

## | 学习目标

- \* 能够理解反射的概述
  - \* 运行时，万物皆对象：Class,Field,Constructor,Method
  - \* 优点：框架灵魂，灵活性，扩展性，减低耦合度，无需硬编码
  - \* 缺点：性能，安全，健壮性，都不太好
- \* 能够掌握常用反射的类
- \* 能够掌握内省机制
  - \* javabeanset ( writeMethod ) , get ( readMethod )
  - \* Introspector,BeanInfo,PropertyDescriptor
- \* 能够掌握BeanUtils
  - \* BeanUtils.populate(obj,args);
- \* 能够掌握数据库的元数据
  - \* DatabaseMetaData
  - \* ParameterMetaData
  - \* ResultSetMetaData
- \* 能够自定义JDBC的框架
  - \* I/O
  - \* 提高开发效率

---

## \* 国庆任务

- \* 总结Oracle
- \* Mysql
- \* 两套sql题
- \* 反射
- \* 连接池：C3p0 , Druid
- \* 能够理解反射的概述

## \* 反射的概述

\* JAVA反射机制是在运行状态中，对于任意一个实体类，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。

\* 反射这一概念最早由编程开发人员Smith在1982年提出

## \* 反射的优点

### \* 框架的灵魂

\* 反射机制是构建框架技术的基础所在，使用反射可以避免将代码写死在框架中。

\* 提高了程序灵活性和扩展性，降低模块的耦合性

\* 反射机制极大的提高了程序的灵活性和扩展性，降低模块的耦合性，提高自身的适应能力。

\* 无需提前硬编码目标类

\* 通过反射机制可以让程序创建和控制任何类的对象，无需提前硬编码目标类

## \* 反射的缺点

### \* 性能问题

\* Java虚拟机不能够对反射动态代码进行优化

\* 反射操作的效率要比正常操作效率低很多。

### \* 安全限制

\* 使用反射通常需要程序的运行没有安全方面的限制。

### \* 程序健壮性

\* 反射允许代码执行一些通常不被允许的操作

\* 反射代码破坏了Java程序结构的抽象性

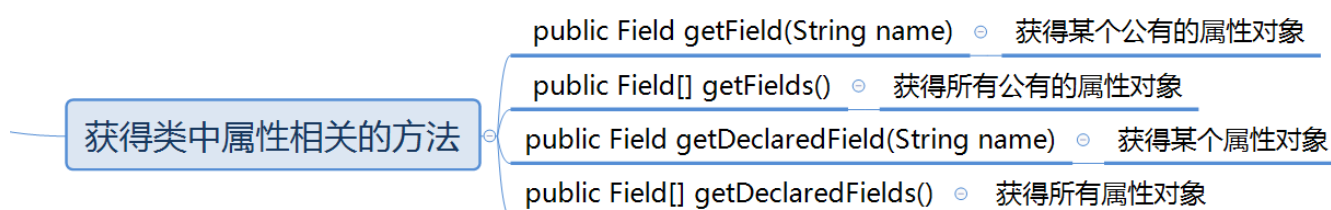
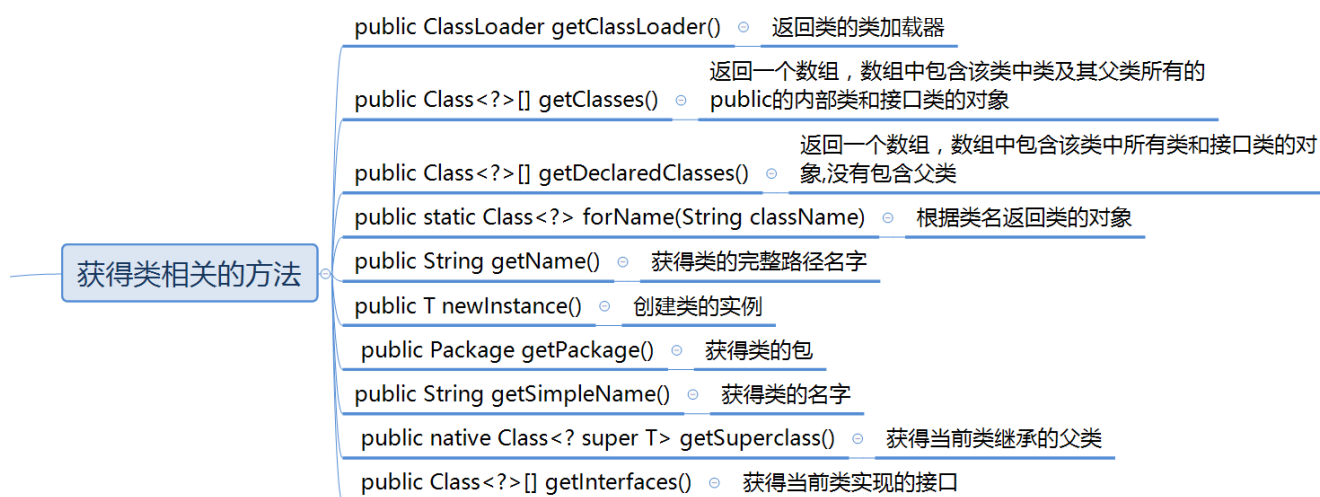
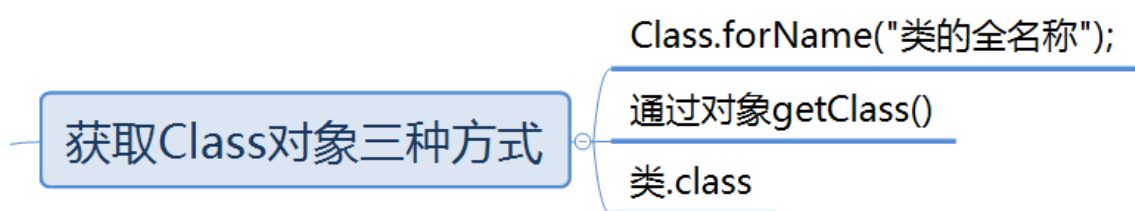
## \* 温馨提醒：

