[为java应用程序设置开机启动项](http://blog.csdn.net/x844010689/article/details/8049715)

分类： [java](http://blog.csdn.net/x844010689/article/category/1186757)2012-10-08 19:41 1603人阅读 [评论](http://blog.csdn.net/x844010689/article/details/8049715#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/x844010689/article/details/8049715#report)

[java](http://www.csdn.net/tag/java)[classloader](http://www.csdn.net/tag/classloader)[jar](http://www.csdn.net/tag/jar)[delete](http://www.csdn.net/tag/delete)[path](http://www.csdn.net/tag/path)[servlet](http://www.csdn.net/tag/servlet)

思路就是将java应用程序打包成.jar文件，然后转成.exe，通过修改注册表来增加删除启动项，即将安装后的.exe执行文件添加到注册表中；

首先将java应用程序打包成.jar文件，可以利用如下代码找到.jar文件的绝对路径，即也可以找到安装后的.exe执行文件

转载：

对于Java程序，无论是未打包的还是打包的JAR或WAR文件，有时候都需要获取它运行所在目录信息，如何做到这一点呢？   
  
在Java处理的文件系统中，目录的表示方式有两种：   
  
（1）绝对目录，它以"/"为起始字符，代表从根目录下开始寻找给出的目录，如/c:/java   
  
（2）相对路径，它以不带“/”的目录名表示，表示以当前Java程序正在运行的目录作为起始目录来寻找给出的目录。如java/classes。在相对路径中，有一些特定的字符，可以代表特的的目录，比如，“.”代表当前目录，“..”代表当前目录的上一级目录。在网上很多给出的例子中，就是利用"."作为目录名，构造File对象的实例，然后通过File对象的方法来获取当前程序运行的目录。   
  
    这种方法虽然简单，但有时不能正确的得出当前程序的运行目录。原因在于，运行Java程序不一定要进入到该程序的类文件或JAR文件所在的目录，只要在运行时指定了正确的类路径信息，就可以在任何目录中运行Java程序，此时利用这种方法只能得到发出运行命令时所在的目录信息。   
  
     从上面的分析可以看出，对于很多Java程序，尤其是WEB程序，利用当前路径的“.”表示法，都不能满足要求。那么怎样才能正确的得到运行目录信息呢？   
  
     在Web程序中，利用Servlet API可以获得一些路径信息，比如HttpServletRequest接口中定义的getRealPath方法，但类似这些方法都依赖于Servlet环境，不便于程序的单元测试。   
  
    本文提供了一种只使用Java标准API的路径探测方法，就是利用ClassLoader抽象类。   
  
    利用java.lang.Class的getClassLoader方法，可以获得给定类的ClassLoader实例，它的getResource方法可以获得当前类装载器中的资源的位置，我们可以利用类文件的名称作为要查找的资源，经过处理后就可获得当前Java程序的运行位置信息，其伪代码如下：   
      
    获得Class参数的所在的类名   
    取得该类所在的包名   
    将包名转换为路径   
    利用getResource得到当前的类文件所在URL   
    利用URL解析出当前Java程序所在的路径

