Filters for prepare and execution phase | |
|  | **Why the Filter is mapped with /\* and how to configure explicit exclusions (since 2.1.7)** In the example above we've mapped the Struts 2 dispatcher to /\*, so Struts 2 has a crack at all incoming requests. This is because Struts 2 serves static content from its jar files, including Dojo JavaScript files (if using S2.0, or the Dojo plugin in S2.1+) and FreeMarker templates for the Struts 2 tags that produce HTML.  If we change the filter mapping to something else, for example /\*.html, we must take this in to account and extract the content that would normally be served from the Struts 2 jar files, or some other solution.  Since Struts 2.1.7, you are able to provide a comma seperated list of patterns for which when matching against the request URL the Filter will just pass by. This is done via the configuration option struts.action.excludePattern, for example in your struts.xml  <struts>  <constant name="struts.action.excludePattern" value=".\*unfiltered.\*,.\*\\.nofilter"/>  ...  </struts> |

**Taglib Example**

Typically, configuring a taglib is neither required nor recommended. The taglib is included in struts-core.jar, and the container will discover it automatically.

If, for some reason, a taglib configuration is needed within web.xml, extract the TLD file from the struts-core.jar META-INF folder, and add a taglib element to the web.xml.

<!-- ... -->

</welcome-file-list>

<taglib>

<taglib-uri>/s</taglib-uri>

<taglib-location>/WEB-INF/struts-tags.tld</taglib-location>

</taglib>

</web-app>

##### 1.4.2struts.xml

###### 1.4.2.1Bean配置

###### 1.4.2.2常量配置

###### 1.4.2.3包配置

###### 1.4.2.4命名空间配置

###### 1.4.2.5包含配置

###### 1.4.2.6拦截器配置

##### 1.4.3配置Action(struts.xml)

##### 1.4.4配置result(struts.xml)

#### 1.5Eclipse Ant 构建Struts2应用程序（来源于struts2官方网站）

##### 1.5.1概述

This tutorial walks through installing the framework and creating a simple application.

|  |  |
| --- | --- |
|  | While the Struts 2 framework is simple to use, creating non-trivial applications assumes a working knowledge of many J2EE technologies, including:   * Java * Filters, JSP, and Tag Libraries * JavaBeans * HTML and HTTP * Web Containers (such as Tomcat) * XML |

For more about supporting technologies, see the [Key Technologies Primer](http://struts.apache.org/primer.html).

**Java Requirements**

Struts 2 requires Servlet API 2.4 or higher, JSP 2.0 or higher, and Java 5 or higher.

**Our First Application**

To get started using Struts 2 we will create a web application with the required Struts 2 artifacts in WEB-INF lib and use Ant to build the application. Then we will create the same application using Maven to manage the artifact dependencies.

You can download all the tutorial examples from Google Code - <http://code.google.com/p/struts2-examples/downloads/list>.

Click on the link for whichever technology (Ant or Maven) you're familiar with.

[Create Struts 2 Web Application With Artifacts In WEB-INF lib and Use Ant To Build The Application](http://struts.apache.org/release/2.1.x/docs/create-struts-2-web-application-with-artifacts-in-web-inf-lib-and-use-ant-to-build-the-application.html)

[Create Struts 2 Web Application Using Maven To Manage Artifacts and To Build The Application](http://struts.apache.org/release/2.1.x/docs/create-struts-2-web-application-using-maven-to-manage-artifacts-and-to-build-the-application.html)

**Children** [**Show Children**](javascript:showChildren())

##### 1.5.2Create Struts 2 Web Application With Artifacts In WEB-INF lib and Use Ant To Build The Application

You can download this complete example, Basic\_Struts2\_Ant, from Google Code - <http://code.google.com/p/struts2-examples/downloads/list>. Click on the link for Basic\_Struts2\_Ant.zip and save the file to your computer. Unzip the file and you should have a folder named Basic\_Struts2\_Ant. In that folder is a README.txt file with instruction on how to build and run the application.

|  |  |
| --- | --- |
|  | This tutorial assumes you already know how to create a Java web application, use Ant to build the web application archive (war) file, and deploy the war file to a Servlet container such as Tomcat or Jetty. |

To create a Struts 2 web application with the Struts 2 artifacts added to the the application's class path manually you will need to download the Struts 2 distribution from the [Apache Struts website](http://struts.apache.org/downloads.html).

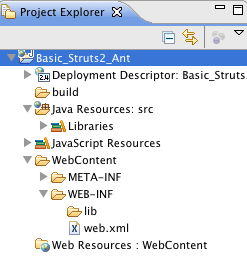
On the Struts 2 download page, click on the link for the current General Availability release. In that release's section you'll find several links. To get started with a basic Struts 2 web application you need to only download the Essential Dependencies Only zip file, which is approximately 12mb.

After downloading this zip file, unzip it. You should have a folder named the same as the current general availability release and in that folder will be a lib folder. The lib folder contains the Struts 2 jar files (e.g. struts2-core-X.X.X.X.jar, where X.X.X.X is the version) and other jar files Struts 2 requires (e.g. xwork-core.X.X.X.jar).

As we create our basic Struts 2 web application we will copy from the Struts 2 distribution lib folder just the jar files our application requires. As we add features to our application in future tutorials we will copy other jar files.

###### Step 1 - Create A Basic Java Web Application

In your Java IDE create a web application project named Basic\_Struts2\_Ant. To follow along with this tutorial your web application should have the following folder structure:



Remember Struts 2 requires Servlet API 2.4 or higher, JSP 2.0 or higher, and Java 5 or higher. The example project, Basic\_Struts2\_Ant, which you can download from Google code <http://code.google.com/p/struts2-examples/downloads/list> was created using Eclipse 3.5 with Java 6.

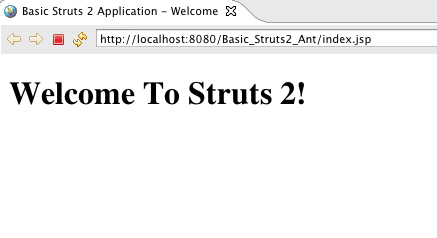
###### Step 2 - Add index.jsp and Ant Build File

Our next step is to add a simple index.jsp to this web application. Create an index.jsp under WebContent with a title of "Basic Struts 2 Application - Welcome" and in the body add an h1 heading of "Welcome to Struts 2!"

Next create an Ant build.xml file that will compile any Java classes and create the war file. The code example download, Basic\_Struts2\_Ant, includes an Ant build.xml.

Run the Ant target that creates the war file (in the example downloaded that is the archive target, which places the war file in the dist folder). Copy the war file into your Servlet container so that it will deploy the war file.

Start up your Servlet container and in a web browser go to <http://localhost:8080/Basic_Struts2_Ant/index.jsp>. You should see the following:



###### Step 3 - Add Struts 2 Jar Files To Class Path

Now that we know we have a working Java web application, lets add the minimal required Struts 2 framework Jar files to our web application's class path. We'll copy these Jar files from the lib folder of the Struts 2 download to the WEB-INF/lib folder in our web application.

Find and copy to WEB-INF\lib these files (note X.X.X.X.jar refers to the version number):

1. commons-fileupload-X.X.X.jar
2. commons-io-X.X.X.jar
3. commons-logging-X.X.X.jar
4. commons-logging-api.X.X.jar
5. freemarker-X.X.X.jar
6. ognl-X.X.X.jar
7. struts2-core-X.X.X.X.jar
8. xwork-core-X.X.X.jar

You can see from the above that the Struts 2 core jar depends on several other artifacts. As we add additional Struts 2 features in future tutorials, we'll need to add additional Struts 2 jar files and other transitive dependencies.

###### Step 4 - Add Logging

To see what's happening under the hood, the example application for this tutorial uses log4j. You'll need to download the log4j jar file and copy it to WEB-INF/lib. The example application uses log4j-1.2.14.jar <http://archive.apache.org/dist/logging/log4j/1.2.14/>.

Setup a log4j.xml configuration in the src folder. You can copy the one from the example application, which contains the following

**log4j.xml**

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE log4j:configuration PUBLIC "-//log4j/log4j Configuration//EN" "log4j.dtd">

<log4j:configuration **xmlns:log4j**="http://jakarta.apache.org/log4j/">

<appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%d %-5p %c.%M:%L - %m%n"/>

</layout>

</appender>

<!-- specify the logging level for loggers from other libraries -->

<logger name="com.opensymphony">

<level value="DEBUG" />

</logger>

<logger name="org.apache.struts2">

<level value="DEBUG" />

</logger>

<!-- for all other loggers log only debug and above log messages -->

<root>

<priority value="INFO"/>

<appender-ref ref="STDOUT" />

</root>

</log4j:configuration>

Note the above log4j configuration specifies the console as the log target.

###### Step 5 - Add Struts 2 Servlet Filter

To enable the Struts 2 framework to work with your web application you need to add a Servlet filter class and filter mapping to web.xml. Below is the filter and filter-mapping nodes you should add to web.xml.

**web.xml Servlet Filter**

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

For more information about configuring the deployment descriptor for Struts 2 see [web.xml](http://struts.apache.org/release/2.1.x/docs/webxml.html). Note the url-pattern node value is /\* meaning the Struts 2 filter will be applied to all URLs for this web application.

###### Step 6 - Create struts.xml

Struts 2 can use either an XML configuration file or annotations (or both) to specify the relationship between a URL, a Java class, and a view page (such as index.jsp). For our basic Struts 2 application, we'll use a minimal xml configuration. Note the file name is struts.xml and it should be in the src folder (struts.xml must be on the web application's root class path).

**struts.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<constant name="struts.devMode" value="true" />

<package name="basicstruts2" extends="struts-default">

<action name="index">

<result>/index.jsp</result>

</action>

</package>

</struts>

This minimal Struts 2 configuration file tells the framework that if the URL ends in index.action to redirect the browser to index.jsp.

For more information about the struts.xml configuration file see [XML](https://cwiki.apache.org/confluence/pages/createpage.action?spaceKey=WW&title=XML&linkCreation=true&fromPageId=14811863).

###### Step 7 - Build and Run the Application

With all of the above in place run the Ant target that creates the war file (in the example download that target is archive). Remove the previously created war file and exploded web application folder from your Servlet container's webapps folder. Copy to your Servlet container's webapps folder the new war you just created.

Start up the Servlet container. View the console where you should see numerous debug messages that tell you the Struts 2 framework is being included in the Basic\_Struts2\_Ant web application.

Open a web browser and go to <http://localhost:8080/Basic_Struts2_Ant/index.action> (note that's index.action not index.jsp at the end of the URL). You should see the same web page as when going to <http://localhost:8080/Basic_Struts2_Ant/index.jsp>. View the log messages written to the console and you should find several that discuss index.action and index.jsp:

**Struts 2 Log Messages**

com.opensymphony.xwork2.DefaultActionProxy.debug:57 - Creating an DefaultActionProxy for namespace / and action name index

...

org.apache.struts2.dispatcher.ServletDispatcherResult.debug:57 - Forwarding to location /index.jsp

Note that the xwork2 artifact is one of the Jar files we copied over to WEB-INF lib and is used heavily by the Struts 2 framework.

**Getting Help**

The [Struts 2 user mailing list](http://struts.apache.org/mail.html) is an excellent place to get help. If you are having a problem getting this Basic Struts 2 application to work search the Struts 2 mailing list. If you don't find an answer to your problem, post a question on the mailing list.

|  |  |
| --- | --- |
| **Next** | Onward to [Hello World Using Struts 2](http://struts.apache.org/release/2.1.x/docs/hello-world-using-struts-2.html) |
| **Prev** | Return to [Tutorials](http://struts.apache.org/release/2.1.x/docs/tutorials.html) |

**Children** [**Show Children**](javascript:showChildren())

#### 1.6Struts2拦截器

#### 1.7类型转换与输入校验

#### 1.8国际化

#### 1.9文件上传

#### 1.10Struts2标签库

##### 1.10.1控制标签

##### 1.10.2数据标签

##### 1.10.3表单标签

### 2.Spring

#### 2.1Getting Started Guides

Designed to be completed in 15-30 minutes, these guides provide quick, hands-on instructions for building the "Hello World" of any development task with Spring. In most cases, the only prerequisites are a JDK and a text editor.

##### 2.1.0Working a Getting Started guide with STS

This guide walks you through using Spring Tool Suite (STS) to build one of the Getting Started guides.

**What you’ll build**

You’ll pick a Spring guide and import it into Spring Tool Suite. Then you can read the guide, work on the code, and run the project.

**What you’ll need**

* About 15 minutes
* [Spring Tool Suite (STS)](http://spring.io/tools/sts/all)
* [JDK 6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later

**Installing STS**

If you don’t have STS installed yet, visit the link up above. From there, you can download a copy for your platform. Then install it according to directions from the installer.

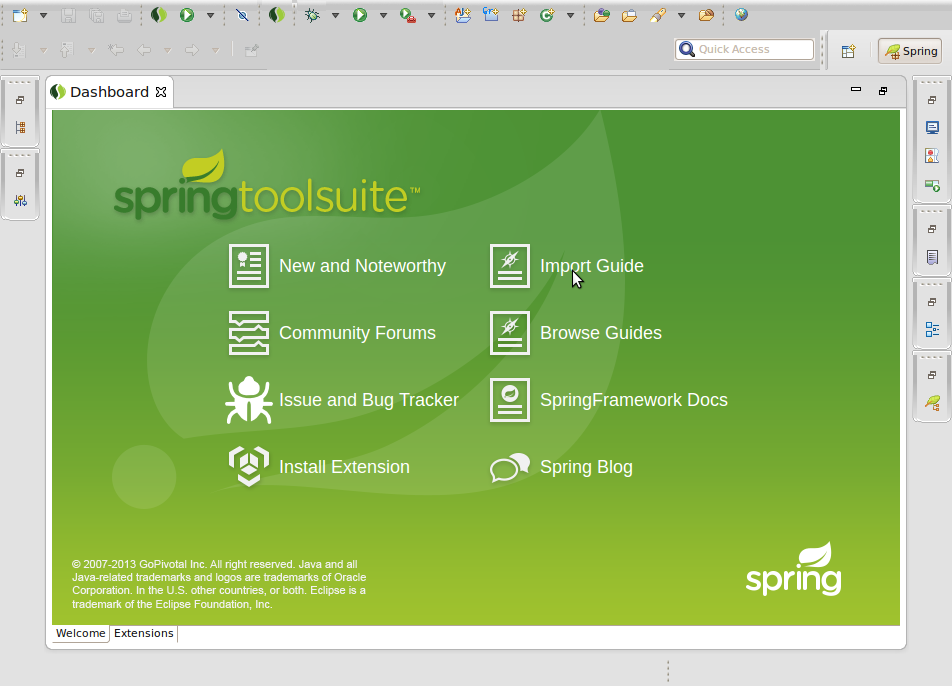
When you’re done, go ahead and launch STS.

**Importing a Getting Started guide**

With STS up and running, click on the tiny "Spring" logo as shown below:



Clicking on that logo will open the Spring Tool Suite dashboard. There are many things to pick from. But for this guide, select **Import Guide** as shown below:



After picking **Import Guide**, you will see a pop-up wizard. This will offer you the chance to search and pick any of the published guides from the Spring website. You can either skim the list, or enter search words to instantly filter the options.

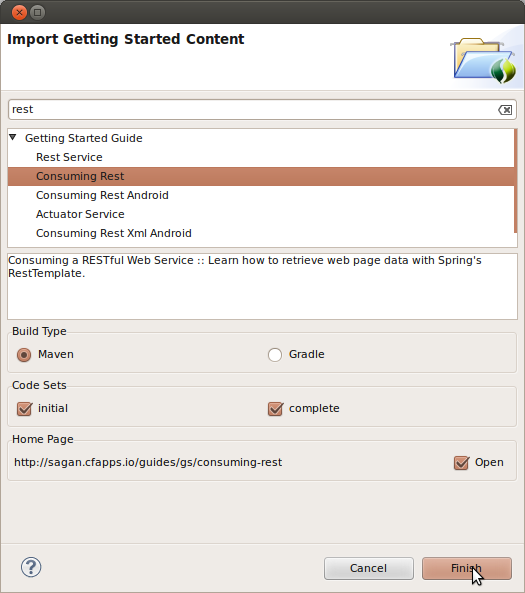
|  |  |
| --- | --- |
|  | The criteria is applied to both the title and the description when offering instant search results. Wildcards are supported. |

You can pick either [Maven](http://spring.io/guides/gs/maven) or [Gradle](http://spring.io/guides/gs/gradle) as the build system to use.

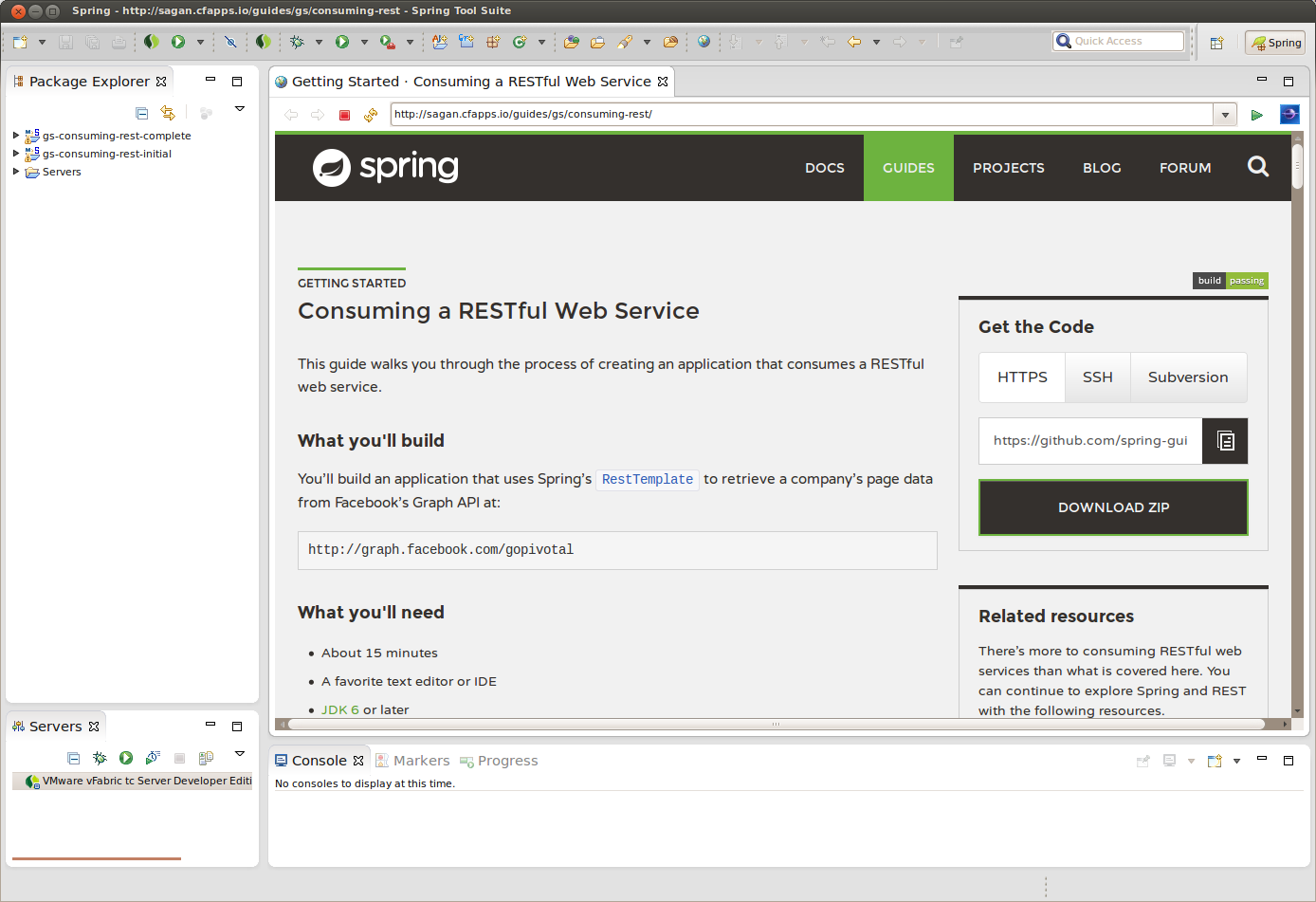
You can also decide whether to grab the **initial** code set, **complete** code set, or both. For most projects, the **initial** code set is an empty project, making it possible for you to copy-and-paste your way through a guide. The **complete** code set is all the code from the guide already entered. If you grab both, you can compare your work against the guide’s and see the differences.

Finally, you can have STS open a browser tab to the guide on the website. This will let you work through a guide without having to leave STS.

For the purpose of this guide, enter **rest** into the instant search box. Then pick [Consuming Rest](http://spring.io/guides/gs/consuming-rest). Pick **Maven** for building, and **initial** and **complete** code sets. Also opt to open the web page as shown below:



STS will create two new projects in your workspace, import the [Consuming Rest](http://spring.io/guides/gs/consuming-rest) code base (both initial and complete), and open a browser tab inside STS as shown below:



From here, you can walk through the guide and navigate to the code files.

**Summary**

Congratulations! You have setup Spring Tool Suite, imported the Consuming Rest getting started guide, and opened a browser tab to walk through it.

##### 2.1.1Building a RESTful Web Service

This guide walks you through the process of creating a "hello world" [RESTful web service](http://spring.io/understanding/REST) with Spring.

**What you’ll build**

You’ll build a service that will accept HTTP GET requests at:

http://localhost:8080/greeting

and respond with a [JSON](http://spring.io/understanding/JSON) representation of a greeting:

{"id":1,"content":"Hello, World!"}

You can customize the greeting with an optional name parameter in the query string:

http://localhost:8080/greeting?name=User

The name parameter value overrides the default value of "World" and is reflected in the response:

{"id":1,"content":"Hello, User!"}

**What you’ll need**

* About 15 minutes
* A favorite text editor or IDE
* [JDK 1.6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later
* [Gradle 1.8+](http://www.gradle.org/downloads) or [Maven 3.0+](http://maven.apache.org/download.cgi)
* You can also import the code from this guide as well as view the web page directly into [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts) and work your way through it from there.

**How to complete this guide**

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Set up the project](http://spring.io/guides/gs/rest-service/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-rest-service/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-rest-service.git>
* cd into gs-rest-service/initial
* Jump ahead to [Create a resource representation class](http://spring.io/guides/gs/rest-service/#initial).

**When you’re finished**, you can check your results against the code in gs-rest-service/complete.

**Set up the project**

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

**Create the directory structure**

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

**Create a Gradle build file**

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-rest-service/blob/master/initial/build.gradle). But you can also use Maven. The pom.xml file is included [right here](https://github.com/spring-guides/gs-rest-service/blob/master/initial/pom.xml). If you are using [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts), you can import the guide directly.

build.gradle

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

jar {

baseName = 'gs-rest-service'

version = '0.1.0'

}

repositories {

mavenCentral()

maven { url "http://repo.spring.io/libs-snapshot" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter-web:0.5.0.M6")

compile("com.fasterxml.jackson.core:jackson-databind")

testCompile("junit:junit:4.11")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

|  |  |
| --- | --- |
|  | This guide is using [Spring Boot](http://spring.io/guides/gs/spring-boot). |

**Create a resource representation class**

Now that you’ve set up the project and build system, you can create your web service.

Begin the process by thinking about service interactions.

The service will handle GET requests for /greeting, optionally with a name parameter in the query string. The GET request should return a 200 OK response with JSON in the body that represents a greeting. It should look something like this:

{

"id": 1,

"content": "Hello, World!"

}

The id field is a unique identifier for the greeting, and content is the textual representation of the greeting.

To model the greeting representation, you create a resource representation class. Provide a plain old java object with fields, constructors, and accessors for the id and content data:

src/main/java/hello/Greeting.java

package hello;

public class Greeting {

private final long id;

private final String content;

public Greeting(long id, String content) {

this.id = id;

this.content = content;

}

public long getId() {

return id;

}

public String getContent() {

return content;

}

}

|  |  |
| --- | --- |
|  | As you see in steps below, Spring uses the [Jackson JSON](http://wiki.fasterxml.com/JacksonHome) library to automatically marshal instances of type Greeting into JSON. |

Next you create the resource controller that will serve these greetings.

**Create a resource controller**

In Spring’s approach to building RESTful web services, HTTP requests are handled by a controller. These components are easily identified by the [@Controller](http://docs.spring.io/spring/docs/3.2.4.RELEASE/javadoc-api/org/springframework/stereotype/Controller.html) annotation, and the GreetingController below handles GET requests for /greeting by returning a new instance of the Greeting class:

src/main/java/hello/GreetingController.java

package hello;

import java.util.concurrent.atomic.AtomicLong;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

@Controller

public class GreetingController {

private static final String template = "Hello, %s!";

private final AtomicLong counter = new AtomicLong();

@RequestMapping("/greeting")

public @ResponseBody Greeting greeting(

@RequestParam(value="name", required=false, defaultValue="World") String name) {

return new Greeting(counter.incrementAndGet(),

String.format(template, name));

}

}

This controller is concise and simple, but there’s plenty going on under the hood. Let’s break it down step by step.

The @RequestMapping annotation ensures that HTTP requests to /greeting are mapped to the greeting() method.

|  |  |
| --- | --- |
|  | The above example does not specify GET vs. PUT, POST, and so forth, because @RequestMapping maps all HTTP operations by default. Use @RequestMapping(method=GET) to narrow this mapping. |

@RequestParam binds the value of the query string parameter name into the name parameter of the greeting() method. This query string parameter is not required; if it is absent in the request, the defaultValue of "World" is used.

The implementation of the method body creates and returns a new Greeting object with id and content attributes based on the next value from the counter, and formats the given name by using the greeting template.

A key difference between a traditional MVC controller and the RESTful web service controller above is the way that the HTTP response body is created. Rather than relying on a [view technology](http://spring.io/understanding/view-templates) to perform server-side rendering of the greeting data to HTML, this RESTful web service controller simply populates and returns a Greeting object. The object data will be written directly to the HTTP response as JSON.

To accomplish this, the [@ResponseBody](http://docs.spring.io/spring/docs/3.2.4.RELEASE/javadoc-api/org/springframework/web/bind/annotation/ResponseBody.html) annotation on the greeting() method tells Spring MVC that it does not need to render the greeting object through a server-side view layer, but that instead that the greeting object returned *is* the response body, and should be written out directly.

The Greeting object must be converted to JSON. Thanks to Spring’s HTTP message converter support, you don’t need to do this conversion manually. Because [Jackson 2](http://wiki.fasterxml.com/JacksonHome) is on the classpath, Spring’s [MappingJackson2HttpMessageConverter](http://docs.spring.io/spring/docs/3.2.4.RELEASE/javadoc-api/org/springframework/http/converter/json/MappingJackson2HttpMessageConverter.html) is automatically chosen to convert the Greeting instance to JSON.

**Make the application executable**

Although it is possible to package this service as a traditional [WAR](http://spring.io/understanding/WAR) file for deployment to an external application server, the simpler approach demonstrated below creates a standalone application. You package everything in a single, executable JAR file, driven by a good old Java main() method. Along the way, you use Spring’s support for embedding the [Tomcat](http://spring.io/understanding/Tomcat) servlet container as the HTTP runtime, instead of deploying to an external instance.

src/main/java/hello/Application.java

package hello;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.SpringApplication;

import org.springframework.context.annotation.ComponentScan;

@ComponentScan

@EnableAutoConfiguration

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

The main() method defers to the [SpringApplication](http://docs.spring.io/spring-boot/docs/0.5.0.M4/api/org/springframework/boot/SpringApplication.html) helper class, providing Application.class as an argument to its run() method. This tells Spring to read the annotation metadata from Application and to manage it as a component in the [Spring application context](http://spring.io/understanding/application-context).

The @ComponentScan annotation tells Spring to search recursively through the hello package and its children for classes marked directly or indirectly with Spring’s [@Component](http://docs.spring.io/spring/docs/3.2.4.RELEASE/javadoc-api/org/springframework/stereotype/Component.html) annotation. This directive ensures that Spring finds and registers the GreetingController, because it is marked with @Controller, which in turn is a kind of @Component annotation.

The [@EnableAutoConfiguration](http://docs.spring.io/spring-boot/docs/0.5.0.M4/api/org/springframework/boot/autoconfigure/EnableAutoConfiguration.html) annotation switches on reasonable default behaviors based on the content of your classpath. For example, because the application depends on the embeddable version of Tomcat (tomcat-embed-core.jar), a Tomcat server is set up and configured with reasonable defaults on your behalf. And because the application also depends on Spring MVC (spring-webmvc.jar), a Spring MVC [DispatcherServlet](http://docs.spring.io/spring/docs/3.2.4.RELEASE/javadoc-api/org/springframework/web/servlet/DispatcherServlet.html) is configured and registered for you — no web.xml necessary! Auto-configuration is a powerful, flexible mechanism. See the [API documentation](http://docs.spring.io/spring-boot/docs/0.5.0.M4/api/org/springframework/boot/autoconfigure/EnableAutoConfiguration.html) for further details.

**Build an executable JAR**

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-rest-service/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-rest-service/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-rest-service-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-rest-service-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

**Run the service**

If you are using Gradle, you can run your service at the command line this way:

./gradlew clean build && java -jar build/libs/gs-rest-service-0.1.0.jar

|  |  |
| --- | --- |
|  | If you are using Maven, you can run your service by typing mvn clean package && java -jar target/gs-rest-service-0.1.0.jar. |

Logging output is displayed. The service should be up and running within a few seconds.

**Test the service**

Now that the service is up, visit <http://localhost:8080/greeting>, where you see:

{"id":1,"content":"Hello, World!"}

Provide a name query string parameter with <http://localhost:8080/greeting?name=User>. Notice how the value of the content attribute changes from "Hello, World!" to "Hello User!":

{"id":2,"content":"Hello, User!"}

This change demonstrates that the @RequestParam arrangement in GreetingController is working as expected. The name parameter has been given a default value of "World", but can always be explicitly overridden through the query string.

Notice also how the id attribute has changed from 1 to 2. This proves that you are working against the same GreetingController instance across multiple requests, and that its counter field is being incremented on each call as expected.

**Summary**

Congratulations! You’ve just developed a RESTful web service with Spring.

##### 2.1.2Scheduling Tasks

This guide walks you through the steps for scheduling tasks with Spring.

**What you’ll build**

You’ll build an application the prints out the current time every five seconds using Spring’s @Scheduled annotation.

**What you’ll need**

* About 15 minutes
* A favorite text editor or IDE
* [JDK 1.6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later
* [Gradle 1.8+](http://www.gradle.org/downloads) or [Maven 3.0+](http://maven.apache.org/download.cgi)
* You can also import the code from this guide as well as view the web page directly into [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts) and work your way through it from there.

**How to complete this guide**

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Set up the project](http://spring.io/guides/gs/scheduling-tasks/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-scheduling-tasks/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-scheduling-tasks.git>
* cd into gs-scheduling-tasks/initial
* Jump ahead to [Create a scheduled task](http://spring.io/guides/gs/scheduling-tasks/#initial).

**When you’re finished**, you can check your results against the code in gs-scheduling-tasks/complete.

**Set up the project**

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

**Create the directory structure**

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

**Create a Gradle build file**

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-scheduling-tasks/blob/master/initial/build.gradle). But you can also use Maven. The pom.xml file is included [right here](https://github.com/spring-guides/gs-scheduling-tasks/blob/master/initial/pom.xml). If you are using [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts), you can import the guide directly.

build.gradle

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

jar {

baseName = 'gs-scheduling-tasks'

version = '0.1.0'

}

repositories {

mavenCentral()

maven { url "http://repo.spring.io/libs-snapshot" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter:0.5.0.M6")

testCompile("junit:junit:4.11")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

|  |  |
| --- | --- |
|  | This guide is using [Spring Boot](http://spring.io/guides/gs/spring-boot). |

**Create a scheduled task**

Now that you’ve set up your project, you can create a scheduled task.

src/main/java/hello/ScheduledTasks.java

package hello;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.springframework.scheduling.annotation.EnableScheduling;

import org.springframework.scheduling.annotation.Scheduled;

@EnableScheduling

public class ScheduledTasks {

private static final SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");

@Scheduled(fixedRate = 5000)

public void reportCurrentTime() {

System.out.println("The time is now " + dateFormat.format(new Date()));

}

}

The key components that make this code perform scheduled tasks are the @EnableScheduling and @Scheduled annotations.

@EnableScheduling ensures that a background task executor is created. Without it, nothing gets scheduled.

You use @Scheduled to configure when a particular method is run. NOTE: This example uses fixedRate, which specifies the interval between method invocations measured from the start time of each invocation. There are [other options](http://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/scheduling.html#scheduling-annotation-support-scheduled), like fixedDelay, which specifies the interval between invocations measured from the completion of the task. You can also [use @Scheduled(cron=". . .") expressions for more sophisticated task scheduling](http://docs.spring.io/spring/docs/4.0.x/javadoc-api/org/springframework/scheduling/support/CronSequenceGenerator.html).

**Make the application executable**

Although scheduled tasks can be embedded in web apps and WAR files, the simpler approach demonstrated below creates a standalone application. You package everything in a single, executable JAR file, driven by a good old Java main() method.

Here you create a new SpringApplication and run it with the ScheduledTasks you defined earlier. This action creates a task executor and allows tasks to be scheduled."

src/main/java/hello/Application.java

package hello;

import org.springframework.boot.SpringApplication;

public class Application {

public static void main(String[] args) throws Exception {

SpringApplication.run(ScheduledTasks.class);

}

}

**Build an executable JAR**

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-scheduling-tasks/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-scheduling-tasks/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-scheduling-tasks-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-scheduling-tasks-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

**Run the service**

If you are using Gradle, you can run your service at the command line this way:

./gradlew clean build && java -jar build/libs/gs-scheduling-tasks-0.1.0.jar

|  |  |
| --- | --- |
|  | If you are using Maven, you can run your service by typing mvn clean package && java -jar target/gs-scheduling-tasks-0.1.0.jar. |

Logging output is displayed. You should see your scheduled task fire every 5 seconds:

[...]

The time is now 13:10:00

The time is now 13:10:05

The time is now 13:10:10

The time is now 13:10:15

**Summary**

Congratulations! You created an application with a scheduled task. Heck, the actual code was shorter than the build file! This technique works in any type of application.

##### 2.1.3Detecting a Device

This guide walks you through the process of using Spring to detect the type of device that is accessing your web site.

**What you’ll build**

You’ll create a Spring MVC application that detects the type of device that is accessing your web site and that switches views dynamically based on that device type.

**What you’ll need**

* About 15 minutes
* A favorite text editor or IDE
* [JDK 1.6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later
* [Gradle 1.8+](http://www.gradle.org/downloads) or [Maven 3.0+](http://maven.apache.org/download.cgi)
* You can also import the code from this guide as well as view the web page directly into [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts) and work your way through it from there.

**How to complete this guide**

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Set up the project](http://spring.io/guides/gs/device-detection/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-device-detection/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-device-detection.git>
* cd into gs-device-detection/initial
* Jump ahead to [Create a web controller](http://spring.io/guides/gs/device-detection/#initial).

**When you’re finished**, you can check your results against the code in gs-device-detection/complete.

**Set up the project**

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

**Create the directory structure**

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

**Create a Gradle build file**

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-device-detection/blob/master/initial/build.gradle). But you can also use Maven. The pom.xml file is included [right here](https://github.com/spring-guides/gs-device-detection/blob/master/initial/pom.xml). If you are using [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts), you can import the guide directly.

build.gradle

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-milestone" }

mavenLocal()

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

jar {

baseName = 'gs-device-detection'

version = '0.1.0'

}

repositories {

mavenCentral()

maven { url "http://repo.spring.io/libs-milestone" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter-web:0.5.0.M6")

compile("org.springframework.mobile:spring-mobile-device:1.1.0.RELEASE")

testCompile("junit:junit:4.11")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

|  |  |
| --- | --- |
|  | This guide is using [Spring Boot](http://spring.io/guides/gs/spring-boot). |

By including the Spring Mobile dependency, Spring Boot configures a [DeviceResolverHandlerInterceptor](http://docs.spring.io/spring-mobile/docs/1.1.x/api/org/springframework/mobile/device/DeviceResolverHandlerInterceptor.html) and [DeviceHandlerMethodArgumentResolver](http://docs.spring.io/spring-mobile/docs/1.1.x/api/org/springframework/mobile/device/DeviceHandlerMethodArgumentResolver.html) automatically. [DeviceResolverHandlerInterceptor](http://docs.spring.io/spring-mobile/docs/1.1.x/api/org/springframework/mobile/device/DeviceResolverHandlerInterceptor.html) examines the User-Agent header in the incoming request, and based on the header value, determines whether the request is coming from a normal (desktop) browser, a mobile (phone) browser, or a tablet browser. The [DeviceHandlerMethodArgumentResolver](http://docs.spring.io/spring-mobile/docs/1.1.x/api/org/springframework/mobile/device/DeviceHandlerMethodArgumentResolver.html) allows Spring MVC to use the resolved [Device](http://docs.spring.io/spring-mobile/docs/1.1.x/api/org/springframework/mobile/device/Device.html) object in a controller method.

**Create a web controller**

In Spring, web endpoints are simply Spring MVC controllers. The following Spring MVC controller handles a GET request and returns a String indicating the type of the device:

src/main/java/hello/DeviceDetectionController.java

package hello;

import org.springframework.mobile.device.Device;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

@Controller

public class DeviceDetectionController {

@RequestMapping("/detect-device")

public @ResponseBody String detectDevice(Device device) {

String deviceType = "unknown";

if (device.isNormal()) {

deviceType = "normal";

} else if (device.isMobile()) {

deviceType = "mobile";

} else if (device.isTablet()) {

deviceType = "tablet";

}

return "Hello " + deviceType + " browser!";

}

}

For this example, rather than rely on a view (such as JSP) to render model data in HTML, this controller simply returns the data to be written directly to the body of the response. In this case, the data is a String that reads "Hello mobile browser!" if the requesting client is a mobile device. The [@ResponseBody](http://docs.spring.io/spring/docs/3.2.x/javadoc-api/org/springframework/web/bind/annotation/ResponseBody.html) annotation tells Spring MVC to write the returned object into the response body, rather than to render a model into a view.

**Make the application executable**

Although it is possible to package this service as a traditional [WAR](http://spring.io/understanding/WAR) file for deployment to an external application server, the simpler approach demonstrated in the next section creates a *standalone application*. You package everything in a single, executable JAR file, driven by a good old Java main() method. And along the way, you use Spring’s support for embedding the [Tomcat](http://spring.io/understanding/Tomcat) servlet container as the HTTP runtime, instead of deploying to an external instance.

src/main/java/hello/Application.java

package hello;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.SpringApplication;

import org.springframework.context.annotation.ComponentScan;

@ComponentScan

@EnableAutoConfiguration

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

The main() method defers to the [SpringApplication](http://docs.spring.io/spring-boot/docs/0.5.0.M6/api/org/springframework/boot/SpringApplication.html) helper class, providing Application.class as an argument to its run() method. This tells Spring to read the annotation metadata from Application and to manage it as a component in the [*Spring application context*](http://spring.io/understanding/application-context).

The @ComponentScan annotation tells Spring to search recursively through the hello package and its children for classes marked directly or indirectly with Spring’s [@Component](http://docs.spring.io/spring/docs/4.0.0.RC1/javadoc-api/org/springframework/stereotype/Component.html) annotation. This directive ensures that Spring finds and registers the DeviceDetectionConfiguration and DeviceDetectionController classes, because they are marked with @Controller, which in turn is a kind of @Component annotation.

The [@EnableAutoConfiguration](http://docs.spring.io/spring-boot/docs/0.5.0.M6/api/org/springframework/boot/autoconfigure/EnableAutoConfiguration.html) annotation switches on reasonable default behaviors based on the content of your classpath. For example, because the application depends on the embeddable version of Tomcat (tomcat-embed-core.jar), a Tomcat server is set up and configured with reasonable defaults on your behalf. And because the application also depends on Spring MVC (spring-webmvc.jar), a Spring MVC [DispatcherServlet](http://docs.spring.io/spring/docs/4.0.0.RC1/javadoc-api/org/springframework/web/servlet/DispatcherServlet.html) is configured and registered for you — no web.xml necessary! Auto-configuration is a powerful, flexible mechanism. See the [API documentation](http://docs.spring.io/spring-boot/docs/0.5.0.M6/api/org/springframework/boot/autoconfigure/EnableAutoConfiguration.html) for further details.

**Build an executable JAR**

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-device-detection/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-device-detection/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-device-detection-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-device-detection-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

**Run the service**

If you are using Gradle, you can run your service at the command line this way:

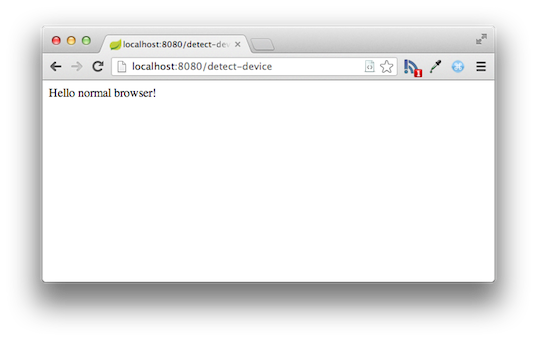
./gradlew clean build && java -jar build/libs/gs-device-detection-0.1.0.jar

|  |  |
| --- | --- |
|  | If you are using Maven, you can run your service by typing mvn clean package && java -jar target/gs-device-detection-0.1.0.jar. |

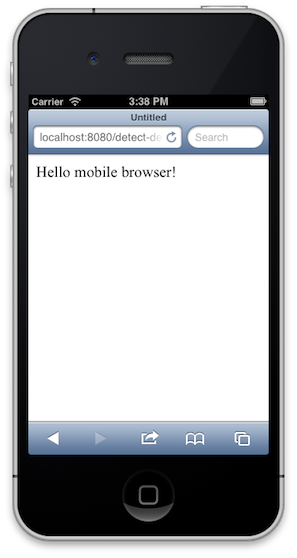
Logging output is displayed. The service should be up and running within a few seconds.

