详解Java高级特性之反射

主要介绍了Java高级特性之反射的相关知识，文中讲解非常细致，代码帮助大家更好的理解和学习，感兴趣的朋友可以了解下

定义

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取信息以及动态调用对象方法的功能称为java语言的反射机制。

用途

在日常的第三方应用开发过程中，经常会遇到某个类的某个成员变量、方法或是属性是私有的或是只对系统应用开放，这时候就可以利用Java的反射机制通过反射来获取所需的私有成员或是方法。当然，也不是所有的都适合反射，之前就遇到一个案例，通过反射得到的结果与预期不符。阅读源码发现，经过层层调用后在最终返回结果的地方对应用的权限进行了校验，对于没有权限的应用返回值是没有意义的缺省值，否则返回实际值起到保护用户的隐私目的。

反射机制的相关类

与Java反射相关的类如下：

类名	用途
Class类	代表类的实体，在运行的Java应用程序中表示类和接口
Field类	代表类的成员变量（成员变量也称为类的属性）
Method类	代表类的方法
Constructor类	代表类的构造方法

Class类

Class代表类的实体，在运行的Java应用程序中表示类和接口。在这个类中提供了很多有用的方法，这里对他们简单的分类介绍。

获得类相关的方法

方法	用途
asSubclass(Class<U> clazz)	把传递的类的对象转换成代表其子类的对象
Cast	把对象转换成代表类或是接口的对象
getClassLoader()	获得类的加载器
getClasses()	返回一个数组，数组中包含该类中所有公共类和接口类的对象
getDeclaredClasses()	返回一个数组，数组中包含该类中所有类和接口类的对象
forName(String className)	根据类名返回类的对象
getName()	获得类的完整路径名字
newInstance()	创建类的实例
getPackage()	获得类的包
getSimpleName()	获得类的名字
getSuperclass()	获得当前类继承的父类的名字
getInterfaces()	获得当前类实现的类或是接口

获得类中属性相关的方法

方法	用途
getAnnotation(Class<A> annotationClass)	返回该类中与参数类型匹配的公有注解对象
getAnnotations()	返回该类所有的公有注解对象
getDeclaredAnnotation(Class<A> annotationClass)	返回该类中与参数类型匹配的所有注解对象
getDeclaredAnnotations()	返回该类所有的注解对象

获得类中构造器相关的方法

方法	用途
getConstructor(Class...<?> parameterTypes)	获得该类中与参数类型匹配的公有构造方法
getConstructors()	获得该类的所有公有构造方法
getDeclaredConstructor(Class...<?> parameterTypes)	获得该类中与参数类型匹配的构造方法
getDeclaredConstructors()	获得该类所有构造方法

获得类中方法相关的方法

方法	用途
getMethod(String name, Class...<?> parameterTypes)	获得该类某个公有的方法
getMethods()	获得该类所有公有的方法
getDeclaredMethod(String name, Class...<?> parameterTypes)	获得该类某个方法
getDeclaredMethods()	获得该类所有方法

类中其他重要的方法

方法	用途
isAnnotation()	如果是注解类型则返回true
isAnnotationPresent(Class<? extends Annotation> annotationClass)	如果是指定类型注解类型则返回true
isAnonymousClass()	如果是匿名类则返回true
isArray()	如果是一个数组类则返回true
isEnum()	如果是枚举类则返回true
isInstance(Object obj)	如果obj是该类的实例则返回true
isInterface()	如果是接口类则返回true
isLocalClass()	如果是局部类则返回true
isMemberClass()	如果是内部类则返回true

Field类

Field代表类的成员变量（成员变量也称为类的属性）。

方法	用途
equals(Object obj)	属性与obj相等则返回true
get(Object obj)	获得obj中对应的属性值
set(Object obj, Object value)	设置obj中对应属性值