具体代码如下：

java代码：

Java代码

1. /\*\*-----------------------------------------------------------------------
2. \*getAppPath需要一个当前程序使用的Java类的class属性参数，它可以返回打包过的
3. \*Java可执行文件（jar，war）所处的系统目录名或非打包Java程序所处的目录
4. \*@param cls为Class类型
5. \*@return 返回值为该类所在的Java程序运行的目录
6. -------------------------------------------------------------------------\*/
7. **public** **static** String getAppPath(Class cls){
8. //检查用户传入的参数是否为空
9. **if**(cls==**null**)
10. **throw** **new** java.lang.IllegalArgumentException("参数不能为空！");
11. ClassLoader loader=cls.getClassLoader();
12. //获得类的全名，包括包名
13. String clsName=cls.getName()+".class";
14. //获得传入参数所在的包
15. Package pack=cls.getPackage();
16. String path="";
17. //如果不是匿名包，将包名转化为路径
18. **if**(pack!=**null**){
19. String packName=pack.getName();
20. //此处简单判定是否是Java基础类库，防止用户传入JDK内置的类库
21. **if**(packName.startsWith("java.")||packName.startsWith("javax."))
22. **throw** **new** java.lang.IllegalArgumentException("不要传送系统类！");
23. //在类的名称中，去掉包名的部分，获得类的文件名
24. clsName=clsName.substring(packName.length()+1);
25. //判定包名是否是简单包名，如果是，则直接将包名转换为路径，
26. **if**(packName.indexOf(".")<0) path=packName+"/";
27. **else**{//否则按照包名的组成部分，将包名转换为路径
28. **int** start=0,end=0;
29. end=packName.indexOf(".");
30. **while**(end!=-1){
31. path=path+packName.substring(start,end)+"/";
32. start=end+1;
33. end=packName.indexOf(".",start);
34. }
35. path=path+packName.substring(start)+"/";
36. }
37. }
38. //调用ClassLoader的getResource方法，传入包含路径信息的类文件名
39. java.net.URL url =loader.getResource(path+clsName);
40. //从URL对象中获取路径信息
41. String realPath=url.getPath();
42. //去掉路径信息中的协议名"file:"
43. **int** pos=realPath.indexOf("file:");
44. **if**(pos>-1) realPath=realPath.substring(pos+5);
45. //去掉路径信息最后包含类文件信息的部分，得到类所在的路径
46. pos=realPath.indexOf(path+clsName);
47. realPath=realPath.substring(0,pos-1);
48. //如果类文件被打包到JAR等文件中时，去掉对应的JAR等打包文件名
49. **if**(realPath.endsWith("!"))
50. realPath=realPath.substring(0,realPath.lastIndexOf("/"));
51. /\*------------------------------------------------------------
52. ClassLoader的getResource方法使用了utf-8对路径信息进行了编码，当路径
53. 中存在中文和空格时，他会对这些字符进行转换，这样，得到的往往不是我们想要
54. 的真实路径，在此，调用了URLDecoder的decode方法进行解码，以便得到原始的
55. 中文及空格路径
56. -------------------------------------------------------------\*/
57. **try**{
58. realPath=java.net.URLDecoder.decode(realPath,"utf-8");
59. }**catch**(Exception e){**throw** **new** RuntimeException(e);}
60. **return** realPath;
61. }//getAppPath定义结束
62. //-----------------------------------------------------------------