**Test the service**

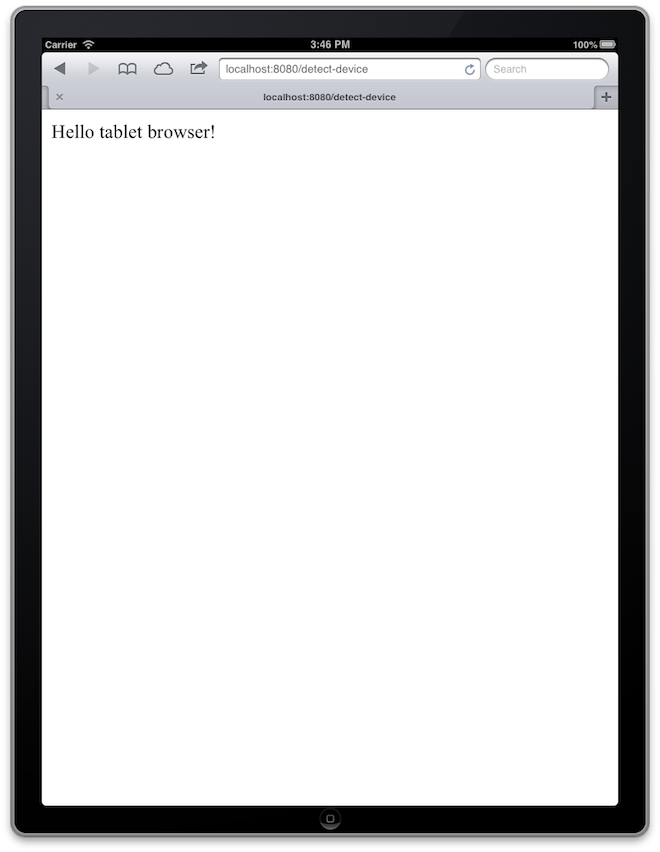
To test the application, point your browser at <http://localhost:8080/detect-device>. In a normal desktop browser, you should see something like this:



If you point a mobile browser at the same URL (such as the iOS Simulator’s browser), you should see something like this (you may have to zoom in on the mobile browser to read the message clearly):



If you point a tablet browser at the URL, you should see something like this:



Note that if you want to use a real mobile device to test this controller, it will not work with the localhost server. You’ll need to find the name of your machine on your network and use that instead of localhost.

**Summary**

Congratulations! You have just developed a simple web page that detects the type of device being used by a client.

##### 2.1.4Consuming a RESTful Web Service

This guide walks you through the process of creating an application that consumes a RESTful web service.

**What you’ll build**

You’ll build an application that uses Spring’s RestTemplate to retrieve a company’s page data from Facebook’s Graph API at:

http://graph.facebook.com/gopivotal

**What you’ll need**

* About 15 minutes
* A favorite text editor or IDE
* [JDK 1.6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later
* [Gradle 1.8+](http://www.gradle.org/downloads) or [Maven 3.0+](http://maven.apache.org/download.cgi)
* You can also import the code from this guide as well as view the web page directly into [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts) and work your way through it from there.

**How to complete this guide**

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Set up the project](http://spring.io/guides/gs/consuming-rest/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-consuming-rest/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-consuming-rest.git>
* cd into gs-consuming-rest/initial
* Jump ahead to [Fetch a REST resource](http://spring.io/guides/gs/consuming-rest/#initial).

**When you’re finished**, you can check your results against the code in gs-consuming-rest/complete.

**Set up the project**

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

**Create the directory structure**

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

**Create a Gradle build file**

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-consuming-rest/blob/master/initial/build.gradle). But you can also use Maven. The pom.xml file is included [right here](https://github.com/spring-guides/gs-consuming-rest/blob/master/initial/pom.xml). If you are using [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts), you can import the guide directly.

build.gradle

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

jar {

baseName = 'gs-consuming-rest'

version = '0.1.0'

}

repositories {

mavenCentral()

maven { url "http://repo.spring.io/libs-snapshot" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter:0.5.0.M6")

compile("org.springframework:spring-web:4.0.0.RC1")

compile("com.fasterxml.jackson.core:jackson-databind:2.2.2")

testCompile("junit:junit:4.11")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

|  |  |
| --- | --- |
|  | This guide is using [Spring Boot](http://spring.io/guides/gs/spring-boot). |

**Fetch a REST resource**

With project setup complete, you can create a simple application that consumes a RESTful service.

Suppose that you want to find out what Facebook knows about Pivotal. Knowing that Pivotal has a page on Facebook and that the ID is "gopivotal", you should be able to query Facebook’s Graph API via this URL:

http://graph.facebook.com/gopivotal

If you request that URL through your web browser or curl, you’ll receive a JSON document that looks something like this:

{

"id": "161112704050757",

"about": "At Pivotal, our mission is to enable customers to build a new class of applications, leveraging big and fast data, and do all of this with the power of cloud independence. ",

"app\_id": "0",

"can\_post": false,

"category": "Internet/software",

"checkins": 0,

"cover": {

"cover\_id": 163344023827625,

"source": "http://sphotos-d.ak.fbcdn.net/hphotos-ak-frc1/s720x720/554668\_163344023827625\_839302172\_n.png",

"offset\_y": 0,

"offset\_x": 0

},

"founded": "2013",

"has\_added\_app": false,

"is\_community\_page": false,

"is\_published": true,

"likes": 126,

"link": "https://www.facebook.com/gopivotal",

"location": {

"street": "1900 South Norfolk St.",

"city": "San Mateo",

"state": "CA",

"country": "United States",

"zip": "94403",

"latitude": 37.552261,

"longitude": -122.292152

},

"name": "Pivotal",

"phone": "650-286-8012",

"talking\_about\_count": 15,

"username": "gopivotal",

"website": "http://www.gopivotal.com",

"were\_here\_count": 0

}

Easy enough, but not terribly useful when fetched through a browser or through curl.

A more useful way to consume a REST web service is programmatically. To help you with that task, Spring provides a convenient template class called [RestTemplate](http://docs.spring.io/spring/docs/4.0.x/javadoc-api/org/springframework/web/client/RestTemplate.html). RestTemplate makes interacting with most RESTful services a one-line incantation. And it can even bind that data to custom domain types.

First, create a domain class to contain the data that you need. If all you need to know are Pivotal’s name, phone number, website URL, and what the gopivotal page is about, then the following domain class should do fine:

src/main/java/hello/Page.java

package hello;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@JsonIgnoreProperties(ignoreUnknown = true)

public class Page {

private String name;

private String about;

private String phone;

private String website;

public String getName() {

return name;

}

public String getAbout() {

return about;

}

public String getPhone() {

return phone;

}

public String getWebsite() {

return website;

}

}

As you can see, this is a simple Java class with a handful of properties and matching getter methods. It’s annotated with @JsonIgnoreProperties from the Jackson JSON processing library to indicate that any properties not bound in this type should be ignored.

**Make the application executable**

Although it is possible to package this service as a traditional [WAR](http://spring.io/understanding/WAR) file for deployment to an external application server, the simpler approach demonstrated below creates a standalone application. You package everything in a single, executable JAR file, driven by a good old Java main() method. Along the way, you use Spring’s support for embedding the [Tomcat](http://spring.io/understanding/Tomcat) servlet container as the HTTP runtime, instead of deploying to an external instance.

Now you can write the Application class that uses RestTemplate to fetch the data from Pivotal’s page at Facebook into a Page object.

src/main/java/hello/Application.java

package hello;

import org.springframework.web.client.RestTemplate;

public class Application {

public static void main(String args[]) {

RestTemplate restTemplate = new RestTemplate();

Page page = restTemplate.getForObject("http://graph.facebook.com/gopivotal", Page.class);

System.out.println("Name: " + page.getName());

System.out.println("About: " + page.getAbout());

System.out.println("Phone: " + page.getPhone());

System.out.println("Website: " + page.getWebsite());

}

}

Because the Jackson JSON processing library is in the classpath, RestTemplate will use it (via a [message converter](http://docs.spring.io/spring/docs/4.0.x/javadoc-api/org/springframework/http/converter/HttpMessageConverter.html)) to convert the incoming JSON data into a Page object. From there, the contents of the Page object will be printed to the console.

Here you’ve only used RestTemplate to make an HTTP GET request. But RestTemplate also supports other HTTP verbs such as POST, PUT, and DELETE.

**Build an executable JAR**

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-consuming-rest/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-consuming-rest/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-consuming-rest-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-consuming-rest-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

**Run the service**

If you are using Gradle, you can run your service at the command line this way:

./gradlew clean build && java -jar build/libs/gs-consuming-rest-0.1.0.jar

|  |  |
| --- | --- |
|  | If you are using Maven, you can run your service by typing mvn clean package && java -jar target/gs-consuming-rest-0.1.0.jar. |

You should see the following output:

Name: Pivotal

About: At Pivotal, our mission is to enable customers to build a new class of applications, leveraging big and fast data, and do all of this with the power of cloud independence.

Phone: 650-286-8012

Website: http://www.gopivotal.com

**Summary**

Congratulations! You have just developed a simple REST client using Spring.

##### 2.1.5Building Java Projects with Gradle

This guide walks you through using Gradle to build a simple Java project.

**What you’ll build**

You’ll create a simple app and then build it using Gradle.

**What you’ll need**

* About 15 minutes
* A favorite text editor or IDE
* [JDK 6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later

**How to complete this guide**

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Set up the project](http://spring.io/guides/gs/gradle/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-gradle/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-gradle.git>
* cd into gs-gradle/initial
* Jump ahead to [Install Gradle](http://spring.io/guides/gs/gradle/#initial).

**When you’re finished**, you can check your results against the code in gs-gradle/complete.

**Set up the project**

First you set up a Java project for Gradle to build. To keep the focus on Gradle, make the project as simple as possible for now.

**Create the directory structure**

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

Within the src/main/java/hello directory, you can create any Java classes you want. For simplicity’s sake and for consistency with the rest of this guide, Spring recommends that you create two classes: HelloWorld.java and Greeter.java.

src/main/java/hello/HelloWorld.java

package hello;

public class HelloWorld {

public static void main(String[] args) {

Greeter greeter = new Greeter();

System.out.println(greeter.sayHello());

}

}

src/main/java/hello/Greeter.java

package hello;

public class Greeter {

public String sayHello() {

return "Hello world!";

}

}

**Install Gradle**

Now that you have a project that you can build with Gradle, you can install Gradle.

Gradle is downloadable as a zip file at <http://www.gradle.org/downloads>. Only the binaries are required, so look for the link to gradle-*version*-bin.zip. (You can also choose gradle-*version*-all.zip to get the sources and documentation as well as the binaries.)

Unzip the file to your computer, and add the bin folder to your path.

To test the Gradle installation, run Gradle from the command-line:

gradle

If all goes well, you see a welcome message:

:help

Welcome to Gradle 1.8.

To run a build, run gradle <task> ...

To see a list of available tasks, run gradle tasks

To see a list of command-line options, run gradle --help

BUILD SUCCESSFUL

Total time: 2.675 secs

You now have Gradle installed.

**Find out what Gradle can do**

Now that Gradle is installed, see what it can do. Before you even create a build.gradle file for the project, you can ask it what tasks are available:

gradle tasks

You should see a list of available tasks. Assuming you run Gradle in a folder that doesn’t already have a *build.gradle* file, you’ll see some very elementary tasks such as this:

:tasks

== All tasks runnable from root project

== Build Setup tasks

setupBuild - Initializes a new Gradle build. [incubating]

wrapper - Generates Gradle wrapper files. [incubating]

== Help tasks

dependencies - Displays all dependencies declared in root project 'gs-gradle'.

dependencyInsight - Displays the insight into a specific dependency in root project 'gs-gradle'.

help - Displays a help message

projects - Displays the sub-projects of root project 'gs-gradle'.

properties - Displays the properties of root project 'gs-gradle'.

tasks - Displays the tasks runnable from root project 'gs-gradle'.

To see all tasks and more detail, run with --all.

BUILD SUCCESSFUL

Total time: 3.077 secs

Even though these tasks are available, they don’t offer much value without a project build configuration. As you flesh out the build.gradle file, some tasks will be more useful. The list of tasks will grow as you add plugins to build.gradle, so you’ll occasionally want to run **tasks** again to see what tasks are available.

Speaking of adding plugins, next you add a plugin that enables basic Java build functionality.

**Build Java code**

Starting simple, create a very basic build.gradle file that has only one line in it:

apply plugin: 'java'

This single line in the build configuration brings a significant amount of power. Run **gradle tasks** again, and you see new tasks added to the list, including tasks for building the project, creating JavaDoc, and running tests.

You’ll use the **gradle build** task frequently. This task compiles, tests, and assembles the code into a JAR file. You can run it like this:

gradle build

After a few seconds, "BUILD SUCCESSFUL" indicates that the build has completed.

To see the results of the build effort, take a look in the *build* folder. Therein you’ll find several directories, including these three notable folders:

* *classes*. The project’s compiled .class files.
* *reports*. Reports produced by the build (such as test reports).
* *libs*. Assembled project libraries (usually JAR and/or WAR files).

The classes folder has .class files that are generated from compiling the Java code. Specifically, you should find HelloWorld.class and Greeter.class.

At this point, the project doesn’t have any library dependencies, so there’s nothing in the **dependency\_cache** folder.

The reports folder should contain a report of running unit tests on the project. Because the project doesn’t yet have any unit tests, that report will be uninteresting.

The libs folder should contain a JAR file that is named after the project’s folder. If you cloned the project from GitHub, then the JAR file is likely named initial.jar. That’s probably not what you’d want it named, though.

**Declare dependencies**

The simple Hello World sample is completely self-contained and does not depend on any additional libraries. Most applications, however, depend on external libraries to handle common and/or complex functionality.

For example, suppose that in addition to saying "Hello World!", you want the application to print the current date and time. You could use the date and time facilities in the native Java libraries, but you can make things more interesting by using the Joda Time libraries.

First, change HelloWorld.java to look like this:

package hello;

import org.joda.time.LocalTime;

public class HelloWorld {

public static void main(String[] args) {

LocalTime currentTime = new LocalTime();

System.out.println("The current local time is: " + currentTime);

Greeter greeter = new Greeter();

System.out.println(greeter.sayHello());

}

}

Here HelloWorld uses Joda Time’s LocalTime class to get and print the current time.

If you ran gradle build to build the project now, the build would fail because you have not declared Joda Time as a compile dependency in the build. You can fix that by adding the following lines to build.gradle:

repositories { mavenCentral() }

dependencies {

compile "joda-time:joda-time:2.2"

}

The first line here indicates that the build should resolve its dependencies from the Maven Central repository. Gradle leans heavily on many conventions and facilities established by the Maven build tool, including the option of using Maven Central as a source of library dependencies.

Within the dependencies block, you declare a single dependency for Joda Time. Specifically, you’re asking for (reading right to left) version 2.2 of the joda-time library, in the joda-time group.

Another thing to note about this dependency is that it is a compile dependency, indicating that it should be available during compile-time (and if you were building a WAR file, included in the /WEB-INF/libs folder of the WAR). Other notable types of dependencies include:

* providedCompile. Required dependencies for compiling the project code, but that will be provided at runtime by a container running the code (for example, the Java Servlet API).
* testCompile. Dependencies used for compiling and running tests, but not required for building or running the project’s runtime code.

Now if you run gradle build, Gradle should resolve the Joda Time dependency from the Maven Central repository and the build will succeed.

**Build your project with Gradle Wrapper**

The Gradle Wrapper is the preferred way of starting a Gradle build. It consists of a batch script for Windows support and a shell script for support on OS X and Linux. These scripts allow you to run a Gradle build without requiring that Gradle be installed on your system. You can install the wrapper into your project by adding the following lines to the build.gradle:

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

Run the following command to download and initialize the wrapper scripts:

gradle wrapper

After this task completes, you will notice a few new files. The two scripts are in the root of the folder, while the wrapper jar and properties files have been added to a new gradle/wrapper folder.

└── initial

└── gradlew

└── gradlew.bat

└── gradle

└── wrapper

└── gradle-wrapper.jar

└── gradle-wrapper.properties

The Gradle Wrapper is now available for building your project. It can be used in the exact same way as an installed version of Gradle. Run the wrapper script to perform the build task, just like you did previously:

./gradlew build

The first time you run the wrapper for a specified version of Gradle, it downloads and caches the Gradle binaries for that version. The Gradle Wrapper files are designed to be committed to source control so that anyone can build the project without having to first install and configure a specific version of Gradle.

Here is the completed build.gradle file:

build.gradle

apply plugin: 'java'

apply plugin: 'eclipse'

repositories { mavenCentral() }

dependencies {

compile "joda-time:joda-time:2.2"

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

**Summary**

Congratulations! You have now created a simple yet effective Gradle build file for building Java projects.

##### 2.1.6Building Java Projects with Maven

This guide walks you through using Maven to build a simple Java project.

**What you’ll build**

You’ll create an application that provides the time of day and then build it with Maven.

**What you’ll need**

* About 15 minutes
* A favorite text editor or IDE
* [JDK 6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later

**How to complete this guide**

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Set up the project](http://spring.io/guides/gs/maven/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-maven/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-maven.git>
* cd into gs-maven/initial
* Jump ahead to [[initial]](http://spring.io/guides/gs/maven/#initial).

**When you’re finished**, you can check your results against the code in gs-maven/complete.

**Set up the project**

First you’ll need to setup a Java project for Maven to build. To keep the focus on Maven, make the project as simple as possible for now.

**Create the directory structure**

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

Within the src/main/java/hello directory, you can create any Java classes you want. To maintain consistency with the rest of this guide, create these two classes: HelloWorld.java and Greeter.java.

src/main/java/hello/HelloWorld.java

package hello;

public class HelloWorld {

public static void main(String[] args) {

Greeter greeter = new Greeter();

System.out.println(greeter.sayHello());

}

}

src/main/java/hello/Greeter.java

package hello;

public class Greeter {

public String sayHello() {

return "Hello world!";

}

}

Now that you have a project that is ready to be built with Maven, the next step is to install Maven.

Maven is downloadable as a zip file at <http://maven.apache.org/download.cgi>. Only the binaries are required, so look for the link to apache-maven-*{version}*-bin.zip or apache-maven-*{version}*-bin.tar.gz.

Once you have downloaded the zip file, unzip it to your computer. Then add the *bin* folder to your path.

To test the Maven installation, run mvn from the command-line:

mvn -v

If all goes well, you should be presented with some information about the Maven installation. It will look similar to (although perhaps slightly different from) the following:

Apache Maven 3.0.5 (r01de14724cdef164cd33c7c8c2fe155faf9602da; 2013-02-19 07:51:28-0600)

Maven home: /usr/share/maven

Java version: 1.7.0\_09, vendor: Oracle Corporation

Java home: /Library/Java/JavaVirtualMachines/jdk1.7.0\_09.jdk/Contents/Home/jre

Default locale: en\_US, platform encoding: UTF-8

OS name: "mac os x", version: "10.8.3", arch: "x86\_64", family: "mac"

Congratulations! You now have Maven installed.

**Define a simple Maven build**

Now that Maven is installed, you need to create a Maven project definition. Maven projects are defined with an XML file named *pom.xml*. Among other things, this file gives the project’s name, version, and dependencies that it has on external libraries.

Create a file named *pom.xml* at the root of the project and give it the following contents:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.springframework.gs</groupId>

<artifactId>gs-maven-initial</artifactId>

<version>0.1.0</version>

<packaging>jar</packaging>

</project>

With the exception of the optional <packaging> element, this is the simplest possible *pom.xml* file necessary to build a Java project. It includes the following details of the project configuration:

* <modelVersion>. POM model version (always 4.0.0).
* <groupId>. Group or organization that the project belongs to. Often expressed as an inverted domain name.
* <artifactId>. Name to be given to the project’s library artifact (for example, the name of its JAR or WAR file).
* <version>. Version of the project that is being built.
* <packaging> - How the project should be packaged. Defaults to "jar" for JAR file packaging. Use "war" for WAR file packaging.

|  |  |
| --- | --- |
|  | When it comes to choosing a versioning scheme, Spring recommends the [semantic versioning]([http://semver.org](http://semver.org/)) approach. |

At this point you have a minimal, yet capable Maven project defined.

**Build Java code**

Maven is now ready to build the project. You can execute several build lifecycle goals with Maven now, including goals to compile the project’s code, create a library package (such as a JAR file), and install the library in the local Maven dependency repository.

To try out the build, issue the following at the command line:

mvn compile

This will run Maven, telling it to execute the *compile* goal. When it’s finished, you should find the compiled *.class* files in the *target/classes* directory.

Since it’s unlikely that you’ll want to distribute or work with *.class* files directly, you’ll probably want to run the *package* goal instead:

mvn package

The *package* goal will compile your Java code, run any tests, and finish by packaging the code up in a JAR file within the *target* directory. The name of the JAR file will be based on the project’s <artifactId> and <version>. For example, given the minimal *pom.xml* file from before, the JAR file will be named *gs-maven-initial-0.1.0.jar*.

|  |  |
| --- | --- |
|  | If you’ve changed the value of <packaging> from "jar" to "war", the result will be a WAR file within the *target* directory instead of a JAR file. |

Maven also maintains a repository of dependencies on your local machine (usually in a *.m2/repository* directory in your home directory) for quick access to project dependencies. If you’d like to install your project’s JAR file to that local repository, then you should invoke the install goal:

mvn install

The *install* goal will compile, test, and package your project’s code and then copy it into the local dependency repository, ready for another project to reference it as a dependency.

Speaking of dependencies, now it’s time to declare dependencies in the Maven build.

**Declare Dependencies**

The simple Hello World sample is completely self-contained and does not depend on any additional libraries. Most applications, however, depend on external libraries to handle common and complex functionality.

For example, suppose that in addition to saying "Hello World!", you want the application to print the current date and time. While you could use the date and time facilities in the native Java libraries, you can make things more interesting by using the Joda Time libraries.

First, change HelloWorld.java to look like this:

package hello;

import org.joda.time.LocalTime;

public class HelloWorld {

public static void main(String[] args) {

LocalTime currentTime = new LocalTime();

System.out.println("The current local time is: " + currentTime);

Greeter greeter = new Greeter();

System.out.println(greeter.sayHello());

}

}

Here HelloWorld uses Joda Time’s LocalTime class to get and print the current time.

If you were to run mvn compile to build the project now, the build would fail because you’ve not declared Joda Time as a compile dependency in the build. You can fix that by adding the following lines to *pom.xml* (within the <project> element):

<dependencies>

<dependency>

<groupId>joda-time</groupId>

<artifactId>joda-time</artifactId>

<version>2.2</version>

</dependency>

</dependencies>

This block of XML declares a list of dependencies for the project. Specifically, it declares a single dependency for the Joda Time library. Within the <dependency> element, the dependency coordinates are defined by three sub-elements:

* <groupId> - The group or organization that the dependency belongs to.
* <artifactId> - The library that is required.
* <version> - The specific version of the library that is required.

By default, all dependencies are scoped as compile dependencies. That is, they should be available at compile-time (and if you were building a WAR file, including in the */WEB-INF/libs* folder of the WAR). Additionally, you may specify a <scope> element to specify one of the following scopes:

* provided - Dependencies that are required for compiling the project code, but that will be provided at runtime by a container running the code (e.g., the Java Servlet API).
* test - Dependencies that are used for compiling and running tests, but not required for building or running the project’s runtime code.

Now if you run mvn compile or mvn package, Maven should resolve the Joda Time dependency from the Maven Central repository and the build will be successful.

Here’s the completed pom.xml file:

pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.springframework</groupId>

<artifactId>gs-maven</artifactId>

<packaging>jar</packaging>

<version>0.1.0</version>

<dependencies>

<dependency>

<groupId>joda-time</groupId>

<artifactId>joda-time</artifactId>

<version>2.2</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-shade-plugin</artifactId>

<version>2.1</version>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

<configuration>

<transformers>

<transformer

implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">

<mainClass>hello.HelloWorld</mainClass>

</transformer>

</transformers>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

|  |  |
| --- | --- |
|  | The completed **pom.xml** file is using the [Maven Shade Plugin](http://maven.apache.org/plugins/maven-shade-plugin/) for the simple convenience of making the JAR file executable. The focus of this guide is getting started with Maven, not using this particular plugin. |

**Summary**

Congratulations! You’ve created a simple yet effective Maven project definition for building Java projects.

##### 2.1.7Accessing Relational Data using JDBC with Spring

This guide walks you through the process of accessing relational data with Spring.

**What you’ll build**

You’ll build an application using Spring’s JdbcTemplate to access data stored in a relational database.

**What you’ll need**

* About 15 minutes
* A favorite text editor or IDE
* [JDK 1.6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later
* [Gradle 1.8+](http://www.gradle.org/downloads) or [Maven 3.0+](http://maven.apache.org/download.cgi)
* You can also import the code from this guide as well as view the web page directly into [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts) and work your way through it from there.

**How to complete this guide**

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Set up the project](http://spring.io/guides/gs/relational-data-access/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-relational-data-access/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-relational-data-access.git>
* cd into gs-relational-data-access/initial
* Jump ahead to [Create a Customer object](http://spring.io/guides/gs/relational-data-access/#initial).

**When you’re finished**, you can check your results against the code in gs-relational-data-access/complete.

**Set up the project**

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

**Create the directory structure**

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

**Create a Gradle build file**

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-relational-data-access/blob/master/initial/build.gradle). But you can also use Maven. The pom.xml file is included [right here](https://github.com/spring-guides/gs-relational-data-access/blob/master/initial/pom.xml). If you are using [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts), you can import the guide directly.

build.gradle

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

jar {

baseName = 'gs-relational-data-access'

version = '0.1.0'

}

repositories {

mavenCentral()

maven { url "http://repo.spring.io/libs-snapshot" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter:0.5.0.M6")

compile("org.springframework:spring-jdbc:4.0.0.RC1")

compile("com.h2database:h2:1.3.172")

testCompile("junit:junit:4.11")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

|  |  |
| --- | --- |
|  | This guide is using [Spring Boot](http://spring.io/guides/gs/spring-boot). |

**Create a Customer object**

The simple data access logic you will work with below below manages first and last names of customers. To represent this data at the application level, create a Customer class.

src/main/java/hello/Customer.java

package hello;

public class Customer {

private long id;

private String firstName, lastName;

public Customer(long id, String firstName, String lastName) {

this.id = id;

this.firstName = firstName;

this.lastName = lastName;

}

@Override

public String toString() {

return String.format(

"Customer[id=%d, firstName='%s', lastName='%s']",

id, firstName, lastName);

}

// getters & setters omitted for brevity

}

**Store and retrieve data**

Spring provides a template class called JdbcTemplate that makes it easy to work with SQL relational databases and JDBC. Most JDBC code is mired in resource acquisition, connection management, exception handling, and general error checking that is wholly unrelated to what the code is meant to achieve. The JdbcTemplate takes care of all of that for you. All you have to do is focus on the task at hand.

src/main/java/hello/Application.java

package hello;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.jdbc.datasource.SimpleDriverDataSource;

public class Application {

public static void main(String args[]) {

// simple DS for test (not for production!)

SimpleDriverDataSource dataSource = new SimpleDriverDataSource();

dataSource.setDriverClass(org.h2.Driver.class);

dataSource.setUsername("sa");

dataSource.setUrl("jdbc:h2:mem");

dataSource.setPassword("");

JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

System.out.println("Creating tables");

jdbcTemplate.execute("drop table customers if exists");

jdbcTemplate.execute("create table customers(" +

"id serial, first\_name varchar(255), last\_name varchar(255))");

String[] names = "John Woo;Jeff Dean;Josh Bloch;Josh Long".split(";");

for (String fullname : names) {

String[] name = fullname.split(" ");

System.out.printf("Inserting customer record for %s %s\n", name[0], name[1]);

jdbcTemplate.update(

"INSERT INTO customers(first\_name,last\_name) values(?,?)",

name[0], name[1]);

}

System.out.println("Querying for customer records where first\_name = 'Josh':");

List<Customer> results = jdbcTemplate.query(

"select \* from customers where first\_name = ?", new Object[] { "Josh" },

new RowMapper<Customer>() {

@Override

public Customer mapRow(ResultSet rs, int rowNum) throws SQLException {

return new Customer(rs.getLong("id"), rs.getString("first\_name"),

rs.getString("last\_name"));

}

});

for (Customer customer : results) {

System.out.println(customer);

}

}

}

In this example you set up a JDBC [DataSource] using Spring’s handy SimpleDriverDataSource. Then, you use the DataSource to construct a JdbcTemplate instance.

|  |  |
| --- | --- |
|  | SimpleDriverDataSource is a convenience class and **not** intended for production. |

After you configure JdbcTemplate, it’s easy to start making calls to the database.

First, you install some DDL using JdbcTemplate's execute method.

Then, you install some records in your newly created table using JdbcTemplate's update method. The first argument to the method call is the query string, the last argument (the array of Object s) holds the variables to be substituted into the query where the "?" characters are.

|  |  |
| --- | --- |
|  | Use ? for arguments to avoid [SQL injection attacks](http://en.wikipedia.org/wiki/SQL_injection) by instructing JDBC to bind variables. |

Finally you use the query method to search your table for records matching the criteria. You again use the "?" arguments to create parameters for the query, passing in the actual values when you make the call. The last argument in the query method is an instance of RowMapper<T>, which you provide. Spring’s done 90% of the work, but it can’t know what you want it to do with the result set data. So, you provide a RowMapper<T> instance that Spring will call for each record, aggregate the results, and return as a collection.

**Build an executable JAR**

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-relational-data-access/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-relational-data-access/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-relational-data-access-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-relational-data-access-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

**Run the service**

If you are using Gradle, you can run your service at the command line this way:

./gradlew clean build && java -jar build/libs/gs-relational-data-access-0.1.0.jar

|  |  |
| --- | --- |
|  | If you are using Maven, you can run your service by typing mvn clean package && java -jar target/gs-relational-data-access-0.1.0.jar. |

You should see the following output:

Creating tables

Inserting customer record for John Woo

Inserting customer record for Jeff Dean

Inserting customer record for Josh Bloch

Inserting customer record for Josh Long

Querying for customer records where first\_name = 'Josh':

Customer[id=3, firstName='Josh', lastName='Bloch']

Customer[id=4, firstName='Josh', lastName='Long']

**Summary**

Congratulations! You’ve just used Spring to develop a simple JDBC client.

##### 2.1.8Consuming a RESTful Web Service with Spring for Android

This Getting Started guide walks you through the process of building an application that uses [Spring for Android](http://spring.io/projects/spring-android)'s RestTemplate to consume a Spring MVC-based [RESTful web service](http://spring.io/understanding/REST).

**What you will build**

You will build an Android client that consumes a Spring-based RESTful web service. Specifically, the client will consume the service created in [Building a RESTful Web Servce](http://spring.io/guides/gs/rest-service).

The Android client will be accessed through an Android emulator, and will consume the service accepting requests at:

http://rest-service.guides.spring.io/greeting

The service will respond with a [JSON](http://spring.io/understanding/JSON) representation of a greeting:

{"id":1,"content":"Hello, World!"}

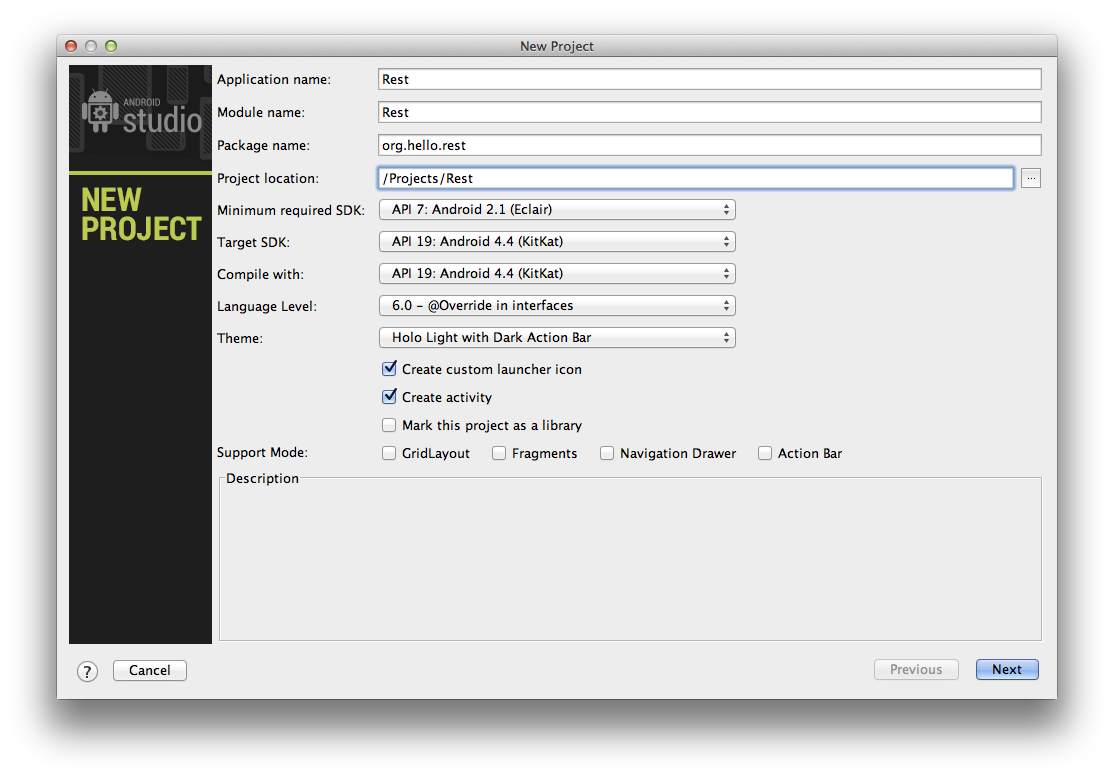
The Android client will render the ID and content into a view.

**What you will need**

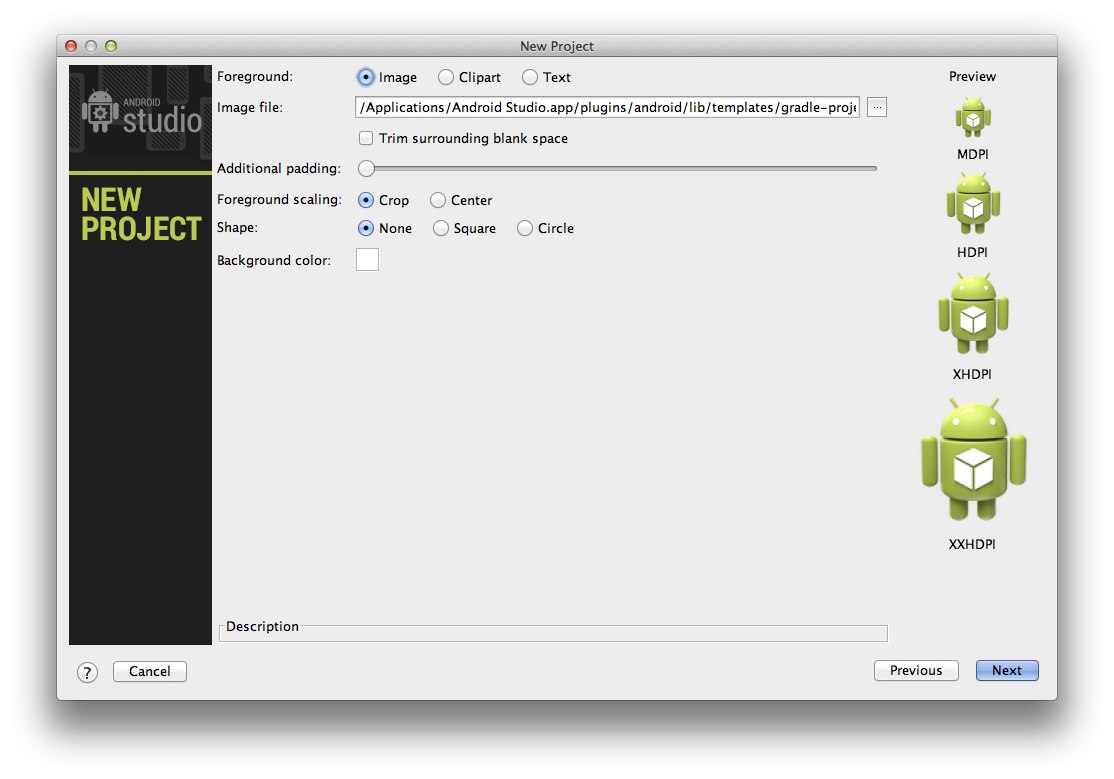
* About 15 minutes
* [Android Studio](http://developer.android.com/sdk/installing/studio.html)
* An internet connection

**Create an Android project**

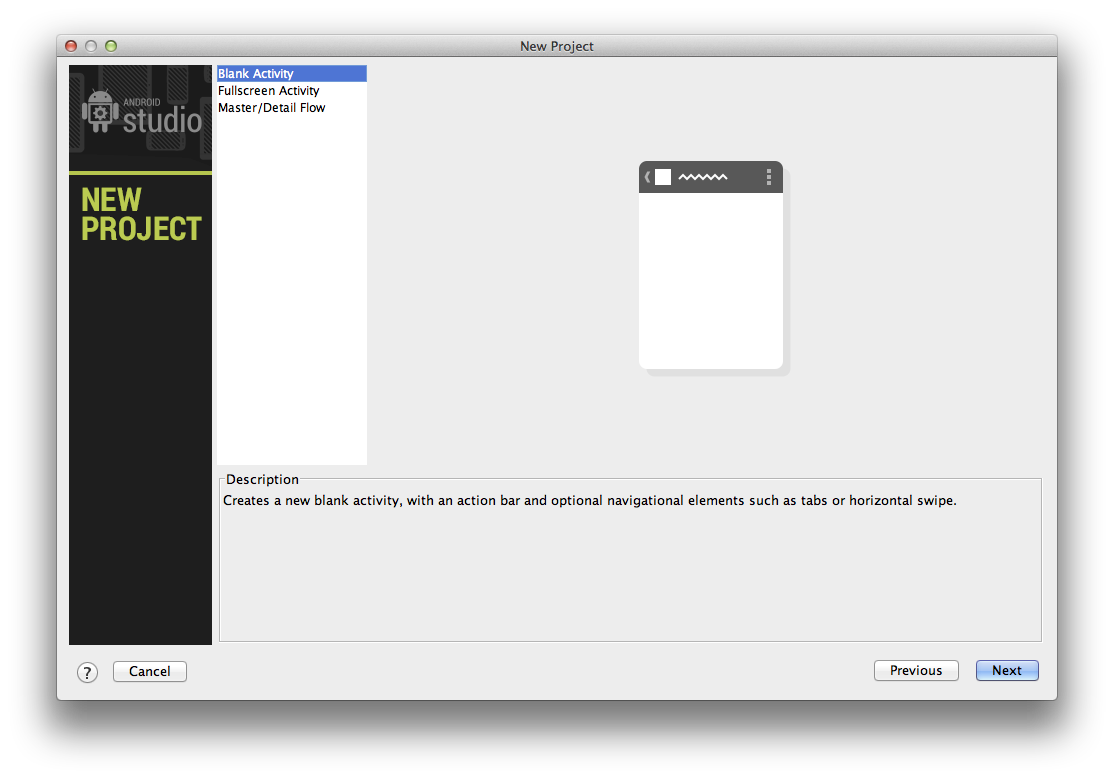
Within Android Studio, create a new project. If you prefer, you can use the project in the initial folder and skip ahead to [Create a representation class](http://spring.io/guides/gs/consuming-rest-android/#initial). When you are finished, you can compare your code to the complete folder and [Run the client](http://spring.io/guides/gs/consuming-rest-android/#run). Use "Rest" for the application and module names, and modify the package name to be "org.hello.rest". Enter the location of your choosing for the project and leave all the other options with their default settings.



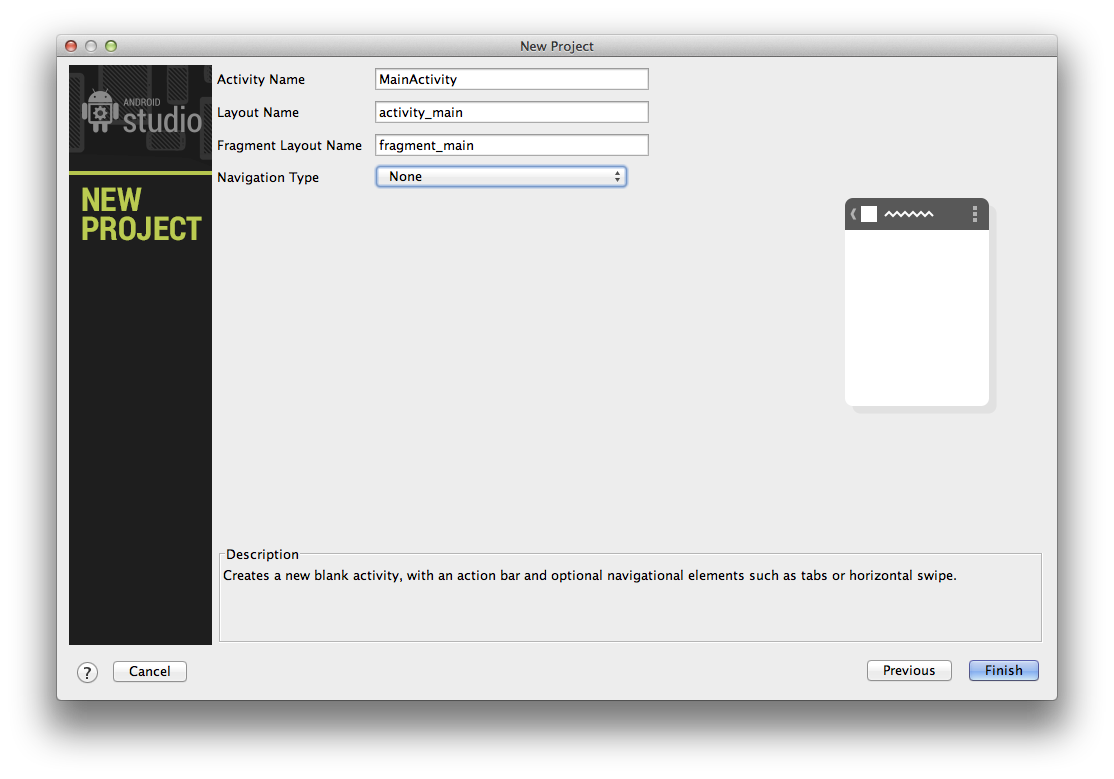
The next screen presents some options for configuring the app icons. Continue with the default options.