\* 反射机制的功能非常强大，但不能滥用。在能不使用反射完成时，尽量不要使用

## \* 能够掌握常用反射的类



## \* Class



## 获得类中构造器相关的方法

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

## 获得该类某个公有的方法

`public Method getMethod(String name, Class<?>... parameterTypes)`

获得该类所有公有的方法 `public Method[] getMethods()`

## 获得该类某个方法

`public Method getDeclaredMethod(String name, Class<?>... parameterTypes)`

获得该类所有方法 `public Method[] getDeclaredMethods()`

## 获得类中方法相关的方法

如果是匿名类则返回true `public boolean isAnonymousClass()`

如果是成员内部类则返回true `public boolean isMemberClass()`

如果是一个数组类则返回true `public native boolean isArray();`

如果是枚举类则返回true `public boolean isEnum()`

如果obj是该类的实例则返回true `public native boolean isInstance(Object obj);`

如果是接口类则返回true `public native boolean isInterface();`

确定指定的对象是否表示基本数据类型 `public native boolean isPrimitive();`

## 类中其他方法

如果该对象表示注解类型，则返回true。 `public boolean isAnnotation()`

如果是指定类型注解类型则返回true `public boolean isAnnotationPresent(Class<? extends Annotation> annotationClass)`

返回该类所有的公有注解对象 `public Annotation[] getAnnotations()`

返回该类中与参数类型匹配的所有注解对象 `public <A extends Annotation> A getDeclaredAnnotation(Class<A> annotationClass)`

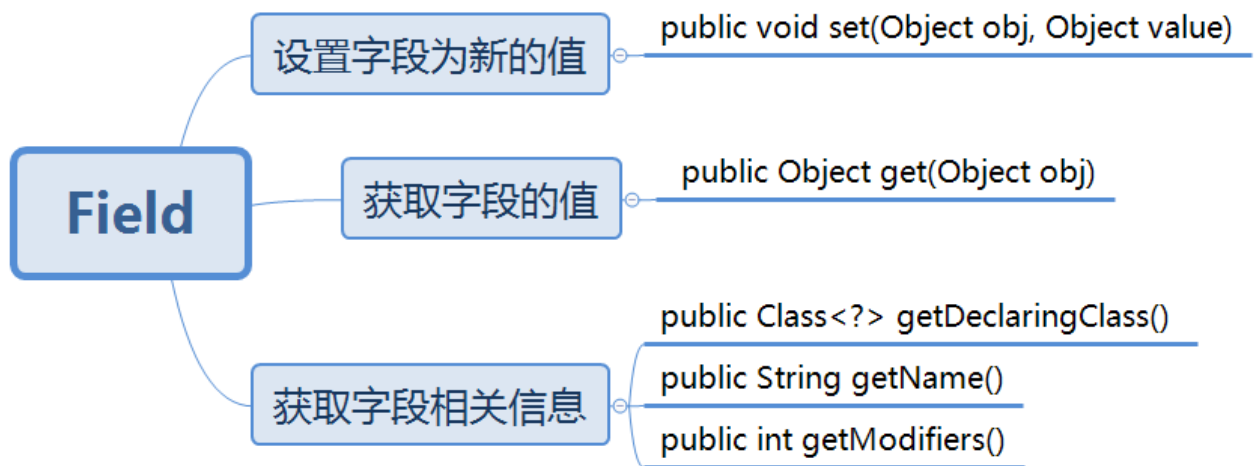
返回该类所有的注解对象 `public Annotation[] getDeclaredAnnotations()`