**[java]** [view plaincopy](http://blog.csdn.net/Ja5on/article/details/5871348)

1. /\*\*-----------------------------------------------------------------------
2. \*getAppPath需要一个当前程序使用的Java类的class属性参数，它可以返回打包过的
3. \*Java可执行文件（jar，war）所处的系统目录名或非打包Java程序所处的目录
4. \*@param cls为Class类型
5. \*@return 返回值为该类所在的Java程序运行的目录
6. -------------------------------------------------------------------------\*/
7. **public** **static** String getAppPath(Class cls){
8. //检查用户传入的参数是否为空
9. **if**(cls==**null**)
10. **throw** **new** java.lang.IllegalArgumentException("参数不能为空！");
11. ClassLoader loader=cls.getClassLoader();
12. //获得类的全名，包括包名
13. String clsName=cls.getName()+".class";
14. //获得传入参数所在的包
15. Package pack=cls.getPackage();
16. String path="";
17. //如果不是匿名包，将包名转化为路径
18. **if**(pack!=**null**){
19. String packName=pack.getName();
20. //此处简单判定是否是Java基础类库，防止用户传入JDK内置的类库
21. **if**(packName.startsWith("java.")||packName.startsWith("javax."))
22. **throw** **new** java.lang.IllegalArgumentException("不要传送系统类！");
23. //在类的名称中，去掉包名的部分，获得类的文件名
24. clsName=clsName.substring(packName.length()+1);
25. //判定包名是否是简单包名，如果是，则直接将包名转换为路径，
26. **if**(packName.indexOf(".")<0) path=packName+"/";
27. **else**{//否则按照包名的组成部分，将包名转换为路径
28. **int** start=0,end=0;
29. end=packName.indexOf(".");
30. **while**(end!=-1){
31. path=path+packName.substring(start,end)+"/";
32. start=end+1;
33. end=packName.indexOf(".",start);
34. }
35. path=path+packName.substring(start)+"/";
36. }
37. }
38. //调用ClassLoader的getResource方法，传入包含路径信息的类文件名
39. java.net.URL url =loader.getResource(path+clsName);
40. //从URL对象中获取路径信息
41. String realPath=url.getPath();
42. //去掉路径信息中的协议名"file:"
43. **int** pos=realPath.indexOf("file:");
44. **if**(pos>-1) realPath=realPath.substring(pos+5);
45. //去掉路径信息最后包含类文件信息的部分，得到类所在的路径
46. pos=realPath.indexOf(path+clsName);
47. realPath=realPath.substring(0,pos-1);
48. //如果类文件被打包到JAR等文件中时，去掉对应的JAR等打包文件名
49. **if**(realPath.endsWith("!"))
50. realPath=realPath.substring(0,realPath.lastIndexOf("/"));
51. /\*------------------------------------------------------------
52. ClassLoader的getResource方法使用了utf-8对路径信息进行了编码，当路径
53. 中存在中文和空格时，他会对这些字符进行转换，这样，得到的往往不是我们想要
54. 的真实路径，在此，调用了URLDecoder的decode方法进行解码，以便得到原始的
55. 中文及空格路径
56. -------------------------------------------------------------\*/
57. **try**{
58. realPath=java.net.URLDecoder.decode(realPath,"utf-8");
59. }**catch**(Exception e){**throw** **new** RuntimeException(e);}
60. **return** realPath;
61. }//getAppPath定义结束
62. //-----------------------------------------------------------------

该方法既可以用于JAR或WAR文件，也可以用于非JAR文件。但要注意以下2点：

1. 不要传递系统的类，作为getAppPath的参数，如java.lang.String.class，当然，也不要传递那些已经位于JDK中的那些类，比如xml相关的一些类等等。
2. 要传递应该是程序中主要的运行类，不要传递程序中的支持类库中的类文件，也就是那些第三方的类库中的类文件，否则得到的将是那些类库的位置。

然后可以通过修改注册表来增加开机启动项:

转载：

需要修改的注册表项   
[HKEY\_LOCAL\_MACHINE/Software/Microsoft/Windows/CurrentVersion/Run]  开机自动运行程序  
[HKEY\_LOCAL\_MACHINE/Software/Microsoft/Windows/CurrentVersion/RunOnce] 开机自动运行程序 且 仅运行一次   
[HKEY\_LOCAL\_MACHINE/Software/Microsoft/Windows/CurrentVersion/RunServices] 开机自动运行服务

JDK 从1.4开始提供操作 Windows 的 API 是 Preferences，因为这个 API 也是跨平台的，所功能比较弱，在 Win32 下只能用来操作 HKCU/Software/JavaSoft 和 HKLM/Software/JavaSoft 下及子节点的数据。

自由访问注册表其他键的值光用 Java 是做不到的，必然方案就是 JNI，这里我使用的是Windows Registry API Native Interface <http://www.trustice.com/java/jnireg/index.shtml> 下的 [registry-3.1.3.zip](http://www.gjt.org/download/time/java/jnireg/registry-3.1.3.zip)(包含源代码)。可以利用它访问、修改、导出注册表项到文件等。解开 registry-3.1.3.zip，在 bin 目录中可以看到两个文件 ICE\_JNIRegistry.dll 和 registry.jar，动态库就是本地代码实现。

com.ice.jni.registry.Registry.main() 就是 registry 的示例代码，动态库 ICE\_JNIRegistry.dll 也是在这个类的静态块中被加载的，记得要把 ICE\_JNIRegistry.dll 放在它能够被加载的位置上，比如你把 registry-3.1.3.zip 解压到 c:/registry-3.1.3，在命令行下你可以进入到这个目录中，并执行。