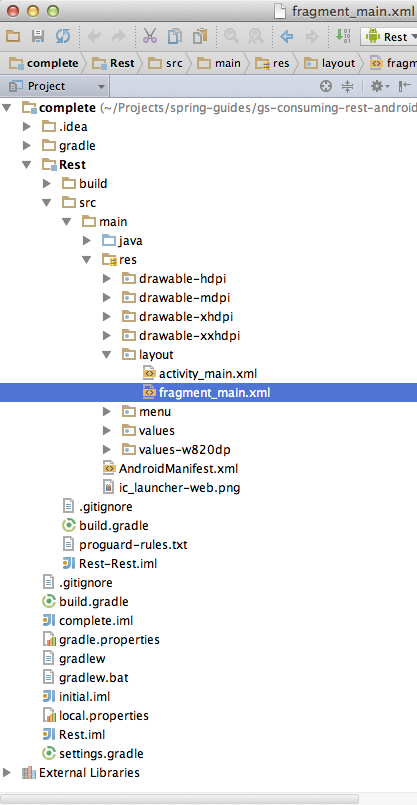
The next screen presents an option to select the type of activity to use. Select "Blank Activity" and continue.



The last screen presents some fields for setting the activity, layout, and fragment names. Again, continue with the default options to finish the project setup.



When the project is created, you will see that several files are added.



To complete this guide, you will edit the following:

* Rest/src/main/AndroidManifest.xml
* Rest/src/main/res/values/strings.xml
* Rest/src/main/res/layout/fragment\_main.xml
* Rest/src/main/res/menu/main.xml
* Rest/build.gradle
* Rest/src/main/java/org/hello/rest/MainActivity.java

**Create an Android Manifest**

When you created the project, an AndroidManifest.xml was also created with a basic implementation. The [Android Manifest](http://developer.android.com/guide/topics/manifest/manifest-intro.html) contains all the information required to run an Android application, and it cannot build without one. The manifest also contains any permissions for which the app is requesting of the Android operating system. In this case, the app needs to access the internet to make an HTTP request.

Add the INTERNET permission so the application can access resources over the internet.

Rest/src/main/AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="org.hello.rest"

android:versionCode="1"

android:versionName="1.0">

<uses-sdk

android:minSdkVersion="7"

android:targetSdkVersion="19" />

<uses-permission android:name="android.permission.INTERNET" />

<application

android:allowBackup="true"

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme">

<activity

android:name="org.hello.rest.MainActivity"

android:label="@string/app\_name">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

**Create string resources**

Text strings can be referenced from the application or from other resource files. This guide uses four text views and a menu item, and each of these UI elements needs a text description. First, remove the hello\_world and action\_settings strings. When you created the project, it included these. These are not used for this guide and can be removed. Next, add id\_label, id\_value, content\_label, content\_value and action\_refresh strings for each UI widget respectively.

Rest/src/main/res/values/strings.xml

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">Rest</string>

<string name="id\_label">The ID is</string>

<string name="id\_value">[id]</string>

<string name="content\_label">The Content is</string>

<string name="content\_value">[content]</string>

<string name="action\_refresh">Refresh</string>

</resources>

**Create a layout**

The layout file is where you define the visual structure for the user interface of your application. When you created the project, Android Studio added a layout fragment. As the name implies, a layout fragment represents a piece of the overall layout. In this case the layout fragment is used to display some text within the main activity. Remove the existing "Hello world!" TextView that was added when you created the project. Then modify the layout fragment to include four TextView widgets. The ids are used to reference these widgets from the code. Note the use of the string resources for the text of each widget.

Rest/src/main/res/layout/fragment\_main.xml

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

android:paddingBottom="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity$PlaceholderFragment">

<TextView

android:id="@+id/id\_label"

android:text="@string/id\_label"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

<TextView

android:id="@+id/content\_label"

android:text="@string/content\_label"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginTop="25dp"

android:layout\_below="@+id/id\_label"

android:layout\_alignParentLeft="true" />

<TextView

android:id="@+id/id\_value"

android:text="@string/id\_value"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentTop="true"

android:layout\_toRightOf="@+id/id\_label"

android:layout\_marginLeft="50dp" />

<TextView

android:id="@+id/content\_value"

android:text="@string/content\_value"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignTop="@+id/content\_label"

android:layout\_alignLeft="@+id/id\_value" />

</RelativeLayout>

The layout includes some information about how to position and size the widgets. Android Studio will display the visual representation of the layout in the preview window:



**Create a menu**

The project includes a menu for the main activity with an existing "Settings" option. Remove the "Settings" menu option and add the "refresh" option. Note again the use of the string resource as the title of the menu item.

Rest/src/main/res/menu/main.xml

<menu xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

tools:context=".MainActivity" >

<item android:id="@+id/action\_refresh"

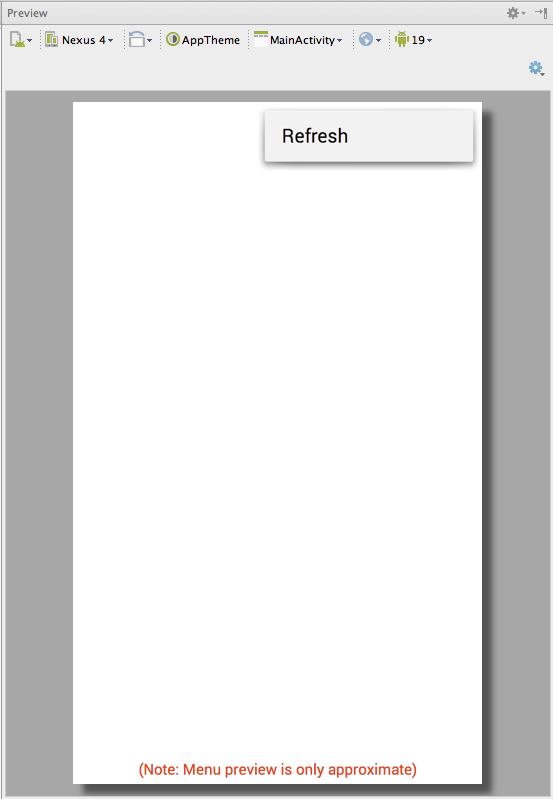
android:title="@string/action\_refresh"

android:orderInCategory="100"

app:showAsAction="never" />

</menu>

Android Studio will display the visual representation of the menu in the preview window:

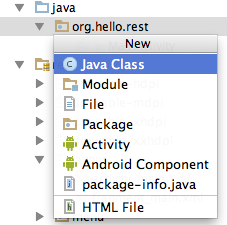


**Create a representation class**

To model the JSON data received from the RESTful HTTP request, you create a representation class that defines the fields. Navigate to the "org.hello.rest" package in the Project navigator. Select "New…" from the "File" menu.



Select "Java Class"



Add the id and content member variables and getters.

Rest/src/main/java/org/hello/rest/Greeting.java

package org.hello.rest;

public class Greeting {

private String id;

private String content;

public String getId() {

return this.id;

}

public String getContent() {

return this.content;

}

}

**Add dependencies**

To utilize Spring for Android’s RestTemplate within an Android app, you need to add the required Maven dependencies to the Gradle build file. RestTemplate makes use of Jackson, which is a powerful JSON processor for Java.

Rest/build.gradle

buildscript {

repositories {

mavenCentral()

}

dependencies {

classpath 'com.android.tools.build:gradle:0.6.+'

}

}

apply plugin: 'android'

repositories {

mavenCentral()

}

android {

compileSdkVersion 19

buildToolsVersion "19.0.0"

defaultConfig {

minSdkVersion 7

targetSdkVersion 19

}

buildTypes {

release {

runProguard false

proguardFile getDefaultProguardFile('proguard-android.txt')

}

}

productFlavors {

defaultFlavor {

proguardFile 'proguard-rules.txt'

}

}

}

dependencies {

compile 'com.android.support:appcompat-v7:+'

compile 'org.springframework.android:spring-android-rest-template:1.0.1.RELEASE'

compile 'com.fasterxml.jackson.core:jackson-databind:2.3.0'

}

**Create an activity**

The Model-View-Controller design pattern (MVC) is used extensively in Android applications. An Activity controls the view, which is represented by the layout you already created. When you created the project, a MainActivity was also created with a default implementation. Modify the MainActivity to make a RESTful HTTP request and update the view. Each modification is explained below.

Rest/src/main/java/org/hello/rest/MainActivity.java

package org.hello.rest;

import android.os.AsyncTask;

import android.os.Bundle;

import android.support.v4.app.Fragment;

import android.support.v7.app.ActionBarActivity;

import android.util.Log;

import android.view.LayoutInflater;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.view.ViewGroup;

import android.widget.TextView;

import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;

import org.springframework.web.client.RestTemplate;

public class MainActivity extends ActionBarActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

if (savedInstanceState == null) {

getSupportFragmentManager().beginTransaction()

.add(R.id.container, new PlaceholderFragment())

.commit();

}

}

@Override

protected void onStart() {

super.onStart();

new HttpRequestTask().execute();

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// Inflate the menu; this adds items to the action bar if it is present.

getMenuInflater().inflate(R.menu.main, menu);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

// Handle action bar item clicks here. The action bar will

// automatically handle clicks on the Home/Up button, so long

// as you specify a parent activity in AndroidManifest.xml.

int id = item.getItemId();

if (id == R.id.action\_refresh) {

new HttpRequestTask().execute();

return true;

}

return super.onOptionsItemSelected(item);

}

/\*\*

\* A placeholder fragment containing a simple view.

\*/

public static class PlaceholderFragment extends Fragment {

public PlaceholderFragment() {

}

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

View rootView = inflater.inflate(R.layout.fragment\_main, container, false);

return rootView;

}

}

private class HttpRequestTask extends AsyncTask<Void, Void, Greeting> {

@Override

protected Greeting doInBackground(Void... params) {

try {

final String url = "http://rest-service.guides.spring.io/greeting";

RestTemplate restTemplate = new RestTemplate();

restTemplate.getMessageConverters().add(new MappingJackson2HttpMessageConverter());

Greeting greeting = restTemplate.getForObject(url, Greeting.class);

return greeting;

} catch (Exception e) {

Log.e("MainActivity", e.getMessage(), e);

}

return null;

}

@Override

protected void onPostExecute(Greeting greeting) {

TextView greetingIdText = (TextView) findViewById(R.id.id\_value);

TextView greetingContentText = (TextView) findViewById(R.id.content\_value);

greetingIdText.setText(greeting.getId());

greetingContentText.setText(greeting.getContent());

}

}

}

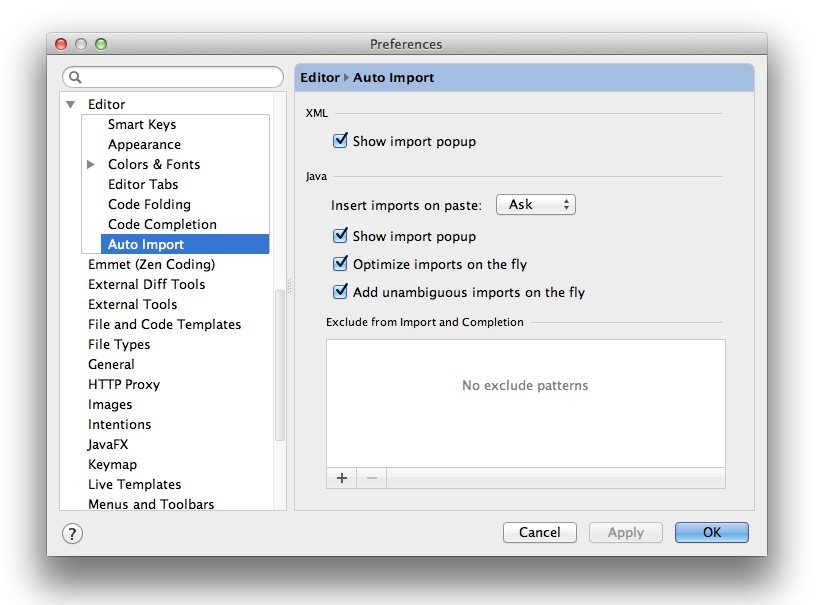
First, add the HttpRequestTask private class. This class inherits from AsyncTask which is a facility provided by Android for performing potentially, long running activities off of the main UI thread. It is important to do this, because otherwise you can lock the UI, causing a user to believe the app has stopped responding or crashed.

Spring provides a template class called RestTemplate. RestTemplate makes interacting with most RESTful services a simple process. Within the doInBackground method of the HttpRequestTask class, RestTemplate is used to make an HTTP request and marshal the JSON response to a Greeting object. When doInBackground returns, the onPostExecute method is called, where the text values of the greetingIdText and greetingContentText widgets are updated with the results of the HTTP request.

Next, add the onStart method which calls the execute method on HttpRequestTask. The onStart method is part of the Activity lifecycle and is called when the activity starts. The result is that the HTTP request is performed when the app loads.

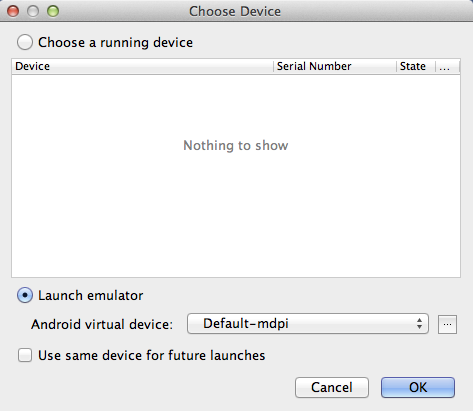
Lastly, update the onOptionsItemSelected method to also execute the HTTP request when the "Refresh" menu item is selected. This allows you to make additional HTTP requests without closing and restarting the app.

Android Studio does not automatically update the imports, however you can change this behavior in the settings. Once enabled it will behave similarly to "Ctrl+Shift+O" in STS or Eclipse. Otherwise, you need to manually update the imports to include the newly referenced classes.

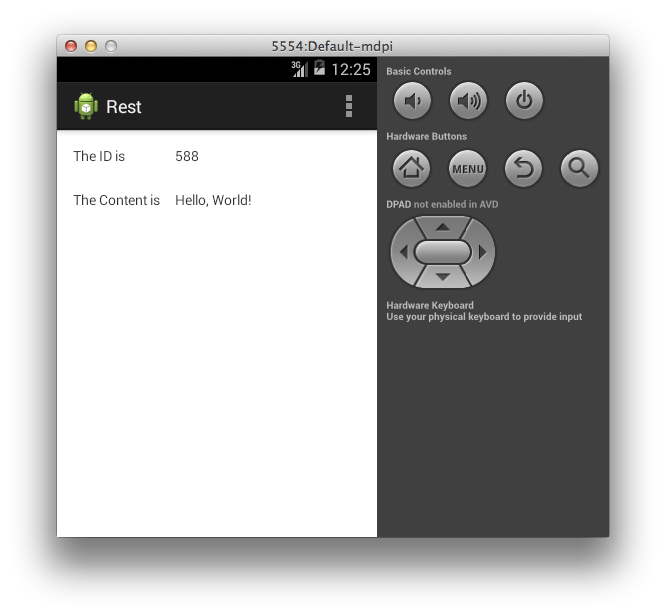


**Run the client**

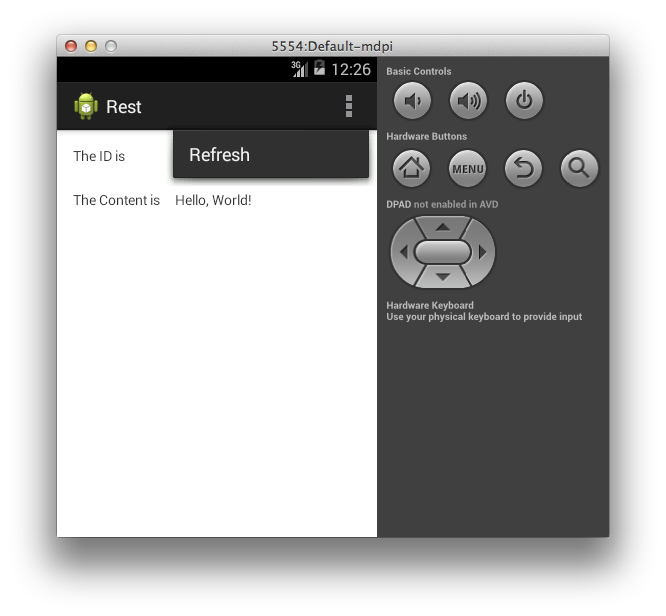
You can now run the app from Android Studio. To do this, click the play button (green triangle) in the toolbar of Android Studio. A dialog box will appear asking you to select the device on which to run the app. You must have an Android device or emulator configured in order to run the app. If you have not configured an Android Virtual Device (AVD), then you can select the elipsis to create a new one.



After you select an Android device, Android Studio will build and deploy the app:



The ID value will increment each time you click the refresh menu button.



**Summary**

Congratulations! You have developed a simple REST client using Spring for Android.

##### 2.1.9Uploading Files

This guide walks you through the process of creating a server application that can receive multi-part file uploads.

What you’ll build

You will create a Spring MVC application that accepts file uploads. You will also build a simple client to upload a test file.

What you’ll need

About 15 minutes

A favorite text editor or IDE

[JDK 1.6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later

[Gradle 1.8+](http://www.gradle.org/downloads) or [Maven 3.0+](http://maven.apache.org/download.cgi)

You can also import the code from this guide as well as view the web page directly into [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts) and work your way through it from there.

How to complete this guide

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To start from scratch, move on to [Set up the project](http://spring.io/guides/gs/uploading-files/#scratch).

To skip the basics, do the following:

[Download](https://github.com/spring-guides/gs-uploading-files/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-uploading-files.git>

cd into gs-uploading-files/initial

Jump ahead to [Create a configuration class](http://spring.io/guides/gs/uploading-files/#initial).

When you’re finished, you can check your results against the code in gs-uploading-files/complete.

Set up the project

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

Create the directory structure

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

Create a Gradle build file

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-uploading-files/blob/master/initial/build.gradle). But you can also use Maven. The pom.xml file is included [right here](https://github.com/spring-guides/gs-uploading-files/blob/master/initial/pom.xml). If you are using [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts), you can import the guide directly.

build.gradle

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

jar {

baseName = 'gs-uploading-files'

version = '0.1.0'

}

repositories {

mavenCentral()

maven { url "http://repo.spring.io/libs-snapshot" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter-web:0.5.0.M6")

testCompile("junit:junit:4.11")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

|  |  |
| --- | --- |
|  | This guide is using [Spring Boot](http://spring.io/guides/gs/spring-boot). |

Create a configuration class

To upload files with Servlet 3.0 containers, you need to register a MultipartConfigElement class (which would be <multipart-config> in web.xml).

src/main/java/hello/Application.java

package hello;

import javax.servlet.MultipartConfigElement;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan

@EnableAutoConfiguration

public class Application {

@Bean

MultipartConfigElement multipartConfigElement() {

return new MultipartConfigElement("");

}

}

This class is used to configure the server application that will receive file uploads, thanks to the @Configuration annotation.

You will soon add a Spring MVC controller, which is why you need both @EnableAutoConfiguration and @ComponentScan. Normally, you would use @EnableWebMvc for a Spring MVC application, but Spring Boot automatically adds this annotation when it detects spring-webmvc on your classpath. @ComponentScan makes it possible to automatically find @Controller-marked classes.

Using @EnableAutoConfiguration, the application will also detect the MultipartConfigElement bean and make itself ready for file uploads.

|  |  |
| --- | --- |
|  | [MultipartConfigElement](http://tomcat.apache.org/tomcat-7.0-doc/servletapi/javax/servlet/MultipartConfigElement.html) is a Servlet 3.0 standard element that defines the limits on uploading files. This component is supported by all compliant containers like Tomcat and Jetty. Here it’s configured to upload to the folder the application runs in with no limits, but you can override these settings if you wish. |

Create a file upload controller

In Spring, REST endpoints are just Spring MVC controllers. The following code provides the web app with the ability to upload files.

src/main/java/hello/FileUploadController.java

package hello;

import java.io.BufferedOutputStream;

import java.io.File;

import java.io.FileOutputStream;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.multipart.MultipartFile;

@Controller

public class FileUploadController {

@RequestMapping(value="/upload", method=RequestMethod.GET)

public @ResponseBody String provideUploadInfo() {

return "You can upload a file by posting to this same URL.";

}

@RequestMapping(value="/upload", method=RequestMethod.POST)

public @ResponseBody String handleFileUpload(@RequestParam("name") String name,

@RequestParam("file") MultipartFile file){

if (!file.isEmpty()) {

try {

byte[] bytes = file.getBytes();

BufferedOutputStream stream =

new BufferedOutputStream(new FileOutputStream(new File(name + "-uploaded")));

stream.write(bytes);

stream.close();

return "You successfully uploaded " + name + " into " + name + "-uploaded !";

} catch (Exception e) {

return "You failed to upload " + name + " => " + e.getMessage();

}

} else {

return "You failed to upload " + name + " because the file was empty.";

}

}

}

The entire class is marked up with @Controller so Spring MVC can pick it up and look for routes.

Each method is tagged with @RequestMapping to flag the path and the REST action. In this case, GET returns a very simple message indicating the POST operation is available.

The handleFileUpload method is geared to handle a two-part message: name and file. It checks to make sure the file is not empty, and if it is empty, the method grabs the bytes. Next, it writes them out through a BufferedOutputStream. Finally, it appends -uploaded to the target filename to clearly show when a file has been uploaded.

|  |  |
| --- | --- |
|  | In a production scenario, you more likely would store the files in a temporary location, a database, or perhaps a NoSQL store like [Mongo’s GridFS](http://docs.mongodb.org/manual/core/gridfs). You also need controls in place to avoid filling up the filesystem while also protecting yourself from vulnerabilities such as uploading executables and overwriting existing files. |

Make the application executable

Although it is possible to package this service as a traditional [WAR](http://spring.io/understanding/WAR) file for deployment to an external application server, the simpler approach demonstrated below creates a standalone application. You package everything in a single, executable JAR file, driven by a good old Java main() method. And along the way, you use Spring’s support for embedding the [Tomcat](http://spring.io/understanding/Tomcat) servlet container as the HTTP runtime, instead of deploying to an external instance.

src/main/java/hello/Application.java

package hello;

import javax.servlet.MultipartConfigElement;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.SpringApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan

@EnableAutoConfiguration

public class Application {

@Bean

MultipartConfigElement multipartConfigElement() {

return new MultipartConfigElement("");

}

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

The main() method defers to the [SpringApplication](http://docs.spring.io/spring-boot/docs/0.5.0.M6/api/org/springframework/boot/SpringApplication.html) helper class, providing Application.class as an argument to its run() method. This tells Spring to read the annotation metadata from Application and to manage it as a component in the [Spring application context](http://spring.io/understanding/application-context).

The @ComponentScan annotation tells Spring to search recursively through the hello package and its children for classes marked directly or indirectly with Spring’s [@Component](http://docs.spring.io/spring/docs/4.0.0.RC1/javadoc-api/org/springframework/stereotype/Component.html) annotation. This directive ensures that Spring finds and registers the FileUploadController, because it is marked with @Controller, which in turn is a kind of @Component annotation.

The [@EnableAutoConfiguration](http://docs.spring.io/spring-boot/docs/0.5.0.M6/api/org/springframework/boot/autoconfigure/EnableAutoConfiguration.html) annotation switches on reasonable default behaviors based on the content of your classpath. For example, because the application depends on the embeddable version of Tomcat (tomcat-embed-core.jar), a Tomcat server is set up and configured with reasonable defaults on your behalf. And because the application also depends on Spring MVC (spring-webmvc.jar), a Spring MVC [DispatcherServlet](http://docs.spring.io/spring/docs/4.0.0.RC1/javadoc-api/org/springframework/web/servlet/DispatcherServlet.html) is configured and registered for you — no web.xml necessary! Because there is a MultipartConfigElement, it configured the DispatcherServlet with multipart file upload functionality. Auto-configuration is a powerful, flexible mechanism. See the [API documentation](http://docs.spring.io/spring-boot/docs/0.5.0.M6/api/org/springframework/boot/autoconfigure/EnableAutoConfiguration.html) for further details.

Build an executable JAR

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-uploading-files/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-uploading-files/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-uploading-files-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-uploading-files-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

Run the service

If you are using Gradle, you can run your service at the command line this way:

./gradlew clean build && java -jar build/libs/gs-uploading-files-0.1.0.jar

|  |  |
| --- | --- |
|  | If you are using Maven, you can run your service by typing mvn clean package && java -jar target/gs-uploading-files-0.1.0.jar. |

That runs the server-side piece that receives file uploads. Logging output is displayed. The service should be up and running within a few seconds.

Create a client and upload a file

So far, you have built a server application capable of receiving file uploads. It would not be of much use unless you also build a client application to upload a file. The easiest way to do that is by using Spring MVC’s RestTemplate.

src/main/java/hello/FileUploader.java

package hello;

import java.io.FileNotFoundException;

import org.springframework.core.io.FileSystemResource;

import org.springframework.util.LinkedMultiValueMap;

import org.springframework.util.MultiValueMap;

import org.springframework.web.client.RestTemplate;

public class FileUploader {

public static void main(String[] args) throws FileNotFoundException {

if (args.length == 0) {

System.out.println("Usage: Requires the name of a file to upload.");

System.exit(1);

}

RestTemplate template = new RestTemplate();

MultiValueMap<String, Object> parts = new LinkedMultiValueMap<String, Object>();

parts.add("name", args[0]);

parts.add("file", new FileSystemResource(args[0]));

String response = template.postForObject("http://localhost:8080/upload", parts, String.class);

System.out.println(response);

}

}

This client application creates a RestTemplate and then loads up a MultiValueMap with the name and the file. This leverages Spring’s FileSystemResource class to properly load the bytes for the file. Then the template uses its postForObject method to POST the file to the server. Because the server was coded to write a textual message straight into the HTTP response, the client application prints that message out to the console.

|  |  |
| --- | --- |
|  | In more sophisticated applications, you probably want to use real HTML and some type of file chooser component to pick the file for upload. |

You just coded some client code to upload a sample file. To run the code, add this to your Gradle build file:

apply plugin: 'application'

mainClassName = "hello.FileUploader"

run {

args 'sample.txt'

}

|  |  |
| --- | --- |
|  | If you clicked on the link up above to view the final build.gradle file, you will have already seen this. There is similar material added to the pom.xml file. |

With the server running in one window, you need to open another window to run the client.

./gradlew run

|  |  |
| --- | --- |
|  | If you are using Maven, you can run the client by typing mvn package exec:java. |

It should produce some output like this in the client window:

You successfully uploaded sample.txt into sample.txt-uploaded !

The controller itself doesn’t print anything out, but instead returns the message posted to the client.

Summary

Congratulations! You have just written a client and server that use Spring to handle file uploads.

##### 2.1.10Authenticating a User with LDAP

This guide walks you through the process creating an application and securing it with the [Spring Security](http://projects.spring.io/spring-security/) LDAP module.

What you’ll build

You’ll build a simple web application that is secured by Spring Security’s embedded Java-based LDAP server. You’ll load the LDAP server with a data file containing a set of users.

What you’ll need

About 15 minutes

A favorite text editor or IDE

[JDK 1.6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later

[Gradle 1.8+](http://www.gradle.org/downloads) or [Maven 3.0+](http://maven.apache.org/download.cgi)

You can also import the code from this guide as well as view the web page directly into [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts) and work your way through it from there.

|  |  |
| --- | --- |
|  | This guide does NOT run with Java 8 at this time. |

How to complete this guide

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To start from scratch, move on to [Set up the project](http://spring.io/guides/gs/authenticating-ldap/#scratch).

To skip the basics, do the following:

[Download](https://github.com/spring-guides/gs-authenticating-ldap/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-authenticating-ldap.git>

cd into gs-authenticating-ldap/initial

Jump ahead to [Create a simple web controller](http://spring.io/guides/gs/authenticating-ldap/#initial).

When you’re finished, you can check your results against the code in gs-authenticating-ldap/complete.

Set up the project

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

Create the directory structure

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

Create a Gradle build file

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-authenticating-ldap/blob/master/initial/build.gradle). But you can also use Maven. The pom.xml file is included [right here](https://github.com/spring-guides/gs-authenticating-ldap/blob/master/initial/pom.xml). If you are using [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts), you can import the guide directly.

build.gradle

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

jar {

baseName = 'gs-authenticating-ldap'

version = '0.1.0'

}

repositories {

mavenCentral()

maven { url "http://repo.spring.io/libs-snapshot" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter-web:0.5.0.M6")

compile("org.springframework.boot:spring-boot-starter-security:0.5.0.M6")

compile("org.springframework.security:spring-security-ldap:3.2.0.RC1")

compile("org.apache.directory.server:apacheds-server-jndi:1.5.5")

testCompile("junit:junit:4.11")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

|  |  |
| --- | --- |
|  | This guide is using [Spring Boot](http://spring.io/guides/gs/spring-boot). |

Create a simple web controller

In Spring, REST endpoints are just Spring MVC controllers. The following Spring MVC controller handles a GET / request by returning a simple message:

src/main/java/hello/HomeController.java

package hello;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

@Controller

public class HomeController {

@RequestMapping("/")

public @ResponseBody String index() {

return "Welcome to the home page!";

}

}

The entire class is marked up with @Controller so Spring MVC can autodetect the controller using it’s built-in scanning features and automatically configure web routes.

The method is tagged with @RequestMapping to flag the path and the REST action. In this case, GET is the default behavior; it returns a message indicating that you are on the home page.

@ResponseBody tells Spring MVC to write the text directly into the HTTP response body, because there aren’t any views. Instead, when you visit the page, you’ll get a simple message in the browser as the focus of this guide is securing the page with LDAP.

Build the unsecured web application

Before you secure the web application, verify that it works. To do that, you need to define some key beans. To do that, create an Application class.

src/main/java/hello/Application.java

package hello;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan

@EnableAutoConfiguration

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

Build an executable JAR

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-authenticating-ldap/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-authenticating-ldap/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-authenticating-ldap-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-authenticating-ldap-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

Run the unsecured web application

If you are using Gradle, you can run your unsecured web application at the command line this way:

./gradlew clean build && java -jar build/libs/gs-authenticating-ldap-0.1.0.jar

|  |  |
| --- | --- |
|  | If you are using Maven, you can run your unsecured web application by typing mvn clean package && java -jar target/gs-authenticating-ldap-0.1.0.jar. |

If you open your browser and visit [http://localhost:8080](http://localhost:8080/), you should see the following plain text:

---- TODO: If this is an output, replace with .'s Welcome to the home page! ---- TODO: If this is an output, replace with .'s

Set up Spring Security

To configure Spring Security, you can use pure Java to configure things properly.

src/main/java/hello/WebSecurityConfig.java

package hello;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(AuthenticationManagerBuilder authManagerBuilder) throws Exception {

authManagerBuilder

.ldapAuthentication()

.userDnPatterns("uid={0},ou=people")

.groupSearchBase("ou=groups")

.contextSource()

.ldif("classpath:test-server.ldif");

}

}

The @EnableWebSecurity turns on a variety of beans needed to use Spring Security.

You also need an LDAP server. Spring Security’s LDAP module includes an embedded server written in pure Java, which is being used for this guide. The ldapAuthentication() method configures things where the username at the login form is plugged into {0} such that it searches uid={0},ou=people,dc=springframework,dc=org in the LDAP server.

Set up user data

LDAP servers can use LDIF (LDAP Data Interchange Format) files to exchange user data. The ldif() method inside WebSecurityConfig pulls in an LDIF data file. This makes it easy to pre-load demonstration data.

src/main/resources/test-server.ldif

dn: ou=groups,dc=springframework,dc=org

objectclass: top

objectclass: organizationalUnit

ou: groups

dn: ou=subgroups,ou=groups,dc=springframework,dc=org

objectclass: top

objectclass: organizationalUnit

ou: subgroups

dn: ou=people,dc=springframework,dc=org

objectclass: top

objectclass: organizationalUnit

ou: people

dn: ou=space cadets,dc=springframework,dc=org

objectclass: top

objectclass: organizationalUnit

ou: space cadets

dn: ou=\"quoted people\",dc=springframework,dc=org

objectclass: top

objectclass: organizationalUnit

ou: "quoted people"

dn: ou=otherpeople,dc=springframework,dc=org

objectclass: top

objectclass: organizationalUnit

ou: otherpeople

dn: uid=ben,ou=people,dc=springframework,dc=org

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

cn: Ben Alex

sn: Alex

uid: ben

userPassword: {SHA}nFCebWjxfaLbHHG1Qk5UU4trbvQ=

dn: uid=bob,ou=people,dc=springframework,dc=org

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

cn: Bob Hamilton

sn: Hamilton

uid: bob

userPassword: bobspassword

dn: uid=joe,ou=otherpeople,dc=springframework,dc=org

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

cn: Joe Smeth

sn: Smeth

uid: joe

userPassword: joespassword

dn: cn=mouse\, jerry,ou=people,dc=springframework,dc=org

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

cn: Mouse, Jerry

sn: Mouse

uid: jerry

userPassword: jerryspassword

dn: cn=slash/guy,ou=people,dc=springframework,dc=org

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

cn: slash/guy

sn: Slash

uid: slashguy

userPassword: slashguyspassword

dn: cn=quote\"guy,ou=\"quoted people\",dc=springframework,dc=org

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

cn: quote\"guy

sn: Quote

uid: quoteguy

userPassword: quoteguyspassword

dn: uid=space cadet,ou=space cadets,dc=springframework,dc=org

objectclass: top

objectclass: person

objectclass: organizationalPerson

objectclass: inetOrgPerson

cn: Space Cadet

sn: Cadet

uid: space cadet

userPassword: spacecadetspassword

dn: cn=developers,ou=groups,dc=springframework,dc=org

objectclass: top

objectclass: groupOfNames

cn: developers

ou: developer

uniqueMember: uid=ben,ou=people,dc=springframework,dc=org

uniqueMember: uid=bob,ou=people,dc=springframework,dc=org

dn: cn=managers,ou=groups,dc=springframework,dc=org

objectclass: top

objectclass: groupOfNames

cn: managers

ou: manager

uniqueMember: uid=ben,ou=people,dc=springframework,dc=org

uniqueMember: cn=mouse\, jerry,ou=people,dc=springframework,dc=org

dn: cn=submanagers,ou=subgroups,ou=groups,dc=springframework,dc=org

objectclass: top

objectclass: groupOfNames

cn: submanagers

ou: submanager

uniqueMember: uid=ben,ou=people,dc=springframework,dc=org

|  |  |
| --- | --- |
|  | Using an LDIF file isn’t standard configuration for a production system. However, it’s very useful for testing purposes or guides. |

Create an Application class

src/main/java/hello/Application.java

package hello;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.SpringApplication;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan

@EnableAutoConfiguration

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

The main() method defers to the [SpringApplication](http://docs.spring.io/spring-boot/docs/%7Bspring_boot_version%7D/api/org/springframework/boot/SpringApplication.html) helper class, providing Application.class as an argument to its run() method. This tells Spring to read the annotation metadata from Application and to manage it as a component in the [Spring application context](http://spring.io/understanding/application-context).

The @ComponentScan annotation tells Spring to search recursively through the hello package and its children for classes marked directly or indirectly with Spring’s [@Component](http://docs.spring.io/spring/docs/4.0.0.RC1/javadoc-api/org/springframework/stereotype/Component.html) annotation. This directive ensures that Spring finds and registers the WebSecurityConfig class, because it is marked with @Configuration, which in turn is a kind of @Component annotation.

The [@EnableAsync](http://docs.spring.io/spring/docs/current/spring-framework-reference/html/scheduling.html#scheduling-annotation-support) annotation switches on Spring’s ability to run @Async methods in a background thread pool.

The [@EnableAutoConfiguration](http://docs.spring.io/spring-boot/docs/%7Bspring_boot_version%7D/api/org/springframework/boot/autoconfigure/EnableAutoConfiguration.html) annotation switches on reasonable default behaviors based on the content of your classpath. For example, it looks for any class that implements the CommandLineRunner interface and invokes its run() method. In this case, it runs the demo code for this guide.

Build an executable JAR

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-authenticating-ldap/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-authenticating-ldap/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-authenticating-ldap-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-authenticating-ldap-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

Run the secured web application

If you are using Gradle, you can run your secured web application at the command line this way:

./gradlew clean build && java -jar build/libs/gs-authenticating-ldap-0.1.0.jar

|  |  |
| --- | --- |
|  | If you are using Maven, you can run your secured web application by typing mvn clean package && java -jar target/gs-authenticating-ldap-0.1.0.jar. |

If you visit the site at [[http://localhost:8080](http://localhost:8080/)]([http://localhost:8080](http://localhost:8080/)), you should be redirected to a login page provided by Spring Security.

Enter username ben and password benspassword. You should see this message in your browser:

---- TODO: If this is an output, replace with .'s Welcome to the home page! ---- TODO: If this is an output, replace with .'s

Summary

Congratulations! You have just written a web application and secured it with [Spring Security](http://docs.spring.io/spring-security/site/docs/3.2.x/reference/html/). In this case, you used an [LDAP-based user store](<http://docs.spring.io/spring-security/site/docs/3.2.x/reference/html/ldap.html>).

##### 2.1.11Building Android Projects with Maven

This guide walks you through the process of building a simple Android project with Maven.

**What you’ll build**

You’ll create an Android application that gives you the time of day, and then build it with Maven.

**What you’ll need**

* About 15 minutes
* A favorite text editor or IDE
* [Android SDK](http://developer.android.com/sdk/index.html)
* An Android device or Emulator

**How to complete this guide**

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Set up the project](http://spring.io/guides/gs/maven-android/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-maven-android/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-maven-android.git>
* cd into gs-maven-android/initial
* Jump ahead to [Install Maven](http://spring.io/guides/gs/maven-android/#initial).

**When you’re finished**, you can check your results against the code in gs-maven-android/complete.

**Set up the project**

First, you set up an Android project for Maven to build. To keep the focus on Maven, make the project as simple as possible for now. If this is your first time working with Android projects, refer to [Installing the Android Development Environment](http://spring.io/guides/gs/android) to help configure your development environment.

**Create the directory structure**

In a project directory of your choosing, create the following subdirectory structure; for example, with the following command on Mac or Linux:

mkdir -p src/main/java/org/hello

└── src

└── main

└── java

└── org

└── hello

**Create an Android manifest**

The [Android Manifest](http://developer.android.com/guide/topics/manifest/manifest-intro.html) contains all the information required to run an Android application, and it cannot build without one.

AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="org.hello"

android:versionCode="1"

android:versionName="1.0.0" >

<application android:label="@string/app\_name" >

<activity

android:name=".HelloActivity"

android:label="@string/app\_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

Add a text string. Text strings can be referenced from the application or from other resource files.

res/values/strings.xml

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">Android Maven</string>

</resources>

Here you define the visual structure for the user interface of your application.

res/layout/hello\_layout.xml

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

>

<TextView

android:id="@+id/text\_view"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

/>

</LinearLayout>

Within the src/main/java/org/hello directory, you can create any Java classes you want. To maintain consistency with the rest of this guide, create the following class:

src/main/java/org/hello/HelloActivity.java

package org.hello;

import android.app.Activity;

import android.os.Bundle;

import android.widget.TextView;

public class HelloActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.hello\_layout);

}

@Override

public void onStart() {

super.onStart();

TextView textView = (TextView) findViewById(R.id.text\_view);

textView.setText("Hello world!");

}

}

**Install Maven**

Now you have a project that you can build with Maven. The next step is to install Maven.

Maven is downloadable as a zip file at <http://maven.apache.org/download.cgi>. Only the binaries are required, so look for the link to apache-maven-{version}-bin.zip or apache-maven-{version}-bin.tar.gz.

Download and unzip the file, then add the *bin* folder to your path.

To test the Maven installation, run mvn from the command-line:

mvn -v

If all goes well, you should see installation information like this:

Apache Maven 3.0.5 (r01de14724cdef164cd33c7c8c2fe155faf9602da; 2013-02-19 08:51:28-0500)

Maven home: /usr/local/apache-maven/apache-maven-3.0.5

Java version: 1.7.0\_21, vendor: Oracle Corporation

Java home: /Library/Java/JavaVirtualMachines/jdk1.7.0\_25.jdk/Contents/Home/jre

Default locale: en\_US, platform encoding: UTF-8

OS name: "mac os x", version: "10.8.3", arch: "x86\_64", family: "mac"

You now have Maven installed.

**Define a simple Maven build**

Now that Maven is installed, you need to create a Maven project definition. You define Maven projects with an XML file named *pom.xml*. Among other things, this file gives the project’s name, version, and dependencies that it has on external libraries.