## 注解相关的方法（目前了解）

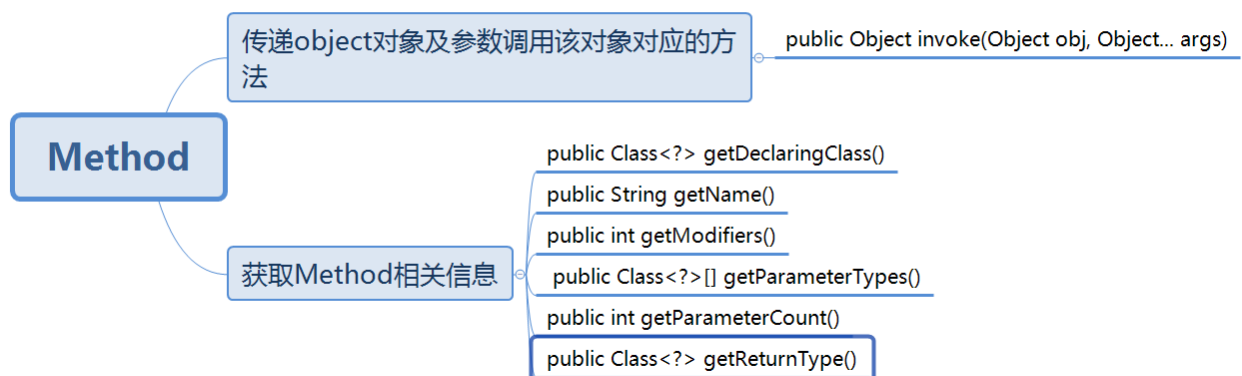
## \* Constructor



\* Field



\* Method



1 \* 准备测试User的对象

2 `public class User implements Comparable<User>{`

```
3     private int id;
4     private String name;
5     private String password;
6     public User() {
7         super();
8     }
9     public User(int id, String name, String password) {
10         super();
11         this.id = id;
12         this.name = name;
13         this.password = password;
14     }
15     //...
16 }
17 * 获取Class对象三种方式
18 @Test
19     public void test1() throws ClassNotFoundException {
20         // 获取Class对象三种方式
21         Class<?> clazz=User.class;
22         Class<?> clazz1=Class.forName("com.lg.bean.User");
23         User user=new User();
24         Class<?> clazz2=user.getClass();
25         System.out.println(clazz);
26         System.out.println(clazz==clazz1);
27         System.out.println(clazz1==clazz2);
28     }
29 * 结果
30 class com.lg.bean.User
31 true
32 true
33
34 * 获得类相关的方法
35 @Test
36     public void test2() throws ClassNotFoundException, InstantiationException,
37         Class<?> clazz=Class.forName("com.lg.bean.User");
38         System.out.println("返回类的类加载器:"+clazz.getClassLoader());
39         System.out.println("获得类的包:"+clazz.getPackage());
40         System.out.println("创建类的实例:"+clazz.newInstance());
41         System.out.println("获得类的完整路径名字:"+clazz.getName());
42         System.out.println("获得类的名字:"+clazz.getSimpleName());
```

```

43         System.out.println("获得当前类继承的父类:"+clazz.getSuperclass());
44         System.out.println("获得当前类实现的接口:"+Arrays.toString(clazz.getInter
45     }
46 * 结果
47 返回类的类加载器:sun.misc.Launcher$AppClassLoader@4e25154f
48 获得类的包:package com.lg.bean
49 创建类的实例:User [id=0, name=null, password=null]
50 获得类的完整路径名字:com.lg.bean.User
51 获得类的名字:User
52 获得当前类继承的父类:class java.lang.Object
53 获得当前类实现的接口:[interface java.lang.Comparable]
54
55 * 测试getClasses和getDeclaredClasses
56 * getClasses : 返回一个数组, 数组中包含该类中类及其父类所有的public的内部类和接口类
57 * getDeclaredClasses: 返回一个数组, 数组中包含该类中所有类和接口类的对象,没有包含父
58 public class A {
59     private class B{
60     }
61     public class C{
62     }
63 }
64 public class D extends A{
65     private class E{
66     }
67     public class F{
68     }
69 }
70 @Test
71 public void test3() throws ClassNotFoundException {
72     Class<?> clazz=Class.forName("com.lg.bean.A");
73     System.out.println("A:"+Arrays.toString(clazz.getClasses()));
74     System.out.println("A:"+Arrays.toString(clazz.getDeclaredClasses()));
75     clazz=Class.forName("com.lg.bean.D");
76     System.out.println("D:"+Arrays.toString(clazz.getClasses()));
77     System.out.println("D:"+Arrays.toString(clazz.getDeclaredClasses()));
78 }
79
80 结果:
81 A:[class com.lg.bean.A$C]
82 A:[class com.lg.bean.A$B, class com.lg.bean.A$C]