代码：

Java代码

1. **package** org.zh.ss.util;
3. **import** com.ice.jni.registry.\*;
4. **import** java.text.SimpleDateFormat;
6. /\*\* \*//\*\*
7. \* java 操作注册表
8. \* @author 李志远
9. \*/
10. **public** **class** RegeditTool {
12. **static** SimpleDateFormat shortDateFormat = **new** SimpleDateFormat("yyyy-MM-dd");
14. /\*\* \*//\*\* \*/
15. /\*\* \*//\*\* Creates a new instance of test \*/
17. // 把信息存储到注册表HKEY\_LOCAL\_MACHINE下的某个节点的某一变量中，有则修改，无则创建
18. **public** **static** **boolean** setValue(String folder, String subKeyNode,
19. String subKeyName, String subKeyValue) {
20. **try** {
21. RegistryKey software = Registry.HKEY\_LOCAL\_MACHINE
22. .openSubKey(folder);
23. RegistryKey subKey = software.createSubKey(subKeyNode, "");
24. subKey
25. .setValue(**new** RegStringValue(subKey, subKeyName,
26. subKeyValue));
27. subKey.closeKey();
28. **return** **true**;
29. } **catch** (NoSuchKeyException e) {
30. e.printStackTrace();
31. } **catch** (NoSuchValueException e) {
32. e.printStackTrace();
33. } **catch** (RegistryException e) {
34. e.printStackTrace();
35. }
36. **return** **false**;
37. }
39. // 删除注册表中某节点下的某个变量
40. **public** **static** **boolean** deleteValue(String folder, String subKeyNode,
41. String subKeyName) {
43. **try** {
44. RegistryKey software = Registry.HKEY\_LOCAL\_MACHINE
45. .openSubKey(folder);
46. RegistryKey subKey = software.createSubKey(subKeyNode, "");
47. subKey.deleteValue(subKeyName);
48. subKey.closeKey();
49. **return** **true**;
50. } **catch** (NoSuchKeyException e) {
51. System.out.println("NOsuchKey\_delete");
52. } **catch** (NoSuchValueException e) {
53. System.out.println("NOsuchValue\_delete");
54. } **catch** (RegistryException e) {
55. e.printStackTrace();
56. }
57. **return** **false**;
58. }
60. // 删除注册表中某节点下的某节点
61. **public** **static** **boolean** deleteSubKey(String folder, String subKeyNode) {
62. **try** {
63. RegistryKey software = Registry.HKEY\_LOCAL\_MACHINE
64. .openSubKey(folder);
65. software.deleteSubKey(subKeyNode);
66. software.closeKey();
67. **return** **true**;
68. } **catch** (NoSuchKeyException e) {
69. e.printStackTrace();
70. } **catch** (RegistryException e) {
71. e.printStackTrace();
72. }
73. **return** **false**;
74. }
76. // 打开注册表项并读出相应的变量名的值
77. **public** **static** String getValue(String folder, String subKeyNode,
78. String subKeyName) {
79. String value = "";
80. **try** {
81. RegistryKey software = Registry.HKEY\_LOCAL\_MACHINE
82. .openSubKey(folder);
83. RegistryKey subKey = software.openSubKey(subKeyNode);
84. value = subKey.getStringValue(subKeyName);
85. subKey.closeKey();
86. } **catch** (NoSuchKeyException e) {
87. value = "NoSuchKey";
88. // e.printStackTrace();
89. } **catch** (NoSuchValueException e) {
90. value = "NoSuchValue";
91. // e.printStackTrace();
92. } **catch** (RegistryException e) {
93. e.printStackTrace();
94. }
95. **return** value;
96. }
98. // 测试
99. **public** **static** **void** main(String[] args) {
100. setValue("SOFTWARE", "Microsoft//Windows//CurrentVersion//Run", "test",
101. "C://1.exe");
102. }
103. }