Create a file named pom.xml at the root of the project and give it the following contents:

pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.hello</groupId>

<artifactId>gs-maven-android</artifactId>

<version>0.1.0</version>

<packaging>apk</packaging>

<dependencies>

<dependency>

<groupId>com.google.android</groupId>

<artifactId>android</artifactId>

<version>4.1.1.4</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>com.jayway.maven.plugins.android.generation2</groupId>

<artifactId>android-maven-plugin</artifactId>

<version>3.6.1</version>

<configuration>

<sdk>

<platform>18</platform>

</sdk>

<deleteConflictingFiles>true</deleteConflictingFiles>

<undeployBeforeDeploy>true</undeployBeforeDeploy>

</configuration>

<extensions>true</extensions>

</plugin>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.6</source>

<target>1.6</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

The <packaging> element specifies an **apk**. This is the simplest possible **pom.xml** file necessary to build an Android project. It includes the following details of the project configuration:

* <modelVersion>. POM model version (always 4.0.0).
* <groupId>. Group or organization that the project belongs to. Often expressed as an inverted domain name.
* <artifactId>. Name to be given to the project’s library artifact (for example, the name of its APK file).
* <version>. Version of the project that is being built.
* <packaging>. How the project should be packaged, in this case as an Android APK.

The <dependencies> section declares a list of dependencies for the project. Specifically, it declares a single dependency for the Android library. Within the <dependency> element, the dependency coordinates are defined by three subelements:

* <groupId>. Group or organization that the dependency belongs to.
* <artifactId>. Library that is required.
* <version>. Specific version of the library that is required.
* <scope>. Scoped as compile dependencies by default. That is, all dependencies should be available at compile-time.

In this case, the <scope> element has a value of provided. Dependencies of this type are required for compiling the project code, but will be provided at runtime by a container running the code. For example, the Android APIs are always available when an Android application is running.

The <build> section declares additional configuration for building an application. Within the build section is a <plugins> section, which contains a list of plugins that add additional functionality to the build process. This is where you define the configuration for the [Android Maven Plugin](https://code.google.com/p/maven-android-plugin). As with dependencies, plugins also have <groupId>, <artifactId>, and <version> elements, and they behave as previously described. The plugin declaration also has these elements:

* <configuration>. Plugin-specific configuration. Here you specify which Android Platform SDK to use in the build.
* <extensions>. Combination of specifying a value of true and apk for <packaging> directs the [Android Maven Plugin] to become involved in the build process.

At this point you have defined a minimal yet capable Maven project.

**Build Android code**

Maven is now ready to build the project. You can execute several build lifecycle goals with Maven now, including goals to compile the project’s code, create a library package (such as a JAR file), and install the library in the local Maven dependency repository.

Try out the build:

mvn compile

This command runs Maven, telling it to execute the *compile* goal. When it’s finished, you should find the compiled *.class* files in the *target/classes* directory.

Because it’s unlikely that you’ll want to distribute or work with .class files directly, you’ll probably want to run the *package* goal instead:

mvn package

The package goal compiles your Java code, runs any tests, and packages the code in a JAR file within the **target** directory. The name of the JAR file is based on the project’s <artifactId> and <version>. For example, given the minimal pom.xml file shown earlier, the JAR file will be named gs-maven-android-0.1.0.jar.

Because you set the value of <packaging> to "apk", the result will be an APK file within the **target** directory in addition to the JAR file. This APK file is now a packaged Android application ready to be deployed to a device or emulator.

The Android Maven plugin provides several more Maven goals that you can use to initiate the various phases of the build process, or interact with the device and emulator. You can see a list of all the available goals by running the following command:

mvn android:help

**Declare dependencies**

The simple Hello World sample is completely self-contained and does not depend on any additional libraries. Most applications, however, depend on external libraries to handle common and/or complex functionality.

For example, suppose you want your application to print the current date and time. Although you could use the date and time facilities in the native Java libraries, you can make things more interesting by using the Joda Time libraries.

To do this, modify HelloActivity.java to look like this:

src/main/java/org/hello/HelloActivity.java

package org.hello;

import org.joda.time.LocalTime;

import android.app.Activity;

import android.os.Bundle;

import android.widget.TextView;

public class HelloActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.hello\_layout);

}

@Override

public void onStart() {

super.onStart();

LocalTime currentTime = new LocalTime();

TextView textView = (TextView) findViewById(R.id.text\_view);

textView.setText("The current local time is: " + currentTime);

}

}

In this example, you use Joda Time’s LocalTime class to retrieve and display the current time.

If you were to run mvn package to build the project now, the build would fail because you have not declared Joda Time as a compile dependency in the build. You can fix that by adding the following lines to <dependencies> section of the **pom.xml**:

<dependency>

<groupId>joda-time</groupId>

<artifactId>joda-time</artifactId>

<version>2.2</version>

</dependency>

Similar to the Android dependency discussed earlier, this block of XML declares a new dependency for the project, specifically the Joda Time library.

**Rebuild Android code with dependencies**

Now if you run mvn compile or mvn package, Maven should resolve the Joda Time dependency from the Maven Central repository and the build will be successful.

Here’s the completed pom.xml file:

pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.hello</groupId>

<artifactId>gs-maven-android</artifactId>

<version>0.1.0</version>

<packaging>apk</packaging>

<dependencies>

<dependency>

<groupId>com.google.android</groupId>

<artifactId>android</artifactId>

<version>4.1.1.4</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>com.jayway.maven.plugins.android.generation2</groupId>

<artifactId>android-maven-plugin</artifactId>

<version>3.6.1</version>

<configuration>

<sdk>

<platform>18</platform>

</sdk>

<deleteConflictingFiles>true</deleteConflictingFiles>

<undeployBeforeDeploy>true</undeployBeforeDeploy>

</configuration>

<extensions>true</extensions>

</plugin>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.6</source>

<target>1.6</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Summary**

Congratulations! You have created a simple yet effective Maven project definition for building Java projects.

##### 2.1.12Messaging with Redis

This guide walks you through the process of using Spring Data Redis to publish and subscribe to messages sent via Redis.

**What you’ll build**

You’ll build an application that uses StringRedisTemplate to publish a string message and has a POJO subscribe for it using MessageListenerAdapter.

|  |  |
| --- | --- |
|  | It may sound strange to be using Spring Data Redis as the means to publish messages, but as you’ll discover, Redis not only provides a NoSQL data store, but a messaging system as well. |

**What you’ll need**

* About 15 minutes
* A favorite text editor or IDE
* [JDK 1.6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later
* [Gradle 1.8+](http://www.gradle.org/downloads) or [Maven 3.0+](http://maven.apache.org/download.cgi)
* You can also import the code from this guide as well as view the web page directly into [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts) and work your way through it from there.
* Redis server (installation instructions below)

**How to complete this guide**

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Set up the project](http://spring.io/guides/gs/messaging-redis/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-messaging-redis/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-messaging-redis.git>
* cd into gs-messaging-redis/initial
* Jump ahead to [Create a Redis message receiver](http://spring.io/guides/gs/messaging-redis/#initial).

**When you’re finished**, you can check your results against the code in gs-messaging-redis/complete.

**Set up the project**

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

**Create the directory structure**

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

**Create a Gradle build file**

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-messaging-redis/blob/master/initial/build.gradle). But you can also use Maven. The pom.xml file is included [right here](https://github.com/spring-guides/gs-messaging-redis/blob/master/initial/pom.xml). If you are using [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts), you can import the guide directly.

build.gradle

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

jar {

baseName = 'gs-messaging-redis'

version = '0.1.0'

}

repositories {

mavenCentral()

maven { url "http://repo.spring.io/libs-snapshot" }

}

dependencies {

compile("org.springframework.data:spring-data-redis:1.0.6.RELEASE")

compile("cglib:cglib:2.2.2")

compile("org.slf4j:slf4j-log4j12:1.7.5")

testCompile("junit:junit:4.11")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

|  |  |
| --- | --- |
|  | This guide is using [Spring Boot](http://spring.io/guides/gs/spring-boot). |

Before you can build a messaging application, you need to set up the server that will handle receiving and sending messages.

Redis is an open source, BSD-licensed, key-value data store that also comes with a messaging system. The server is freely available at <http://redis.io/download>. You can download it manually, or if you use a Mac with homebrew:

brew install redis

Once you unpack Redis, you can launch it with default settings.

redis-server

You should see a message like this:

[35142] 01 May 14:36:28.939 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf

[35142] 01 May 14:36:28.940 \* Max number of open files set to 10032

\_.\_

\_.-``\_\_ ''-.\_

\_.-`` `. `\_. ''-.\_ Redis 2.6.12 (00000000/0) 64 bit

.-`` .-```. ```\/ \_.,\_ ''-.\_

( ' , .-` | `, ) Running in stand alone mode

|`-.\_`-...-` \_\_...-.``-.\_|'` \_.-'| Port: 6379

| `-.\_ `.\_ / \_.-' | PID: 35142

`-.\_ `-.\_ `-./ \_.-' \_.-'

|`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|

| `-.\_`-.\_ \_.-'\_.-' | http://redis.io

`-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'

|`-.\_`-.\_ `-.\_\_.-' \_.-'\_.-'|

| `-.\_`-.\_ \_.-'\_.-' |

`-.\_ `-.\_`-.\_\_.-'\_.-' \_.-'

`-.\_ `-.\_\_.-' \_.-'

`-.\_ \_.-'

`-.\_\_.-'

[35142] 01 May 14:36:28.941 # Server started, Redis version 2.6.12

[35142] 01 May 14:36:28.941 \* The server is now ready to accept connections on port 6379

**Create a Redis message receiver**

In any messaging-based application, there are message publishers and messaging receivers. To create the message receiver, implement a receiver with a method to respond to messages:

src/main/java/hello/Receiver.java

package hello;

public class Receiver {

public void receiveMessage(String message) {

System.out.println("Received <" + message + ">");

}

}

The Receiver is a simple POJO that defines a method for receiving messages. As you’ll see when you register the Receiver as a message listener, you can name the message-handling method whatever you want.

**Register the listener and send a message**

Spring Data Redis provides all the components you need to send and receive messages with Redis. Specifically, you need to configure:

* A connection factory
* A message listener container
* A Redis template

You’ll use the Redis template to send messages and you will register the Receiver with the message listener container so that it will receive messages. The connection factory drives both the template and the message listener container, enabling them to connect to the Redis server.

This example sets up a JedisConnectionFactory, a Redis connection factory based on the [Jedis](https://github.com/xetorthio/jedis) Redis library. That connection factory is injected into both the message listener container and the Redis template.

src/main/java/hello/Application.java

package hello;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;

import org.springframework.data.redis.core.StringRedisTemplate;

import org.springframework.data.redis.listener.PatternTopic;

import org.springframework.data.redis.listener.RedisMessageListenerContainer;

import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;

@Configuration

public class Application {

@Bean

JedisConnectionFactory connectionFactory() {

return new JedisConnectionFactory();

}

@Bean

RedisMessageListenerContainer container(final JedisConnectionFactory connectionFactory) {

RedisMessageListenerContainer container = new RedisMessageListenerContainer() {{

setConnectionFactory(connectionFactory);

}};

container.addMessageListener(listenerAdapter(), new PatternTopic("chat"));

return container;

}

@Bean

MessageListenerAdapter listenerAdapter() {

return new MessageListenerAdapter(new Receiver(), "receiveMessage");

}

@Bean

StringRedisTemplate template(JedisConnectionFactory connectionFactory) {

return new StringRedisTemplate(connectionFactory);

}

public static void main(String[] args) throws InterruptedException {

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(Application.class);

StringRedisTemplate template = ctx.getBean(StringRedisTemplate.class);

System.out.println("Sending message...");

template.convertAndSend("chat", "Hello from Redis!");

ctx.close();

}

}

The bean defined in the listenerAdapter() method is registered as a message listener in the message listener container defined in container() and will listen for messages on the "chat" topic. Because the Receiver class is a POJO, it needs to be wrapped in a message listener adapter that implements the MessageListener interface required by addMessageListener(). The message listener adapter is also configured to call the receiveMessage() method on Receiver when a message arrives.

The connection factory and message listener container beans are all you need to listen for messages. To send a message you also need a Redis template. Here, it is a bean configured as a StringRedisTemplate, an implementation of RedisTemplate that is focused on the common use of Redis where both keys and values are `String`s.

The main() method kicks everything off by creating a Spring application context. The application context then starts the message listener container, and the message listener container bean starts listening for messages. The main() method then retrieves the StringRedisTemplate bean from the application context and uses it to send a "Hello from Redis!" message on the "chat" topic. Finally, it closes the Spring application context and the application ends.

**Build an executable JAR**

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-messaging-redis/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-messaging-redis/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-messaging-redis-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-messaging-redis-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

**Run the service**

If you are using Gradle, you can run your service at the command line this way:

./gradlew clean build && java -jar build/libs/gs-messaging-redis-0.1.0.jar

|  |  |
| --- | --- |
|  | If you are using Maven, you can run your service by typing mvn clean package && java -jar target/gs-messaging-redis-0.1.0.jar. |

You should see the following output:

Sending message...

Received <Hello from Redis!>

**Summary**

Congratulations! You’ve just developed a simple publish-and-subscribe application with Spring and Redis.

##### 2.1.13Building Android Projects with Gradle

This guide walks you through the process of using Gradle to build a simple Android project.

**What you’ll build**

You’ll create a simple Spring app for Android and then build it with Gradle.

**What you’ll need**

* About 15 minutes
* A favorite text editor or IDE
* [Android SDK](http://developer.android.com/sdk/index.html)
* An Android device or Emulator

**How to complete this guide**

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to [Set up the project](http://spring.io/guides/gs/gradle-android/#scratch).

To **skip the basics**, do the following:

* [Download](https://github.com/spring-guides/gs-gradle-android/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-gradle-android.git>
* cd into gs-gradle-android/initial
* Jump ahead to [Install Gradle](http://spring.io/guides/gs/gradle-android/#initial).

**When you’re finished**, you can check your results against the code in gs-gradle-android/complete.

**Set up the project**

First, you need to set up an Android project for Gradle to build. To keep the focus on Gradle, make the project as simple as possible for now. If this is your first time working with Android projects, refer to [Getting Started with Android](http://spring.io/guides/gs/android) to help configure your development environment.

**Create the directory structure**

In a project directory of your choosing, create the following subdirectory structure; for example, with the following command on Mac or Linux:

mkdir -p src/main/java/org/hello

└── src

└── main

└── java

└── org

└── hello

**Create an Android manifest**

The [Android Manifest](http://developer.android.com/guide/topics/manifest/manifest-intro.html) contains all the information required to run an Android application, and it cannot build without one.

src/main/AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="org.hello"

android:versionCode="1"

android:versionName="1.0.0" >

<application android:label="@string/app\_name" >

<activity

android:name=".HelloActivity"

android:label="@string/app\_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

Add a text string. Text strings can be referenced from the application or from other resource files.

src/main/res/values/strings.xml

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">Android Gradle</string>

</resources>

Here you define the visual structure for the user interface of your application.

src/main/res/layout/hello\_layout.xml

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

>

<TextView

android:id="@+id/text\_view"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

/>

</LinearLayout>

Within the src/main/java/org/hello directory, you can create any Java classes you want. To maintain consistency with the rest of this guide, create the following class:

src/main/java/org/hello/HelloActivity.java

package org.hello;

import android.app.Activity;

import android.os.Bundle;

import android.widget.TextView;

public class HelloActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.hello\_layout);

}

@Override

public void onStart() {

super.onStart();

TextView textView = (TextView) findViewById(R.id.text\_view);

textView.setText("Hello world!");

}

}

**Install Gradle**

Now that you have a project that you can build with Gradle, you can install Gradle.

**1.** Download the latest version of Gradle (1.8 as of this writing) from the {Gradle Downloads} page.

|  |  |
| --- | --- |
|  | Only the binaries are required, so look for the link to gradle-1.8-bin.zip. Alternatively, you can choose gradle-1.8-all.zip to download the sources and documentation as well as the binaries. |

**2.** Unzip the archive and place it in a location of your choosing. For example, on Linux or Mac, you may want to place it in the root of your user directory. See the [Installing Gradle](http://www.gradle.org/docs/current/userguide/installation.html) page for additional details.

**3.** Configure the GRADLE\_HOME environment variable based on the location where you installed Gradle.

**Mac/Linux**

export GRADLE\_HOME=/<installation location>/gradle-1.8

export PATH=${PATH}:$GRADLE\_HOME/bin

**Windows**

set GRADLE\_HOME=C:\<installation location>\gradle-1.8

set PATH=%PATH%;%GRADLE\_HOME%\bin

**4.** Test the Gradle installation with following command:

$ gradle

If the installation is correct, you see a welcome message:

:help

Welcome to Gradle 1.8.

To run a build, run gradle <task> ...

To see a list of available tasks, run gradle tasks

To see a list of command-line options, run gradle --help

BUILD SUCCESSFUL

Total time: 2.923 secs

You now have Gradle installed.

**Find out what Gradle can do**

Before you even create a build.gradle file for the project, you can ask Gradle what tasks are available:

gradle tasks

You should see a list of available tasks. Assuming you run Gradle in a folder that does not contain a build.gradle file, you see basic tasks such as the following:

:tasks

== All tasks runnable from root project

== Help tasks

dependencies - Displays all dependencies declared in root project 'gs-gradle-android'.

dependencyInsight - Displays the insight into a specific dependency in root project 'gs-gradle-android'.

help - Displays a help message

projects - Displays the sub-projects of root project 'gs-gradle-android'.

properties - Displays the properties of root project 'gs-gradle-android'.

tasks - Displays the tasks runnable from root project 'gs-gradle-android' (some of the displayed tasks may belong to subprojects).

To see all tasks and more detail, run with --all.

BUILD SUCCESSFUL

Total time: 2.706 secs

Even though these tasks are available, they do not offer much value without a project build configuration. As you flesh out the build.gradle file, some of these tasks become more useful. The list of tasks will grow as you add plugins to the build.gradle file. You can run **gradle tasks** again to see what new tasks are available.

**Build Android code**

The most simple Android project has the following build.gradle file:

build.gradle

buildscript {

repositories {

mavenCentral()

}

dependencies {

classpath 'com.android.tools.build:gradle:0.5.6'

}

}

apply plugin: 'android'

android {

buildToolsVersion "18.0.1"

compileSdkVersion 18

}

This build configuration brings a significant amount of power. Run **gradle tasks** again, and you see new tasks for building the project, creating JavaDoc, and running tests. The complete output can be seen below:

:tasks

== All tasks runnable from root project

== Android tasks

androidDependencies - Displays the Android dependencies of the project

signingReport - Displays the signing info for each variant

== Build tasks

assemble - Assembles all variants of all applications and secondary packages.

assembleDebug - Assembles all Debug builds

assembleRelease - Assembles all Release builds

assembleTest - Assembles the Test build for the Debug build

build - Assembles and tests this project.

buildDependents - Assembles and tests this project and all projects that depend on it.

buildNeeded - Assembles and tests this project and all projects it depends on.

clean - Deletes the build directory.

== Build Setup tasks

setupBuild - Initializes a new Gradle build. [incubating]

wrapper - Generates Gradle wrapper files. [incubating]

== Help tasks

dependencies - Displays all dependencies declared in root project 'initial'.

dependencyInsight - Displays the insight into a specific dependency in root project 'initial'.

help - Displays a help message

projects - Displays the sub-projects of root project 'initial'.

properties - Displays the properties of root project 'initial'.

tasks - Displays the tasks runnable from root project 'initial'.

== Install tasks

installDebug - Installs the Debug build

installTest - Installs the Test build for the Debug build

uninstallAll - Uninstall all applications.

uninstallDebug - Uninstalls the Debug build

uninstallRelease - Uninstalls the Release build

uninstallTest - Uninstalls the Test build for the Debug build

== Verification tasks

check - Runs all checks.

connectedCheck - Runs all device checks on currently connected devices.

connectedInstrumentTest - Installs and runs the tests for Build 'Debug' on connected devices.

deviceCheck - Runs all device checks using Device Providers and Test Servers.

Rules

-----

Pattern: build<ConfigurationName>: Assembles the artifacts of a configuration.

Pattern: upload<ConfigurationName>: Assembles and uploads the artifacts belonging to a configuration.

Pattern: clean<TaskName>: Cleans the output files of a task.

To see all tasks and more detail, run with --all.

BUILD SUCCESSFUL

Total time: 7.007 secs

You’ll use the **gradle build** task frequently. This task compiles, tests, and packages the code into an APK file. You can run it like this:

gradle build

After a few seconds, you see "BUILD SUCCESSFUL". The complete output can be seen below:

:prepareDebugDependencies

:compileDebugAidl UP-TO-DATE

:generateDebugBuildConfig UP-TO-DATE

:mergeDebugAssets UP-TO-DATE

:compileDebugRenderscript UP-TO-DATE

:mergeDebugResources UP-TO-DATE

:processDebugManifest UP-TO-DATE

:processDebugResources UP-TO-DATE

:compileDebug UP-TO-DATE

:dexDebug UP-TO-DATE

:processDebugJavaRes UP-TO-DATE

:validateDebugSigning

:packageDebug UP-TO-DATE

:assembleDebug UP-TO-DATE

:prepareReleaseDependencies

:compileReleaseAidl UP-TO-DATE

:generateReleaseBuildConfig UP-TO-DATE

:mergeReleaseAssets UP-TO-DATE

:compileReleaseRenderscript UP-TO-DATE

:mergeReleaseResources UP-TO-DATE

:processReleaseManifest UP-TO-DATE

:processReleaseResources UP-TO-DATE

:compileRelease UP-TO-DATE

:dexRelease UP-TO-DATE

:processReleaseJavaRes UP-TO-DATE

:packageRelease UP-TO-DATE

:assembleRelease UP-TO-DATE

:assemble UP-TO-DATE

:check UP-TO-DATE

:build UP-TO-DATE

BUILD SUCCESSFUL

Total time: 6.944 secs

You can see results of the build process in the build folder. Here you see several folders related to various parts of the build or application. The assembled Android package resides in the apk folder. The APK file here is ready to be deployed to a device or emulator.

**Declare dependencies**

The simple Hello World sample is completely self-contained and does not depend on any additional libraries. Most applications, however, depend on external libraries to handle common and/or complex functionality.

For example, suppose you want the application to print the current date and time. You could use the date and time facilities in the native Java libraries, but you can make things more interesting by using the Joda Time libraries.

To do this, modify HelloActivity.java to look like this:

src/main/java/org/hello/HelloActivity.java

package org.hello;

import org.joda.time.LocalTime;

import android.app.Activity;

import android.os.Bundle;

import android.widget.TextView;

public class HelloActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.hello\_layout);

}

@Override

public void onStart() {

super.onStart();

LocalTime currentTime = new LocalTime();

TextView textView = (TextView) findViewById(R.id.text\_view);

textView.setText("The current local time is: " + currentTime);

}

}

In this example, we are using Joda Time’s LocalTime class to retrieve and display the current time.

If you ran gradle build to build the project now, the build would fail because you have not declared Joda Time as a compile dependency in the build. You can fix that by adding the following lines to build.gradle:

repositories { mavenCentral() }

dependencies {

compile "joda-time:joda-time:2.2"

}

The first line here indicates that the build should resolve its dependencies from the Maven Central repository. Gradle leans heavily on many conventions and facilities established by the Maven build tool, including the option of using Maven Central as a source of library dependencies.

Within the dependencies block, you declare a single dependency for Joda Time. Specifically, you are asking for (reading right to left) version 2.2 of the joda-time library, in the joda-time group.

Another thing to note about this dependency is that it is a compile dependency, indicating that it should be available during compile-time. Now if you run gradle build, Gradle should resolve the Joda Time dependency from the Maven Central repository and the build will be successful.

**Build your project with Gradle Wrapper**

The Gradle Wrapper is the preferred way of starting a Gradle build. It consists of a batch script for Windows support and a shell script for support on OS X and Linux. These scripts allow you to run a Gradle build without requiring that Gradle be installed on your system. You can install the wrapper into your project by adding the following lines to the build.gradle:

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

Run the following command to download and initialize the wrapper scripts:

gradle wrapper

After this task completes, you will notice a few new files. The two scripts are in the root of the folder, while the wrapper jar and properties files have been added to a new gradle/wrapper folder.

└── initial

└── gradlew

└── gradlew.bat

└── gradle

└── wrapper

└── gradle-wrapper.jar

└── gradle-wrapper.properties

The Gradle Wrapper is now available for building your project. It can be used in the exact same way as an installed version of Gradle. Run the wrapper script to perform the build task, just like you did previously:

./gradlew build

The first time you run the wrapper for a specified version of Gradle, it downloads and caches the Gradle binaries for that version. The Gradle Wrapper files are designed to be committed to source control so that anyone can build the project without having to first install and configure a specific version of Gradle.

Here is the completed build.gradle file:

build.gradle

buildscript {

repositories {

mavenCentral()

}

dependencies {

classpath 'com.android.tools.build:gradle:0.5.6'

}

}

apply plugin: 'android'

android {

buildToolsVersion "18.0.1"

compileSdkVersion 18

}

repositories {

mavenCentral()

}

dependencies {

compile("joda-time:joda-time:2.2")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

**Summary**

Congratulations! You have created a simple yet effective Gradle build file for building Android projects.

##### 2.1.14Accessing Data with Neo4j

This guide walks you through the process of using Spring Data to build an application with Neo4j.

What you’ll build

You’ll use Neo4j’s [NoSQL](http://spring.io/understanding/NoSQL) graph-based data store to build an embedded Neo4j server, store entities and relationships, and develop queries.

What you’ll need

About 15 minutes

A favorite text editor or IDE

[JDK 1.6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later

[Gradle 1.8+](http://www.gradle.org/downloads) or [Maven 3.0+](http://maven.apache.org/download.cgi)

You can also import the code from this guide as well as view the web page directly into [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts) and work your way through it from there.

How to complete this guide

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To start from scratch, move on to [Set up the project](http://spring.io/guides/gs/accessing-data-neo4j/#scratch).

To skip the basics, do the following:

[Download](https://github.com/spring-guides/gs-accessing-data-neo4j/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-accessing-data-neo4j.git>

cd into gs-accessing-data-neo4j/initial

Jump ahead to [Define a simple entity](http://spring.io/guides/gs/accessing-data-neo4j/#initial).

When you’re finished, you can check your results against the code in gs-accessing-data-neo4j/complete.

Set up the project

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

Create the directory structure

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

Create a Gradle build file

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-accessing-data-neo4j/blob/master/initial/build.gradle). But you can also use Maven. The pom.xml file is included [right here](https://github.com/spring-guides/gs-accessing-data-neo4j/blob/master/initial/pom.xml). If you are using [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts), you can import the guide directly.

build.gradle

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-milestone" }

mavenLocal()

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

jar {

baseName = 'gs-accessing-data-neo4j'

version = '0.1.0'

}

repositories {

mavenCentral()

maven { url "http://repo.spring.io/libs-milestone" }

maven { url "http://m2.neo4j.org" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter:0.5.0.M6")

compile("org.springframework:spring-context:4.0.0.RC1")

compile("org.springframework:spring-tx:4.0.0.RC1")

compile("org.springframework.data:spring-data-neo4j:2.3.2.RELEASE")

compile("javax.validation:validation-api:1.0.0.GA")

testCompile("junit:junit:4.11")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

|  |  |
| --- | --- |
|  | This guide is using [Spring Boot](http://spring.io/guides/gs/spring-boot). |

Define a simple entity

Neo4j captures entities and their relationships, with both aspects being of equal importance. Imagine you are modeling a system where you store a record for each person. But you also want to track a person’s co-workers (teammates in this example). With Neo4j, you can capture all that with some simple annotations.

src/main/java/hello/Person.java

package hello;

import java.util.HashSet;

import java.util.Set;

import org.neo4j.graphdb.Direction;

import org.springframework.data.neo4j.annotation.Fetch;

import org.springframework.data.neo4j.annotation.GraphId;

import org.springframework.data.neo4j.annotation.NodeEntity;

import org.springframework.data.neo4j.annotation.RelatedTo;

@NodeEntity

public class Person {

@GraphId Long id;

public String name;

public Person() {}

public Person(String name) { this.name = name; }

@RelatedTo(type="TEAMMATE", direction=Direction.BOTH)

public @Fetch Set<Person> teammates;

public void worksWith(Person person) {

if (teammates == null) {

teammates = new HashSet<Person>();

}

teammates.add(person);

}

public String toString() {

String results = name + "'s teammates include\n";

if (teammates != null) {

for (Person person : teammates) {

results += "\t- " + person.name + "\n";

}

}

return results;

}

}

Here you have a Person class that has only one attribute, the name. You have two constructors, an empty one as well as one for the name. To use Neo4j later on, you need the empty constructor. The name-based one is for convenience.

|  |  |
| --- | --- |
|  | In this guide, the typical getters and setters are omitted for brevity. |

The Person class is annotated @NodeEntity. When Neo4j stores it, it results in the creation of a new node. This class also has an id marked @GraphId. Neo4j uses @GraphId internally to track the data.

The next important piece is the set of teammates. It is a simple Set<Person>, but marked up as @RelatedTo. This means that every member of this set is expected to also exist as a separate Person node. Note how the direction is set to BOTH. This means that when you generate a TEAMMATE relationship in one direction, it exists in the other direction as well. There is also a @Fetch annotation on this field as well. This causes the teammates to be eagerly retrieved. Otherwise you would have to use neo4jTemplate.fetch().

With the worksWith() method, you can easily link people together.

Finally, you have a convenient toString() method to print out the person’s name and that person’s co-workers.

Create simple queries

Spring Data Neo4j is focused on storing data in Neo4j. But it inherits functionality from the Spring Data Commons project, including the ability to derive queries. Essentially, you don’t have to learn the query language of Neo4j, but can simply write a handful of methods and the queries are written for you.

To see how this works, create an interface that queries Person nodes.

src/main/java/hello/PersonRepository.java

package hello;

import org.springframework.data.neo4j.repository.GraphRepository;

public interface PersonRepository extends GraphRepository<Person> {

Person findByName(String name);

Iterable<Person> findByTeammatesName(String name);

}

PersonRepository extends the GraphRepository class and plugs in the type it operates on: Person. Out-of-the-box, this interface comes with many operations, including standard CRUD (create-read-update-delete) operations.

But you can define other queries as needed by simply declaring their method signature. In this case, you added findByName, which seeks nodes of type Person`and finds the one that matches on `name. You also have findByTeammatesName, which looks for a Person node, drills into each entry of the teammates field, and matches based on the teammate’s name.

Let’s wire this up and see what it looks like!

Create an Application class

Create an Application class with all the components.

src/main/java/hello/Application.java

package hello;

import java.io.File;

import org.neo4j.graphdb.Transaction;

import org.neo4j.kernel.EmbeddedGraphDatabase;

import org.neo4j.kernel.impl.util.FileUtils;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.neo4j.config.EnableNeo4jRepositories;

import org.springframework.data.neo4j.config.Neo4jConfiguration;

import org.springframework.data.neo4j.core.GraphDatabase;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

@Configuration

@EnableNeo4jRepositories

public class Application extends Neo4jConfiguration implements CommandLineRunner {

@Bean

EmbeddedGraphDatabase graphDatabaseService() {

return new EmbeddedGraphDatabase("accessingdataneo4j.db");

}

@Autowired

PersonRepository personRepository;

@Autowired

GraphDatabase graphDatabase;

public void run(String... args) throws Exception {

Person greg = new Person("Greg");

Person roy = new Person("Roy");

Person craig = new Person("Craig");

System.out.println("Before linking up with Neo4j...");

for (Person person : new Person[]{greg, roy, craig}) {

System.out.println(person);

}

Transaction tx = graphDatabase.beginTx();

try {

personRepository.save(greg);

personRepository.save(roy);

personRepository.save(craig);

greg = personRepository.findByName(greg.name);

greg.worksWith(roy);

greg.worksWith(craig);

personRepository.save(greg);

roy = personRepository.findByName(roy.name);

roy.worksWith(craig);

// We already know that roy works with greg

personRepository.save(roy);

// We already know craig works with roy and greg

tx.success();

} finally {

tx.finish();

}

System.out.println("Lookup each person by name...");

for (String name: new String[]{greg.name, roy.name, craig.name}) {

System.out.println(personRepository.findByName(name));

}

System.out.println("Looking up who works with Greg...");

for (Person person : personRepository.findByTeammatesName("Greg")) {

System.out.println(person.name + " works with Greg.");

}

}

public static void main(String[] args) throws Exception {

FileUtils.deleteRecursively(new File("accessingdataneo4j.db"));

SpringApplication.run(Application.class, args);

}

}

In the configuration, you need to add the @EnableNeo4jRepositories annotation as well as extend the Neo4jConfiguration class to conveniently spin up needed components.

One piece that’s missing is the graph database service bean. In this case, you are using the EmbeddedGraphDatabase, which creates and reuses a file-based data store at accessingdataneo4j.db.

|  |  |
| --- | --- |
|  | In a production environment, you would probably connect to a standalone, running Neo4j server instead. |

You autowire an instance of PersonRepository that you defined earlier. Spring Data Neo4j will dynamically create a concrete class that implements that interface and will plug in the needed query code to meet the interface’s obligations.

The public static void main uses Spring Boot’s SpringApplication.run() to launch the application and invoke the CommandLineRunner that builds the relationships.

In this case, you create three local Person s, Greg, Roy, and Craig. Initially, they only exist in memory. It’s also important to note that no one is a teammate of anyone (yet).

To store anything in Neo4j, you must start a transaction using the graphDatabase. In there, you will save each person. Then, you fetch each person, and link them together.

At first, you find Greg and indicate that he works with Roy and Craig, then persist him again. Remember, the teammate relationship was marked as BOTH, that is, bidirectional. That means that Roy and Craig will have been updated as well.

That’s why when you need to update Roy, it’s critical that you fetch that record from Neo4j first. You need the latest status on Roy’s teammates before adding Craig to the list.

Why is there no code that fetches Craig and adds any relationships? Because you already have! Greg earlier tagged Craig as a teammate, and so did Roy. That means there is no need to update Craig’s relationships again. You can see it as you iterate over each team member and print their information to the console.

Finally, check out that other query where you look backwards, answering the question "who works with whom?"

Build an executable JAR

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-accessing-data-neo4j/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-accessing-data-neo4j/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-accessing-data-neo4j-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-accessing-data-neo4j-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

Run the service

If you are using Gradle, you can run your service at the command line this way:

./gradlew clean build && java -jar build/libs/gs-accessing-data-neo4j-0.1.0.jar

|  |  |
| --- | --- |
|  | If you are using Maven, you can run your service by typing mvn clean package && java -jar target/gs-accessing-data-neo4j-0.1.0.jar. |

You should see something like this (with other stuff like queries as well):

Before linking up with Neo4j...

Greg's teammates include

Roy's teammates include

Craig's teammates include

Lookup each person by name...

Greg's teammates include

- Craig

- Roy

Roy's teammates include

- Craig

- Greg

Craig's teammates include

- Roy

- Greg

Looking up who works with Greg...

Roy works with Greg.

Craig works with Greg.

You can see from the output that initially no one is connected by any relationship. Then after adding people in, they are tied together. Finally, you can see the handy query that looks up people based on teammate.

Summary

Congratulations! You just set up an embedded Neo4j server, stored some simple, related entities, and developed some quick queries.

##### 2.1.15Validating Form Input

This guide walks you through the process of configuring a web application form to support validation.

What you’ll build

You’ll build a simple Spring MVC application that take user input and checks the input using standard validation annotations. You’ll also see how to display the error message on the screen so the user can re-enter a valid input.

What you’ll need

About 15 minutes

A favorite text editor or IDE

[JDK 1.6](http://www.oracle.com/technetwork/java/javase/downloads/index.html) or later

[Gradle 1.8+](http://www.gradle.org/downloads) or [Maven 3.0+](http://maven.apache.org/download.cgi)

You can also import the code from this guide as well as view the web page directly into [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts) and work your way through it from there.

How to complete this guide

Like all Spring’s [Getting Started guides](http://spring.io/guides/gs), you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To start from scratch, move on to [Set up the project](http://spring.io/guides/gs/validating-form-input/#scratch).

To skip the basics, do the following:

[Download](https://github.com/spring-guides/gs-validating-form-input/archive/master.zip) and unzip the source repository for this guide, or clone it using [Git](http://spring.io/understanding/Git): git clone <https://github.com/spring-guides/gs-validating-form-input.git>

cd into gs-validating-form-input/initial

Jump ahead to [Create a Person object](http://spring.io/guides/gs/validating-form-input/#initial).

When you’re finished, you can check your results against the code in gs-validating-form-input/complete.

Set up the project

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with [Gradle](http://gradle.org/) and [Maven](https://maven.apache.org/) is included here. If you’re not familiar with either, refer to [Building Java Projects with Gradle](http://spring.io/guides/gs/gradle) or [Building Java Projects with Maven](http://spring.io/guides/gs/maven).

Create the directory structure

In a project directory of your choosing, create the following subdirectory structure; for example, with mkdir -p src/main/java/hello on \*nix systems:

└── src

└── main

└── java

└── hello

Create a Gradle build file

Below is the [initial Gradle build file](https://github.com/spring-guides/gs-validating-form-input/blob/master/initial/build.gradle). But you can also use Maven. The pom.xml file is included [right here](https://github.com/spring-guides/gs-validating-form-input/blob/master/initial/pom.xml). If you are using [Spring Tool Suite (STS)](http://spring.io/guides/gs/sts), you can import the guide directly.

build.gradle

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-milestone" }

mavenLocal()

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

jar {

baseName = 'gs-validating-form-input'

version = '0.1.0'

}

repositories {

mavenCentral()

maven { url "http://repo.spring.io/libs-milestone" }

}

dependencies {

compile("org.springframework.boot:spring-boot-starter-web:0.5.0.M6")

compile("org.hibernate:hibernate-validator:4.3.1.Final")

compile("org.thymeleaf:thymeleaf-spring3:2.0.16")

testCompile("junit:junit:4.11")

}

task wrapper(type: Wrapper) {

gradleVersion = '1.8'

}

|  |  |
| --- | --- |
|  | This guide is using [Spring Boot](http://spring.io/guides/gs/spring-boot). |

Create a Person object

The application involves validating a user’s age, so first you need to create a class to represent a person.

src/main/java/hello/Person.java

package hello;

import javax.validation.constraints.Min;

import javax.validation.constraints.NotNull;

public class Person {

@NotNull

@Min(18)

private Integer age;

public String toString() {

return "Person(" + age + ")";

}

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

}

The Person class only has one attribute, age. It is flagged with standard validation annotations:

@NotNull won’t allow an empty value

@Min(18) won’t allow if the age is less than 18

In addition to that, you can also see getters/setters for age as well as a convenient toString() method.

Create a web controller

Now that you have defined an entity, it’s time to create a simple web controller.

src/main/java/hello/WebController.java

package hello;

import javax.validation.Valid;

import org.springframework.stereotype.Controller;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

@Controller

public class WebController {

@RequestMapping(value="/", method=RequestMethod.GET)

public String showForm(Person person) {

return "form";

}

@RequestMapping(value="/", method=RequestMethod.POST)

public String enterAge(@Valid Person person, BindingResult bindingResult,

RedirectAttributes redirectAttributes) {

if (bindingResult.hasErrors()) {

redirectAttributes.addFlashAttribute("error", bindingResult.getFieldError().getDefaultMessage());

return "redirect:/";

}

return "results";

}

}

This controller has a GET and a POST method, both mapped to /.

The showForm method returns the form template. It includes a Person in its method signature so the template can associate form attributes with a Person.

The enterAge method accepts three arguments:

A person object marked up with @Valid to gather the attributes filled out in the form you’re about to build.

A bindingResult object so you can test for and retrieve validation errors.

A redirectAttributes object so you can create a flash-scoped error message to show the user what went wrong in the event of an error.

You can retrieve all the attributes from the form bound to the Person object. In the code, you test for errors, and if so, add a flash attribute named error, and redirect the user back to the / page. If there are no errors, you return the results template.

Build an HTML front end

Now you build the "main" page.

src/main/webapp/form.html

<html>

<body>

<form action="#" th:action="@{/}" th:object="${person}" method="post">

<table>

<tr>

<td>How old are you?</td>

<td><input type="text" th:field="\*{age}" /></td>

<td><div id="errors" th:text="${error}" /></td>

</tr>

<tr>

<td><button type="submit">Submit</button></td>

</tr>

</table>

</form>

</body>

</html>

The page contains a simple form with each field in a separate slot of a table. The form is geared to post towards /enterAge. It is marked as being backed up by the person object that you saw in the GET method in the web controller. This is known as a bean-backed form. There is only one field in the Person bean, and you can see it tagged with th:field="\*{age}".

Right next to that entry field is a <div> with th:text="${error}". This gives you a place to insert an error message.

Finally, you have a button to submit. In general, if the user enters an age that violates the @Valid constraints, it will bounce back to this page with the error message on display. If a valid age is entered, the user is routed to the next web page.

src/main/webapp/results.html

<html>

<body>

Congratulations! You are old enough to sign up for this site.

</body>

</html>

|  |  |
| --- | --- |
|  | In this simple example, these web pages don’t have any sophisticated CSS or JavaScript. But for any production web site, it’s valuable to learn how to style your web pages. |

Create an Application class

For this application, you are using the template language of [Thymeleaf](http://www.thymeleaf.org/doc/html/Thymeleaf-Spring3.html). This application needs more than raw HTML.

src/main/java/hello/Application.java

package hello;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.SpringApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.thymeleaf.spring3.SpringTemplateEngine;

import org.thymeleaf.spring3.view.ThymeleafViewResolver;

import org.thymeleaf.templateresolver.ServletContextTemplateResolver;

@Configuration

@ComponentScan

@EnableAutoConfiguration

public class Application {

@Bean

ServletContextTemplateResolver templateResolver() {

ServletContextTemplateResolver resolver = new ServletContextTemplateResolver();

resolver.setSuffix(".html");

resolver.setTemplateMode("HTML5");

return resolver;

}

@Bean

SpringTemplateEngine engine() {

SpringTemplateEngine engine = new SpringTemplateEngine();

engine.setTemplateResolver(templateResolver());

return engine;

}

@Bean

ThymeleafViewResolver viewResolver() {

ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();

viewResolver.setTemplateEngine(engine());

return viewResolver;

}

public static void main(String[] args) throws Exception {

SpringApplication.run(Application.class, args);

}

}

To activate Spring MVC, you would normally add @EnableWebMvc to the Application class. But Spring Boot’s @EnableAutoConfiguration already adds this annotation when it detects spring-webmvc on your classpath. The application also has @ComponentScan to find the annotated @Controller class and its methods.