```

```

83 D:[class com.lg.bean.D$F, class com.lg.bean.A$C]
84 D:[class com.lg.bean.D$E, class com.lg.bean.D$F]
85
86 * 测试获得Field的方法
87 public class Person {
88     private int id;
89     public String name;
90     protected int age;
91     String password;
92 }
93 @Test
94 public void test4() throws ClassNotFoundException, InstantiationException,
95     IllegalAccessException, NoSuchFieldException, SecurityException {
96     Class<?> clazz=Class.forName("com.lg.bean.Person");
97     System.out.println("获得某个公有的属性对象:"+clazz.getField("name"));
98     System.out.println("获得所有公有的属性对象:"+Arrays.toString(clazz.getFields()));
99     System.out.println("获得某个属性对象:"+clazz.getDeclaredField("age"));
100    System.out.println("获得所有属性对象:"+Arrays.toString(clazz.getDeclaredFields()));
101 }
102
103 结果:
104 获得某个公有的属性对象:public java.lang.String com.lg.bean.Person.name
105 获得所有公有的属性对象:[public java.lang.String com.lg.bean.Person.name]
106 获得某个属性对象:protected int com.lg.bean.Person.age
107 获得所有属性对象:[private int com.lg.bean.Person.id,
108     public java.lang.String com.lg.bean.Person.name,
109     protected int com.lg.bean.Person.age,
110     java.lang.String com.lg.bean.Person.password]
111
112 * 测试获取构造器的方法
113 public class Person {
114     private int id;
115     public String name;
116     protected int age;
117     String password;
118
119     private Person(String name) {
120         this.name=name;
121     }
122     Person(){

```



```

123
124     }
125     public Person(String name,int age) {
126
127     }
128     protected Person(int id) {
129         this.id=id;
130     }
131 }
132 @Test
133 public void test5() throws ClassNotFoundException,
134     InstantiationException, NoSuchMethodException, SecurityException{
135     Class<?> clazz=Class.forName("com.lg.bean.Person");
136     System.out.println("获得该类中与参数类型匹配的公有构造方法:"+clazz.getConstruc
137     System.out.println("获得该类的所有公有构造方法:"+Arrays.toString(clazz.getCon
138     System.out.println("获得该类中与参数类型匹配的构造方法:"+clazz.getDeclaredCons
139     System.out.println("获得该类所有构造方法:"+Arrays.toString(clazz.getDeclared
140     }

```

141 结果:

```

142 获得该类中与参数类型匹配的公有构造方法:public com.lg.bean.Person(java.lang.String,
143 获得该类的所有公有构造方法:[public com.lg.bean.Person(java.lang.String,int)]
144 获得该类中与参数类型匹配的构造方法:private com.lg.bean.Person(java.lang.String)
145 获得该类所有构造方法:[protected com.lg.bean.Person(int),
146     public com.lg.bean.Person(java.lang.String,int),
147     com.lg.bean.Person(), \
148     private com.lg.bean.Person(java.lang.String)]
149

```

150 \* 测试获取方法

```

151 public class Person {
152     private int id;
153     public String name;
154     protected int age;
155     String password;
156
157     private Person(String name) {
158         this.name=name;
159     }
160     Person(){
161
162     }

```

```

163     public Person(String name,int age) {
164
165     }
166     protected Person(int id) {
167         this.id=id;
168     }
169     private void say1(String msg) {
170
171     }
172     boolean isOk(boolean isOk) {
173         return isOk;
174     }
175     protected String say2(String msg) {
176         return msg;
177     }
178     public String say3(String msg,int age) {
179         return msg+age;
180     }
181     private static void say12(String msg) {
182
183     }
184     static boolean isOk1(boolean isOk) {
185         return isOk;
186     }
187     protected static String say21(String msg) {
188         return msg;
189     }
190     public static String say31(String msg,int age) {
191         return msg+age;
192     }
193 }
194
195 @Test
196     public void test6() throws ClassNotFoundException,
197         InstantiationException, NoSuchMethodException, SecurityException{
198         Class<?> clazz=Class.forName("com.lg.bean.Person");
199         System.out.println("获得该类某个公有的方法:"+clazz.getMethod("say3",String.class));
200         System.out.println("获得该类所有公有的方法:"+Arrays.toString(clazz.getMethods()));
201         System.out.println("获得该类某个方法:"+clazz.getDeclaredMethod("say1", String.class));
202         System.out.println("获得该类所有方法:"+Arrays.toString(clazz.getDeclaredMethods()));