**[java]** [view plaincopy](http://blog.csdn.net/Ja5on/article/details/5871348)

1. **package** org.zh.ss.util;
3. **import** com.ice.jni.registry.\*;
4. **import** java.text.SimpleDateFormat;
6. /\*\* \*//\*\*
7. \* java 操作注册表
8. \* @author 李志远
9. \*/
10. **public** **class** RegeditTool {
12. **static** SimpleDateFormat shortDateFormat = **new** SimpleDateFormat("yyyy-MM-dd");
14. /\*\* \*//\*\* \*/
15. /\*\* \*//\*\* Creates a new instance of test \*/
17. // 把信息存储到注册表HKEY\_LOCAL\_MACHINE下的某个节点的某一变量中，有则修改，无则创建
18. **public** **static** **boolean** setValue(String folder, String subKeyNode,
19. String subKeyName, String subKeyValue) {
20. **try** {
21. RegistryKey software = Registry.HKEY\_LOCAL\_MACHINE
22. .openSubKey(folder);
23. RegistryKey subKey = software.createSubKey(subKeyNode, "");
24. subKey
25. .setValue(**new** RegStringValue(subKey, subKeyName,
26. subKeyValue));
27. subKey.closeKey();
28. **return** **true**;
29. } **catch** (NoSuchKeyException e) {
30. e.printStackTrace();
31. } **catch** (NoSuchValueException e) {
32. e.printStackTrace();
33. } **catch** (RegistryException e) {
34. e.printStackTrace();
35. }
36. **return** **false**;
37. }
39. // 删除注册表中某节点下的某个变量
40. **public** **static** **boolean** deleteValue(String folder, String subKeyNode,
41. String subKeyName) {
43. **try** {
44. RegistryKey software = Registry.HKEY\_LOCAL\_MACHINE
45. .openSubKey(folder);
46. RegistryKey subKey = software.createSubKey(subKeyNode, "");
47. subKey.deleteValue(subKeyName);
48. subKey.closeKey();
49. **return** **true**;
50. } **catch** (NoSuchKeyException e) {
51. System.out.println("NOsuchKey\_delete");
52. } **catch** (NoSuchValueException e) {
53. System.out.println("NOsuchValue\_delete");
54. } **catch** (RegistryException e) {
55. e.printStackTrace();
56. }
57. **return** **false**;
58. }
60. // 删除注册表中某节点下的某节点
61. **public** **static** **boolean** deleteSubKey(String folder, String subKeyNode) {
62. **try** {
63. RegistryKey software = Registry.HKEY\_LOCAL\_MACHINE
64. .openSubKey(folder);
65. software.deleteSubKey(subKeyNode);
66. software.closeKey();
67. **return** **true**;
68. } **catch** (NoSuchKeyException e) {
69. e.printStackTrace();
70. } **catch** (RegistryException e) {
71. e.printStackTrace();
72. }
73. **return** **false**;
74. }
76. // 打开注册表项并读出相应的变量名的值
77. **public** **static** String getValue(String folder, String subKeyNode,
78. String subKeyName) {
79. String value = "";
80. **try** {
81. RegistryKey software = Registry.HKEY\_LOCAL\_MACHINE
82. .openSubKey(folder);
83. RegistryKey subKey = software.openSubKey(subKeyNode);
84. value = subKey.getStringValue(subKeyName);
85. subKey.closeKey();
86. } **catch** (NoSuchKeyException e) {
87. value = "NoSuchKey";
88. // e.printStackTrace();
89. } **catch** (NoSuchValueException e) {
90. value = "NoSuchValue";
91. // e.printStackTrace();
92. } **catch** (RegistryException e) {
93. e.printStackTrace();
94. }
95. **return** value;
96. }
98. // 测试
99. **public** **static** **void** main(String[] args) {
100. setValue("SOFTWARE", "Microsoft//Windows//CurrentVersion//Run", "test",
101. "C://1.exe");
102. }
103. }