Method类

Method代表类的方法。

方法	用途
invoke(Object obj, Object... args)	传递object对象及参数调用该对象对应的方法

Constructor类

Constructor代表类的构造方法。

方法	用途
newInstance(Object... initargs)	根据传递的参数创建类的对象

示例

为了演示反射的使用，首先构造一个与书籍相关的model——Book.java，然后通过反射方法示例创建对象、反射私有构造方法、反射私有属性、反射私有方法，最后给出两个比较复杂的反射示例——获得当前ZenMode和关机Shutdown。

被反射类Book.java

```
public class Book{
  private final static String TAG = "BookTag";

  private String name;
  private String author;

  @Override
  public String toString() {
    return "Book{" +
        "name='" + name + '\'' +
        ", author='" + author + '\'' +
```

```
        '}';
  }

  public Book() {
  }

  private Book(String name, String author) {
    this.name = name;
    this.author = author;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getAuthor() {
    return author;
  }

  public void setAuthor(String author) {
    this.author = author;
  }

  private String declaredMethod(int index) {
    String string = null;
    switch (index) {
      case 0:
        string = "I am declaredMethod 1 !";
        break;
      case 1:
        string = "I am declaredMethod 2 !";
        break;
      default:
        string = "I am declaredMethod 1 !";
    }

    return string;
  }
}
```

反射逻辑封装在ReflectClass.java

```
public class ReflectClass {
  private final static String TAG = "peter.log.ReflectClass";

  // 创建对象
  public static void reflectNewInstance() {
    try {
      Class<?> classBook = Class.forName("com.android.peter.reflectdemo.Book");
      Object objectBook = classBook.newInstance();
      Book book = (Book) objectBook;
      book.setName("Android进阶之光");
      book.setAuthor("刘望舒");
      Log.d(TAG,"reflectNewInstance book = " + book.toString());
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }

  // 反射私有的构造方法
  public static void reflectPrivateConstructor() {
    try {
      Class<?> classBook = Class.forName("com.android.peter.reflectdemo.Book");
      Constructor<?> declaredConstructorBook = classBook.getDeclaredConstructor(String.class,String.class);
      declaredConstructorBook.setAccessible(true);
      Object objectBook = declaredConstructorBook.newInstance("Android开发艺术探索","任玉刚");
      Book book = (Book) objectBook;
      Log.d(TAG,"reflectPrivateConstructor book = " + book.toString());
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }

  // 反射私有属性
  public static void reflectPrivateField() {
    try {
      Class<?> classBook = Class.forName("com.android.peter.reflectdemo.Book");
```

```
    Object objectBook = classBook.newInstance();
    Field fieldTag = classBook.getDeclaredField("TAG");
    fieldTag.setAccessible(true);
    String tag = (String) fieldTag.get(objectBook);
    Log.d(TAG,"reflectPrivateField tag = " + tag);
  } catch (Exception ex) {
    ex.printStackTrace();
  }
}

// 反射私有方法
public static void reflectPrivateMethod() {
  try {
    Class<?> classBook = Class.forName("com.android.peter.reflectdemo.Book");
    Method methodBook = classBook.getDeclaredMethod("declaredMethod",int.class);
    methodBook.setAccessible(true);
    Object objectBook = classBook.newInstance();
    String string = (String) methodBook.invoke(objectBook,0);

    Log.d(TAG,"reflectPrivateMethod string = " + string);
  } catch (Exception ex) {
    ex.printStackTrace();
  }
}

// 获得系统Zenmode值
public static int getZenMode() {
  int zenMode = -1;
  try {
    Class<?> cServiceManager = Class.forName("android.os.ServiceManager");
    Method mGetService = cServiceManager.getMethod("getService", String.class);
    Object oNotificationManagerService = mGetService.invoke(null, Context.NOTIFICATION_SERVICE);
    Class<?> cINotificationManagerStub = Class.forName("android.app.INotificationManager$Stub");
    Method mAsInterface = cINotificationManagerStub.getMethod("asInterface",IBinder.class);
    Object oINotificationManager = mAsInterface.invoke(null,oNotificationManagerService);
    Method mGetZenMode = cINotificationManagerStub.getMethod("getZenMode");
    zenMode = (int) mGetZenMode.invoke(oINotificationManager);
  } catch (Exception ex) {
    ex.printStackTrace();
  }

  return zenMode;
}

// 关闭手机
public static void shutDown() {
  try {
    Class<?> cServiceManager = Class.forName("android.os.ServiceManager");
    Method mGetService = cServiceManager.getMethod("getService",String.class);
    Object oPowerManagerService = mGetService.invoke(null,Context.POWER_SERVICE);
    Class<?> cIPowerManagerStub = Class.forName("android.os.IPowerManager$Stub");
    Method mShutdown = cIPowerManagerStub.getMethod("shutdown",boolean.class,String.class,boolean.class);
    Method mAsInterface = cIPowerManagerStub.getMethod("asInterface",IBinder.class);
    Object oIPowerManager = mAsInterface.invoke(null,oPowerManagerService);
    mShutdown.invoke(oIPowerManager,true,null,true);

  } catch (Exception ex) {
    ex.printStackTrace();
  }
}

public static void shutdownOrReboot(final boolean shutdown, final boolean confirm) {
  try {
    Class<?> ServiceManager = Class.forName("android.os.ServiceManager");
    // 获得ServiceManager的getService方法
    Method getService = ServiceManager.getMethod("getService", java.lang.String.class);
    // 调用getService获取RemoteService
    Object oRemoteService = getService.invoke(null, Context.POWER_SERVICE);
    // 获得IPowerManager.Stub类
    Class<?> cStub = Class.forName("android.os.IPowerManager$Stub");
    // 获得asInterface方法
    Method asInterface = cStub.getMethod("asInterface", android.os.IBinder.class);
    // 调用asInterface方法获取IPowerManager对象
    Object oIPowerManager = asInterface.invoke(null, oRemoteService);
    if (shutdown) {
      // 获得shutdown()方法
      Method shutdownMethod = oIPowerManager.getClass().getMethod(
          "shutdown", boolean.class, String.class, boolean.class);
      // 调用shutdown()方法
      shutdownMethod.invoke(oIPowerManager, confirm, null, false);
    } else {
```

```
        // 获得reboot()方法
        Method rebootMethod = oIPowerManager.getClass().getMethod("reboot",
            boolean.class, String.class, boolean.class);
        // 调用reboot()方法
        rebootMethod.invoke(oIPowerManager, confirm, null, false);
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

调用相应反射逻辑方法