```

```

203     }
204
205 结果:
206 获得该类某个公有的方法:public java.lang.String com.lg.bean.Person.say3(java.lang.
207 获得该类所有公有的方法:[public java.lang.String com.lg.bean.Person.say3(java.lang
208         public static java.lang.String com.lg.bean.Person.say31(java.lang.
209         public final void java.lang.Object.wait() throws java.lang.Interru
210         public final void java.lang.Object.wait(long,int) throws java.lang
211         public final native void java.lang.Object.wait(long) throws java.l
212         public boolean java.lang.Object.equals(java.lang.Object),
213         public java.lang.String java.lang.Object.toString(),
214         public native int java.lang.Object.hashCode(),
215         public final native java.lang.Class java.lang.Object.getClass(),
216         public final native void java.lang.Object.notify(),
217         public final native void java.lang.Object.notifyAll()]
218 获得该类某个方法:private void com.lg.bean.Person.say1(java.lang.String)
219 获得该类所有方法:[private void com.lg.bean.Person.say1(java.lang.String),
220         public java.lang.String com.lg.bean.Person.say3(java.lang.String,int
221         boolean com.lg.bean.Person.isOk(boolean),
222         protected java.lang.String com.lg.bean.Person.say2(java.lang.String)
223         private static void com.lg.bean.Person.say12(java.lang.String),
224         static boolean com.lg.bean.Person.isOk1(boolean),
225         public static java.lang.String com.lg.bean.Person.say31(java.lang.St
226         protected static java.lang.String com.lg.bean.Person.say21(java.lang
227
228
229 * 构造器的测试
230 @Test
231     @Test
232     public void test8() throws ClassNotFoundException,
233             InstantiationException,
234             IllegalAccessException,
235             NoSuchFieldException, \
236             SecurityException{
237         Class<?> clazz=Class.forName("com.lg.bean.User");
238         // 无参数构造
239         Object obj= clazz.newInstance();// 通常使用这种, 很多框架都使用这种,
240                                     // 所以JavaBean有个要求要提供无参数的构造
241         System.out.println(obj);
242         Field id = clazz.getDeclaredField("id");

```

```

243     Field name = clazz.getDeclaredField("name");
244     Field password = clazz.getDeclaredField("password");
245     // java.lang.IllegalAccessException:
246     // Class com.lg.test.Test1 can not access a member of class
247     // com.lg.bean.User with modifiers "private"
248     // 暴力反射
249     id.setAccessible(true);
250     name.setAccessible(true);
251     password.setAccessible(true);
252     id.set(obj, 18);
253     name.set(obj, "xiaohei");
254     password.set(obj, "123");
255     System.out.println(id.get(obj));
256     System.out.println(name.get(obj));
257     System.out.println(password.get(obj));
258     System.out.println(obj);
259 }
260 结果:
261 User [id=0, name=null, password=null]
262 18
263 xiaohei
264 123
265 User [id=18, name=xiaohei, password=123]
266
267
268 * 测试方法
269 @Test
270     public void test9() throws ClassNotFoundException, InstantiationException,
271         Class<?> clazz=Class.forName("com.lg.bean.User");
272         // 无参数构造
273         Object obj= clazz.newInstance();// 通常使用这种, 很多框架都使用这种,
274         // 所以JavaBean有个要求要提供无参数的构造
275         System.out.println(obj);
276         Method setIdMethod = clazz.getDeclaredMethod("setId", int.class);
277         Method getIdMethod = clazz.getDeclaredMethod("getId");
278         Method setNameMethod = clazz.getDeclaredMethod("setName", String.class);
279         Method getNameMethod = clazz.getDeclaredMethod("getName");
280         Method setPasswordMethod = clazz.getDeclaredMethod("setPassword", String.class);
281         Method getPasswordMethod = clazz.getDeclaredMethod("getPassword");
282         setIdMethod.invoke(obj, 18);

```

```

283     setNameMethod.invoke(obj, "xiaohei");
284     setPasswordMethod.invoke(obj, "123");
285     System.out.println(getIdMethod.invoke(obj));
286     System.out.println(getNameMethod.invoke(obj));
287     System.out.println(getPasswordMethod.invoke(obj));
288     System.out.println(obj);
289 }
290 结果:
291 User [id=0, name=null, password=null]
292 18
293 xiaohei
294 123
295 User [id=18, name=xiaohei, password=123]
296
297 * 通用型的编写
298 @Test
299     public void test10() throws Exception {
300         Class<?> clazz = Class.forName("com.lg.bean.User");
301         // 调用无参数构造
302         Object obj = clazz.newInstance();
303         Map<String,Object> map=new HashMap<String,Object>();
304         map.put("id", 18);
305         map.put("name", "xiaohei");
306         map.put("password","123456");
307         Field[] fields = clazz.getDeclaredFields();
308         for (int i = 0; i < fields.length; i++) {
309             String fieldName = fields[i].getName();
310             Object arg = map.get(fieldName);
311             String methodName="set"+fieldName.substring(0,1).toUpperCase()+fie
312             Class<?> type=fields[i].getType();
313             Method method = clazz.getMethod(methodName,type);
314             method.invoke(obj, arg);
315         }
316         System.out.println(obj);
317     }
318 * 结果
319 User [id=0, name=null, password=null]
320 18
321 xiaohei
322 123

```

```
323 User [id=18, name=xiaohei, password=123]
```

```
324
```

```
325
```