The extra beans shown in this configuration are used to wire up Thymeleaf and integrate it with Spring MVC. The first one takes view names, appends .html, and looks for that file in src/main/webapp/. The rest are used to perform proper resolution and rendering.

Build an executable JAR

Now that your Application class is ready, you simply instruct the build system to create a single, executable jar containing everything. This makes it easy to ship, version, and deploy the service as an application throughout the development lifecycle, across different environments, and so forth.

Below are the Gradle steps, but if you are using Maven, you can find the updated pom.xml [right here](https://github.com/spring-guides/gs-validating-form-input/blob/master/complete/pom.xml) and build it by typing mvn clean package.

Update your Gradle build.gradle file’s buildscript section, so that it looks like this:

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:0.5.0.M6")

}

}

Further down inside build.gradle, add the following to the list of applied plugins:

apply plugin: 'spring-boot'

You can see the final version of build.gradle [right here](https://github.com/spring-guides/gs-validating-form-input/blob/master/complete/build.gradle).

The [Spring Boot gradle plugin](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-tools/spring-boot-gradle-plugin) collects all the jars on the classpath and builds a single "über-jar", which makes it more convenient to execute and transport your service. It also searches for the public static void main() method to flag as a runnable class.

Now run the following command to produce a single executable JAR file containing all necessary dependency classes and resources:

./gradlew build

If you are using Gradle, you can run the JAR by typing:

java -jar build/libs/gs-validating-form-input-0.1.0.jar

If you are using Maven, you can run the JAR by typing:

java -jar target/gs-validating-form-input-0.1.0.jar

|  |  |
| --- | --- |
|  | The procedure above will create a runnable JAR. You can also opt to [build a classic WAR file](http://spring.io/guides/gs/convert-jar-to-war/) instead. |

Run the web application

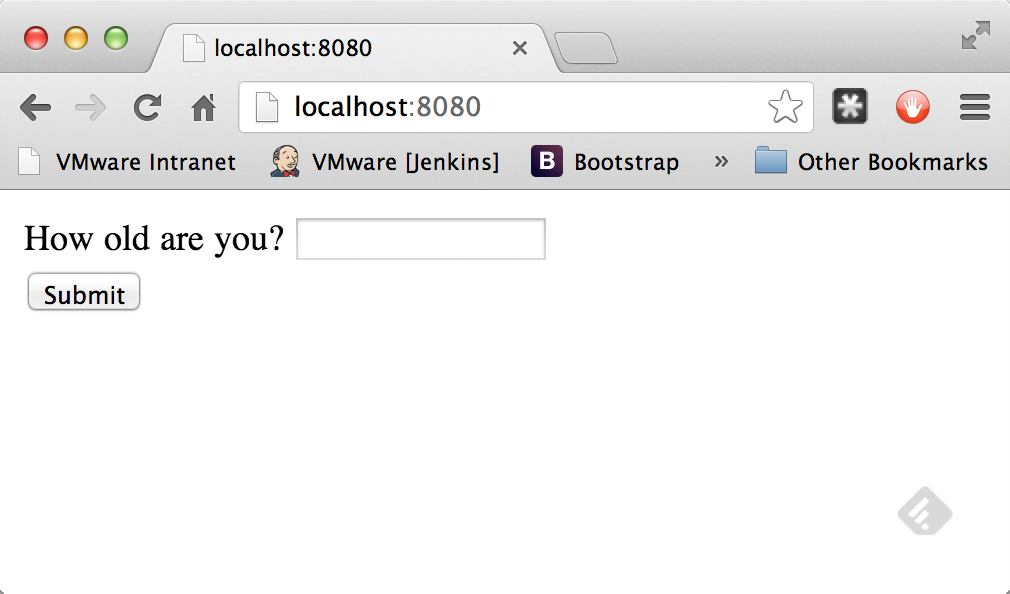
If you are using Gradle, you can run your web application at the command line this way:

./gradlew clean build && java -jar build/libs/gs-validating-form-input-0.1.0.jar

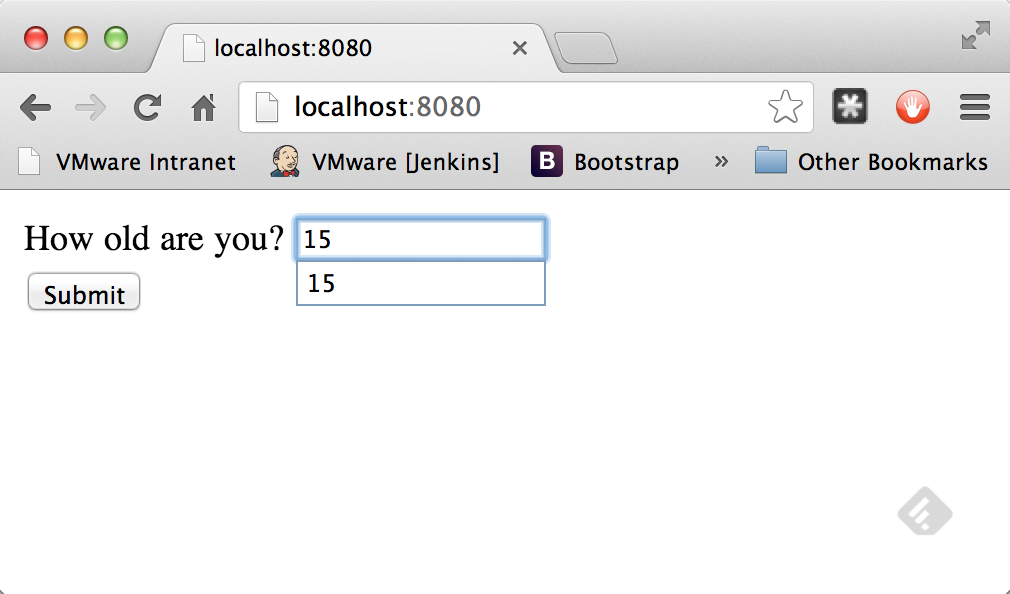
|  |  |
| --- | --- |
|  | If you are using Maven, you can run your web application by typing mvn clean package && java -jar target/gs-validating-form-input-0.1.0.jar. |

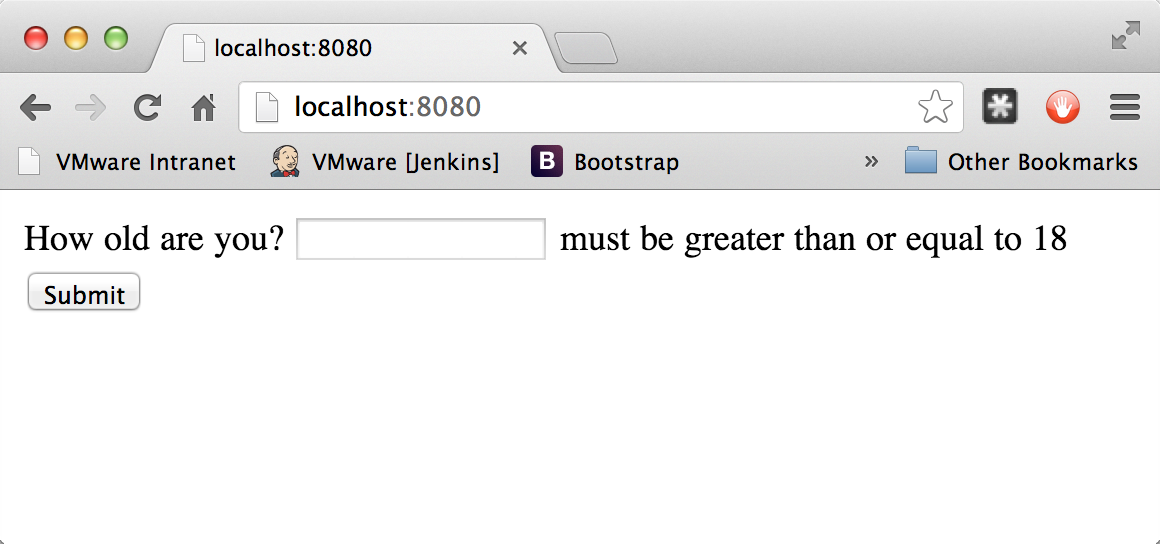
The application should be up and running within a few seconds.

If you visit <http://localhost:8080/>, you should see something like this:

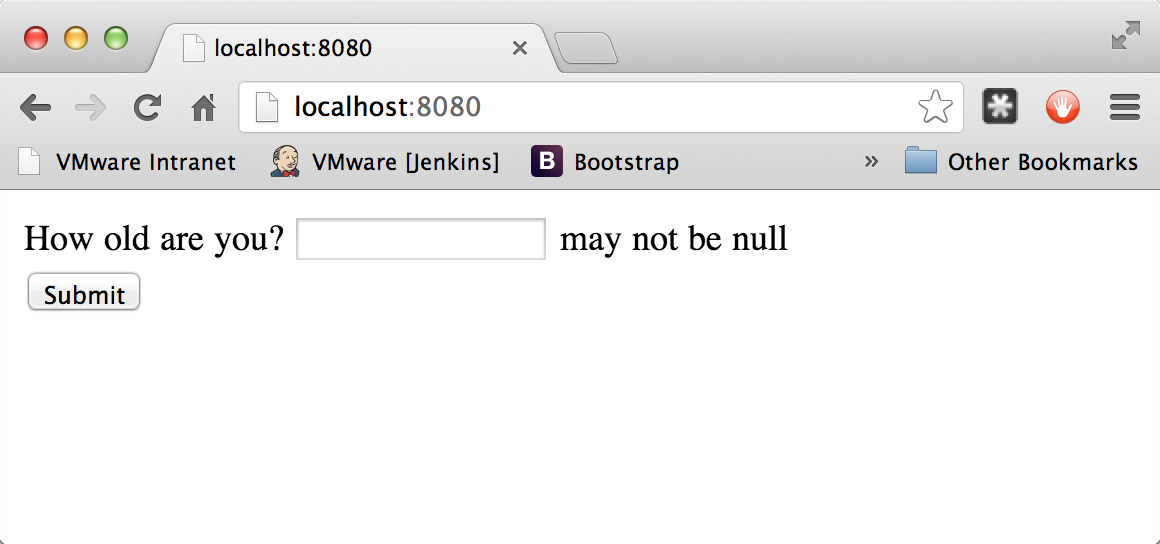


What happens if you enter 15 and click on Submit?





Here you can see that because it violated the constraints in the Person class, you get bounced back to the "main" page. If you click on Submit with nothing in the entry box, you get a different error.



If you enter a valid age, you end up on the results page!



Summary

Congratulations! You have coded a simple web application with validation built into a domain object. This way you can ensure the data meets certain criteria and that the user inputs it correctly.

#### 2.2Spring MVC 简介

  Spring MVC，即 Spring Model-View-Controller，是一个实现了通用开发模式（模型-视图-控制器）的Web框架，它通过一个DispatcherServlet处理HTTP请求、完成资源映射、递交请求给控制器完成业务逻辑，相应数据则通过Model传递给视图解析器解析为相应的页面或数据流返回给客户端。

    这里，我们可以通过Spring官方给出的图示大致了解其内部的工作机制：

[](http://static.oschina.net/uploads/space/2013/0724/175447_j7Eb_1170792.jpg)

    DispatcherServlet作为前端控制器（Front Controller）过滤所有客户端发来的请求，检查请求路径并根据配置好的映射规则，将请求提交给指定的控制器（Controller），完成业务逻辑处理（比如，数据库访问），生成数据模型（model），传递给指定视图解析器（Spring内部已为我们定义好的一系列模板，拿来用即可）解析为相应的视图数据，最后返回客户端响应。

        其实，好比我们最初学习Java Web开发一样，首先要在web.xml文件中配置DispatcherServlet:

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |  |
| --- | --- | --- |
| 01 | <!-- Spring Dispatcher Servlet--> | |
| 02 | <servlet> |

|  |  |  |
| --- | --- | --- |
| 03 | | <servlet-name>SpringDispatcher</servlet-name> |
| 04 | <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class> | | |

|  |  |
| --- | --- |
| 05 | <init-param> |
| 06 | <param-name>contextConfigLocation</param-name> | |

|  |  |  |  |
| --- | --- | --- | --- |
| 07 | <param-value>classpath:config/springDispatcher.xml</param-value> | | |
| 08 | | </init-param> |

|  |  |  |
| --- | --- | --- |
| 09 | <load-on-startup>1</load-on-startup> | |
| 10 | </servlet> |

|  |  |
| --- | --- |
| 11 | <servlet-mapping> |
| 12 | <servlet-name>SpringDispatcher</servlet-name> | |

|  |  |  |
| --- | --- | --- |
| 13 | <url-pattern>/</url-pattern> | |
| 14 | </servlet-mapping> |

    这里配置了名为SpringDispatcher的Servlet，处理所有客户端的请求，contextConfigLocation参数指明了同时要加载的Spring MVC配置信息。

    既然SpringDispatcher会过滤所有的请求，那如果请求的是静态资源的话，我们这样做就有点得不偿失了。不过不用担心，Spring MVC为我们提供了处理静态资源的解决办法：

    在springDispatcher.xml文件中，引入spring mvc标记，并添加<mvc:resource>标签即可：

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |
| --- | --- |
| 01 | <?xml version="1.0" encoding="UTF-8"?> |
| 02 | <beans xmlns="http://www.springframework.org/schema/beans" | |

|  |  |
| --- | --- |
| 03 | xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" |
| 04 | xmlns:mvc="http://www.springframework.org/schema/mvc" |

|  |  |
| --- | --- |
| 05 | xsi:schemaLocation=" |
| 06 | http://www.springframework.org/schema/mvc | |

|  |  |  |  |
| --- | --- | --- | --- |
| 07 | http://www.springframework.org/schema/mvc/spring-mvc.xsd | | |
| 08 | | http://www.springframework.org/schema/beans |

|  |  |  |  |
| --- | --- | --- | --- |
| 09 | http://www.springframework.org/schema/beans/spring-beans.xsd> | | |
| 10 | | <!-- Handle requests for static resources --> |

|  |  |  |
| --- | --- | --- |
| 11 | <mvc:resources mapping="/resources/\*\*"  location="/resources/"/> | |
| 12 | </beans> |

    如上所配置，<mvc:resources>会将所有直接返回以/resources/开始的静态资源请求，而不会通过SpringDispatcher进行处理。

    DispatcherServlet配置好后，接下来就需要创建我们的控制器类了，Spring MVC里我们可以通过组件扫描来注册我们所写的控制器，自动织入所需的bean：

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |  |
| --- | --- | --- |
| 1 | | http://www.springframework.org/schema/context |
| 2 | http://www.springframework.org/schema/context/spring-context.xsd> | | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | <!-- Enable annotation-driven features --> | |

|  |  |  |
| --- | --- | --- |
| 5 | <mvc:annotation-driven/> | |
| 6 |  |

|  |  |
| --- | --- |
| 7 | <!-- Enable component-scan features --> |
| 8 | <context:component-scan base-package="com.alan"/> | |

     控制器类如下：

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |
| --- | --- |
| 01 | @Controller |
| 02 | @RequestMapping("/user") | |

|  |  |  |
| --- | --- | --- |
| 03 | public class UserController { | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | private final UserService userService; | |
| 06 |  |

|  |  |
| --- | --- |
| 07 | @Autowired |
| 08 | public UserController(UserService userService) { | |

|  |  |  |
| --- | --- | --- |
| 09 | this.userService = userService; | |
| 10 | } |

|  |  |
| --- | --- |
| 11 |  |
| 12 | @RequestMapping("/queryAll") | |

|  |  |  |
| --- | --- | --- |
| 13 | public String queryAll(@RequestParam("type") int type, Model model) { | |
| 14 | if(type == 1) { |

|  |  |  |
| --- | --- | --- |
| 15 | List<User> users = userService.findAll(); | |
| 16 | model.addAttribute("users", users); |

|  |  |
| --- | --- |
| 17 | } |
| 18 | return "UserList"; | |

|  |  |  |
| --- | --- | --- |
| 19 | } | |
| 20 | } |

    通过注解技术，我们可以很方便的将我们的业务类注册给控制器，在初始化时由Spring容器帮我们完成依赖注入。其中@RequestMapping注解则告诉Spring所有以“/user”开始的请求将由UserController来处理，而"/user/queryAll"则交由queryAll方法处理。@RequestParam则会接收URL请求参数，这里为type,并且自动转化为对应的参数类型。Model即为数据模型，由Spring提供，我们可以将处理后的结果数据绑定到Model上，返回Model给指定视图解析器。queryAll方法最后的return "UserList"意思是告诉视图解析器返回哪一个页面。这里我们需要再增加视图解析器的配置到springDsipatcher.xml中：

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |  |
| --- | --- | --- |
| 1 | | <!-- Configure view resolver --> |
| 2 | <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"> | | |

|  |  |  |
| --- | --- | --- |
| 3 | <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/> | |
| 4 | <property name="prefix" value="/WEB-INF/views/"/> |

|  |  |  |
| --- | --- | --- |
| 5 | <property name="suffix" value=".jsp"/> | |
| 6 | </bean> |

    以上XML配置表明，我们使用JstlView视图（支持JSTL），并且返回“/WEB-INF/views/”下面所有以“.jsp”为后缀的文件。本例中，即为“ /WEB-INF/views/UserList.jsp ”。

    是不是很easy呢？不过，可能有的同学要问了，如果我不想返回页面，而是直接返回字符串或者IO流怎么办。好吧，Spring框架当然也想到了。我们可以在方法上增加@ResponseBody标记以表明直接返回响应内容，而不是转交视图解析器处理，就像下面这样：

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |  |
| --- | --- | --- |
| 01 | @RequestMapping("/queryAll2") | |
| 02 | @ResponseBody |

|  |  |  |
| --- | --- | --- |
| 03 | public String queryAll2(@RequestParam("type") int type) { | |
| 04 | JSONObject jsonObj = new JSONObject(); |

|  |  |
| --- | --- |
| 05 | if(type == 2) { |
| 06 | List<User> users = userService.findAll(); | |

|  |  |  |
| --- | --- | --- |
| 07 | jsonObj.put("users", users); | |
| 08 | } |

|  |  |  |
| --- | --- | --- |
| 09 | return jsonObj.toString(); | |
| 10 | } |

    我们即可很轻松的返回字符串数据，比如返回JSON字符串。

    InternalResourceViewResolver通常使用转发的方式返回页面数据。如果我们需要重定向到某个页面，则可以在方法返回的时候增加“redirect:”标记即可：

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |
| --- | --- |
| 1 | return "redirect:/UserList"; |

    当然，如果需要转发至某个Controller的方法也很方便：

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |
| --- | --- |
| 1 | return "forward:/user/queryAll2?type=2" |

    此外，经常使用Struts的同学也比较喜欢表单数据绑定到Model的功能以减轻自己获取所有请求参数的繁琐工作，Spring亦有提供实现。即可通过@ModelAttribute标记获取表单参数（这需要Spring MVC所提供的form tag库的支持）：

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |
| --- | --- |
| 1 | @Controller |
| 2 | public class HelloWorldController { | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | @RequestMapping(value = "/helloWorld", method=RequestMethod.POST) | |

|  |  |  |
| --- | --- | --- |
| 5 | public String helloWorld(@ModelAttribute User user) { | |
| 6 | return "helloWorld"; |

|  |  |  |
| --- | --- | --- |
| 7 | } | |
| 8 | } |

    好像还少了点什么对不对？是的，我如果要在Controller里获得ServletContext对象呢？别担心，只要我们implements ServletContext即可，这样我们就可以在Controller里获取Servlet上下文，因而什么request、response都不在话下了，这里给出一个上传JPG图片的例子：

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |
| --- | --- |
| 01 | @Controller |
| 02 | @RequestMapping("/upload") | |

|  |  |  |
| --- | --- | --- |
| 03 | public class FileUploadController implements ServletContextAware { | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | private ServletContext servletContext; | |
| 06 |  |

|  |  |  |
| --- | --- | --- |
| 07 | public void setServletContext(ServletContext servletContext) { | |
| 08 | this.servletContext = servletContext; |

|  |  |  |
| --- | --- | --- |
| 09 | } | |
| 10 |  |

|  |  |
| --- | --- |
| 11 | @RequestMapping("/jpg") |
| 12 | public String uploadJPG(@RequestParam("image") MultipartFile image, Model model) { | |

|  |  |
| --- | --- |
| 13 | try { |
| 14 | if(!image.isEmpty()) { | |

|  |  |  |
| --- | --- | --- |
| 15 | validateImage(image, "image/jpeg"); | |
| 16 | saveImage(image); |

|  |  |  |
| --- | --- | --- |
| 17 | model.addAttribute("img", "resources/upload" + File.separator + image.getName()); | |
| 18 | } |

|  |  |
| --- | --- |
| 19 | } catch (UploadImageException e) { |
| 20 | model.addAttribute("msg", e.getMessage()); | |

|  |  |  |
| --- | --- | --- |
| 21 | return "uploadError"; | |
| 22 | } |

|  |  |  |
| --- | --- | --- |
| 23 | return "images"; | |
| 24 | } |

|  |  |
| --- | --- |
| 25 |  |
| 26 | private void saveImage(MultipartFile image) throws UploadImageException { | |

|  |  |  |
| --- | --- | --- |
| 27 | String name = image.getName(); | |
| 28 | try { |

|  |  |
| --- | --- |
| 29 | File file = new File(servletContext.getRealPath("/resources/upload") + File.separator + name); |
| 30 | FileUtils.writeByteArrayToFile(file, image.getBytes()); |

|  |  |
| --- | --- |
| 31 | } catch (IOException e) { |
| 32 | throw new UploadImageException("保存图片出错！"); | |

|  |  |  |
| --- | --- | --- |
| 33 | } | |
| 34 |  |

|  |  |  |
| --- | --- | --- |
| 35 | } | |
| 36 |  |

|  |  |  |
| --- | --- | --- |
| 37 | private void validateImage(MultipartFile image, String type) throws UploadImageException { | |
| 38 | if(!image.getContentType().equals(type)) { |

|  |  |  |
| --- | --- | --- |
| 39 | throw new UploadImageException("只接受JPG格式的文件！"); | |
| 40 | } |

|  |  |  |
| --- | --- | --- |
| 41 | } | |
| 42 |  |

|  |  |
| --- | --- |
| 43 | } |

    还有，还有，记得我们经常会用到Session来保存一些共享数据，Spring MVC里可以在Controller上加上@SessionAttributes标记来完成这个功能：

[view source](http://my.oschina.net/moson/blog/146808#viewSource)



[print](http://my.oschina.net/moson/blog/146808#printSource)[?](http://my.oschina.net/moson/blog/146808#about)

|  |  |
| --- | --- |
| 01 | @Controller |
| 02 | @RequestMapping("/user") | |

|  |  |
| --- | --- |
| 03 | @SessionAttributes("user") |
| 04 | public class UserController { | |

|  |  |
| --- | --- |
| 05 |  |
| 06 | private final UserService userService; | |

|  |  |
| --- | --- |
| 07 |  |
| 08 | @Autowired | |

|  |  |  |
| --- | --- | --- |
| 09 | public UserController(UserService userService) { | |
| 10 | this.userService = userService; |

|  |  |
| --- | --- |
| 11 | } |
| 12 | /\*\* | |

|  |  |
| --- | --- |
| 13 | ....... |
| 14 | \*/ | |

|  |  |
| --- | --- |
| 15 | } |

    惊喜远远不仅如此，Spring MVC还提供了更多，这里不再一一列举，有了以上的这些简单介绍，想必 你 对Spring MVC的开发模式有了一定的了解，不得不说这个框架用起来其实还是蛮方便、蛮体贴的。

#### 2.3Spring简介

Spring是一个开源框架，它由Rod Johnson创建。它是为了解决企业应用开发的复杂性而创建的。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、[可测试性](http://zhidao.baidu.com/search?word=%E5%8F%AF%E6%B5%8B%E8%AF%95%E6%80%A7&fr=qb_search_exp&ie=utf8)和[松耦合](http://zhidao.baidu.com/search?word=%E6%9D%BE%E8%80%A6%E5%90%88&fr=qb_search_exp&ie=utf8)的角度而言，任何Java应用都可以从Spring中受益。

  ◆目的：解决企业应用开发的复杂性

  ◆功能：使用基本的JavaBean代替EJB，并提供了更多的企业应用功能

  ◆范围：任何Java应用

  简单来说，Spring是一个[轻量级](http://zhidao.baidu.com/search?word=%E8%BD%BB%E9%87%8F%E7%BA%A7&fr=qb_search_exp&ie=utf8)的[控制反转](http://zhidao.baidu.com/search?word=%E6%8E%A7%E5%88%B6%E5%8F%8D%E8%BD%AC&fr=qb_search_exp&ie=utf8)(IoC)和面向[切面](http://zhidao.baidu.com/search?word=%E5%88%87%E9%9D%A2&fr=qb_search_exp&ie=utf8)(AOP)的容器框架。

  ◆轻量——从大小与开销两方面而言Spring都是轻量的。完整的Spring框架可以在一个大小只有1MB多的JAR文件里发布。并且Spring所需的处理开销也是微不足道的。此外，Spring是非侵入式的：典型地，Spring应用中的对象不依赖于Spring的特定类。

  ◆[控制反转](http://zhidao.baidu.com/search?word=%E6%8E%A7%E5%88%B6%E5%8F%8D%E8%BD%AC&fr=qb_search_exp&ie=utf8)——Spring通过一种称作[控制反转](http://zhidao.baidu.com/search?word=%E6%8E%A7%E5%88%B6%E5%8F%8D%E8%BD%AC&fr=qb_search_exp&ie=utf8)（IoC）的技术促进了[松耦合](http://zhidao.baidu.com/search?word=%E6%9D%BE%E8%80%A6%E5%90%88&fr=qb_search_exp&ie=utf8)。当应用了IoC，一个对象依赖的其它对象会通过被动的方式传递进来，而不是这个对象自己创建或者查找依赖对象。你可以认为IoC与JNDI相反——不是对象从容器中查找依赖，而是容器在对象初始化时不等对象请求就主动将依赖传递给它。

  ◆面向[切面](http://zhidao.baidu.com/search?word=%E5%88%87%E9%9D%A2&fr=qb_search_exp&ie=utf8)——Spring提供了面向[切面](http://zhidao.baidu.com/search?word=%E5%88%87%E9%9D%A2&fr=qb_search_exp&ie=utf8)编程的丰富支持，允许通过分离应用的[业务逻辑](http://zhidao.baidu.com/search?word=%E4%B8%9A%E5%8A%A1%E9%80%BB%E8%BE%91&fr=qb_search_exp&ie=utf8)与系统级服务（例如审计（auditing）和事务（transaction）管理）进行[内聚性](http://zhidao.baidu.com/search?word=%E5%86%85%E8%81%9A%E6%80%A7&fr=qb_search_exp&ie=utf8)的开发。应用对象只实现它们应该做的——完成[业务逻辑](http://zhidao.baidu.com/search?word=%E4%B8%9A%E5%8A%A1%E9%80%BB%E8%BE%91&fr=qb_search_exp&ie=utf8)——[仅此而已](http://zhidao.baidu.com/search?word=%E4%BB%85%E6%AD%A4%E8%80%8C%E5%B7%B2&fr=qb_search_exp&ie=utf8)。它们并不负责（甚至是意识）其它的系统级关注点，例如日志或事务支持。

  ◆容器——Spring包含并管理应用对象的配搜索置和生命周期，在这个意义上它是一种容器，你可以配置你的每个bean如何被创建——基于一个可配置原型（prototype），你的bean可以创建一个单独的实例或者每次需要时都生成一个新的实例——以及它们是如何相互关联的。然而，Spring不应该被[混同](http://zhidao.baidu.com/search?word=%E6%B7%B7%E5%90%8C&fr=qb_search_exp&ie=utf8)于传统的重量级的EJB容器，它们经常是庞大与笨重的，难以使用。

  ◆框架——Spring可以将简单的组件配置、组合成为复杂的应用。在Spring中，应用对象被声明式地组合，典型地是在一个XML文件里。Spring也提供了很多基础功能（[事务管理](http://zhidao.baidu.com/search?word=%E4%BA%8B%E5%8A%A1%E7%AE%A1%E7%90%86&fr=qb_search_exp&ie=utf8)、[持久化](http://zhidao.baidu.com/search?word=%E6%8C%81%E4%B9%85%E5%8C%96&fr=qb_search_exp&ie=utf8)框架集成等等），将应用逻辑的开发留给了你。

  所有Spring的这些特征使你能够编写更干净、更可管理、并且更易于测试的代码。它们也为Spring中的各种模块提供了基础支持。

 几天前根据JavaEye在Twitter上看到的消息，Spring 3.0RC1版发布了。目前Spring官方还没有给出下载，是由Amazon的Arjen Poutsma老兄在Twitter上分享的地址。本篇文章是JavaEye的博主ixu在试用Spring 3.0之后的一些小感想：

  下面说说初步看到的变化：

  1、项目结构与构建变化

  解压后的立即发现，Spring 3.0的项目结构已经发现了巨大变化：

  1、Spring3采用多项目结构源码组织，不再是以前的单一方式，共26个项目，差不多每个项目对于一个分发的jar包，不过有些项目是空的，或者是为了构建而设。

  2、不再提供完整打包文件spring.jar，而是20个jar(或称bundle)，一方面应该也是向osgi靠拢。

  Spring 3.0的readme中说道：

  Note that this release does not contain a 'spring.jar' file anymore, in contrast to previous Spring generations. Furthermore, the jar file names follow bundle repository conventions now.

  ([51CTO](http://zhidao.baidu.com/search?word=51CTO&fr=qb_search_exp&ie=utf8)编辑快译：与之前的Spring版本相反，此次发布不再包括spring.jar文件了。新版本中的jar文件命名由bundle版本库的规则所决定。)

  3、采用Ivy为主构建方式，当然仍然有Maven，项目结构由Maven管理。另外没有打包全部的依赖包了，整个下载包比2.5的小了近一半

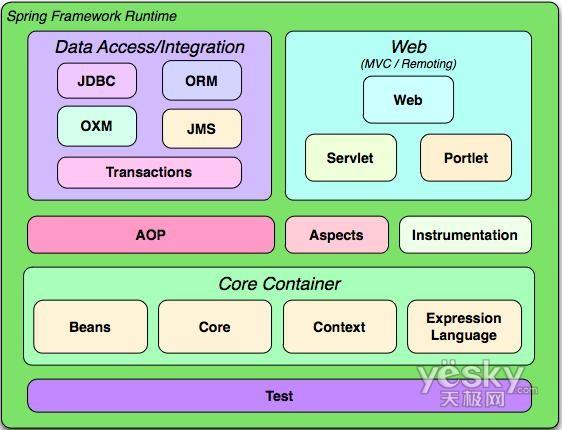
  4、Spring3已经完全采用Java5/6开发和编译构建，因此应该是不再支持Java1.4及更早版本了

  2、[框架结构](http://zhidao.baidu.com/search?word=%E6%A1%86%E6%9E%B6%E7%BB%93%E6%9E%84&fr=qb_search_exp&ie=utf8)的变化

[框架结构](http://zhidao.baidu.com/search?word=%E6%A1%86%E6%9E%B6%E7%BB%93%E6%9E%84&fr=qb_search_exp&ie=utf8)的架构图也进一步演变了，不再是原来那个简单的方块图：

  Spring3架构图

  跟原来的相比，DAO、ORM、JEE等模块被划归到了一起，成为“数据访问/集成”部分，Web层突出了自己的MVC(Servlet)和Portlet，核心容器增加了表达式语言。另外，对测试的支持也放到了整个架构中来了。所以整个框架重新划分成了五部分。

[](http://g.hiphotos.baidu.com/zhidao/pic/item/ac4bd11373f08202acc45d4f4bfbfbedab641b68.jpg)

### 3.Hibernate

#### 3.1．Eclipse安装Hibernate插件(本节重点是安装插件)

##### 3.1.1网络参考：How to install Hibernate / JBoss Tools in Eclipse IDE

原文地址：http://www.mkyong.com/hibernate/how-to-install-hibernate-tools-in-eclipse-ide/

[Hibernate Tools](http://www.hibernate.org/subprojects/tools.html) is a handy tool for Java’s developers to generate tedious hibernate related stuffs like mapping files and annotation code. The common use case is the “reverse engineering” feature to generate Hibernate model class, hbm mapping file or annotation code from database tables.

**Note**  
Hibernate Tools is bundled as the core component of JBoss Tools. So, after installed JBoss tools, you installed the Hibernate tools as well.

Here’s a guide to show you how to install Hibernate / JBoss Tools in Eclipse IDE.

**1. Know your Eclipse & JBoss Tools version to download**

First, you have to find out the correct version of Hibernate/JBoss tool for your Eclipse IDE. Go here – <http://www.jboss.org/tools/download> for the available combination version to download.

For example,

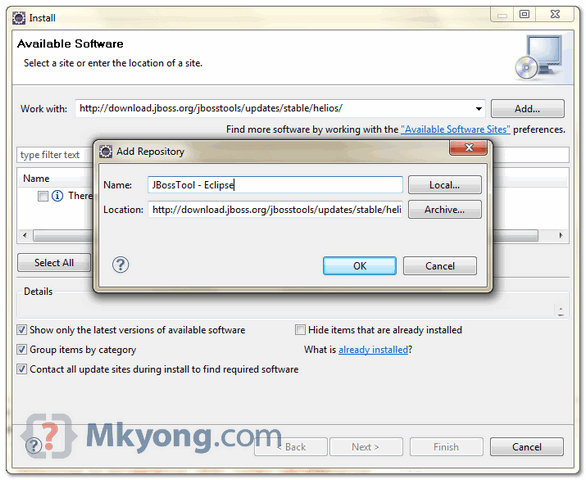
1. If you are using Eclipse 3.6 / Helios , download JBoss Tools 3.2
2. If you are using Eclipse 3.5 / Galileo, download JBoss Tools 3.1

**2. Eclipse update site for JBoss Tools**

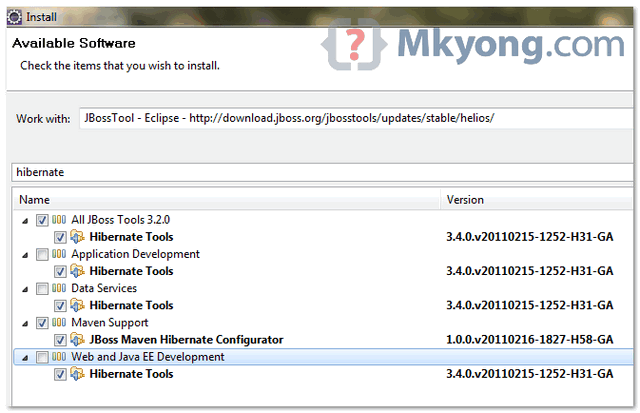
Point to your correct version, right click to copy the Eclipse update site for JBoss tools. For Eclipse 3.6, the URL is ” *http://download.jboss.org/jbosstools/updates/stable/helios/* ”

**3. Install It**

In Eclipse IDE, menu bar, select “**Help**” >> “**Install New Software …**” , put the Eclipse update site URL.



Type “**hibernate**” in the filter box, to list down the necessary components for Hibernate tools. Select all the “**Hibernate Tools**” components and click next to download.



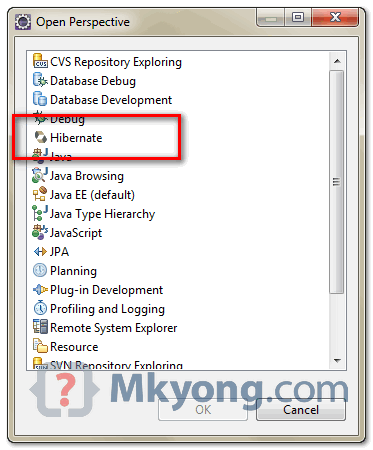
**Warning**  
Do not select all components, it will take much longer time download many unnecessary components. You want Hibernate tools only, not others.

**4. Restart Eclipse**

After the download progress is completed, restart Eclipse to take effect.

**5. Verification**

If Hibernate tools is installed properly, you are able to see the “**Hibernate Perspective**” in “**Windows**” >> “**Open Perspective**” >> “**Others**“.



Done.

**Reference**

1. <http://www.hibernate.org/subprojects/tools.html>
2. <http://www.jboss.org/tools/download>

##### 3.1.2How to generate Hibernate mapping files & annotation with Hibernate Tools

原文地址：<http://www.mkyong.com/hibernate/how-to-generate-code-with-hibernate-tools/>

In this article, we show you how to use **Hibernate / JBoss Tools** to generate Hibernate mapping files (hbm) and annotation code from database automatically.

Tools in this article

1. Eclipse v3.6 (Helios)
2. JBoss / Hibernate Tools v3.2
3. Oracle 11g
4. JDK 1.6

**Note**  
Before proceed, please [Install Hibernate / JBoss Tools in Eclipse IDE](http://www.mkyong.com/hibernate/how-to-install-hibernate-tools-in-eclipse-ide/).

**1. Hibernate Perspective**

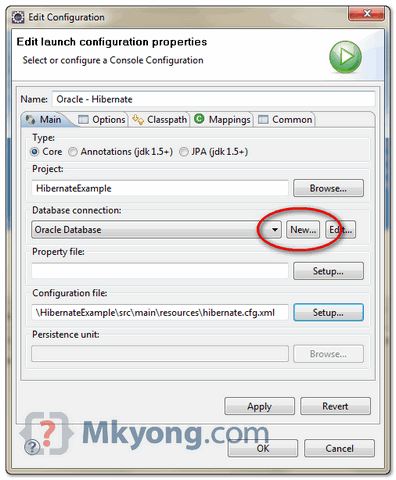
Open your “**Hibernate Perspective**“. In Eclipse IDE, select “**Windows**” >> “**Open Perspective**” >> “**Others…**” , choose “**Hibernate**“.

**2. New Hibernate Configuration**

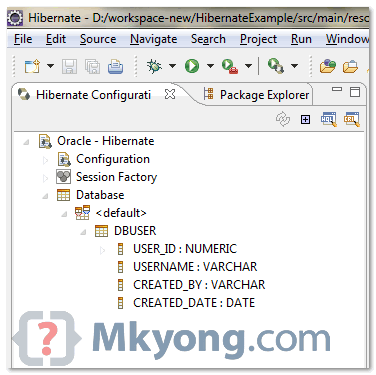
In Hibernate Perspective, right click and select “**Add Configuration…**”

In “Edit Configuration” dialog box,

1. In “**Project**” box, click on the “Browse..” button to select your project.
2. In “**Database Connection**” box, click “New..” button to create your database settings.
3. In “**Configuration File**” box, click “Setup” button to create a new or use existing “Hibernate configuration file”, hibernate.cfg.xml.



See your list of your tables in “**Hibernate Perspective**“.



Sample of “hibernate.cfg.xml“, connect to Oracle 11g database.

**<?xml** version="1.0" encoding="utf-8"**?>**

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

**<hibernate-configuration>**

**<session-factory>**

**<property** name="hibernate.connection.driver\_class"**>**oracle.jdbc.driver.OracleDriver**</property>**

**<property** name="hibernate.connection.url"**>**jdbc:oracle:thin:@127.0.0.1:1521:MKYONG**</property>**

**<property** name="hibernate.connection.username"**>**mkyong**</property>**

**<property** name="hibernate.connection.password"**>**password**</property>**

**<property** name="hibernate.dialect"**>**org.hibernate.dialect.Oracle10gDialect**</property>**

**<property** name="hibernate.default\_schema"**>**MKYONG**</property>**

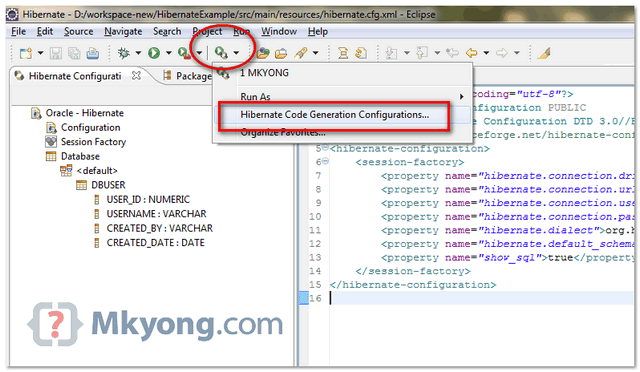
**</session-factory>**

**</hibernate-configuration>**

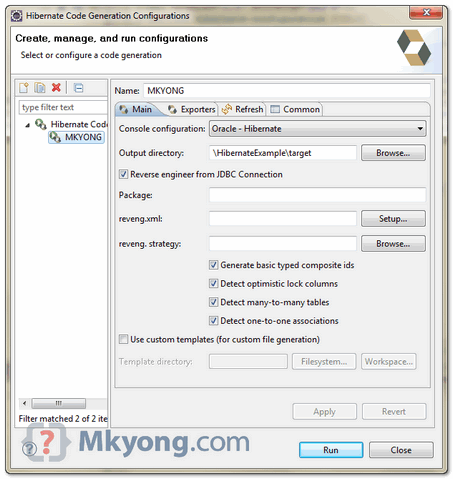
**3. Hibernate Code Generation**

Now, you are ready to generate the Hibernate mapping files and annotation codes.

