Java 核心技术(高阶)

第三章 反射

第三节 反射应用

华东师范大学 陈良育

概要

- 数据库连接
- 数组扩充器
- 动态执行方法
- Json和Java对象互转
- Tomcat的Servlet对象创建
- MyBatis的OR/M
- Spring的Bean容器
- org.reflections包介绍

数据库连接

- JDBC
  - Connection，连接到各个不同数据库

```
//构建Java和数据库之间的桥梁介质
try{
      Class.forName("com.mysql.jdbc.Driver");
      //Class.forName(className, true, currentLoader)
      //通知类加载器加载此类的class文件
      System.out.println("注册驱动成功!");
}catch(ClassNotFoundException e1){
      System.out.println("注册驱动失败!");
      e1.printStackTrace();
      return;
}
```

```
//构建Java和数据库之间的桥梁: URL, 用户名, 密码
conn = DriverManager.getConnection(url, "root", "123456");
//DriverManager将会挑选加载合适的驱动类，并采用getConnection方法连接
```

数组扩充

- 给定一个数组(任意类型)，将其长度扩大一倍
  - Java的数组一旦创建，其长度是不再更改的
  - 新建一个大数组(相同类型)，然后将旧数组的内容拷贝过去

```
public static Object goodCopy(Object oldArray, int newLength) {
    // Array类型
    Class c = oldArray.getClass();

    // 获取数组中的单个元素类型
    Class componentType = c.getComponentType();

    // 旧数组长度
    int oldLength = Array.getLength(oldArray);

    // 新数组
    Object newArray = Array.newInstance(componentType, newLength);

    // 拷贝旧数据
    System.arraycopy(oldArray, 0, newArray, 0, oldLength);
    return newArray;
}
```

```
int[] a = { 1, 2, 3, 4, 5 };
a = (int[]) goodCopy(a, 10);
for (int i : a) {
    System.out.println(i);
}
```

动态执行方法

- 给定类名、方法名，即可执行
  - 加上定时器，即可做定时任务执行

```
class Worker {
    public static void hello() {
        System.out.println("Hello java!");
    }
}

class MyTask extends TimerTask {
    public void run() {
        try {
            Method m = Class.forName("Worker")
                    .getClass().getMethod("hello");
            m.invoke(null);// 静态方法可以不用new对象
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
Timer timer = new Timer();

Calendar now = Calendar.getInstance();
now.set(Calendar.SECOND,
        now.get(Calendar.SECOND) + 1);
Date runDate = now.getTime();

MyTask task2 = new MyTask();
timer.scheduleAtFixedRate(task2, runDate, 3000);
```

Json和Java对象互转

- Json： {“name":"Jo","email":"a@b.com"}

```
class Person
{
    private String name;
    private String email;
    public String getName() {
        return name;
    }
    public String getEmail() {
        return email;
    }
}
```

```
Gson gson = new Gson();
String s =  "{\"name\":\"Jo\""
        + ",\"email\":\"a@b.com\"}";
Person p = gson.fromJson(s, Person.class);

System.out.println(p.getName());
System.out.println(p.getEmail());
```

```
Field[] fields = raw.getDeclaredFields();
for (Field field : fields) {
  boolean serialize = excludeField(field, true);
  boolean deserialize = excludeField(field, false);
  if (!serialize && !deserialize) {
    continue;
  }
  accessor.makeAccessible(field);
```

```
Object fieldValue = typeAdapter.read(reader);
if (fieldValue != null || !isPrimitive) {
  field.set(value, fieldValue);
}
```

Tomcat的Servlet创建

```
<!-- web.xml -->
<!-- servlet definition -->

<servlet>
  <servlet-name>Init</servlet-name>
  <servlet-class>service.Init</servlet-class>
</servlet>
```

```
public Object newInstance(final String className, final ClassLoader classLoader)
        throws IllegalAccessException, NamingException, InvocationTargetException,
        InstantiationException, ClassNotFoundException, IllegalArgumentException,
        NoSuchMethodException, SecurityException {
    Class<?> clazz = classLoader.loadClass(className);
    return newInstance(clazz.getConstructor().newInstance(), clazz);
}
```

MyBatis的OR/M

- MyBatis: OR/M(Object-Relation Mapping)
  - 数据库表和Java的POJO/DAO类对应关系

```
public class Reflector {
  private final Class<?> type;
  private final String[] readablePropertyNames;
  private final String[] writablePropertyNames;
  private final Map<String, Invoker> setMethods = new HashMap<>();
  private final Map<String, Invoker> getMethods = new HashMap<>();
  private final Map<String, Class<?>> setTypes = new HashMap<>();
  private final Map<String, Class<?>> getTypes = new HashMap<>();
  private Constructor<?> defaultConstructor;

  private Map<String, String> caseInsensitivePropertyMap = new HashMap<>();

  public Reflector(Class<?> clazz) {
    type = clazz;
    addDefaultConstructor(clazz);
    addGetMethods(clazz);
    addSetMethods(clazz);
    addFields(clazz);
    readablePropertyNames = getMethods.keySet().toArray(new String[0]);
    writablePropertyNames = setMethods.keySet().toArray(new String[0]);
```

Spring Framework的Bean容器

- Spring Framework: Java EE的主要框架
  - IoC 的Bean容器(HashMap)

```
/**
 * Actually load bean definitions from the specified XML file.
 * @param inputSource the SAX InputSource to read from
 * @param resource the resource descriptor for the XML file
 * @return the number of bean definitions found
 * @throws BeanDefinitionStoreException in case of loading or parsing errors
 * @see #doLoadDocument
 * @see #registerBeanDefinitions
 */
protected int doLoadBeanDefinitions(InputSource inputSource, Resource resource)
    throws BeanDefinitionStoreException {

  try {
    Document doc = doLoadDocument(inputSource, resource);
    int count = registerBeanDefinitions(doc, resource);
    if (logger.isDebugEnabled()) {
      logger.debug("Loaded " + count + " bean definitions from " + resource);
    }
```

```
protected void invokeCustomInitMethod(String beanName, final Object bean,
    throws Throwable {

  String initMethodName = mbd.getInitMethodName();
  Method initMethod = (mbd.isNonPublicAccessAllowed() ?
      BeanUtils.findMethod(bean.getClass(), initMethodName) :
      ClassUtils.getMethodIfAvailable(bean.getClass(), initMethodName));

  if (System.getSecurityManager() != null) {
  }
  else {
    try {
      ReflectionUtils.makeAccessible(methodToInvoke);
      methodToInvoke.invoke(bean);
    }
    catch (InvocationTargetException ex) {
      throw ex.getTargetException();
    }
  }
```

org.reflections

- Reflection的增强工具包
  - https://github.com/ronmamo/reflections
  - Java runtime metadata analysis
    - 获取某类的所有子类型
    - 获取有特殊annotation的类型或者成员变量/方法
    - 根据正则表达式获取资源(类/成员变量/方法)
    - 根据组合条件查询相应的方法
    - ……

总结

- 了解Java反射的广泛用途
- 了解反射在诸多框架的底层作用

谢 谢！