## \* 能够掌握内省机制

### \* 内省概述

\* 内省(Introspector)是Java语言对JavaBean类属性、方法的一种缺省处理方法

### \* 内省常见的API



```
1 * 测试
2 @Test
3 public void test9() throws Exception {
4     Class<?> clazz = Class.forName("com.lg.bean.User");
5     // 调用无参数构造
6     Object obj = clazz.newInstance();
7     Map<String, Object> map = new HashMap<String, Object>();
8     map.put("id", 18);
9     map.put("name", "xiaohei");
10    map.put("password", "123456");
11    BeanInfo beanInfo = Introspector.getBeanInfo(clazz);
12    PropertyDescriptor[] propertyDescriptors = beanInfo.getPropertyDescriptors();
13    for (PropertyDescriptor pd : propertyDescriptors) {
14        Object arg = map.get(pd.getName());
15        if (arg == null) {
```

```

16         continue;
17     }
18     Method writeMethod = pd.getWriteMethod();
19     writeMethod.invoke(obj, arg);
20 }
21 System.out.println(obj);
22 }
23
24 * 结果
25 User [id=18, name=xiaohei, password=123]

```

### \* 能够掌握BeanUtils

\* 操作内省API比较繁琐，Apache组织开发了一套用于操作JavaBean的API——  
beanutils，简化程序代码的编写

```

1 * BeanUtils开发
2 * 导入jar
3 * commons-beanutils-1.9.3.jar
4 * commons-logging-1.2-sources.jar
5 * 代码
6 @Test
7     public void test10() throws Exception {
8         Class<?> clazz = Class.forName("com.lg.bean.User");
9         // 调用无参数构造
10        Object obj = clazz.newInstance();
11        Map<String,Object> map=new HashMap<String,Object>();
12        map.put("id", 18);
13        map.put("name", "xiaohei");
14        map.put("password", "123456");
15        BeanUtils.populate(obj, map);
16        System.out.println(obj);
17    }
18 * 结果
19 User [id=18, name=xiaohei, password=123456]