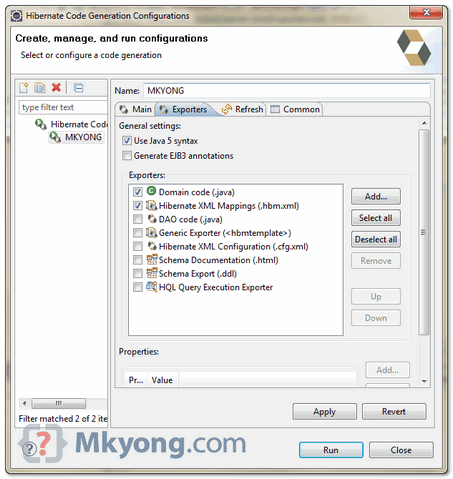
- In “Hibernate Perspective”, click “**Hibernate code generation**” icon (see below figure) and select “Hibernate Code Generation Configuration”



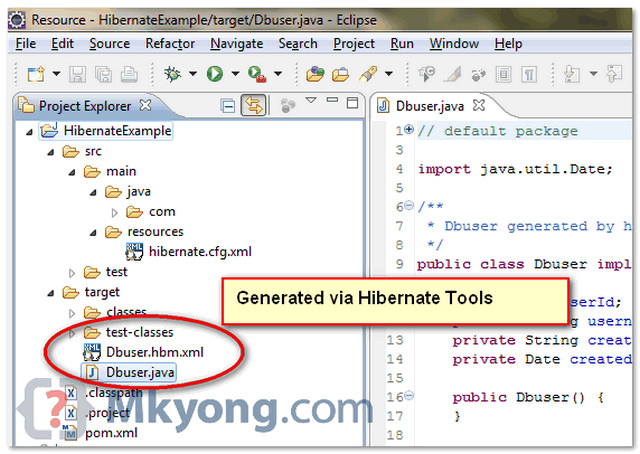
- Create a new configuration, select your “**console configuration**” (configured in step 2), puts your “**Output directory**” and checked option “**Reverse engineer from JDBC Connection**“.



- In “**Exporter**” tab, select what you want to generate, Model , mapping file (hbm) , DAO, annotation code and etc.



See result



**Note**  
The generated Hibernate mapping file and annotations code are very clean, standard and easy to modify. Try explore more features.

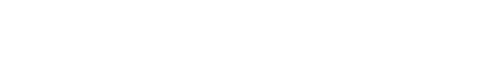
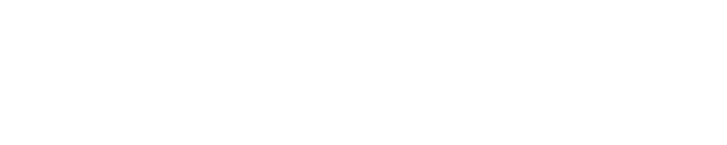
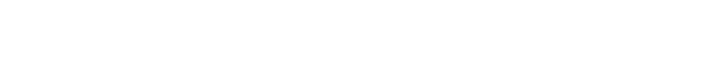
##### 3.1.3直接利用包开发Hibernate（网络版未经本地实践证明是否可行）

###### 一、 Hibernate 的体系结构和工作流程

Hibernate 是关系对象映射框架，对 JDBC 进行了轻量级的对象封装，使 Java 程序员能 够使用面向对象的编程思想来操作数据库。

Hibernate 体系结构：

Application



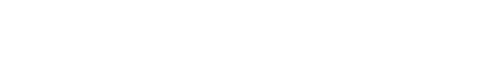
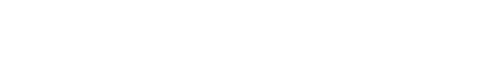
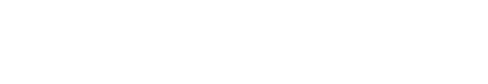
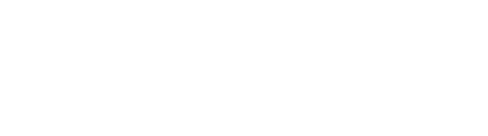
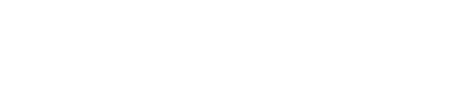
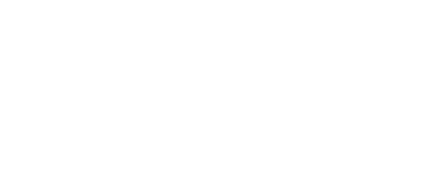
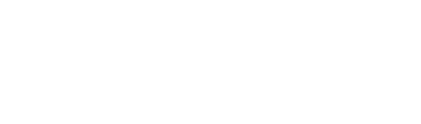
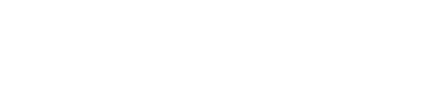
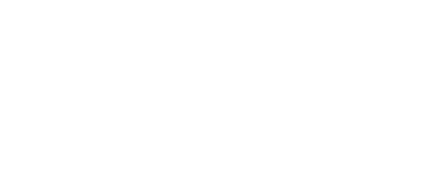
Persistent Objects

Hibernate

Hibernate 配置 ORM 映射

Database

Hibernate 工作流程：



开始

启动 Hibernate

由 SessionFactory 实例创建连接。

Session s = sf.openSession()

构建 Configuration 实例，并初 始 化 该 变 量 中 的 所 有 变 量 。

Configuration cfg = Configuration.configure();

由 Session 实例创建事务操作接口 Transaction 的一个实例。Transaction tx = s.beginTrasaction()

加载 hibernate.cfg.xml 文件至该 实例（内存）

通过 Session 接口提供的各种方法操 作数据库访问

通过 hibernate.cfg.xml 文件中 的 mapping 节点配置，加载 hbm.xml 文件至实例

利用 configuration 实例构建一 个 SessionFactory 实 例 。 SessionFactory sf = cfg.buildSessionFactory

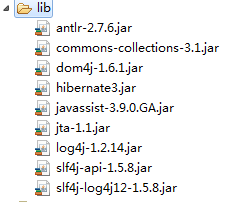
提交数据库操作结果 tx.commit()

关闭 Session 连接 s.close()

结束

###### 二、 Hibernate 开发详解

1. 添加 Hibernate Jar 包支持：
2. 添加 Hibernate 配置文件



Hibernate 配置文件用于配置数据库连接和 Hibernate 运行时所需要的各种属性， 该文件以 xml 文档或 Java 属性文档(properties)形式存在于应用程序的 classpath 中，默认文件名使用 Hibernate.cfg.xml，hibernate 初始化时会自动在 classpath 中寻找这个文件，对读取配置信息。

Hibernate.cfg.xml 配置文件：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN" ["http://hibernate.sourceforge.net/hibernate](http://hibernate.sourceforge.net/hibernate-configuration-3)-[configuration-3](http://hibernate.sourceforge.net/hibernate-configuration-3)

.0.dtd">

<hibernate-configuration>

<session-factory name=*"java:hibernate/SessionFactory"*>

<!-- 定义数据库连接驱动 -->

<property name=*"hibernate.connection.driver\_class"*>

</property>

<!-- 定义数据库服务器地址 -->

<property name=*"hibernate.connection.url"*>

</property>

<!-- 数据库用户名 -->

<property name=*"hibernate.connection.username"*>

</property>

<!-- 数据库用户对应的密码 -->

<property name=*"hibernate.connection.password"*>

</property>

<!-- 数据库对应的方言 -->

<property name=*"hibernate.dialect"*>

</property>

<!-- 在操作数据库时是否打印SQL语句 -->

<property name=*"hibernate.show\_sql"*>

</property>

<!-- 打开 hbm2ddl.auto 选项将自动生成数据库模式（schema）－ 直 接加入数据库中 -->

<property name=*"hbm2ddl.auto"*>update</property>

<!-- 配置ORM映射文件 -->

<mapping resource=*"\*.hbm.xml"*"></mapping>

</session-factory>

</hibernate-configuration>

3. 创建持久类 注意：

1) 持久化类和类中的方法不能为 final 修饰，在 Hibernate 中使用到代理。 2) 创建的持久类建议实现 Serializable 序列化接口。

3) 为持久类创建一个标识属性 id，该属性映射数据库表的主键字段。

4) 为类的各属性提供访问器

5) 为类实现 hashcode 方法和 equals 方法 4. 定义 ORM 映射文件

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

["http://hibernate.sourceforge.net/hibernate](http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd)-[mapping-3.0.dtd"](http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd)>

<!-- 定义ORM映射文件，对应包名为package属性值 -->

<hibernate-mapping package=*""*>

<!-- 定义映射文件对应的类，以及数据库表 -->

<class name=*""* table=*""*>

<!-- 指定类的标识列，该标识列对应数据库表的主键字段，type为字段的 类型 -->

<id name=*""* column=*""* type=*""*>

<!-- 定义标识列的生成策略 native:自动生成 assigned:自定义

uuid:用一个 128-bit 的 UUID 算法生成字符串类型的标识符， 这在一个网络中是唯一的（使用了IP 地址）。UUID 被编码为一个 32 位 16 进制数 字的字符串。

-->

<generator class=*""*/>

</id>

<!-- 定义PO类的其他属性信息 -->

<property name=*""* column=*""* type=*""*>

</property>

<!-- 定义持久化类的多对一关联，该类的该属性引用其他表的主键 class:引用类 unique:唯一约束，若为true则关联为一对一关联 cascade:级联方式

-->

<many-to-one name=*""* column=*""* class=*""* unique=*"" cascade=""*>

</many-to-one>

<!--定义持久化类的一对多关联 inverse:若为true则指定该类为主控方 cascade:级联方式

-->

<set name=*""* inverse=*"" cascade=""*>

<!-- 指定外键 -->

<key column=*""*/>

<!-- 定义引用类 -->

<one-to-many class=*""* />

</set>

</class>

</hibernate-mapping>

**Hibernate 内置的映射类型**

|  |  |  |
| --- | --- | --- |
| Hibenate 映射类型 | JAVA 类型 | 标准 SQL 类型 |
| integer/int | java.lang.Integer/int | INTEGER |
| long | java.lang.Long/long | BIGINT |
| short | java.lang.Short/short | SMALLINT |
| byte | java.lang.Byte/byte | TINYINT |
| float | java.lang.Float/float | FLOAT |
| double | java.lang.Double/double | DOUBLE |
| big\_decimal | java.math.BigDecimal | NUMBERIC |
| character | java.lang.Character/java.lang.String  /char | CHAR(1) |
| string | java.lang.String | VARCHAR |
| boolean/yes\_no/tru e\_false | java.lang.Boolean/boolean | BIT |
| date | java.util.Date/java.sql.Date | TIMESTAMP |
| timestamp | java.util.Date/java.util.Timestamp | TIMESTAMP |
| calendar | java.util.Calendar | TIMESTAMP |

|  |  |  |
| --- | --- | --- |
| calendar\_date | java.util.Calendar | DATE |
| binary | Byte[] | BLOB |
| text | java.lang.String | TEXT |
| serializable | 实现 java.io.Serializable 接口的类 | BLOB |
| clob | java.sql.Clob | CLOB |
| blob | java.sql.Blob | BLOB |
| class | java.lang.Class | VARCHAR |
| locale | java.util.Locale | VARCHAR |
| timezone | java.util.TimeZone | VARCHAR |
| currency | java.util.Currency | VARCHAR |

**Hibernate 内置标识符生成器**

|  |  |
| --- | --- |
| 标识生成器 | 描述 |
| increment |  |
| identity | 采用数据库提供的主键生成策略，如 mysql，ms sql 数据库 |
| sequence | Hibernate 提供底层数据库的序列来生成标识符，前提是底层数据库 支持序列，如 oracle |
| Hilo | 通过 hi/lo 算法实现主键生成策略，需要额外的数据表保存主键生 成历史状态 |
| seqhilo | 与 hilo 类似，只是主键历史状态保存在 sequence 中，只支持 sequence 的数据库 |
| native | 由 Hibernate 根据数据库适配器的定义，自动选择 identity、 sequence 或 hilo 中的一种作为主键生成方式 |
| uuid.hex | Hibernate 采用 128 位的 UUID 算法生成标识符 |
| uuid.string | 与 uuid.hex 类似，只是生成的主键未进行编码，长度为 16 位，对 于某些数据库不适用 |
| assigned | 主键应用逻辑产生，数据库由 Hibernate 保存，主键已经设置完毕， 无需 Hibernate 敢于 |
| foreign | 使用外部表的字段作为主键 |
| select | Hibernat3 中新引入的主键获取机制，针对老系统的改造工程 |

1. 编写 SessionFactory 生成类

**package** com.silence.hibernate;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.cfg.Configuration;

//用于创建SessionFactory对象

**public class** HibernateUtil {

//定义sessionFactory对象

**private static final** SessionFactory *sessionFactory* =

*buildSessionFactory*();

**private** HibernateUtil() {}

//根据配置初始化SessionFactory

**private static** SessionFactory buildSessionFactory() {

**return new**

Configuration().configure().buildSessionFactory();;

}

//获取SessionFactory对象

**public static** SessionFactory getSessionfactory() {

**return** *sessionFactory*;

}

}

1. 进行 Hibernate 数据检索
   1. **Criteria Query 检索方式：**

Criteria Query 通过面向对象化的设计，将数据查询条件封装成一个对象。使 用 Session.createCriteia 方法实例化 Criteria 对象。

Criteria 本身只是一个查询容器，具体的查询条件需要通过 add 方法添加到 Criteria 实例中，add 方法的参数使用 Expression 的静态方法，Expression 类具体描述了查询条件。

针对 SQL 语法 Expression 提供的查询限定机制

|  |  |
| --- | --- |
| 方法 | 描述 |
| Expression.eq | 对应 sql 为：field=value 表达式 |
| Expression.allEq | 参数为一个 Map 对象，其中包含了多个属性-值对应 的关系，相当于多个 Expression.eq 表达式 |
| Expression.gt | 对应 sql 为：field>value 表达式 |
| Expression.ge | 对应 sql 为：field>=value 表达式 |
| Expression.lt | 对应 sql 为：field<value 表达式 |
| Expression.le | 对应 sql 为：field<=value 表达式 |
| Expression.between | 对应 sql 为：between 表达式 |
| Expression.like | 对应 sql 为：field like value 表达式 |
| Expression.in | 对应 sql 为：field in …表达式 |
| Expression.eqProper ty | 用 于 比 较 两 个 属 性 之 间 的 值 ， 对应 sql 为： field\_1=field\_2 表达式 |
| Expression.gtProper ty | 用 于 比 较 两 个 属 性 之 间 的 值 ， 对 应 sql 为： field\_1>field\_2 表达式 |
| Expression.geProper ty | 用 于 比 较 两 个 属 性 之 间 的 值 ， 对 应 sql 为： field\_1>=field\_2 表达式 |
| Expression.ltProper | 用 于 比 较 两 个 属 性 之 间 的 值 ， 对 应 sql 为： |

|  |  |
| --- | --- |
| ty | field\_1<field\_2 表达式 |
| Expression.leProper ty | 用 于 比 较 两 个 属 性 之 间 的 值 ， 对 应 sql 为： field\_1<=field\_2 表达式 |
| Expression.and | and 关 系 组 合 ， Expression.add(Expression.like(),Expression.e q()) |
| Expression.or | or 关系组合 |
| Expression.sql | 用于补充 SQL，可以直接执行 SQL 语句限定查询条件 |

* 1. **Hibernate Query Language(HQL)检索方式**

HQL 使用 Query 对象进行操作。Query 对象使用 Session.createQuery 方法实 例化。

* 1. **DetachedCriteria 检索方式**

该检索方式为 Hibernate3 提供的一种新的 Criteria 实现，可以脱离 Session 实例独立存在，这样可以将某些通用的 Criteria 查询条件进行抽离，每次使 用时再与当前 Session 实例绑定以获得更好的代码重用效果。

创建 DetachedCriteria 对象使用 DetachedCriteria.forClass 方法，使用 add 方 法 添 加 查 询 条 件 ， 使 用 getExecutableCriteria(Session) 方 法 实 例 化 Criteria 对象。

###### 三、 实体对象生命周期

1. **Transient(自由状态)** 实体对象在内存中自由存在，它与数据库中的记录无关，对象刚刚使用 new 语 句创建，还未被持久化，不存在于 Session 缓存当中。

**特征：**

不处于 Hibernate 容器当中，不被任何一个 session 的实例所关联 在数据库中没有对应的记录

**使对象处于自由态的方法：**

当通过 new 语句创建一个 java 对象时，他处于自由状态。此时不和数据 库中的任何记录对应。

Session 的 delete 方法能使一个在持久或游离状态的对象变换为自由状 态。对游离状态的对象，使用 delete 方法从数据库中删除与之对应的记录。 对于持久状态的对象，delete 方法从数据库中删除与之对应的记录，并把它从 session 缓存中删除。

1. **Persistent(持久状态)**

实体对象处于由 Hibernate 框架管理的状态，这种状态下实体对象已经被初始 化，加入到 Session 缓存中，处于 Persistent 状态的对象，其变更将由 Hibernate 固化到数据库中。

**特征：**

位于 Hibernate 的管理容器中，持久对象中被一个 session 实例所关联 持久对象与数据库中的相关记录对应

Session 在清理缓存时，会根据持久对象的属性变换，来同步更新数据库， Session 许多方法能够触发 java 对象转变为持久对象。

Session 的 save 方法把自由对象转变为持久对象。

Session 的 load 方法和 get 方法返回的对象总是处于持久状态。 Session 的 find 方法返回的 list 集合中存放的对象都是持久对象。 Session 的 update、saveOrUpdate 和 lock 方法是游离对象转变为持久状

态。

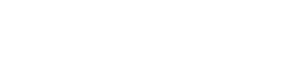
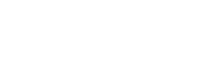
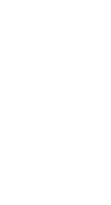
1. **Detached(游离状态)**

处于持久状态的对象，其对应的 session 实例关闭后，此对象就处于游离状态。 Session 实例可以看做是 Persistent 对象的宿主，一旦宿主失效，则从属的 Persistent 对象随即进入游离状态。

**特征：**

不再位于 Hibernate 的管理容器中，游离对象已经不再被 session 关联。 游离对象由持久对象转变过来，因此在数据库中可能还存在与之对应的记

录。 对象状态转换图



new 语句

自由状态

垃圾回收

Save saveOrUpdate

Delete

Get Load Find Iterate

等

Evict、Close、Clear

持久状态

游离状态

Update、saveOrUpdate lock

垃圾回收

###### 四、 Hibernate 数据缓存

Hibernate 缓存包括 Session 和 SessionFactory 的缓冲。

1. **Session 缓存：**

Sesion 缓存时内置的，也被称为 Hibernat 一级缓存，属于应用事务级缓存。 Session 的实现类 SessionImpl 中维护了一个 Map 集合，此数据集合中保持了所有 与当前 Session 相关联的数据对象。

**作用：**

减少访问数据库的频率，应用程序先从内存中读取持久对象的速度显然比在数 据库中查询数据速度快。

保证缓存中的对象与数据库中的相关记录保持同步，当缓存中持久对象的属性 发生变化时，session 并不会立即执行相关 sql 语句，以便减少数据库的访问次数。

Session 缓存由 Hibernate 自动维护，但是可以使用以下方法使用程序处理： Session.evict 方法：将某个特定对象从 Session 中删除。 Session.clear 方法：清空 Session 缓存。

1. **SessionFactory 缓存：**

SessionFactory 缓存从属于本 SessionFactory 的所有 Session 共享，也成为二级 缓存，Session 在进行数据查询操作时，会首先在一级缓存中进行查询，如果一级 缓存未找到，则将在二级缓存中进行查询。

###### 五、 事务管理

在事务管理方面，Hibernate 将其委托给 JDBC 或 JTA，以实现事务管理和调度。 1. Hiberante 配置文件设置隔离级别：

使用 JDBC 连接数据库时为默认隔离级别，分为读操作已提交(Read Committed)和 可重读(Repeatable Read)。

在 Hibernate.cfg.xml 中可以修改隔离基本。

<property name=*"hibernate.connection.isolation"*>4</property>

级别含义：

1 表示读操作未提交

2 表示读操作已提交

4 表示可重读

8 表示可串行化

1. 在 Hibernate 中使用 JDBC 事务

在 hibernate.cfg.xml 配置文件中为 hibernate 添加 JDBC 事务管理。

<!-- Hibernate将事务管理委托给JDBC -->

<property name=*"hibernate.transaction.factory\_class"*> org.hibernate.transaction.JDBCTransactionFactory

</property>

Hibernate 的 Session 在取得数据库连接后立刻取消 JDBC 设置的自动提交模式， 即 Hibernate 在执行 Session 的 beginTransaction 方法后，自动调用 JDBC 层的 setAutoCommit(false)。为了实现 JDBC 事务管理，在程序开始时需要获取事务， 并在事务结束后提交事务。

Session session = **null**; Transaction tx = **null**; **try** {

session = HibernateUtil.*getSessionfactory*().openSession(); tx = session.beginTransaction();

tx.commit();

} **catch** (HibernateException e) { e.printStackTrace(); tx.rollback();

} **finally** {

session.close();

}

1. 在 Hibernate 中使用 JTA 事务

JTA(Java Transaction API) 是事务服务的 JavaEE 解决方案，JTA 提供了跨 Session 的事务管理能力。

JDBC 事务由 Connection 管理，事务管理实际上是在 JDBC Connection 中实现的。 事务周期限于 Connection 的生命周期中。同样，对基于 JDBC Trancation 的 Hiberante 事务管理机制而言，事务管理在 Session 所委托的 JDBC Connection 中 实现，事务周期限于 Session 的生命周期内。

JTA 事务管理由 JTA 容器实现，JTA 容器对当前加入事务的众多 Connction 进行调 度，实现其事务性要求。JTA 的事务周期可横跨 JDBC Connection 生命周期。同样， 对于基于 JTA 的事务管理而言，JTA 事务可以横跨多个 Session。

在 hibernate.cfg.xml 配置文件中为 hibernate 添加 JTA 事务管理

<!-- Hibernate将事务管理委托给JTA-->

<property name=*"hibernate.transaction.factory\_class"*> org.hibernate.transaction.JTATransactionFactory

</property>

Session session\_1 = **null**; Session session\_2 = **null**; UserTransaction tx = **null**; **try** {

session\_1 = HibernateUtil.*getSessionfactory*().openSession();

session\_2 = HibernateUtil.*getSessionfactory*().openSession();

tx = (UserTransaction) **new**

InitialContext().lookup("myTransaction"); tx.begin();

session\_1.flush(); session\_2.flush(); tx.commit();

} **catch** (HibernateException e) { e.printStackTrace();

**try** { tx.rollback();

} **catch** (Exception e1) { e1.printStackTrace();

}

} **catch** (Exception e) { e.printStackTrace();

} **finally** {

session\_1.close(); session\_2.close();

}

###### 六、 持久层操作：

Session 是 Hibernate 向应用程序提供操作数据库的接口。 1. 数据库加载操作：

* 1. **Session 的 get 和 load 方法**

Session 的 get 和 load 方法都是可以根据指导的实体类和 id 从数据库中读取 记录，并且返回与之对应的实体对象。

**区别：**

如果为找到符合条件的记录，load 方法抛出 OjbectNotFoundException 异常，而 get 方法返回 null

Load 方法可以返回实体的代理实体（延迟加载实体对象时，Hibernate3 默认为延迟加载），而该 get 方法无法直接返回实体对象本身。

Load 方法可以充分利用内部缓存和二级缓存，而 get 方法仅仅在内部缓 存中进行查找，若未发现，则越过二级缓存进行 sql 查询。

* 1. **Session 的 find 和 iterate 方法（Hibernate 2） 区别：**

Find 方法拥有实现从数据库中获取所有符合条件的记录并构造成相应的 实体对象，实体对象构造完毕后，将之纳入缓存，而 iterate 方法是执行 N+1 查询

Find 方法对缓存只写不读，而 iterate 方法充分利用缓存。在系统只读 或读取比较频繁的情况下，通过 iterate 可以获得较好的性能。

在进行海量信息查询时，最好结合 iterate 和 evict 方法逐条处理记录。 **在 Hibernate3 中使用 Session.createQuery().list()替代 find 方法，**

**使用 Session.createQuery().iterate()方法替代 iterate 方法 3) 查询缓存**

查询缓存用于特定情况下产生作用： **完全相同的 Select SQL 重复执行**

**在两次查询之间，此 Select SQL 所对应的数据库表没有发生过变化。** Hibernate 默认关闭这一特征，需改变 Hibernate.cfg.xml 中配置以开启缓存 查询：

<!-- 开启缓存查询 -->

<property name=*"hibernate.cache.use\_query\_cache"*>true</property>

使用方法：在程序代码区调用 Query 查询之前通过 Query.setCacheable(true) 语句将 cacheable 属性设置为 true。

**4) 延迟加载**

所谓延迟加载是只在需要数据的时候，才真正的执行数据加载操作。

**（1) 实体对象的延迟加载** 为使用实体的延迟加载，必需在实体的映射文件中将 class 元素的 lazy 属性值设置为 true，以打开实体对象的延迟加载功能。

**（2) 集合类型的延迟加载**

修改映射文件的 set 元素的 lazy 属性值为 true

**（3) 属性的延迟加载**

修改映射文件的 property 元素的 lazy 属性值为 true **5) 数据的保持、更新和删除**

Session.save 方法 该方法用于实现将实体对象的信息保存到数据库的操作。 Session.update 方法：

该方法用于实现将实体对象信息更新到数据库的操作，update 方法具体功能 如下：

根据待更改实体对象的 Key 在当前 Session 内部缓存中进行查找，如果发 现则认为当前实体以处于持久化状态，执行返回。对于已经处于持久状态的实 体对象在调用 update 方法时，不会产生任何作用。

Update 方法本省并没有立刻发送 update 语句完成数据库更新操作，update 语句将在 session.flush 方法中执行，在使用 Transaction.commit 提交数据 库事务之前调用 session.flush 方法。

#### 3.2Eclipse开发Hibernate(成功运行)

##### 3.2.1新建一个Java Project添加lib目录

新建文件夹后添加一个lib目录，将开发Hibernate应用所需的jar包放在里面，然后添加到classPath。

##### 3.2.2添Hibernate配置文件Hibernate.cfg.xml(可以放在任意文件夹)

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/JavaDB</property>

<property name=*"hibernate.connection.username"*>root</property>

<property name=*"hibernate.connection.password"*>8625809</property>

<property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"hibernate.default\_schema"*>Javadb</property>

<property name=*"hibernate.show\_sql"*>true</property>

<mapping resource=*"\com\dbclasses\Student.hbm.xml（注意反斜杠！同时记得把括号和里面的内容删了）"*></mapping>

</session-factory>

</hibernate-configuration>

##### 3.2.3编写 SessionFactory 生成类

package com.hibernate;//包名自定到时候import进去就可以了

import java.sql.SQLException;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

/\*\*

\* Configures and provides access to Hibernate sessions, tied to the

\* current thread of execution. Follows the Thread Local Session

\* pattern, see {@link http://hibernate.org/42.html}.

\*/

public class HibernateSessionFactory {

/\*\*

\* Location of hibernate.cfg.xml file.

\* NOTICE: Location should be on the classpath as Hibernate uses

\* #resourceAsStream style lookup for its configuration file. That

\* is place the config file in a Java package - the default location

\* is the default Java package.<br><br>

\* Defaults: <br>

\* <code>CONFIG\_FILE\_LOCATION = "/hibernate.conf.xml"</code>

\* You can change location with setConfigFile method

\* session will be rebuilded after change of config file

\*/

//private static String CONFIG\_FILE\_LOCATION = "/hibernate.cfg.xml";

private static final ThreadLocal threadLocal = new ThreadLocal();

private static Configuration configuration = new Configuration();

private static SessionFactory sessionFactory;

private static String configFile = "/com/hibernate/hibernate.cfg.xml";

private HibernateSessionFactory() {

}

/\*\*

\* Returns the ThreadLocal Session instance. Lazy initialize

\* the <code>SessionFactory</code> if needed.

\* @return Session

\* @throws HibernateException

\*/

public static Session getCurrentSession() throws HibernateException {

Session session = (Session) threadLocal.get();

try {

if (session == null || !session.isOpen()|| session.connection().isClosed()) {

if (sessionFactory == null) {

rebuildSessionFactory();

}

session = (sessionFactory != null) ? sessionFactory.openSession()

: null;

threadLocal.set(session);

}

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return session;

}

/\*\*

\* Rebuild hibernate session factory

\*

\*/

public static void rebuildSessionFactory() {

try {

configuration.configure(configFile);

sessionFactory = configuration.buildSessionFactory();

}

catch (Exception e) {

System.err.println("%%%% Error Creating SessionFactory %%%%");

e.printStackTrace();

}

}

/\*\*

\* Close the single hibernate session instance.

\*

\* @throws HibernateException

\*/

public static void closeCurrentSession() throws HibernateException {

Session session = (Session) threadLocal.get();

threadLocal.set(null);

if (session != null) {

session.close();

}

}

/\*\*

\* return session factory

\*

\*/

public static SessionFactory getSessionFactory() {

return sessionFactory;

}

/\*\*

\* return session factory

\* session factory will be rebuilded in the next call

\*/

public static void setConfigFile(String configFile) {

HibernateSessionFactory.configFile = configFile;

sessionFactory = null;

}

/\*\*

\* return hibernate configuration

\*

\*/

public static Configuration getConfiguration() {

return configuration;

}

}

##### 3.2.4新建实体类和该实体类的映射文件

###### 实体类代码

**package** com.dbclasses;//包名自定，到时候import进去就可以了

**public** **class** Student {

**private** String id;

**private** String name;

**private** String no;

**private** String deptno;

**public** **void** setId(String id){

**this**.id = id;

}

**public** String getId(){

**return** id;

}

**public** **void** setName(String id){

**this**.name = id;

}

**public** String getName(){

**return** name;

}

**public** **void** setNo(String id){

**this**.no = id;

}

**public** String getNo(){

**return** no;

}

**public** **void** setDeptno(String id){

**this**.deptno = id;

}

**public** String getDeptno(){

**return** deptno;

}

**public** Student(){

}

}

###### 映射文件代码Student.hbm.xml(位置随意写出路径即可)

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.dbclasses.Student"* table=*"student"*>

<id name=*"no"* column=*"sno"* type=*"string"*>

<generator class=*"assigned"*></generator>

</id>

<property name=*"id"* column=*"sid"*></property>

<property name=*"name"* column=*"sname"*></property>

<property name=*"deptno"* column=*"sdeptno"*></property>

</class>

</hibernate-mapping>

##### 3.2.5测试代码(主要是写一个主函数来运行测试上面的配置)

###### 代码

package com.test;

import com.hibernate.HibernateSessionFactory;

import com.dbclasses.\*;

import org.hibernate.Session;

import org.hibernate.Transaction;

public class HibernateTest {

public static void main(String[] args){

Student s = new Student();

s.setDeptno("g4217");

s.setId("440921199008043819");

s.setName("许巍");

s.setNo("201005090209");

Session session = HibernateSessionFactory.getCurrentSession();

Transaction transaction = null;

try{

transaction = session.beginTransaction();

session.save(s);

transaction.commit();

System.out.println("添加一条记录！");

}

catch(Exception e){

e.printStackTrace();

transaction.rollback();

}

HibernateSessionFactory.closeCurrentSession();

}

}

###### 总结

开发Hibernate框架程序并不需要网上所说的那样要下载Hibernate Tools才能进行开发，主要有了必须的jar包就可以了，而且这种不经过工具，完全纯手工的操作，更能了解Hibernate是如何工作的，虽然只是表面上，但也更好地理解如何使用它。第一步，就是要下载Hibernate包，在网上很难找（反正我是找了半天没找到，然后就用Install new software安装了一下，网址在上面3.1.1网络参考：。。。结果安装好了却不能用，然后就打开插件目录把安装有的都给Hibernate包找到，然后就有了）；第二步，新建一个Java项目，添加一个lib目录，将Hibernate的包都放到里面去，然后去到项目属性添加classpath（即C#中的添加引用）；第三步，创建HibernateSessionFactory类，添加Hibernate配置文件hibernate.cfg.xml；第四步，添加实体类（对应数据库里的某个表）和映射文件，并通过<mapping source=””>添加到hibernate.cfg.xml里面去。做好上面工作之后，就可以用上面的HibernateSessionFactory获取Session对象，然后利用Session对象操作实体类，也即操作数据表进行增删改查等操作。最后就是要学习Session对象。

#### 3.3

## EJB（Enterprise JavaBean，WebService、Entity Bean等）

## 关于SSH框架的整合

[**Eclipse搭建SSH（Struts2+Spring3+Hibernate3）框架项目教程**](http://blog.csdn.net/yeohcooller/article/details/9316923)

分类： [JavaWeb相关](http://blog.csdn.net/yeohcooller/article/category/889140) [Java](http://blog.csdn.net/yeohcooller/article/category/1104268) [IEDs、Eclipse](http://blog.csdn.net/yeohcooller/article/category/919908) [JSP、Servlet](http://blog.csdn.net/yeohcooller/article/category/890982) [Spring](http://blog.csdn.net/yeohcooller/article/category/1502281) [Hibernate](http://blog.csdn.net/yeohcooller/article/category/1502283) [Struts2](http://blog.csdn.net/yeohcooller/article/category/1502285) [SSH](http://blog.csdn.net/yeohcooller/article/category/1501325) 2013-07-13 21:46 2923人阅读 [评论](http://blog.csdn.net/yeohcooller/article/details/9316923#comments)(24) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/yeohcooller/article/details/9316923#report)

目录[(?)[+]](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. [前言](http://blog.csdn.net/yeohcooller/article/details/9316923#t0)
   1. [这篇博文的目的](http://blog.csdn.net/yeohcooller/article/details/9316923#t1)
   2. [读博文前应该注意](http://blog.csdn.net/yeohcooller/article/details/9316923#t2)
2. [创建Struts项目](http://blog.csdn.net/yeohcooller/article/details/9316923#t3)
   1. [创建工程](http://blog.csdn.net/yeohcooller/article/details/9316923#t4)
   2. [编写代码](http://blog.csdn.net/yeohcooller/article/details/9316923#t5)
   3. [运行效果](http://blog.csdn.net/yeohcooller/article/details/9316923#t6)
3. [在Struts基础上整合Hibernate](http://blog.csdn.net/yeohcooller/article/details/9316923#t7)
   1. [整合Hibernate](http://blog.csdn.net/yeohcooller/article/details/9316923#t8)
   2. [编写代码](http://blog.csdn.net/yeohcooller/article/details/9316923#t9)
   3. [运行效果](http://blog.csdn.net/yeohcooller/article/details/9316923#t10)
4. [在Struts  Hibernate项目中整合Spring](http://blog.csdn.net/yeohcooller/article/details/9316923#t11)
   1. [整合Spring](http://blog.csdn.net/yeohcooller/article/details/9316923#t12)
   2. [编写代码](http://blog.csdn.net/yeohcooller/article/details/9316923#t13)
   3. [运行结果](http://blog.csdn.net/yeohcooller/article/details/9316923#t14)
5. [总结](http://blog.csdn.net/yeohcooller/article/details/9316923#t15)
   1. [唠叨](http://blog.csdn.net/yeohcooller/article/details/9316923#t16)
   2. [如何快速开发SSH框架项目](http://blog.csdn.net/yeohcooller/article/details/9316923#t17)

**前言**

确实好久没有写过了，一直以来把写博文当作自己学习的总结，当作做过的笔记，随时都可以拿出来看看。不过最近习惯了用OneNote和印象笔记，所以就很少在论坛写博文。但是偶尔看到几篇被转载了，也小小的虚荣了下。这种虚荣大神们已经习以为常，他们是不能理解的。

好吧，转入正题。

参加工作从事Java Web开发将近两年，也没有完整的搭建出一个SSH框架的项目，说出来确实有点不好意思，并且前一段一个朋友问我搭建过SSH框架没，我都没有好意思说没有。所以才有了搭建一个SSH框架项目的想法，因此有了今天的这篇博文。

**这篇博文的目的**

1. 尝试搭建一个完整的SSH框架项目。
2. 给以后的自己，也给别人一个参考。

**读博文前应该注意：**

1. 本文提纲：本文通过一个用户注册的实例讲解SSH的整合。创建Struts项目，整合Hibernate，整合Spring。最后总结如何熟练创建SSH项目。
2. 仅是创建SSH项目，对于其他的扩展例如Struts的国际化，Hibernate的缓存优化，Spring的AOP等，本博文涉及不到。想学习更多的东西请搜索其他博文。
3. 本项目的环境：Windows 8-64位，Eclipse Indigo Service Release 2，Tomcat 7.0，Struts-2.2.3.1，spring-framework-3.2.3.RELEASE（本来使用Spring3.1，但是整合Spring的时候启动服务器报错。错误信息竟然是Spring框架中的方法参数空指针。。。对比Spring3.2版本后，发现在3.2种增加了判断此参数是否为空，所以最终改为用Spring3.2.3的版本。详细错误信息见本目录下5），hibernate-release-4.0.0.CR6（整合Spring3时，可以启动服务器，但是点击注册后调用UserDao类的getHibernateTemplate().save(Obj)时浏览器页面报错，但是服务器没有反应。提示org.hibernate.SessionFactory中有没有openSession(Interceptor interceptor)   
   方法，但是Spring框架中的代码Session session = (entityInterceptor != null ? sessionFactory.openSession(entityInterceptor) : sessionFactory.openSession())显示用到该方法，虽然调用这段代码的方法传进来的entityInterceptor就为null。。。网上搜索无果，所以最终在整合Spring的时候使用Hibernate-3.6.10.Final。详细错误信息见本目录下6），MySQL 5.5.28下的project数据库。
4. user表的创建脚本。

**[sql]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

* 1. create table **user**(
  2. userId int auto\_increment,
  3. userName varchar(16) not null,
  4. password varchar(16) not null,
  5. gender int not null,
  6. primary key(userId)
  7. );

create table user(

userId int auto\_increment,

userName varchar(16) not null,

password varchar(16) not null,

gender int not null,

primary key(userId)

);

1. 使用Spring3.1时出现错误的详细信息。

**[plain]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

* 1. SEVERE: Context initialization failed
  2. org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'sessionFactory' defined in ServletContext resource [/WEB-INF/applicationContext.xml]: Invocation of init method failed; nested exception is java.lang.NullPointerException
  3. at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1445)
  4. at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:522)
  5. at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:459)
  6. at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:294)
  7. at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:225)
  8. at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:291)
  9. at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:193)
  10. at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:567)
  11. at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:913)
  12. at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:464)
  13. at org.springframework.web.context.ContextLoader.configureAndRefreshWebApplicationContext(ContextLoader.java:381)
  14. at org.springframework.web.context.ContextLoader.initWebApplicationContext(ContextLoader.java:283)
  15. at org.springframework.web.context.ContextLoaderListener.contextInitialized(ContextLoaderListener.java:111)
  16. at org.apache.catalina.core.StandardContext.listenerStart(StandardContext.java:4723)
  17. at org.apache.catalina.core.StandardContext$1.call(StandardContext.java:5226)
  18. at org.apache.catalina.core.StandardContext$1.call(StandardContext.java:5221)
  19. at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:334)
  20. at java.util.concurrent.FutureTask.run(FutureTask.java:166)
  21. at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1110)
  22. at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:603)
  23. at java.lang.Thread.run(Thread.java:722)
  24. Caused by: java.lang.NullPointerException
  25. at org.springframework.orm.hibernate4.LocalSessionFactoryBuilder.scanPackages(LocalSessionFactoryBuilder.java:138)
  26. at org.springframework.orm.hibernate4.LocalSessionFactoryBean.afterPropertiesSet(LocalSessionFactoryBean.java:294)
  27. at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCapableBeanFactory.java:1504)
  28. at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1442)
  29. ... 20 more
  31. 七月 13, 2013 6:48:44 下午 org.apache.catalina.core.StandardContext listenerStart
  32. SEVERE: Exception sending context initialized event to listener instance of class org.springframework.web.context.ContextLoaderListener
  33. org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'sessionFactory' defined in ServletContext resource [/WEB-INF/applicationContext.xml]: Invocation of init method failed; nested exception is java.lang.NullPointerException
  34. at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1445)
  35. at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:522)
  36. at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:459)
  37. at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:294)
  38. at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:225)
  39. at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:291)
  40. at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:193)
  41. at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:567)
  42. at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:913)
  43. at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:464)
  44. at org.springframework.web.context.ContextLoader.configureAndRefreshWebApplicationContext(ContextLoader.java:381)
  45. at org.springframework.web.context.ContextLoader.initWebApplicationContext(ContextLoader.java:283)
  46. at org.springframework.web.context.ContextLoaderListener.contextInitialized(ContextLoaderListener.java:111)
  47. at org.apache.catalina.core.StandardContext.listenerStart(StandardContext.java:4723)
  48. at org.apache.catalina.core.StandardContext$1.call(StandardContext.java:5226)
  49. at org.apache.catalina.core.StandardContext$1.call(StandardContext.java:5221)
  50. at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:334)
  51. at java.util.concurrent.FutureTask.run(FutureTask.java:166)
  52. at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1110)
  53. at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:603)
  54. at java.lang.Thread.run(Thread.java:722)
  55. Caused by: java.lang.NullPointerException
  56. at org.springframework.orm.hibernate4.LocalSessionFactoryBuilder.scanPackages(LocalSessionFactoryBuilder.java:138)
  57. at org.springframework.orm.hibernate4.LocalSessionFactoryBean.afterPropertiesSet(LocalSessionFactoryBean.java:294)
  58. at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCapableBeanFactory.java:1504)
  59. at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1442)
  60. ... 20 more

SEVERE: Context initialization failed

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'sessionFactory' defined in ServletContext resource [/WEB-INF/applicationContext.xml]: Invocation of init method failed; nested exception is java.lang.NullPointerException