```
try {
      // 创建对象
      ReflectClass.reflectNewInstance();

      // 反射私有的构造方法
      ReflectClass.reflectPrivateConstructor();

      // 反射私有属性
      ReflectClass.reflectPrivateField();

      // 反射私有方法
      ReflectClass.reflectPrivateMethod();
    } catch (Exception ex) {
      ex.printStackTrace();
    }

    Log.d(TAG," zenmode = " + ReflectClass.getZenMode());
```

Log输出结果如下：

```
08-27 15:11:37.999 11987-11987/com.android.peter.reflectdemo D/peter.log.ReflectClass: reflectNewInstance
book = Book{name='Android进阶之光', author='刘望舒'}
08-27 15:11:38.000 11987-11987/com.android.peter.reflectdemo D/peter.log.ReflectClass:
reflectPrivateConstructor book = Book{name='Android开发艺术探索', author='任玉刚'}
08-27 15:11:38.000 11987-11987/com.android.peter.reflectdemo D/peter.log.ReflectClass: reflectPrivateField tag
= BookTag
08-27 15:11:38.000 11987-11987/com.android.peter.reflectdemo D/peter.log.ReflectClass: reflectPrivateMethod
string = I am declaredMethod 1！
08-27 15:11:38.004 11987-11987/com.android.peter.reflectdemo D/peter.log.ReflectDemo:  zenmode = 0
```

以上就是详解Java高级特性之反射的详细内容，更多关于JAVA高级特性反射的资料请关注我们其它相关文章！