```

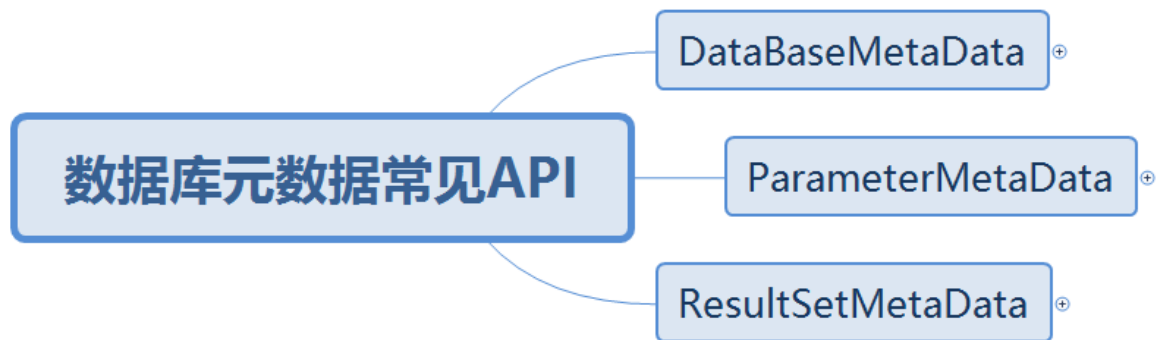
### \* 能够掌握数据库的元数据

#### \* 元数据的概述

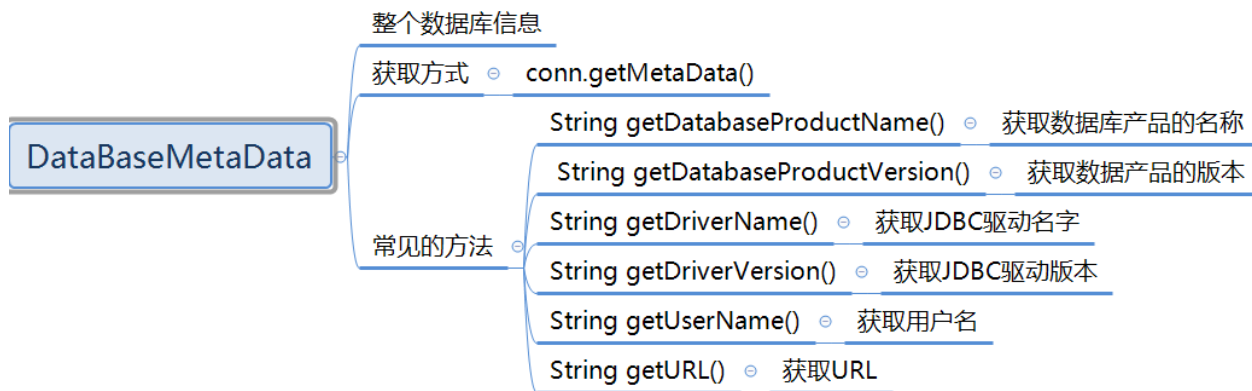
\* 元数据(MetaData)，即定义数据的数据。

\* 打个比方，就好像我们要想搜索一首歌(歌本身是数据)，而我们可以通过歌名，作者，专辑等信息来搜索，那么这些歌名，作者，专辑等等就是这首歌的元数据

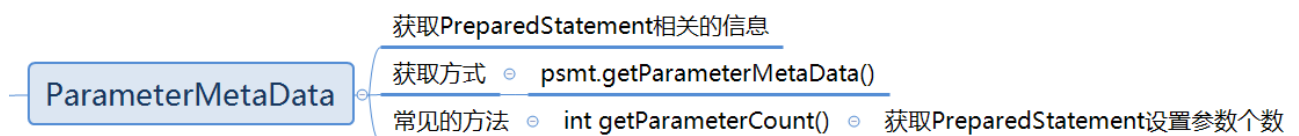
\* 数据库元数据常见的API



\* DataBaseMetaData

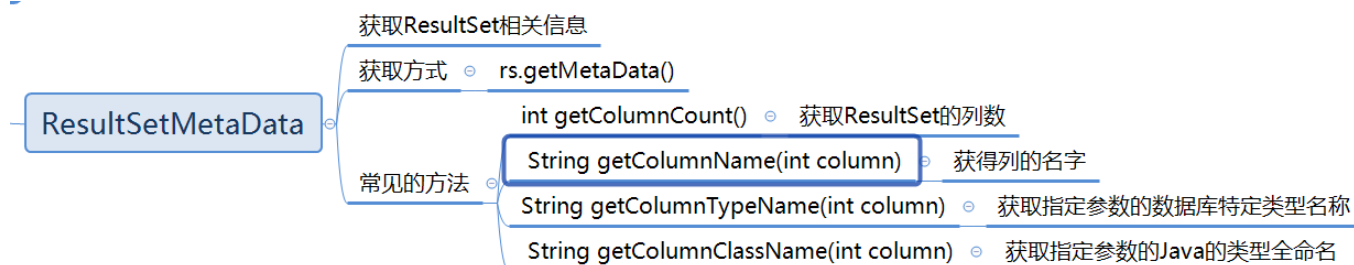


\* ParameterMetaData



\* ResultSetMetaData





```
1 * 测试DataBaseMetaData
2 @Test
3     public void test1() throws SQLException {
4         Connection conn = ConnectionUtils.getConn();
5         DatabaseMetaData metaData = conn.getMetaData();
6         String productName=metaData.getDatabaseProductName();
7         String productVersion=metaData.getDatabaseProductVersion();
8         String driverName=metaData.getDriverName();
9         String driverVersion=metaData.getDriverVersion();
10        String url=metaData.getURL();
11        String username=metaData.getUserName();
12        System.out.println(productName);
13        System.out.println(productVersion);
14        System.out.println(driverName);
15        System.out.println(driverVersion);
16        System.out.println(url);
17        System.out.println(username);
18        ConnectionUtils.close(conn, null,null);
19    }
20 * 结果
21 Oracle
22 Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
23 With the Partitioning, OLAP, Data Mining and Real Application Testing options
24 Oracle JDBC driver
25 11.2.0.1.0
26 jdbc:oracle:thin:@192.168.1.121:1521/orcl
27 SCOTT
28
29 * 测试ParameterMetaData
30 @Test
```

```

31 public void test2() throws SQLException {
32     Connection conn = ConnectionUtils.getConn();
33     // 1 获得ParameterMetaData
34     String sql = "select * from t_user where id=?";
35     PreparedStatement psmt = conn.prepareStatement(sql);
36     psmt.setInt(1, 5);
37     ParameterMetaData parameterMetaData = psmt.getParameterMetaData();
38     int parameterCount = parameterMetaData.getParameterCount();
39     System.out.println("参数的个数:"+parameterCount);
40     ConnectionUtils.close(conn, psmt, null);
41 }

```

结果:

参数的个数:1

\* 测试ResultSetMetaData

@Test

```

47 public void test3() throws SQLException {
48     Connection conn = ConnectionUtils.getConn();
49     String sql = "select * from t_user where id=?";
50     PreparedStatement psmt = conn.prepareStatement(sql);
51     psmt.setInt(1, 5);
52     ResultSet rs = psmt.executeQuery();
53     ResultSetMetaData metaData = rs.getMetaData();
54     int columnCount = metaData.getColumnCount();
55     System.out.println("columnCount:" + columnCount);
56     for (int i = 0; i < columnCount; i++) {
57         String columnName = metaData.getColumnName(i + 1);
58         String columnTypeName = metaData.getColumnTypeName(i + 1);
59         System.out.println("columnName:" + columnName);
60         System.out.println("columnTypeName:" + columnTypeName);
61     }
62     ConnectionUtils.close(conn, psmt, rs);
63 }

```

结果:

columnCount:3

columnName:ID

columnTypeName:NUMBER

columnName:NAME

columnTypeName:VARCHAR2

```
71 columnName:PASSWORD
72 columnTypeName:VARCHAR2
73
```