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1445)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:522)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:459)

at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:294)

at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:225)

at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:291)

at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:193)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:567)

at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:913)

at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:464)

at org.springframework.web.context.ContextLoader.configureAndRefreshWebApplicationContext(ContextLoader.java:381)

at org.springframework.web.context.ContextLoader.initWebApplicationContext(ContextLoader.java:283)

at org.springframework.web.context.ContextLoaderListener.contextInitialized(ContextLoaderListener.java:111)

at org.apache.catalina.core.StandardContext.listenerStart(StandardContext.java:4723)

at org.apache.catalina.core.StandardContext$1.call(StandardContext.java:5226)

at org.apache.catalina.core.StandardContext$1.call(StandardContext.java:5221)

at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:334)

at java.util.concurrent.FutureTask.run(FutureTask.java:166)

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1110)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:603)

at java.lang.Thread.run(Thread.java:722)

Caused by: java.lang.NullPointerException

at org.springframework.orm.hibernate4.LocalSessionFactoryBuilder.scanPackages(LocalSessionFactoryBuilder.java:138)

at org.springframework.orm.hibernate4.LocalSessionFactoryBean.afterPropertiesSet(LocalSessionFactoryBean.java:294)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCapableBeanFactory.java:1504)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1442)

... 20 more

七月 13, 2013 6:48:44 下午 org.apache.catalina.core.StandardContext listenerStart

SEVERE: Exception sending context initialized event to listener instance of class org.springframework.web.context.ContextLoaderListener

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'sessionFactory' defined in ServletContext resource [/WEB-INF/applicationContext.xml]: Invocation of init method failed; nested exception is java.lang.NullPointerException

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1445)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:522)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:459)

at org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:294)

at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:225)

at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:291)

at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:193)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:567)

at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:913)

at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:464)

at org.springframework.web.context.ContextLoader.configureAndRefreshWebApplicationContext(ContextLoader.java:381)

at org.springframework.web.context.ContextLoader.initWebApplicationContext(ContextLoader.java:283)

at org.springframework.web.context.ContextLoaderListener.contextInitialized(ContextLoaderListener.java:111)

at org.apache.catalina.core.StandardContext.listenerStart(StandardContext.java:4723)

at org.apache.catalina.core.StandardContext$1.call(StandardContext.java:5226)

at org.apache.catalina.core.StandardContext$1.call(StandardContext.java:5221)

at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:334)

at java.util.concurrent.FutureTask.run(FutureTask.java:166)

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1110)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:603)

at java.lang.Thread.run(Thread.java:722)

Caused by: java.lang.NullPointerException

at org.springframework.orm.hibernate4.LocalSessionFactoryBuilder.scanPackages(LocalSessionFactoryBuilder.java:138)

at org.springframework.orm.hibernate4.LocalSessionFactoryBean.afterPropertiesSet(LocalSessionFactoryBean.java:294)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCapableBeanFactory.java:1504)

at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1442)

... 20 more

1. 整合Spring时候，使用Hibernate时点击注册报错信息。

**[plain]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

* 1. java.lang.reflect.InvocationTargetException
  2. sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  3. sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
  4. sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  5. java.lang.reflect.Method.invoke(Method.java:601)
  6. com.opensymphony.xwork2.DefaultActionInvocation.invokeAction(DefaultActionInvocation.java:452)
  7. com.opensymphony.xwork2.DefaultActionInvocation.invokeActionOnly(DefaultActionInvocation.java:291)
  8. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:254)
  9. com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor.doIntercept(DefaultWorkflowInterceptor.java:176)
  10. com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)
  11. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  12. com.opensymphony.xwork2.validator.ValidationInterceptor.doIntercept(ValidationInterceptor.java:263)
  13. org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor.doIntercept(AnnotationValidationInterceptor.java:68)
  14. com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)
  15. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  16. com.opensymphony.xwork2.interceptor.ConversionErrorInterceptor.intercept(ConversionErrorInterceptor.java:138)
  17. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  18. com.opensymphony.xwork2.interceptor.ParametersInterceptor.doIntercept(ParametersInterceptor.java:207)
  19. com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)
  20. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  21. com.opensymphony.xwork2.interceptor.ParametersInterceptor.doIntercept(ParametersInterceptor.java:207)
  22. com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)
  23. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  24. com.opensymphony.xwork2.interceptor.StaticParametersInterceptor.intercept(StaticParametersInterceptor.java:190)
  25. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  26. org.apache.struts2.interceptor.MultiselectInterceptor.intercept(MultiselectInterceptor.java:75)
  27. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  28. org.apache.struts2.interceptor.CheckboxInterceptor.intercept(CheckboxInterceptor.java:94)
  29. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  30. org.apache.struts2.interceptor.FileUploadInterceptor.intercept(FileUploadInterceptor.java:243)
  31. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  32. com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor.intercept(ModelDrivenInterceptor.java:100)
  33. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  34. com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor.intercept(ScopedModelDrivenInterceptor.java:141)
  35. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  36. org.apache.struts2.interceptor.debugging.DebuggingInterceptor.intercept(DebuggingInterceptor.java:270)
  37. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  38. com.opensymphony.xwork2.interceptor.ChainingInterceptor.intercept(ChainingInterceptor.java:145)
  39. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  40. com.opensymphony.xwork2.interceptor.PrepareInterceptor.doIntercept(PrepareInterceptor.java:171)
  41. com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)
  42. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  43. com.opensymphony.xwork2.interceptor.I18nInterceptor.intercept(I18nInterceptor.java:176)
  44. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  45. org.apache.struts2.interceptor.ServletConfigInterceptor.intercept(ServletConfigInterceptor.java:164)
  46. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  47. com.opensymphony.xwork2.interceptor.AliasInterceptor.intercept(AliasInterceptor.java:190)
  48. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  49. com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor.intercept(ExceptionMappingInterceptor.java:187)
  50. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  51. org.apache.struts2.impl.StrutsActionProxy.execute(StrutsActionProxy.java:52)
  52. org.apache.struts2.dispatcher.Dispatcher.serviceAction(Dispatcher.java:498)
  53. org.apache.struts2.dispatcher.ng.ExecuteOperations.executeAction(ExecuteOperations.java:77)
  54. org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter.doFilter(StrutsPrepareAndExecuteFilter.java:91)
  56. root cause
  58. java.lang.NoSuchMethodError: org.hibernate.SessionFactory.openSession()Lorg/hibernate/classic/Session;
  59. org.springframework.orm.hibernate3.SessionFactoryUtils.doGetSession(SessionFactoryUtils.java:323)
  60. org.springframework.orm.hibernate3.SessionFactoryUtils.getSession(SessionFactoryUtils.java:235)
  61. org.springframework.orm.hibernate3.HibernateTemplate.getSession(HibernateTemplate.java:457)
  62. org.springframework.orm.hibernate3.HibernateTemplate.doExecute(HibernateTemplate.java:392)
  63. org.springframework.orm.hibernate3.HibernateTemplate.executeWithNativeSession(HibernateTemplate.java:374)
  64. org.springframework.orm.hibernate3.HibernateTemplate.save(HibernateTemplate.java:683)
  65. com.ynwi.ssh.daoImpl.UserDao.saveObject(UserDao.java:12)
  66. com.ynwi.ssh.serviceImpl.UserManagerImpl.regUser(UserManagerImpl.java:23)
  67. com.ynwi.ssh.action.RegisterAction.execute(RegisterAction.java:29)
  68. sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  69. sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
  70. sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  71. java.lang.reflect.Method.invoke(Method.java:601)
  72. com.opensymphony.xwork2.DefaultActionInvocation.invokeAction(DefaultActionInvocation.java:452)
  73. com.opensymphony.xwork2.DefaultActionInvocation.invokeActionOnly(DefaultActionInvocation.java:291)
  74. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:254)
  75. com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor.doIntercept(DefaultWorkflowInterceptor.java:176)
  76. com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)
  77. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  78. com.opensymphony.xwork2.validator.ValidationInterceptor.doIntercept(ValidationInterceptor.java:263)
  79. org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor.doIntercept(AnnotationValidationInterceptor.java:68)
  80. com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)
  81. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  82. com.opensymphony.xwork2.interceptor.ConversionErrorInterceptor.intercept(ConversionErrorInterceptor.java:138)
  83. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  84. com.opensymphony.xwork2.interceptor.ParametersInterceptor.doIntercept(ParametersInterceptor.java:207)
  85. com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)
  86. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  87. com.opensymphony.xwork2.interceptor.ParametersInterceptor.doIntercept(ParametersInterceptor.java:207)
  88. com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)
  89. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  90. com.opensymphony.xwork2.interceptor.StaticParametersInterceptor.intercept(StaticParametersInterceptor.java:190)
  91. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  92. org.apache.struts2.interceptor.MultiselectInterceptor.intercept(MultiselectInterceptor.java:75)
  93. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  94. org.apache.struts2.interceptor.CheckboxInterceptor.intercept(CheckboxInterceptor.java:94)
  95. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  96. org.apache.struts2.interceptor.FileUploadInterceptor.intercept(FileUploadInterceptor.java:243)
  97. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  98. com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor.intercept(ModelDrivenInterceptor.java:100)
  99. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  100. com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor.intercept(ScopedModelDrivenInterceptor.java:141)
  101. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  102. org.apache.struts2.interceptor.debugging.DebuggingInterceptor.intercept(DebuggingInterceptor.java:270)
  103. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  104. com.opensymphony.xwork2.interceptor.ChainingInterceptor.intercept(ChainingInterceptor.java:145)
  105. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  106. com.opensymphony.xwork2.interceptor.PrepareInterceptor.doIntercept(PrepareInterceptor.java:171)
  107. com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)
  108. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  109. com.opensymphony.xwork2.interceptor.I18nInterceptor.intercept(I18nInterceptor.java:176)
  110. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  111. org.apache.struts2.interceptor.ServletConfigInterceptor.intercept(ServletConfigInterceptor.java:164)
  112. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  113. com.opensymphony.xwork2.interceptor.AliasInterceptor.intercept(AliasInterceptor.java:190)
  114. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  115. com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor.intercept(ExceptionMappingInterceptor.java:187)
  116. com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)
  117. org.apache.struts2.impl.StrutsActionProxy.execute(StrutsActionProxy.java:52)
  118. org.apache.struts2.dispatcher.Dispatcher.serviceAction(Dispatcher.java:498)
  119. org.apache.struts2.dispatcher.ng.ExecuteOperations.executeAction(ExecuteOperations.java:77)
  120. org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter.doFilter(StrutsPrepareAndExecuteFilter.java:91)

java.lang.reflect.InvocationTargetException

sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)

sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

java.lang.reflect.Method.invoke(Method.java:601)

com.opensymphony.xwork2.DefaultActionInvocation.invokeAction(DefaultActionInvocation.java:452)

com.opensymphony.xwork2.DefaultActionInvocation.invokeActionOnly(DefaultActionInvocation.java:291)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:254)

com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor.doIntercept(DefaultWorkflowInterceptor.java:176)

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.validator.ValidationInterceptor.doIntercept(ValidationInterceptor.java:263)

org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor.doIntercept(AnnotationValidationInterceptor.java:68)

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ConversionErrorInterceptor.intercept(ConversionErrorInterceptor.java:138)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ParametersInterceptor.doIntercept(ParametersInterceptor.java:207)

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ParametersInterceptor.doIntercept(ParametersInterceptor.java:207)

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.StaticParametersInterceptor.intercept(StaticParametersInterceptor.java:190)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.interceptor.MultiselectInterceptor.intercept(MultiselectInterceptor.java:75)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.interceptor.CheckboxInterceptor.intercept(CheckboxInterceptor.java:94)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.interceptor.FileUploadInterceptor.intercept(FileUploadInterceptor.java:243)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor.intercept(ModelDrivenInterceptor.java:100)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor.intercept(ScopedModelDrivenInterceptor.java:141)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.interceptor.debugging.DebuggingInterceptor.intercept(DebuggingInterceptor.java:270)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ChainingInterceptor.intercept(ChainingInterceptor.java:145)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.PrepareInterceptor.doIntercept(PrepareInterceptor.java:171)

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.I18nInterceptor.intercept(I18nInterceptor.java:176)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.interceptor.ServletConfigInterceptor.intercept(ServletConfigInterceptor.java:164)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.AliasInterceptor.intercept(AliasInterceptor.java:190)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor.intercept(ExceptionMappingInterceptor.java:187)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.impl.StrutsActionProxy.execute(StrutsActionProxy.java:52)

org.apache.struts2.dispatcher.Dispatcher.serviceAction(Dispatcher.java:498)

org.apache.struts2.dispatcher.ng.ExecuteOperations.executeAction(ExecuteOperations.java:77)

org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter.doFilter(StrutsPrepareAndExecuteFilter.java:91)

root cause

java.lang.NoSuchMethodError: org.hibernate.SessionFactory.openSession()Lorg/hibernate/classic/Session;

org.springframework.orm.hibernate3.SessionFactoryUtils.doGetSession(SessionFactoryUtils.java:323)

org.springframework.orm.hibernate3.SessionFactoryUtils.getSession(SessionFactoryUtils.java:235)

org.springframework.orm.hibernate3.HibernateTemplate.getSession(HibernateTemplate.java:457)

org.springframework.orm.hibernate3.HibernateTemplate.doExecute(HibernateTemplate.java:392)

org.springframework.orm.hibernate3.HibernateTemplate.executeWithNativeSession(HibernateTemplate.java:374)

org.springframework.orm.hibernate3.HibernateTemplate.save(HibernateTemplate.java:683)

com.ynwi.ssh.daoImpl.UserDao.saveObject(UserDao.java:12)

com.ynwi.ssh.serviceImpl.UserManagerImpl.regUser(UserManagerImpl.java:23)

com.ynwi.ssh.action.RegisterAction.execute(RegisterAction.java:29)

sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)

sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

java.lang.reflect.Method.invoke(Method.java:601)

com.opensymphony.xwork2.DefaultActionInvocation.invokeAction(DefaultActionInvocation.java:452)

com.opensymphony.xwork2.DefaultActionInvocation.invokeActionOnly(DefaultActionInvocation.java:291)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:254)

com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor.doIntercept(DefaultWorkflowInterceptor.java:176)

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.validator.ValidationInterceptor.doIntercept(ValidationInterceptor.java:263)

org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor.doIntercept(AnnotationValidationInterceptor.java:68)

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ConversionErrorInterceptor.intercept(ConversionErrorInterceptor.java:138)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ParametersInterceptor.doIntercept(ParametersInterceptor.java:207)

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ParametersInterceptor.doIntercept(ParametersInterceptor.java:207)

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.StaticParametersInterceptor.intercept(StaticParametersInterceptor.java:190)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.interceptor.MultiselectInterceptor.intercept(MultiselectInterceptor.java:75)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.interceptor.CheckboxInterceptor.intercept(CheckboxInterceptor.java:94)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.interceptor.FileUploadInterceptor.intercept(FileUploadInterceptor.java:243)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor.intercept(ModelDrivenInterceptor.java:100)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor.intercept(ScopedModelDrivenInterceptor.java:141)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.interceptor.debugging.DebuggingInterceptor.intercept(DebuggingInterceptor.java:270)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ChainingInterceptor.intercept(ChainingInterceptor.java:145)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.PrepareInterceptor.doIntercept(PrepareInterceptor.java:171)

com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.I18nInterceptor.intercept(I18nInterceptor.java:176)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.interceptor.ServletConfigInterceptor.intercept(ServletConfigInterceptor.java:164)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.AliasInterceptor.intercept(AliasInterceptor.java:190)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor.intercept(ExceptionMappingInterceptor.java:187)

com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:248)

org.apache.struts2.impl.StrutsActionProxy.execute(StrutsActionProxy.java:52)

org.apache.struts2.dispatcher.Dispatcher.serviceAction(Dispatcher.java:498)

org.apache.struts2.dispatcher.ng.ExecuteOperations.executeAction(ExecuteOperations.java:77)

org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter.doFilter(StrutsPrepareAndExecuteFilter.java:91)

**创建Struts项目**

**创建工程**

1. 在Eclipse中创建一个Dynamic Web Project，本文项目名称为SSHProject。
2. 把Struts内apps文件夹下的struts2-blank war包用压缩工具解压，然后把WEB-INF/lib内的jar包和WEB-INF下的web.xml文件分别复制到SSHProject项目下的WEB-INF/lib内和WEB-INF下。
3. 在src目录下创建相应package。
4. 在WebContent下创建Register.jsp文件，并分别创建注册成功和注册失败的Success.jsp和Fail.jsp。

操作完成后的结构如图所示：



**编写代码**

* web.xml的内容。

**[html]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** id="WebApp\_9" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"**>**
6. **<display-name>**SSH Project**</display-name>**
8. **<filter>**
9. **<filter-name>**SSH**</filter-name>**
10. **<filter-class>**org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter**</filter-class>**
11. **</filter>**
13. **<filter-mapping>**
14. **<filter-name>**SSH**</filter-name>**
15. **<url-pattern>**/\***</url-pattern>**
16. **</filter-mapping>**
18. **<welcome-file-list>**
19. **<welcome-file>**index.html**</welcome-file>**
20. **</welcome-file-list>**
22. **</web-app>**

<?xml version="1.0" encoding="UTF-8"?>

<web-app id="WebApp\_9" version="2.4" 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 http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>SSH Project</display-name>

<filter>

<filter-name>SSH</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>SSH</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

* Register.jsp的内容。

**[html]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. **<**%@ page language="java" contentType="text/html; charset=UTF-8"
2. pageEncoding="UTF-8"%**>**
3. **<**%@ taglib prefix="s" uri="/struts-tags"%**>**
4. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"**>**
5. **<html>**
6. **<head>**
7. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
8. **<title>**User Register Page**</title>**
9. **</head>**
10. **<body>**
12. **<s:form** action="register"**>**
13. **<s:textfield** name="user.username" label="用户名"**></s:textfield>**
14. **<s:password** name="user.password" label="密码"**></s:password>**
15. **<s:select** list="#{'1':'男','0':'女'}" listKey="key" listValue="value"
16. name="user.gender" label="性别" value="1"**></s:select>**
17. **<s:submit** value="注册"**></s:submit>**
18. **</s:form>**
20. **</body>**
21. **</html>**

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<%@ taglib prefix="s" uri="/struts-tags"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>User Register Page</title>

</head>

<body>

<s:form action="register">

<s:textfield name="user.username" label="用户名"></s:textfield>

<s:password name="user.password" label="密码"></s:password>

<s:select list="#{'1':'男','0':'女'}" listKey="key" listValue="value"

name="user.gender" label="性别" value="1"></s:select>

<s:submit value="注册"></s:submit>

</s:form>

</body>

</html>

* 用户表单类UserForm.java的内容。

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.beans;
3. public class UserForm {
5. private String username;
6. private String password;
7. private int gender;
9. public String getUsername() {
10. return username;
11. }
13. public void setUsername(String username) {
14. this.username = username;
15. }
17. public String getPassword() {
18. return password;
19. }
21. public void setPassword(String password) {
22. this.password = password;
23. }
25. public int getGender() {
26. return gender;
27. }
29. public void setGender(int gender) {
30. this.gender = gender;
31. }
33. }

package com.ynwi.ssh.beans;

public class UserForm {

private String username;

private String password;

private int gender;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public int getGender() {

return gender;

}

public void setGender(int gender) {

this.gender = gender;

}

}

* 注册Action类RegisterAction.java的内容。

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.action;
3. import com.opensymphony.xwork2.ActionSupport;
4. import com.ynwi.ssh.beans.UserForm;
5. import com.ynwi.ssh.service.UserManager;
6. import com.ynwi.ssh.serviceImpl.UserManagerImpl;
8. public class RegisterAction extends ActionSupport {
10. private static final long serialVersionUID = 1L;
12. private UserForm user;
14. private UserManager userManager;
16. public UserForm getUser() {
17. return user;
18. }
20. public void setUser(UserForm user) {
21. this.user = user;
22. }
24. public UserManager getUserManager() {
25. return userManager;
26. }
28. public void setUserManager(UserManager userManager) {
29. this.userManager = userManager;
30. }
32. public String execute() {
33. try {
34. this.setUserManager(new UserManagerImpl());
35. userManager.regUser(user);
36. return SUCCESS;
38. } catch (Exception e) {
39. e.printStackTrace();
40. return ERROR;
41. }
42. }
44. }

package com.ynwi.ssh.action;

import com.opensymphony.xwork2.ActionSupport;

import com.ynwi.ssh.beans.UserForm;

import com.ynwi.ssh.service.UserManager;

import com.ynwi.ssh.serviceImpl.UserManagerImpl;

public class RegisterAction extends ActionSupport {

private static final long serialVersionUID = 1L;

private UserForm user;

private UserManager userManager;

public UserForm getUser() {

return user;

}

public void setUser(UserForm user) {

this.user = user;

}

public UserManager getUserManager() {

return userManager;

}

public void setUserManager(UserManager userManager) {

this.userManager = userManager;

}

public String execute() {

try {

this.setUserManager(new UserManagerImpl());

userManager.regUser(user);

return SUCCESS;

} catch (Exception e) {

e.printStackTrace();

return ERROR;

}

}

}

* 验证文件RegisterAction-validation.xml的内容。该xml文件为RegisterAction的验证文件，命名格式固定，为< ActionName >-validation.xml。

**[html]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. **<?xml** version="1.0" encoding="GBK"**?>**
2. <!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
3. "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd"**>**
5. **<validators>**
6. <!-- 添加对用户名的校验 -->
7. **<field** name="user.username"**>**
8. **<field-validator** type="requiredstring"**>**
9. **<param** name="trim"**>**true**</param>**
10. **<message>**用户名不能为空**</message>**
11. **</field-validator>**
12. **<field-validator** type="regex"**>**
13. **<param** name="expression"**>**<![CDATA[(\w{6,16})]]>**</param>**
14. **<message>**用户名输入不合法，必须为长度在6~16中间的数字或字母**</message>**
15. **</field-validator>**
16. **</field>**
18. <!-- 添加对密码的校验 -->
19. **<field** name="user.password"**>**
20. **<field-validator** type="requiredstring"**>**
21. **<param** name="trim"**>**true**</param>**
22. **<message>**密码不能为空**</message>**
23. **</field-validator>**
24. **<field-validator** type="regex"**>**
25. **<param** name="expression"**>**<![CDATA[(\w{6,16})]]>**</param>**
26. **<message>**密码输入不合法，必须为长度在6~16之间的数字或者字母**</message>**
27. **</field-validator>**
28. **</field>**
29. **</validators>**

<?xml version="1.0" encoding="GBK"?>

<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"

"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

<validators>

<!-- 添加对用户名的校验 -->

<field name="user.username">

<field-validator type="requiredstring">

<param name="trim">true</param>

<message>用户名不能为空</message>

</field-validator>

<field-validator type="regex">

<param name="expression"><![CDATA[(\w{6,16})]]></param>

<message>用户名输入不合法，必须为长度在6~16中间的数字或字母</message>

</field-validator>

</field>

<!-- 添加对密码的校验 -->

<field name="user.password">

<field-validator type="requiredstring">

<param name="trim">true</param>

<message>密码不能为空</message>

</field-validator>

<field-validator type="regex">

<param name="expression"><![CDATA[(\w{6,16})]]></param>

<message>密码输入不合法，必须为长度在6~16之间的数字或者字母</message>

</field-validator>

</field>

</validators>

* 业务逻辑接口UserManaer.java的内容。

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.service;
3. import com.ynwi.ssh.beans.UserForm;
5. public interface UserManager {
7. public void regUser(UserForm user);
9. }

package com.ynwi.ssh.service;

import com.ynwi.ssh.beans.UserForm;

public interface UserManager {

public void regUser(UserForm user);

}

* 业务逻辑实现类UserManagerImpl.java的内容

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.serviceImpl;
3. import com.ynwi.ssh.beans.UserForm;
4. import com.ynwi.ssh.service.UserManager;
6. public class UserManagerImpl implements UserManager {
8. @Override
9. public void regUser(UserForm user) {
11. }
13. }

package com.ynwi.ssh.serviceImpl;

import com.ynwi.ssh.beans.UserForm;

import com.ynwi.ssh.service.UserManager;

public class UserManagerImpl implements UserManager {

@Override

public void regUser(UserForm user) {

}

}

* 配置文件struts.xml的内容。

**[html]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. **<?xml** version="1.0" encoding="UTF-8" **?>**
2. <!DOCTYPE struts PUBLIC
3. "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4. "http://struts.apache.org/dtds/struts-2.0.dtd"**>**
6. **<struts>**
8. **<include** file="struts-default.xml" **/>**
9. **<package** name="ynwi" extends="struts-default"**>**
10. **<action** name="register" class="com.ynwi.ssh.action.RegisterAction"**>**
11. **<result** name="input"**>**/Register.jsp**</result>**
12. **<result** name="success"**>**/Success.jsp**</result>**
13. **<result** name="error"**>**/Fail.jsp**</result>**
14. **</action>**
15. **</package>**
17. **</struts>**

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<include file="struts-default.xml" />

<package name="ynwi" extends="struts-default">

<action name="register" class="com.ynwi.ssh.action.RegisterAction">

<result name="input">/Register.jsp</result>

<result name="success">/Success.jsp</result>

<result name="error">/Fail.jsp</result>

</action>

</package>

</struts>

Success.jsp和Fail.jsp仅仅只是一句提示注册成功和失败的话，这里就不在列出。

**运行效果**

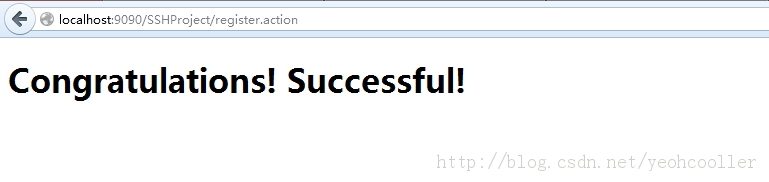
* 输入访问Register.jsp的地址http://localhost:9090/SSHProject/Register.jsp后。



* 输入用户名和密码不合法时。



* 注册成功时。

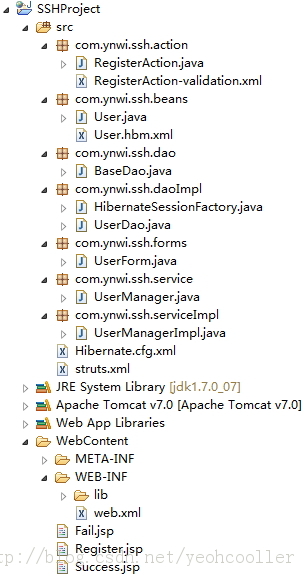


**在Struts基础上整合Hibernate**

**整合Hibernate**

1. 将Hibernate内的lib/required下的jar包和lib/jpa下的jar包复制到SSHProject项目的lib目录下。
2. 在上面Struts项目的基础上添加com.ynwi.ssh.dao和com.ynwi.ssh.daoImpl包。分别创建BaseDao接口和UserDao类。
3. 在数据库中创建对应的表user来写入要注册的用户。创建表的脚本见前言说明部分。
4. 创建user表的Hibernate映射文件。**因为这里要创建user表对应的POJO类，所以我这里把之前com.ynwi.ssh.beans包改成com.ynwi.ssh.forms包，用来存放页面传过来的表单数据，新创建com.ynwi.ssh.beans包，用来创建POJO对象来对应数据库中的表。路径：POJO类同目录。**
5. 建立Hibernate配置文件。路径：src目录下。
6. 编写Session工厂类。**整合Spring后，数据源的创建和SessionFactory的创建都交给Spring去管理，就无需这个工厂类了，所以这里我临时创建在daoImpl包里面。**
7. 修改业务逻辑实现类。

操作完成后的结构如图所示：



**编写代码**

* 接口类BaseDao.java的内容。**注意：接口里的两个get/set方法是为了让实现类覆盖。这样使用BaseDao dao = new UserDao()的时候，dao实体中有这两个方法。**

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.dao;
3. import org.hibernate.HibernateException;
4. import org.hibernate.Session;
6. public interface BaseDao {
8. public void saveObject(Object obj) throws HibernateException;
10. public Session getSession();
12. public void setSession(Session session);
13. }

package com.ynwi.ssh.dao;

import org.hibernate.HibernateException;

import org.hibernate.Session;

public interface BaseDao {

public void saveObject(Object obj) throws HibernateException;

public Session getSession();

public void setSession(Session session);

}

* 实现类UserDao.java的内容。

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.daoImpl;
3. import org.hibernate.HibernateException;
4. import org.hibernate.Session;
6. import com.ynwi.ssh.dao.BaseDao;
8. public class UserDao implements BaseDao {
10. private Session session;
12. @Override
13. public Session getSession() {
14. return session;
15. }
17. @Override
18. public void setSession(Session session) {
19. this.session = session;
20. }
22. @Override
23. public void saveObject(Object obj) throws HibernateException {
24. session.save(obj);
25. }
27. }

package com.ynwi.ssh.daoImpl;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import com.ynwi.ssh.dao.BaseDao;

public class UserDao implements BaseDao {

private Session session;

@Override

public Session getSession() {

return session;

}

@Override

public void setSession(Session session) {

this.session = session;

}

@Override

public void saveObject(Object obj) throws HibernateException {

session.save(obj);

}

}

* 表user的POJO类User.java的内容，因为跟UserForm.java内容差不多，所以不再列出。**这个比UserForm.java多一个userId属性，因为在表user里面有userId的column。在POJO类里需要跟这个column有映射。**
* POJO类User.java的Hibernate映射文件User.hbm.xml的内容。**代码<generator class="native"></generator>说明将根据本地数据库的设置来创建userId，如MySQL中将表user的userId设置为自动增长，则在保存记录是，userId将自动增长。对于需要在程序中指定主键的值，则将generator的class属性设置为assigned。**

**[html]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. **<?xml** version="1.0" encoding='UTF-8'**?>**
2. <!DOCTYPE hibernate-mapping PUBLIC
3. "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4. "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" **>**
6. **<hibernate-mapping** package="com.ynwi.ssh.beans"**>**
7. **<class** name="User" table="user"**>**
8. **<id** name="userId" column="userId"**>**
9. **<generator** class="native"**></generator>**
10. **</id>**
11. **<property** name="username" column="userName" type="java.lang.String"
12. not-null="true" length="16"**></property>**
13. **<property** name="password" column="password" type="java.lang.String"
14. not-null="true" length="16" **/>**
15. **<property** name="gender" column="gender" type="java.lang.Integer" length="1" **/>**
16. **</class>**
17. **</hibernate-mapping>**

<?xml version="1.0" encoding='UTF-8'?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

<hibernate-mapping package="com.ynwi.ssh.beans">

<class name="User" table="user">

<id name="userId" column="userId">

<generator class="native"></generator>

</id>

<property name="username" column="userName" type="java.lang.String"

not-null="true" length="16"></property>

<property name="password" column="password" type="java.lang.String"

not-null="true" length="16" />

<property name="gender" column="gender" type="java.lang.Integer" length="1" />

</class>

</hibernate-mapping>

* Hibernate配置文件Hibernate.cfg.xml的内容。

**[html]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. **<?xml** version='1.0' encoding='utf-8'**?>**
2. <!DOCTYPE hibernate-configuration PUBLIC
3. "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4. "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"**>**
6. **<hibernate-configuration>**
7. **<session-factory>**
8. <!-- Database connection settings -->
9. **<property** name="connection.driver\_class"**>**com.mysql.jdbc.Driver**</property>**
10. **<property** name="connection.url"**>**jdbc:mysql://localhost:3306/project**</property>**
11. **<property** name="connection.username"**>**root**</property>**
12. **<property** name="connection.password"**>**root**</property>**
14. <!-- JDBC connection pool (use the built-in) -->
15. <!-- <property name="connection.pool\_size">1</property> -->
17. <!-- SQL dialect -->
18. **<property** name="dialect"**>**org.hibernate.dialect.MySQLDialect**</property>**
20. <!-- Enable Hibernate's automatic session context management -->
21. <!-- <property name="current\_session\_context\_class">thread</property> -->
23. <!-- Disable the second-level cache  -->
24. <!-- <property name="cache.provider\_class">org.hibernate.cache.internal.NoCacheProvider</property> -->
26. <!-- Echo all executed SQL to stdout -->
27. **<property** name="show\_sql"**>**true**</property>**
29. <!-- Drop and re-create the database schema on startup -->
30. <!-- <property name="hbm2ddl.auto">update</property> -->
31. **<mapping** resource="com/ynwi/ssh/beans/User.hbm.xml"**/>**
32. **</session-factory>**
33. **</hibernate-configuration>**

<?xml version='1.0' encoding='utf-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- Database connection settings -->

<property name="connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="connection.url">jdbc:mysql://localhost:3306/project</property>

<property name="connection.username">root</property>

<property name="connection.password">root</property>

<!-- JDBC connection pool (use the built-in) -->

<!-- <property name="connection.pool\_size">1</property> -->

<!-- SQL dialect -->

<property name="dialect">org.hibernate.dialect.MySQLDialect</property>

<!-- Enable Hibernate's automatic session context management -->

<!-- <property name="current\_session\_context\_class">thread</property> -->

<!-- Disable the second-level cache -->

<!-- <property name="cache.provider\_class">org.hibernate.cache.internal.NoCacheProvider</property> -->

<!-- Echo all executed SQL to stdout -->

<property name="show\_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<!-- <property name="hbm2ddl.auto">update</property> -->

<mapping resource="com/ynwi/ssh/beans/User.hbm.xml"/>

</session-factory>

</hibernate-configuration>

* 工厂类SessionFactory的内容。（这是使用Hibernate4时候的SessionFactory，Hibernate3时候sessionFactory的创建估计会不一样，请自行查阅。）

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.daoImpl;
3. import org.hibernate.HibernateException;
4. import org.hibernate.Session;
5. import org.hibernate.SessionFactory;
6. import org.hibernate.cfg.Configuration;
7. import org.hibernate.service.ServiceRegistry;
8. import org.hibernate.service.ServiceRegistryBuilder;
10. public class HibernateSessionFactory {
12. private static final String CFG\_FILE\_LOCATION = "/Hibernate.cfg.xml";
14. private static final ThreadLocal<Session> threadLocal = new ThreadLocal<Session>();
16. private static final Configuration cfg = new Configuration()
17. .configure(CFG\_FILE\_LOCATION);
19. private static ServiceRegistryBuilder builder = new ServiceRegistryBuilder()
20. .applySettings(cfg.getProperties());
22. private static ServiceRegistry registry;
24. private static SessionFactory sessionFactory;
26. public static Session currentSession() throws HibernateException {
27. Session session = threadLocal.get();
29. if (session == null || session.isOpen() == false) {
31. if (sessionFactory == null) {
32. try {
33. registry = builder.buildServiceRegistry();
34. sessionFactory = cfg.buildSessionFactory(registry);
35. } catch (Exception e) {
36. e.printStackTrace();
37. }
38. }
40. session = sessionFactory.openSession();
41. threadLocal.set(session);
43. }
45. return session;
46. }
48. public static void closeSession() throws HibernateException {
49. Session session = threadLocal.get();
50. threadLocal.set(null);
51. if (session != null) {
52. session.close();
53. }
54. }
56. }

package com.ynwi.ssh.daoImpl;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

import org.hibernate.service.ServiceRegistry;

import org.hibernate.service.ServiceRegistryBuilder;

public class HibernateSessionFactory {

private static final String CFG\_FILE\_LOCATION = "/Hibernate.cfg.xml";

private static final ThreadLocal<Session> threadLocal = new ThreadLocal<Session>();

private static final Configuration cfg = new Configuration()

.configure(CFG\_FILE\_LOCATION);

private static ServiceRegistryBuilder builder = new ServiceRegistryBuilder()

.applySettings(cfg.getProperties());

private static ServiceRegistry registry;

private static SessionFactory sessionFactory;

public static Session currentSession() throws HibernateException {

Session session = threadLocal.get();

if (session == null || session.isOpen() == false) {

if (sessionFactory == null) {

try {

registry = builder.buildServiceRegistry();

sessionFactory = cfg.buildSessionFactory(registry);

} catch (Exception e) {

e.printStackTrace();

}

}

session = sessionFactory.openSession();

threadLocal.set(session);

}

return session;

}

public static void closeSession() throws HibernateException {

Session session = threadLocal.get();

threadLocal.set(null);

if (session != null) {

session.close();

}

}

}

* 修改后的业务逻辑实现类的内容。

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.serviceImpl;
3. import org.hibernate.HibernateException;
4. import org.hibernate.Session;
5. import org.hibernate.Transaction;
7. import com.ynwi.ssh.beans.User;
8. import com.ynwi.ssh.dao.BaseDao;
9. import com.ynwi.ssh.daoImpl.HibernateSessionFactory;
10. import com.ynwi.ssh.daoImpl.UserDao;
11. import com.ynwi.ssh.forms.UserForm;
12. import com.ynwi.ssh.service.UserManager;
14. public class UserManagerImpl implements UserManager {
16. private BaseDao dao;
18. private Session session;
20. public UserManagerImpl() {
21. dao = new UserDao();
22. }
24. @Override
25. public void regUser(UserForm userForm) throws HibernateException {
26. session = HibernateSessionFactory.currentSession();
27. dao.setSession(session);
28. // 获取事务
29. Transaction ts = session.beginTransaction();
30. // 构造User对象
31. User user = new User();
32. user.setUsername(userForm.getUsername());
33. user.setPassword(userForm.getPassword());
34. user.setGender(userForm.getGender());
35. // 保存User对象
36. dao.saveObject(user);
37. // 提交事务
38. ts.commit();
39. // 关闭Session
40. HibernateSessionFactory.closeSession();
41. }
43. }

package com.ynwi.ssh.serviceImpl;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.Transaction;

import com.ynwi.ssh.beans.User;

import com.ynwi.ssh.dao.BaseDao;

import com.ynwi.ssh.daoImpl.HibernateSessionFactory;

import com.ynwi.ssh.daoImpl.UserDao;

import com.ynwi.ssh.forms.UserForm;

import com.ynwi.ssh.service.UserManager;

public class UserManagerImpl implements UserManager {

private BaseDao dao;

private Session session;

public UserManagerImpl() {

dao = new UserDao();

}

@Override

public void regUser(UserForm userForm) throws HibernateException {

session = HibernateSessionFactory.currentSession();

dao.setSession(session);

// 获取事务

Transaction ts = session.beginTransaction();

// 构造User对象

User user = new User();

user.setUsername(userForm.getUsername());

user.setPassword(userForm.getPassword());

user.setGender(userForm.getGender());

// 保存User对象

dao.saveObject(user);

// 提交事务

ts.commit();

// 关闭Session

HibernateSessionFactory.closeSession();

}

}

**运行效果**

* 输入访问Register.jsp的地址http://localhost:9090/SSHProject/Register.jsp后。



* 点击注册前后数据库里面的数据。

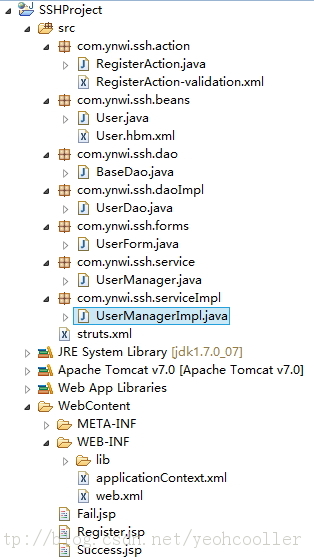


**在Struts + Hibernate项目中整合Spring**

**整合Spring**

1. 将Spring内libs目录下包含所有的jar包（不需要复制结尾为sources和javadoc的jar包）到SSHProject项目的lib目录下。
2. 编写Spring的配置文件applicationContext.xml。路径：src目录下，需要在web.xml配置context-param指定路径，或者把该文件放在WEB-INF下，跟web.xml同目录。这里由于Spring配置数据源的需要，需要把Hibernate内lib/optional/c3p0下的c3p0-0.9.1.jar复制到lib不目下。
3. 修改BaseDao和UserDao。在引入Spring后，需要用Spring进行统一的事务管理，数据源和sessionFactory都交给Spring去生成，因此接口类和实现类BaseDao和UserDao都需要做相应的修改。Spring提供了HibernateDaoSupport类来完成对数据的操作，因此UserDao在实现BaseDao的同时还需要继承HibernateDaoSupport类。并将先前session的操作修改成HibernateTemplate（可通过getHibernateTemplate（）方法来获得）的操作。
4. 修改业务逻辑实现类。**在没有加入Spring之前，业务逻辑实现类的Session的获得，dao的实例化，以及事务的管理都是该类执行管理的。加入Spring后，这些都交给Spring去管理。该类的dao的实例化由Spring注入。**
5. 修改用户注册的RegisterAction类。同样，RegisterAction类中的userManager的实例化也由Spring注入。
6. 删除Hibernate的配置文件Hibernate.cfg.xml和工厂类HibernateSesseionFactory类。他们的工作已经交给Spring去做，已经不再有用。
7. 修改web.xml，加载Spring。**要想启动时加载Spring的配置文件，需要在web.xml中配置对应的监听器（listenser），并制定Spring的配置文件。**
8. 修改Struts的配置文件struts.xml。把原来指定的名为register的action的class由原来的路径变为applicationContext.xml文件中该Action的id。

