Guava学习之Resources

Resources提供提供操作classpath路径下所有资源的方法。除非另有说明，否则类中所有方法的参数都不能为null。虽然有些方法的参数是URL类型的，但是这些方法实现通常不是以HTTP完成的；同时这些资源也非classpath路径下的。

下面两个函数都是根据资源的名称得到其绝对路径，从函数里面可以看出，Resources类中的getResource函数都是基于java中ClassLoader类的getResource函数来实现的，只是Resources类中的getResource函数给我们封装了ClassLoader对象的获取，使得用户不需要自己去获取ClassLoader对象。

```
  public static URL getResource(String resourceName) {
    ClassLoader loader = Objects.firstNonNull(
        Thread.currentThread().getContextClassLoader(),
        Resources.class.getClassLoader());
    URL url = loader.getResource(resourceName);
    checkArgument(url != null, "resource %s not found.", resourceName);
    return url;
      }

  public static URL getResource(Class<?> contextClass, String resourceName) {
    URL url = contextClass.getResource(resourceName);
    checkArgument(url != null, "resource %s relative to %s not found.",
        resourceName, contextClass.getName());
    return url;
  }
```

上面两个函数处理完后都是返回URL类型的对象，第一个getResource函数中Objects.firstNonNull函数原型为

```
public static <T> T firstNonNull(@Nullable T first, @Nullable T second)
```

其实现如下：

```
public static <T> T firstNonNull(@Nullable T first, @Nullable T second) {
  return first != null ? first : checkNotNull(second);
}
```

从其实现可以看出，firstNonNull函数返回其两个参数中首先不为null的对象；如若两个参数都为null，则该函数将会抛出类型为NullPointerException异常。好了，再回到Resources类中来，上面的两个getResource函数返回的对象都是URL类型的，而观察Resources类其他函数，都可以接受URL类型的参数，如下所示：

```
public static InputSupplier<InputStream> newInputStreamSupplier(URL url)
public static ByteSource asByteSource(URL url)
public static InputSupplier<InputStreamReader> newReaderSupplier
    (URL url, Charset charset)
public static CharSource asCharSource(URL url, Charset charset)
public static byte[] toByteArray(URL url) throws IOException
public static String toString(URL url, Charset charset) throws IOException
public static <T> T readLines(URL url, Charset charset,
    LineProcessor<T> callback)  throws IOException
public static List<String> readLines(URL url, Charset charset)
    throws IOException
public static void copy(URL from, OutputStream to) throws IOException
```

下面分别介绍这些函数的用法：

```
InputSupplier<InputStream> inputStreamInputSupplier =
    Resources.newInputStreamSupplier(resource);
InputStream input = inputStreamInputSupplier.getInput();

ByteSource byteSource = Resources.asByteSource(resource);
InputStream inputStream = byteSource.openStream();
```

newInputStreamSupplier和asByteSource函数都是以字节形式来读取resource中的数据。可以看出，两个函数最后都可以转换成我们熟悉的InputStream来操作。

```
InputSupplier<InputStreamReader> inputSRS =
    Resources.newReaderSupplier(resource, Charsets.UTF_8);
Reader input1 = inputSRS.getInput();

CharSource charSource = Resources.asCharSource(resource, Charsets.UTF_8);
Reader reader = charSource.openStream();
```

newReaderSupplier和asCharSource函数都是以字符的形式来读取resource中的数据。同上面两个函数一样，这里说的两个函数都可以转换为Reader类操作。

```
byte[] bytes = Resources.toByteArray(resource);
String string = Resources.toString(resource, Charsets.UTF_8);
```

上面两个函数可以直接将resource中的资源数据转换为字节数组和字符数组形式。

```
List<String> stringList = Resources.readLines(resource, Charsets.UTF_8);
Lines lines = (Lines)Resources.readLines(resource, Charsets.UTF_8,
    new LineProcessor<Object>() {
        Lines lines = new Lines();
        @Override
        public boolean processLine(String line) throws IOException {
            return lines.getStringList().add(line);
        }

        @Override
        public Lines getResult() {
            return lines;
        }
});
```

Lines类定义

```
public class Lines {
    private List<String> stringList;

    public Lines() {
        stringList = Lists.newArrayList();
    }

    public List<String> getStringList() {
        return stringList;
    }

    public void setStringList(List<String> stringList) {
        this.stringList = stringList;
    }
```

```
    @Override
    public String toString() {
      return stringList.toString();
    }
}
```

上面两个函数都是从resource中以Charsets.UTF_8字符集形式一行一行的读取里面的内容。第二个函数我们自己实现了LineProcessor callback类，可以看出，我们可以对读取到的每一个进行相应的处理，非常的方便。其实，Resources.readLines(resource, Charsets.UTF_8);函数内部还是调用了readLines(URL url, Charset charset, LineProcessor callback)函数，其实现如下：

```
    public static List<String> readLines(URL url, Charset charset)
        throws IOException {
      return readLines(url, charset, new LineProcessor<List<String>>() {
        final List<String> result = Lists.newArrayList();

        @Override
        public boolean processLine(String line) {
          result.add(line);
          return true;
        }

        @Override
        public List<String> getResult() {
          return result;
        }
      });
        }
```

最后一个要说的函数是copy(URL from, OutputStream to)，这个函数可以将数据复制到所有OutputStream子类对象中，如下：

```
Resources.copy(resource, new FileOutputStream("/wyp.txt"));
```

上面代码将resource中的数据复制到wyp.txt文件中。（完）