OGNL是Object-Graph Navigation Language的缩写，它是一种功能强大的表达式语言（Expression Language，简称为EL），通过它简单一致的表达式语法，可以存取对象的任意属性，调用对象的方法，遍历整个对象的结构图，实现字段类型转化等功能。它使用相同的表达式去存取对象的属性。

**OGNL的三要素：**

一、表达式：

　　　　表达式（Expression）是整个OGNL的核心内容，所有的OGNL操作都是针对表达式解析后进行的。通过表达式来告诉OGNL操作到底要干些什么。因此，表达式其实是一个带有语法含义的字符串，整个字符串将规定操作的类型和内容。OGNL表达式支持大量的表达式，如“链式访问对象”、表达式计算、甚至还支持Lambda表达式。

二、Root对象：

　　　　OGNL的Root对象可以理解为OGNL的操作对象。当我们指定了一个表达式的时候，我们需要指定这个表达式针对的是哪个具体的对象。而这个具体的对象就是Root对象，这就意味着，如果有一个OGNL表达式，那么我们需要针对Root对象来进行OGNL表达式的计算并且返回结果。

三、上下文环境：

　　　　有个Root对象和表达式，我们就可以使用OGNL进行简单的操作了，如对Root对象的赋值与取值操作。但是，实际上在OGNL的内部，所有的操作都会在一个特定的数据环境中运行。这个数据环境就是上下文环境（Context）。OGNL的上下文环境是一个Map结构，称之为OgnlContext。Root对象也会被添加到上下文环境当中去。

**OGNL 的基本语法：**

　　1. 对Root对象的访问

OGNL使用的是一种链式的风格进行对象的访问。具体代码如下：

@Test

public void testOgnl()

{

User user = new User("rcx", "123");

Address address = new Address("110003", "沈阳市和平区");

user.setAddress(address);

try

{

System.out.println(Ognl.getValue("name", user));

System.out.println(Ognl.getValue("address", user));

System.out.println(Ognl.getValue("address.port", user));

//输出结果：

//rcx

//com.rcx.ognl.Address@dda25b

//110003

}

catch (OgnlException e)

{

e.printStackTrace();

}

}

　　2. 对上下文对象的访问

使用OGNL的时候如果不设置上下文对象，系统会自动创建一个上下文对象，如果传入的参数当中包含了上下文对象则会使用传入的上下文对象。当访问上下文环境当中的参数时候，需要在表达式前面加上'#'，表示了与访问Root对象的区别。具体代码如下：

@Test

public void testOgnl1()

{

User user = new User("rcx", "123");

Address address = new Address("110003", "沈阳市和平区");

user.setAddress(address);

Map<String, Object> context = new HashMap<String, Object>();

context.put("init", "hello");

context.put("user", user);

try

{

System.out.println(Ognl.getValue("#init", context, user));

System.out.println(Ognl.getValue("#user.name", context, user));

System.out.println(Ognl.getValue("name", context, user));

//输出结果：

//hello

//rcx

//rcx

}

catch (OgnlException e)

{

e.printStackTrace();

}

}

这段代码很好的区分了访问Root对象和访问上下文对象的区别。

　　3. 对静态变量的访问

在OGNL表达式当中也可以访问静态变量或者调用静态方法，格式如@[class]@[field/method()]。具体代码如下：

public class Constant

{

public final static String ONE = "one";

public static void get()

{}

public static String getString()

{

return "string";

}

}

@Test

public void testOgnl2()

{

try

{

Object object = Ognl.getValue("@com.rcx.ognl.Constant@ONE", null);

Object object1 = Ognl.getValue("@com.rcx.ognl.Constant@get()", null);

Object object2 = Ognl.getValue("@com.rcx.ognl.Constant@getString()", null);

System.out.println(object);

System.out.println(object1);

System.out.println(object2);

//one

//null 当返回值是void的时候输出的是null

//string

}

catch (OgnlException e)

{

e.printStackTrace();

}

}

　　4. 方法的调用

如果需要调用Root对象或者上下文对象当中的方法也可以使用.+方法的方式来调用。甚至可以传入参数。代码如下：

@Test

public void testOgnl3()

{

User user = new User();

Map<String, Object> context = new HashMap<String, Object>();

context.put("name", "rcx");

context.put("password", "password");

try

{

System.out.println(Ognl.getValue("getName()", context, user));

Ognl.getValue("setName(#name)", context, user);

System.out.println(Ognl.getValue("getName()", context, user));

//输出结果

//null

//rcx

}

catch (OgnlException e)

{

e.printStackTrace();