操作完成后的结构如图所示：



**编写代码**

* Spring的配置文件applicationContext.xml的内容。

**[html]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://www.springframework.org/schema/beans
5. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"**>**
7. <!-- 定义数据源的信息 -->
8. **<bean** id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
9. destroy-method="close"**>**
10. **<property** name="driverClass"**>**
11. **<value>**com.mysql.jdbc.Driver**</value>**
12. **</property>**
13. **<property** name="jdbcUrl"**>**
14. **<value>**jdbc:mysql://localhost/project**</value>**
15. **</property>**
16. **<property** name="user"**>**
17. **<value>**root**</value>**
18. **</property>**
19. **<property** name="password"**>**
20. **<value>**root**</value>**
21. **</property>**
22. **<property** name="maxPoolSize"**>**
23. **<value>**80**</value>**
24. **</property>**
25. **<property** name="minPoolSize"**>**
26. **<value>**1**</value>**
27. **</property>**
28. **<property** name="initialPoolSize"**>**
29. **<value>**1**</value>**
30. **</property>**
31. **<property** name="maxIdleTime"**>**
32. **<value>**20**</value>**
33. **</property>**
34. **</bean>**
36. <!--定义Hibernate的SessionFactory -->
37. <!-- SessionFactory使用的数据源为上面的数据源 -->
38. <!-- 指定了Hibernate的映射文件和配置信息 -->
39. **<bean** id="sessionFactory"
40. class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"**>**
41. **<property** name="dataSource"**>**
42. **<ref** local="dataSource" **/>**
43. **</property>**
44. **<property** name="mappingResources"**>**
45. **<list>**
46. **<value>**com/ynwi/ssh/beans/User.hbm.xml**</value>**
47. **</list>**
48. **</property>**
49. **<property** name="hibernateProperties"**>**
50. **<props>**
51. **<prop** key="hibernate.dialect"**>**org.hibernate.dialect.MySQLDialect**</prop>**
52. **<prop** key="show\_sql"**>**true**</prop>**
53. **<prop** key="hibernate.jdbc.batch\_size"**>**20**</prop>**
54. **</props>**
55. **</property>**
56. **</bean>**
58. **<bean** id="transactionManager"
59. class="org.springframework.orm.hibernate3.HibernateTransactionManager"**>**
60. **<property** name="sessionFactory" ref="sessionFactory" **/>**
61. **</bean>**
63. **<bean** id="baseDao" class="com.ynwi.ssh.daoImpl.UserDao"**>**
64. **<property** name="sessionFactory"**>**
65. **<ref** bean="sessionFactory" **/>**
66. **</property>**
67. **</bean>**
69. <!--用户注册业务逻辑类 -->
70. **<bean** id="userManager" class="com.ynwi.ssh.serviceImpl.UserManagerImpl"**>**
71. **<property** name="dao"**>**
72. **<ref** bean="baseDao" **/>**
73. **</property>**
74. **</bean>**
76. <!-- 用户注册的Action -->
77. **<bean** id="regAction" class="com.ynwi.ssh.action.RegisterAction"**>**
78. **<property** name="userManager"**>**
79. **<ref** bean="userManager" **/>**
80. **</property>**
81. **</bean>**
83. <!-- more bean definitions go here -->
85. **</beans>**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<!-- 定义数据源的信息 -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"

destroy-method="close">

<property name="driverClass">

<value>com.mysql.jdbc.Driver</value>

</property>

<property name="jdbcUrl">

<value>jdbc:mysql://localhost/project</value>

</property>

<property name="user">

<value>root</value>

</property>

<property name="password">

<value>root</value>

</property>

<property name="maxPoolSize">

<value>80</value>

</property>

<property name="minPoolSize">

<value>1</value>

</property>

<property name="initialPoolSize">

<value>1</value>

</property>

<property name="maxIdleTime">

<value>20</value>

</property>

</bean>

<!--定义Hibernate的SessionFactory -->

<!-- SessionFactory使用的数据源为上面的数据源 -->

<!-- 指定了Hibernate的映射文件和配置信息 -->

<bean id="sessionFactory"

class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="dataSource">

<ref local="dataSource" />

</property>

<property name="mappingResources">

<list>

<value>com/ynwi/ssh/beans/User.hbm.xml</value>

</list>

</property>

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

<prop key="show\_sql">true</prop>

<prop key="hibernate.jdbc.batch\_size">20</prop>

</props>

</property>

</bean>

<bean id="transactionManager"

class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<bean id="baseDao" class="com.ynwi.ssh.daoImpl.UserDao">

<property name="sessionFactory">

<ref bean="sessionFactory" />

</property>

</bean>

<!--用户注册业务逻辑类 -->

<bean id="userManager" class="com.ynwi.ssh.serviceImpl.UserManagerImpl">

<property name="dao">

<ref bean="baseDao" />

</property>

</bean>

<!-- 用户注册的Action -->

<bean id="regAction" class="com.ynwi.ssh.action.RegisterAction">

<property name="userManager">

<ref bean="userManager" />

</property>

</bean>

<!-- more bean definitions go here -->

</beans>

* 修改后的接口类Dao和实现类UserDao的内容。

BaseDao.java

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.dao;
3. import org.hibernate.HibernateException;
5. public interface BaseDao {
7. public void saveObject(Object obj) throws HibernateException;
9. }

package com.ynwi.ssh.dao;

import org.hibernate.HibernateException;

public interface BaseDao {

public void saveObject(Object obj) throws HibernateException;

}

UserDao.java

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.daoImpl;
3. import org.hibernate.HibernateException;
4. import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
6. import com.ynwi.ssh.dao.BaseDao;
8. public class UserDao extends HibernateDaoSupport implements BaseDao {
10. @Override
11. public void saveObject(Object obj) throws HibernateException {
12. getHibernateTemplate().save(obj);
13. }
15. }

package com.ynwi.ssh.daoImpl;

import org.hibernate.HibernateException;

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import com.ynwi.ssh.dao.BaseDao;

public class UserDao extends HibernateDaoSupport implements BaseDao {

@Override

public void saveObject(Object obj) throws HibernateException {

getHibernateTemplate().save(obj);

}

}

* 修改后的业务逻辑实现类UserManagerImpl的内容。

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.serviceImpl;
3. import org.hibernate.HibernateException;
4. import org.springframework.beans.BeanUtils;
6. import com.ynwi.ssh.beans.User;
7. import com.ynwi.ssh.dao.BaseDao;
8. import com.ynwi.ssh.forms.UserForm;
9. import com.ynwi.ssh.service.UserManager;
11. public class UserManagerImpl implements UserManager {
13. private BaseDao dao;
15. public void setDao(BaseDao dao) {
16. this.dao = dao;
17. }
19. @Override
20. public void regUser(UserForm userForm) throws HibernateException {
21. User user = new User();
22. BeanUtils.copyProperties(userForm, user);
23. dao.saveObject(user);
24. }
26. }

package com.ynwi.ssh.serviceImpl;

import org.hibernate.HibernateException;

import org.springframework.beans.BeanUtils;

import com.ynwi.ssh.beans.User;

import com.ynwi.ssh.dao.BaseDao;

import com.ynwi.ssh.forms.UserForm;

import com.ynwi.ssh.service.UserManager;

public class UserManagerImpl implements UserManager {

private BaseDao dao;

public void setDao(BaseDao dao) {

this.dao = dao;

}

@Override

public void regUser(UserForm userForm) throws HibernateException {

User user = new User();

BeanUtils.copyProperties(userForm, user);

dao.saveObject(user);

}

}

* 修改后的用户注册Action类RegisterAction的内容。

**[java]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. package com.ynwi.ssh.action;
3. import com.opensymphony.xwork2.ActionSupport;
4. import com.ynwi.ssh.forms.UserForm;
5. import com.ynwi.ssh.service.UserManager;
7. public class RegisterAction extends ActionSupport {
9. private static final long serialVersionUID = 1L;
11. private UserForm user;
13. private UserManager userManager;
15. public UserForm getUser() {
16. return user;
17. }
19. public void setUser(UserForm user) {
20. this.user = user;
21. }
23. public void setUserManager(UserManager userManager) {
24. this.userManager = userManager;
25. }
27. public String execute() {
28. try {
29. userManager.regUser(user);
30. return SUCCESS;
32. } catch (Exception e) {
33. e.printStackTrace();
34. return ERROR;
35. }
36. }
38. }

package com.ynwi.ssh.action;

import com.opensymphony.xwork2.ActionSupport;

import com.ynwi.ssh.forms.UserForm;

import com.ynwi.ssh.service.UserManager;

public class RegisterAction extends ActionSupport {

private static final long serialVersionUID = 1L;

private UserForm user;

private UserManager userManager;

public UserForm getUser() {

return user;

}

public void setUser(UserForm user) {

this.user = user;

}

public void setUserManager(UserManager userManager) {

this.userManager = userManager;

}

public String execute() {

try {

userManager.regUser(user);

return SUCCESS;

} catch (Exception e) {

e.printStackTrace();

return ERROR;

}

}

}

* 修改后的web.xml的内容。

**[html]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** id="WebApp\_9" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"**>**
6. **<display-name>**SSH Project**</display-name>**
8. **<filter>**
9. **<filter-name>**SSH**</filter-name>**
10. **<filter-class>**org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter**</filter-class>**
11. **</filter>**
13. **<filter-mapping>**
14. **<filter-name>**SSH**</filter-name>**
15. **<url-pattern>**/\***</url-pattern>**
16. **</filter-mapping>**
18. **<listener>**
19. **<listener-class>**org.springframework.web.context.ContextLoaderListener**</listener-class>**
20. **</listener>**
22. **<welcome-file-list>**
23. **<welcome-file>**index.html**</welcome-file>**
24. **</welcome-file-list>**
26. **</web-app>**

<?xml version="1.0" encoding="UTF-8"?>

<web-app id="WebApp\_9" version="2.4" 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 http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>SSH Project</display-name>

<filter>

<filter-name>SSH</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>SSH</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

* 修改后的Struts配置文件struts.xml的内容。

**[html]** [view plaincopyprint?](http://blog.csdn.net/yeohcooller/article/details/9316923)

1. **<?xml** version="1.0" encoding="UTF-8" **?>**
2. <!DOCTYPE struts PUBLIC
3. "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4. "http://struts.apache.org/dtds/struts-2.0.dtd"**>**
6. **<struts>**
8. **<include** file="struts-default.xml" **/>**
9. **<package** name="ynwi" extends="struts-default"**>**
10. **<action** name="register" class="regAction"**>**
11. **<result** name="input"**>**/Register.jsp**</result>**
12. **<result** name="success"**>**/Success.jsp**</result>**
13. **<result** name="error"**>**/Fail.jsp**</result>**
14. **</action>**
15. **</package>**
17. **</struts>**

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<include file="struts-default.xml" />

<package name="ynwi" extends="struts-default">

<action name="register" class="regAction">

<result name="input">/Register.jsp</result>

<result name="success">/Success.jsp</result>

<result name="error">/Fail.jsp</result>

</action>

</package>

</struts>

**运行结果**

* 输入访问Register.jsp的地址http://localhost:9090/SSHProject/Register.jsp后。



* 点击注册前后数据库里面的数据。



**总结**

**唠叨**

本来想创建Struts2 + Spring3 + Hibernate4的项目，结果行动之后才发现版本之间不兼容的现象挺严重（我自己是这么认为的，如果有大神看出来是我开发中出的问题导致麻烦告知下），不得不把Hibernate4换成Hibernate3。因为中间换框架，所以导致此博文中整合Hibernate部分是否会报错，不过最终的项目是可以执行的。我会把项目打包上传到csdn资源上。可以[点击这里](http://download.csdn.net/download/yeohcooller/5748981)下载。

**如何快速开发SSH框架项目**

1. 创建Dynamic Web Project并且导入jar包。
2. 我习惯上把classes文件夹由build文件夹内转移到WEB-INF下。在项目上右键Build Path——〉Configure Build Path。设置Default output folder。
3. 在WEB\_INF下面创建web.xml并配置Struts的Filter和Spring的Listener。
4. 在WEB-INF下面创建applicationContext.xml并配置数据源，sessionFactory，Transaction以及各Beans信息。
5. 在src目录下创建struts.xml用来配置action信息，模板可以从Struts空项目里面拿。**这里的action对应的class应该为applicationContext.xml内对应的class的bean的id。**

## 常见问题及解答

### 关于Eclipse IDE

#### 关于以JDK运行Eclipse

有些插件需要用JDK运行Eclipse（Why？），这时候：

机器上装了好几个版本的jdk，要用指定的jdk启动eclipse，只需修改eclipse.ini文件即可。

参考：

-vm  
C:/jdk1.6.0\_18/bin/javaw.exe   （//-- 注：放第一行）  
--launcher.XXMaxPermSize  
256m  
-vmargs  
-Xms128m  
-Xmx512m

其中红色的两行为指定的jdk启动地址，注意：javaw.exe使用的是bin里的。

运行eclipse时，报如下错误：Version 1.3.1\_01 of the JVM is not suitable for this product.Version 1.5 or greater is required.时也可以使用该配置来解决。

#### 1.2安装ADT（Android插件）

Android开发环境配置需要先下载ADT和Android SDK，然后安装ADT。方法：install new software —>add—>Archive（不要解压ADT）选择ADT压缩包

#### 1.3代码在自家的Eclipse里就跑不起来了

实例：在融云的Demo测试的时候，注册登录一直出错，而在其他同事的电脑上就没有问题，可以跑，然后考虑了处理器（ARM还是Intel）问题，JDK问题，甚至下载了一个新的Eclipse和重新配置了环境路径都没有解决。最后想起了import项目的时候出现了一个乱码的问题，当时没注意就把乱码的一部分给改过来就算了，然后设置了一下Eclipse的默认编码格式为UTF-8（所以说同意UTF-8为默认的编码格式是很重要的！）

解决方法：修改Eclipse的默认编码方式为项目的文件的编码方式，建议统一使用UTF-8（啊~多么痛的领悟~）。

### 关于Tamcat

### 关于Java

#### 1xxx cannot be resolved to a type

##### 1.1问题

**HttpServlet cannot be resolved to a type**

##### 1.2解答

这类问题通常是找不到对应的包，或者写错类名所导致的，解决方法就是找到这个类对应的包，HttpServlet是Tomcat中的包，在lib目录中可以找到，然后右键项目，属性->Java Build Path->Add External JARs，去Tomcat的目录，选择servlet-api.jar包添加引用即可。

#### 2.JDK不同版本导致的关于泛型的警告

##### 2.1问题

JDK1.7版本使用非泛型的ArrayList，定义如下变量：

**private** ArrayList listeners = **new** ArrayList(); //非泛型

提示警告如下：

ArrayList is a raw type. References to generic type ArrayList<E> should be parameterized

##### 2.2解答

来源于CSND：<http://bbs.csdn.net/topics/390020116>

Jdk1.5以上使用了泛型，所以未使用泛型会有一个警告。

解决方案：

* 1. 如果不想加泛型，可以在方法上加上这个注解 @SuppressWarnings("unchecked")
  2. 如果不想用泛型，可以调下你的jdk版本，调到1.4或者以下

## Java库

### [java](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava).[lang](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang).ClassLoader

#### 英文文档

A class loader is an object that is responsible（负责的，可靠的，有责任的） for loading classes. The class ClassLoader is an abstract class. Given the [binary name](http://download.oracle.com/javase/7/docs/api/java/lang/ClassLoader.html#name) of a class, a class loader should attempt to locate（vt:位于；查找…的地点。Vi:定位，定居） or generate（使形成；发生；生殖） data that constitutes（构成；组成） a definition for the class. A typical strategy（战略；策略） is to transform the name into a file name and then read a "class file" of that name from a file system.

Every [Class](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82Class) object contains a [reference](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82Class%E2%98%82getClassLoader%E2%98%82) to the ClassLoader that defined it.

Class objects for array classes are not created by class loaders, but are created automatically as required by the Java runtime. The class loader for an array class, as returned by [Class.getClassLoader()](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82Class%E2%98%82getClassLoader%E2%98%82) is the same as the class loader for its element type; if the element type is a primitive type, then the array class has no class loader.

Applications implement subclasses of ClassLoader in order to extend the manner in which the Java virtual machine dynamically loads classes.

Class loaders may typically be used by security managers to indicate security domains.

The ClassLoader class uses a delegation model to search for classes and resources. Each instance of ClassLoader has an associated parent class loader. When requested to find a class or resource, a ClassLoader instance will delegate the search for the class or resource to its parent class loader before attempting to find the class or resource itself. The virtual machine's built-in class loader, called the "bootstrap class loader", does not itself have a parent but may serve as the parent of a ClassLoader instance.

Class loaders that support concurrent loading of classes are known as *parallel capable* class loaders and are required to register themselves at their class initialization time by invoking the [ClassLoader.registerAsParallelCapable](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82%E2%98%82registerAsParallelCapable) method. Note that the ClassLoader class is registered as parallel capable by default. However, its subclasses still need to register themselves if they are parallel capable.   
In environments in which the delegation model is not strictly hierarchical, class loaders need to be parallel capable, otherwise class loading can lead to deadlocks because the loader lock is held for the duration of the class loading process (see [loadClass](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82%E2%98%82loadClass) methods).

Normally, the Java virtual machine loads classes from the local file system in a platform-dependent manner. For example, on UNIX systems, the virtual machine loads classes from the directory defined by the CLASSPATH environment variable.

However, some classes may not originate from a file; they may originate from other sources, such as the network, or they could be constructed by an application. The method [defineClass](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82%E2%98%82defineClass%E2%98%82String%E2%98%82byte%5b%5d%E2%98%82int%E2%98%82int) converts an array of bytes into an instance of class Class. Instances of this newly defined class can be created using [Class.newInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82Class%E2%98%82newInstance).

The methods and constructors of objects created by a class loader may reference other classes. To determine the class(es) referred to, the Java virtual machine invokes the [loadClass](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82%E2%98%82loadClass) method of the class loader that originally created the class.

For example, an application could create a network class loader to download class files from a server. Sample code might look like:

ClassLoader loader = new NetworkClassLoader(host, port);

Object main = loader.loadClass("Main", true).newInstance();

 . . .

The network class loader subclass must define the methods [findClass](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82%E2%98%82findClass) and loadClassData to load a class from the network. Once it has downloaded the bytes that make up the class, it should use the method [defineClass](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82%E2%98%82defineClass) to create a class instance. A sample implementation is:

class NetworkClassLoader extends ClassLoader {

String host;

int port;

public Class findClass(String name) {

byte[] b = loadClassData(name);

return defineClass(name, b, 0, b.length);

}

private byte[] loadClassData(String name) {

// load the class data from the connection

 . . .

}

}

##### Binary names

Any class name provided as a [String](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82String) parameter to methods in ClassLoader must be a binary name as defined by *The Java™ Language Specification*.

Examples of valid class names include:

"java.lang.String"

"javax.swing.JSpinner$DefaultEditor"

"java.security.KeyStore$Builder$FileBuilder$1"

"java.net.URLClassLoader$3$1"

**Since:**

1.0

**See Also:**

[resolveClass(Class)](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(ClassLoader.class%E2%98%83ClassLoader%E2%98%82%E2%98%82resolveClass%E2%98%82Class)

#### 中文自译

ClassLoader是一个负责载入类的对象，给出一个类的二进制名称，ClassLoader就会尝试定位或生成定义这个类的数据（算是类元数据吧）。一个典型的策略是将类名转化为一个文件名，然后从文件系统读取该文件名的类文件。每一个类对象（Class）都包含有一个定义它的ClassLoader的引用

### [java](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava).[util](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util).ResourceBundle

#### 英文文档

Resource bundles contain locale-specific objects. When your program needs a locale-specific resource, a String for example, your program can load it from the resource bundle that is appropriate for the current user's locale. In this way, you can write program code that is largely independent of the user's locale isolating most, if not all, of the locale-specific information in resource bundles.

This allows you to write programs that can:

* be easily localized, or translated, into different languages
* handle multiple locales at once
* be easily modified later to support even more locales

Resource bundles belong to families whose members share a common base name, but whose names also have additional components that identify their locales. For example, the base name of a family of resource bundles might be "MyResources". The family should have a default resource bundle which simply has the same name as its family - "MyResources" - and will be used as the bundle of last resort if a specific locale is not supported. The family can then provide as many locale-specific members as needed, for example a German one named "MyResources\_de".

**Each resource bundle in a family contains the same items,** but the items have been translated for the locale represented by that resource bundle. For example, both "MyResources" and "MyResources\_de" may have a String that's used on a button for canceling operations. In "MyResources" the String may contain "Cancel" and in "MyResources\_de" it may contain "Abbrechen".

If there are different resources for different countries, you can make specializations: for example, "MyResources\_de\_CH" contains objects for the German language (de) in Switzerland (CH). If you want to only modify some of the resources in the specialization, you can do so.

When your program needs a locale-specific object, it loads the ResourceBundle class using the [getBundle](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle%E2%98%82%E2%98%82getBundle%E2%98%82java.lang.String%E2%98%82java.util.Locale) method:

ResourceBundle myResources =

ResourceBundle.getBundle("MyResources", currentLocale);

Resource bundles contain key/value pairs. The keys uniquely identify a locale-specific object in the bundle. Here's an example of a ListResourceBundle that contains two key/value pairs:

public class MyResources extends ListResourceBundle {

protected Object[][] getContents() {

return new Object[][] {

// LOCALIZE THE SECOND STRING OF EACH ARRAY (e.g., "OK")

{"OkKey", "OK"},

{"CancelKey", "Cancel"},

// END OF MATERIAL TO LOCALIZE

};

}

}

Keys are always Strings. In this example, the keys are "OkKey" and "CancelKey". In the above example, the values are also Strings--"OK" and "Cancel"--but they don't have to be. The values can be any type of object.

You retrieve an object from resource bundle using the appropriate getter method. Because "OkKey" and "CancelKey" are both strings, you would use getString to retrieve them:

button1 = new Button(myResources.getString("OkKey"));

button2 = new Button(myResources.getString("CancelKey"));

The getter methods all require the key as an argument and return the object if found. If the object is not found, the getter method throws a MissingResourceException.

Besides getString, ResourceBundle also provides a method for getting string arrays, getStringArray, as well as a generic getObject method for any other type of object. When using getObject, you'll have to cast the result to the appropriate type. For example:

int[] myIntegers = (int[]) myResources.getObject("intList");

The Java Platform provides two subclasses of ResourceBundle, ListResourceBundle and PropertyResourceBundle, that provide a fairly simple way to create resources. As you saw briefly in a previous example, ListResourceBundle manages its resource as a list of key/value pairs. PropertyResourceBundle uses a properties file to manage its resources.

If ListResourceBundle or PropertyResourceBundle do not suit your needs, you can write your own ResourceBundle subclass. Your subclasses must override two methods: handleGetObject and getKeys().

##### ResourceBundle.Control

The [ResourceBundle.Control](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle%E2%98%82ResourceBundle.Control) class provides information necessary to perform the bundle loading process by the getBundle factory methods that take a ResourceBundle.Control instance. You can implement your own subclass in order to enable non-standard resource bundle formats, change the search strategy, or define caching parameters. Refer to the descriptions of the class and the [getBundle](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle%E2%98%82%E2%98%82getBundle%E2%98%82String%E2%98%82Locale%E2%98%82ClassLoader%E2%98%82Control) factory method for details.

##### Cache Management

Resource bundle instances created by the getBundle factory methods are cached by default, and the factory methods return the same resource bundle instance multiple times if it has been cached. getBundle clients may clear the cache, manage the lifetime of cached resource bundle instances using time-to-live values, or specify not to cache resource bundle instances. Refer to the descriptions of the [getBundle factory method](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle%E2%98%82%E2%98%82getBundle%E2%98%82String%E2%98%82Locale%E2%98%82ClassLoader%E2%98%82Control), [clearCache](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle%E2%98%82%E2%98%82clearCache%E2%98%82ClassLoader), [ResourceBundle.Control.getTimeToLive](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle%E2%98%82Control%E2%98%82getTimeToLive%E2%98%82String%E2%98%82Locale), and [ResourceBundle.Control.needsReload](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle%E2%98%82Control%E2%98%82needsReload%E2%98%82String%E2%98%82Locale%E2%98%82String%E2%98%82ClassLoader%E2%98%82ResourceBundle%E2%98%82long) for details.

##### Example

The following is a very simple example of a ResourceBundle subclass, MyResources, that manages two resources (for a larger number of resources you would probably use a Map). Notice that you don't need to supply a value if a "parent-level" ResourceBundle handles the same key with the same value (as for the okKey below).

// default (English language, United States)

public class MyResources extends ResourceBundle {

public Object handleGetObject(String key) {

if (key.equals("okKey")) return "Ok";

if (key.equals("cancelKey")) return "Cancel";

return null;

}

public Enumeration<String> getKeys() {

return Collections.enumeration(keySet());

}

// Overrides handleKeySet() so that the getKeys() implementation

// can rely on the keySet() value.

protected Set<String> handleKeySet() {

return new HashSet<String>(Arrays.asList("okKey", "cancelKey"));

}

}

// German language

public class MyResources\_de extends MyResources {

public Object handleGetObject(String key) {

// don't need okKey, since parent level handles it.

if (key.equals("cancelKey")) return "Abbrechen";

return null;

}

protected Set<String> handleKeySet() {

return new HashSet<String>(Arrays.asList("cancelKey"));

}

}

You do not have to restrict yourself to using a single family of ResourceBundles. For example, you could have a set of bundles for exception messages, ExceptionResources (ExceptionResources\_fr, ExceptionResources\_de, ...), and one for widgets, WidgetResource (WidgetResources\_fr, WidgetResources\_de, ...); breaking up the resources however you like.

**Since:**

JDK1.1

**See Also:**

[ListResourceBundle](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle%E2%98%82ListResourceBundle)

[PropertyResourceBundle](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle%E2%98%82PropertyResourceBundle)

[MissingResourceException](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle%E2%98%82MissingResourceException)

#### [ResourceBundle](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82ResourceBundle) [java](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava).[util](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util).[ResourceBundle](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle).getBundle([String](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82String) baseName, [Locale](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82Locale) locale, [ClassLoader](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82ClassLoader) loader)

Gets a resource bundle using the specified base name, locale, and class loader.

This method behaves the same as calling [getBundle(String, Locale, ClassLoader, Control)](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82%E2%98%82getBundle%E2%98%82String%E2%98%82Locale%E2%98%82ClassLoader%E2%98%82Control) passing a default instance of [Control](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82Control). The following describes this behavior.

getBundle uses the base name, the specified locale, and the default locale (obtained from [Locale.getDefault](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82java.util.Locale%E2%98%82getDefault%E2%98%82)) to generate a sequence of candidate bundle names. If the specified locale's language, script, country, and variant are all empty strings, then the base name is the only candidate bundle name. Otherwise, a list of candidate locales is generated from the attribute values of the specified locale (language, script, country and variant) and appended to the base name. Typically, this will look like the following:

baseName + "\_" + language + "\_" + script + "\_" + country + "\_" + variant

baseName + "\_" + language + "\_" + script + "\_" + country

baseName + "\_" + language + "\_" + script

baseName + "\_" + language + "\_" + country + "\_" + variant

baseName + "\_" + language + "\_" + country

baseName + "\_" + language

Candidate bundle names where the final component is an empty string are omitted, along with the underscore. For example, if country is an empty string, the second and the fifth candidate bundle names above would be omitted. Also, if script is an empty string, the candidate names including script are omitted. For example, a locale with language "de" and variant "JAVA" will produce candidate names with base name "MyResource" below.

MyResource\_de\_\_JAVA

MyResource\_de

In the case that the variant contains one or more underscores ('\_'), a sequence of bundle names generated by truncating the last underscore and the part following it is inserted after a candidate bundle name with the original variant. For example, for a locale with language "en", script "Latn, country "US" and variant "WINDOWS\_VISTA", and bundle base name "MyResource", the list of candidate bundle names below is generated:

MyResource\_en\_Latn\_US\_WINDOWS\_VISTA

MyResource\_en\_Latn\_US\_WINDOWS

MyResource\_en\_Latn\_US

MyResource\_en\_Latn

MyResource\_en\_US\_WINDOWS\_VISTA

MyResource\_en\_US\_WINDOWS

MyResource\_en\_US

MyResource\_en

**Note:** For some Locales, the list of candidate bundle names contains extra names, or the order of bundle names is slightly modified. See the description of the default implementation of [getCandidateLocales](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82Control%E2%98%82getCandidateLocales%E2%98%82String%E2%98%82Locale) for details.

getBundle then iterates over the candidate bundle names to find the first one for which it can instantiate an actual resource bundle. It uses the default controls' [getFormats](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82Control%E2%98%82getFormats) method, which generates two bundle names for each generated name, the first a class name and the second a properties file name. For each candidate bundle name, it attempts to create a resource bundle:

* First, it attempts to load a class using the generated class name. If such a class can be found and loaded using the specified class loader, is assignment compatible with ResourceBundle, is accessible from ResourceBundle, and can be instantiated, getBundle creates a new instance of this class and uses it as the result resource bundle.
* Otherwise, getBundle attempts to locate a property resource file using the generated properties file name. It generates a path name from the candidate bundle name by replacing all "." characters with "/" and appending the string ".properties". It attempts to find a "resource" with this name using [ClassLoader.getResource](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82java.lang.ClassLoader%E2%98%82getResource%E2%98%82java.lang.String). (Note that a "resource" in the sense of getResource has nothing to do with the contents of a resource bundle, it is just a container of data, such as a file.) If it finds a "resource", it attempts to create a new [PropertyResourceBundle](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82PropertyResourceBundle) instance from its contents. If successful, this instance becomes the result resource bundle.

This continues until a result resource bundle is instantiated or the list of candidate bundle names is exhausted. If no matching resource bundle is found, the default control's [getFallbackLocale](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82Control%E2%98%82getFallbackLocale) method is called, which returns the current default locale. A new sequence of candidate locale names is generated using this locale and and searched again, as above.

If still no result bundle is found, the base name alone is looked up. If this still fails, a MissingResourceException is thrown.

Once a result resource bundle has been found, its parent chain is instantiated. If the result bundle already has a parent (perhaps because it was returned from a cache) the chain is complete.

Otherwise, getBundle examines the remainder of the candidate locale list that was used during the pass that generated the result resource bundle. (As before, candidate bundle names where the final component is an empty string are omitted.) When it comes to the end of the candidate list, it tries the plain bundle name. With each of the candidate bundle names it attempts to instantiate a resource bundle (first looking for a class and then a properties file, as described above).

Whenever it succeeds, it calls the previously instantiated resource bundle's [setParent](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82%E2%98%82setParent%E2%98%82java.util.ResourceBundle) method with the new resource bundle. This continues until the list of names is exhausted or the current bundle already has a non-null parent.

Once the parent chain is complete, the bundle is returned.

**Note:** getBundle caches instantiated resource bundles and might return the same resource bundle instance multiple times.

**Note:**The baseName argument should be a fully qualified class name. However, for compatibility with earlier versions, Sun's Java SE Runtime Environments do not verify this, and so it is possible to access PropertyResourceBundles by specifying a path name (using "/") instead of a fully qualified class name (using ".").

**Example:**

The following class and property files are provided:

MyResources.class

MyResources.properties

MyResources\_fr.properties

MyResources\_fr\_CH.class

MyResources\_fr\_CH.properties

MyResources\_en.properties

MyResources\_es\_ES.class

The contents of all files are valid (that is, public non-abstract subclasses of ResourceBundle for the ".class" files, syntactically correct ".properties" files). The default locale is Locale("en", "GB").

Calling getBundle with the locale arguments below will instantiate resource bundles as follows:

|  |  |
| --- | --- |
| Locale("fr", "CH") | MyResources\_fr\_CH.class, parent MyResources\_fr.properties, parent MyResources.class |
| Locale("fr", "FR") | MyResources\_fr.properties, parent MyResources.class |
| Locale("de", "DE") | MyResources\_en.properties, parent MyResources.class |
| Locale("en", "US") | MyResources\_en.properties, parent MyResources.class |
| Locale("es", "ES") | MyResources\_es\_ES.class, parent MyResources.class |

The file MyResources\_fr\_CH.properties is never used because it is hidden by the MyResources\_fr\_CH.class. Likewise, MyResources.properties is also hidden by MyResources.class.

**Parameters:**

**baseName** the base name of the resource bundle, a fully qualified class name

**locale** the locale for which a resource bundle is desired

**loader** the class loader from which to load the resource bundle

**Returns:**

a resource bundle for the given base name and locale

**Throws:**

[java.lang.NullPointerException](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82java.lang.NullPointerException) - if baseName, locale, or loader is null

[MissingResourceException](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.util(ResourceBundle.class%E2%98%83ResourceBundle~getBundle~Ljava.lang.String;~Ljava.util.Locale;~Ljava.lang.ClassLoader;%E2%98%82MissingResourceException) - if no resource bundle for the specified base name can be found

**Since:**

1.2

### [java](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava).[text](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text).MessageFormat

#### 英文文档

MessageFormat provides a means to produce concatenated messages in a language-neutral way. Use this to construct messages displayed for end users.

MessageFormat takes a set of objects, formats them, then inserts the formatted strings into the pattern at the appropriate places.

**Note:** MessageFormat differs from the other Format classes in that you create a MessageFormat object with one of its constructors (not with a getInstance style factory method). The factory methods aren't necessary because MessageFormat itself doesn't implement locale specific behavior. Any locale specific behavior is defined by the pattern that you provide as well as the subformats used for inserted arguments.

***Patterns and Their Interpretation***

MessageFormat uses patterns of the following form:

*MessageFormatPattern:*

*String*

*MessageFormatPattern* *FormatElement* *String*

*FormatElement:*

{ *ArgumentIndex* }

{ *ArgumentIndex* , *FormatType* }

{ *ArgumentIndex* , *FormatType* , *FormatStyle* }

*FormatType: one of*

number date time choice

*FormatStyle:*

short

medium

long

full

integer

currency

percent

*SubformatPattern*

Within a *String*, a pair of single quotes can be used to quote any arbitrary characters except single quotes. For example, pattern string "'{0}'" represents string "{0}", not a *FormatElement*. A single quote itself must be represented by doubled single quotes '' throughout a *String*. For example, pattern string "'{''}'" is interpreted as a sequence of '{ (start of quoting and a left curly brace), '' (a single quote), and }' (a right curly brace and end of quoting), *not* '{' and '}' (quoted left and right curly braces): representing string "{'}", *not* "{}".

A *SubformatPattern* is interpreted by its corresponding subformat, and subformat-dependent pattern rules apply. For example, pattern string "{1,number,$'#',##}" (*SubformatPattern* with underline) will produce a number format with the pound-sign quoted, with a result such as: "$#31,45". Refer to each Format subclass documentation for details.

Any unmatched quote is treated as closed at the end of the given pattern. For example, pattern string "'{0}"} is treated as pattern "'{0}'"}.

Any curly braces within an unquoted pattern must be balanced. For example, "ab {0} de" and "ab '}' de" are valid patterns, but "ab {0'}' de", "ab } de" and "''{''" are not.

**Warning:**

The rules for using quotes within message format patterns unfortunately have shown to be somewhat confusing. In particular, it isn't always obvious to localizers whether single quotes need to be doubled or not. Make sure to inform localizers about the rules, and tell them (for example, by using comments in resource bundle source files) which strings will be processed by MessageFormat. Note that localizers may need to use single quotes in translated strings where the original version doesn't have them.

The *ArgumentIndex* value is a non-negative integer written using the digits '0' through '9', and represents an index into the arguments array passed to the format methods or the result array returned by the parse methods.

The *FormatType* and *FormatStyle* values are used to create a Format instance for the format element. The following table shows how the values map to Format instances. Combinations not shown in the table are illegal. A *SubformatPattern* must be a valid pattern string for the Format subclass used.

|  |  |  |
| --- | --- | --- |
| **FormatType** | **FormatStyle** | **Subformat Created** |
| *(none)* | *(none)* | null |
| number | *(none)* | [NumberFormat.getInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82NumberFormat%E2%98%82getInstance%E2%98%82Locale)(getLocale()) |
| integer | [NumberFormat.getIntegerInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82NumberFormat%E2%98%82getIntegerInstance%E2%98%82Locale)(getLocale()) |
| currency | [NumberFormat.getCurrencyInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82NumberFormat%E2%98%82getCurrencyInstance%E2%98%82Locale)(getLocale()) |
| percent | [NumberFormat.getPercentInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82NumberFormat%E2%98%82getPercentInstance%E2%98%82Locale)(getLocale()) |
| *SubformatPattern* | new [DecimalFormat](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DecimalFormat%E2%98%82DecimalFormat%E2%98%82String%E2%98%82DecimalFormatSymbols)(subformatPattern, [DecimalFormatSymbols.getInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DecimalFormatSymbols%E2%98%82getInstance%E2%98%82Locale)(getLocale())) |
| date | *(none)* | [DateFormat.getDateInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82getDateInstance%E2%98%82int%E2%98%82Locale)([DateFormat.DEFAULT](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82DEFAULT), getLocale()) |
| short | [DateFormat.getDateInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82getDateInstance%E2%98%82int%E2%98%82Locale)([DateFormat.SHORT](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82SHORT), getLocale()) |
| medium | [DateFormat.getDateInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82getDateInstance%E2%98%82int%E2%98%82Locale)([DateFormat.DEFAULT](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82DEFAULT), getLocale()) |
| long | [DateFormat.getDateInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82getDateInstance%E2%98%82int%E2%98%82Locale)([DateFormat.LONG](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82LONG), getLocale()) |
| full | [DateFormat.getDateInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82getDateInstance%E2%98%82int%E2%98%82Locale)([DateFormat.FULL](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82FULL), getLocale()) |
| *SubformatPattern* | new [SimpleDateFormat](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82SimpleDateFormat%E2%98%82SimpleDateFormat%E2%98%82String%E2%98%82Locale)(subformatPattern, getLocale()) |
| time | *(none)* | [DateFormat.getTimeInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82getTimeInstance%E2%98%82int%E2%98%82Locale)([DateFormat.DEFAULT](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82DEFAULT), getLocale()) |
| short | [DateFormat.getTimeInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82getTimeInstance%E2%98%82int%E2%98%82Locale)([DateFormat.SHORT](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82SHORT), getLocale()) |
| medium | [DateFormat.getTimeInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82getTimeInstance%E2%98%82int%E2%98%82Locale)([DateFormat.DEFAULT](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82DEFAULT), getLocale()) |
| long | [DateFormat.getTimeInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82getTimeInstance%E2%98%82int%E2%98%82Locale)([DateFormat.LONG](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82LONG), getLocale()) |
| full | [DateFormat.getTimeInstance](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82getTimeInstance%E2%98%82int%E2%98%82Locale)([DateFormat.FULL](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat%E2%98%82FULL), getLocale()) |
| *SubformatPattern* | new [SimpleDateFormat](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82SimpleDateFormat%E2%98%82SimpleDateFormat%E2%98%82String%E2%98%82Locale)(subformatPattern, getLocale()) |
| choice | *SubformatPattern* | new [ChoiceFormat](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82ChoiceFormat%E2%98%82ChoiceFormat%E2%98%82String)(subformatPattern) |

***Usage Information***

Here are some examples of usage. In real internationalized programs, the message format pattern and other static strings will, of course, be obtained from resource bundles. Other parameters will be dynamically determined at runtime.

The first example uses the static method MessageFormat.format, which internally creates a MessageFormat for one-time use:

int planet = 7;

String event = "a disturbance in the Force";

String result = MessageFormat.format(

"At {1,time} on {1,date}, there was {2} on planet {0,number,integer}.",

planet, new Date(), event);

The output is:

At 12:30 PM on Jul 3, 2053, there was a disturbance in the Force on planet 7.

The following example creates a MessageFormat instance that can be used repeatedly:

int fileCount = 1273;

String diskName = "MyDisk";

Object[] testArgs = {new Long(fileCount), diskName};

MessageFormat form = new MessageFormat(

"The disk \"{1}\" contains {0} file(s).");

System.out.println(form.format(testArgs));

The output with different values for fileCount:

The disk "MyDisk" contains 0 file(s).

The disk "MyDisk" contains 1 file(s).

The disk "MyDisk" contains 1,273 file(s).

For more sophisticated patterns, you can use a ChoiceFormat to produce correct forms for singular and plural:

MessageFormat form = new MessageFormat("The disk \"{1}\" contains {0}.");

double[] filelimits = {0,1,2};

String[] filepart = {"no files","one file","{0,number} files"};

ChoiceFormat fileform = new ChoiceFormat(filelimits, filepart);

form.setFormatByArgumentIndex(0, fileform);

int fileCount = 1273;

String diskName = "MyDisk";

Object[] testArgs = {new Long(fileCount), diskName};

System.out.println(form.format(testArgs));

The output with different values for fileCount:

The disk "MyDisk" contains no files.

The disk "MyDisk" contains one file.

The disk "MyDisk" contains 1,273 files.

You can create the ChoiceFormat programmatically, as in the above example, or by using a pattern. See [ChoiceFormat](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82ChoiceFormat) for more information.

form.applyPattern(

"There {0,choice,0#are no files|1#is one file|1<are {0,number,integer} files}.");

**Note:** As we see above, the string produced by a ChoiceFormat in MessageFormat is treated as special; occurrences of '{' are used to indicate subformats, and cause recursion. If you create both a MessageFormat and ChoiceFormat programmatically (instead of using the string patterns), then be careful not to produce a format that recurses on itself, which will cause an infinite loop.

When a single argument is parsed more than once in the string, the last match will be the final result of the parsing. For example,

MessageFormat mf = new MessageFormat("{0,number,#.##}, {0,number,#.#}");

Object[] objs = {new Double(3.1415)};

String result = mf.format( objs );

// result now equals "3.14, 3.1"

objs = null;

objs = mf.parse(result, new ParsePosition(0));

// objs now equals {new Double(3.1)}

Likewise, parsing with a MessageFormat object using patterns containing multiple occurrences of the same argument would return the last match. For example,

MessageFormat mf = new MessageFormat("{0}, {0}, {0}");

String forParsing = "x, y, z";

Object[] objs = mf.parse(forParsing, new ParsePosition(0));

// result now equals {new String("z")}

***Synchronization***

Message formats are not synchronized. It is recommended to create separate format instances for each thread. If multiple threads access a format concurrently, it must be synchronized externally.

**Author:**

Mark Davis

**See Also:**

[java.util.Locale](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82java.util.Locale)

[Format](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82Format)

[NumberFormat](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82NumberFormat)

[DecimalFormat](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DecimalFormat)

[DecimalFormatSymbols](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DecimalFormatSymbols)

[ChoiceFormat](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82ChoiceFormat)

[DateFormat](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82DateFormat)

[SimpleDateFormat](eclipse-javadoc:%E2%98%82=openjump/C:%5C/Program%20Files%5C/Java%5C/jdk1.7.0_25%5C/jre%5C/lib%5C/rt.jar%3Cjava.text(MessageFormat.class%E2%98%83MessageFormat%E2%98%82SimpleDateFormat)