}

}

　　从代码可以看出来，赋值的时候可以选择上下文当中的元素进行给Root对象的name属性赋值。

　　5. 对数组和集合的访问

OGNL支持对数组按照数组下标的顺序进行访问。此方式也适用于对集合的访问，对于Map支持使用键进行访问。代码如下：

@Test

public void testOgnl4()

{

User user = new User();

Map<String, Object> context = new HashMap<String, Object>();

String[] strings = {"aa", "bb"};

ArrayList<String> list = new ArrayList<String>();

list.add("aa");

list.add("bb");

Map<String, String> map = new HashMap<String, String>();

map.put("key1", "value1");

map.put("key2", "value2");

context.put("list", list);

context.put("strings", strings);

context.put("map", map);

try

{

System.out.println(Ognl.getValue("#strings[0]", context, user));

System.out.println(Ognl.getValue("#list[0]", context, user));

System.out.println(Ognl.getValue("#list[0 + 1]", context, user));

System.out.println(Ognl.getValue("#map['key1']", context, user));

System.out.println(Ognl.getValue("#map['key' + '2']", context, user));

//输出如下：

//aa

//aa

//bb

//value1

//value2

}

catch (OgnlException e)

{

e.printStackTrace();

}

}

　从上面代码不仅看到了访问数组与集合的方式同时也可以看出来OGNL表达式当中支持操作符的简单运算。有如下所示：

　　2 + 4 //整数相加（同时也支持减法、乘法、除法、取余[ % / mod]、）

　　"hell" + "lo" //字符串相加

　　i++ //递增、递减

　　i == j //判断

　　var in list //是否在容器当中

　　6. 投影与选择

　　OGNL支持类似数据库当中的选择与投影功能。

　　投影：选出集合当中的相同属性组合成一个新的集合。语法为collection.{XXX}，XXX就是集合中每个元素的公共属性。

　　选择：选择就是选择出集合当中符合条件的元素组合成新的集合。语法为collection.{Y XXX}，其中Y是一个选择操作符，XXX是选择用的逻辑表达式。

　　　　选择操作符有3种：

　　　　　　? ：选择满足条件的所有元素

　　　　　　^：选择满足条件的第一个元素

　　　　　　$：选择满足条件的最后一个元素

示例代码如下：

@Test

public void testOgnl5()

{

Person p1 = new Person(1, "name1");

Person p2 = new Person(2, "name2");

Person p3 = new Person(3, "name3");

Person p4 = new Person(4, "name4");

Map<String, Object> context = new HashMap<String, Object>();

ArrayList<Person> list = new ArrayList<Person>();

list.add(p1);

list.add(p2);

list.add(p3);

list.add(p4);

context.put("list", list);

try

{

ArrayList<Integer> list2 = (ArrayList<Integer>) Ognl.getValue("#list.{id}", context,

list);

ArrayList<String> list3 = (ArrayList<String>) Ognl.getValue("#list.{id + '-' + name}",

context, list);

ArrayList<Person> list4 = (ArrayList<Person>) Ognl.getValue("#list.{? #this.id > 2}",

context, list);

ArrayList<Person> list5 = (ArrayList<Person>) Ognl.getValue("#list.{^ #this.id > 2}",

context, list);

ArrayList<Person> list6 = (ArrayList<Person>) Ognl.getValue("#list.{$ #this.id > 2}",

context, list);

System.out.println(list2);

System.out.println(list3);

System.out.println(list4);

System.out.println(list5);

System.out.println(list6);

//[1, 2, 3, 4]

//[1-name1, 2-name2, 3-name3, 4-name4]

//[Person [id=3, name=name3], Person [id=4, name=name4]]

//[Person [id=3, name=name3]]

//[Person [id=4, name=name4]]

//

}

catch (OgnlException e)

{

e.printStackTrace();

}

}

　　7. 创建对象

　　OGNL支持直接使用表达式来创建对象。主要有三种情况：

　　构造List对象：使用{},中间使用','进行分割如{"aa", "bb", "cc"}

　　构造Map对象：使用#{}，中间使用',进行分割键值对，键值对使用':'区分，如#{"key1" : "value1", "key2" : "value2"}

　　构造任意对象：直接使用已知的对象的构造方法进行构造。

示例代码如下：

@Test

public void testOgnl6()

{

try

{

Map<String, String> map = (Map<String, String>)Ognl.getValue("#{'key1':'value1'}", null);

System.out.println(map);

List<String> list = (List<String>)Ognl.getValue("{'key1','value1'}", null);

System.out.println(list);

Object object = Ognl.getValue("new java.lang.Object()", null);

System.out.println(object);

//{key1=value1}

//[key1, value1]

//java.lang.Object@dda25b

}

catch (OgnlException e)

{

e.printStackTrace();

}

}