\* 能够自定义JDBC的框架

```
1 * Student
2 public class Student {
3     private String sid;
4     private String sname;
5     private int age;
6     private String gender;
7 }
8 * 观察StudentDaoImpl
9 public class StudentDaoImpl implements StudentDao {
10
11     @Override
12     public boolean add(Student student) {
13         if (student == null) {
14             throw new IllegalArgumentException("student is null");
15         }
16         Connection con = ConnectionUtils.getConn();
17         PreparedStatement st = null;
18         int result = 0;
19         try {
20             String sql = "insert into stu(sid,sname,age,gender) values(?,?,?,?)";
21             st = con.prepareStatement(sql);
22             st.setString(1, student.getSid());
23             st.setString(2, student.getSname());
24             st.setInt(3, student.getAge());
25             st.setString(4, student.getGender());
26             result = st.executeUpdate();
27         } catch (SQLException e) {
28             e.printStackTrace();
29         } finally {
30             ConnectionUtils.close(con, st, null);
31         }
32     }
33 }
```

```

32         return result > 0;
33     }
34
35     @Override
36     public boolean update(String sid, String name) {
37         Connection con = ConnectionUtils.getConnection();
38         PreparedStatement st = null;
39         int result = 0;
40         try {
41             String sql = "update stu set sname=? where sid=?";
42             st = con.prepareStatement(sql);
43             st.setString(1, name);
44             st.setString(2, sid);
45             result = st.executeUpdate();
46         } catch (SQLException e) {
47             e.printStackTrace();
48         } finally {
49             ConnectionUtils.close(con, st, null);
50         }
51         return result > 0;
52     }
53
54     @Override
55     public boolean delete(String sid) {
56         Connection con = ConnectionUtils.getConnection();
57         PreparedStatement st = null;
58         int result = 0;
59         try {
60             String sql = "delete from stu where sid=?";
61             st = con.prepareStatement(sql);
62             st.setString(1, sid);
63             result = st.executeUpdate();
64         } catch (SQLException e) {
65             e.printStackTrace();
66         } finally {
67             ConnectionUtils.close(con, st, null);
68         }
69         return result > 0;
70     }
71

```

```

72  @Override
73  public Student queryById(String id) {
74      String sql = "select sid,sname,age,gender from stu where sid=?";
75      Connection con = ConnectionUtils.getConnection();
76      PreparedStatement st = null;
77      ResultSet rs = null;
78      Student student = null;
79      try {
80          st = con.prepareStatement(sql);
81          st.setString(1, id);
82          rs = st.executeQuery();
83          if (rs.next()) {
84              String sid = rs.getString("sid");
85              String sname = rs.getString("sname");
86              String gender = rs.getString("gender");
87              try {
88                  int age = Integer.parseInt(rs.getString("age"));
89                  student = new Student(sid, sname, age, gender);
90              } catch (NumberFormatException e) {
91                  student = new Student(sid, sname, 0, gender);
92              }
93          }
94      } catch (SQLException e) {
95          e.printStackTrace();
96      } finally {
97          ConnectionUtils.close(con, st, rs);
98      }
99
100     return student;
101 }
102
103 @Override
104 public List<Student> queryAllStudents() {
105     String sql = "select sid,sname,age,gender from stu ";
106     Connection con = ConnectionUtils.getConnection();
107     PreparedStatement st = null;
108     ResultSet rs = null;
109     List<Student> students = new ArrayList<Student>();
110     try {
111         st = con.prepareStatement(sql);

```

```

112         rs = st.executeQuery();
113         while (rs.next()) {
114             String sid = rs.getString("sid");
115             String sname = rs.getString("sname");
116             String gender = rs.getString("gender");
117             try {
118                 int age = Integer.parseInt(rs.getString("age"));
119                 students.add(new Student(sid, sname, age, gender));
120             } catch (NumberFormatException e) {
121                 students.add(new Student(sid, sname, 0, gender));
122             }
123         }
124     } catch (SQLException e) {
125         e.printStackTrace();
126     } finally {
127         ConnectionUtils.close(con, st, rs);
128     }
129     return students;
130 }
131
132 // offset: row:多少行
133 @Override
134 public List<Student> queryStudents(int offset, int row) {
135     String sql = "select sid,sname,age,gender from (select rownum rn,sid,sr";
136     Connection con = ConnectionUtils.getConn();
137     PreparedStatement st = null;
138     ResultSet rs = null;
139     List<Student> students = new ArrayList<Student>();
140     try {
141         st = con.prepareStatement(sql);
142         st.setInt(1, row);
143         st.setInt(2, offset);
144         rs = st.executeQuery();
145         while (rs.next()) {
146             String sid = rs.getString("sid");
147             String sname = rs.getString("sname");
148             String gender = rs.getString("gender");
149             try {
150                 int age = Integer.parseInt(rs.getString("age"));
151                 students.add(new Student(sid, sname, age, gender));

```

```

152         } catch (NumberFormatException e) {
153             students.add(new Student(sid, sname, 0, gender));
154         }
155     }
156     } catch (SQLException e) {
157         e.printStackTrace();
158     } finally {
159         ConnectionUtils.close(con, st, rs);
160     }
161     return students;
162 }
163
164 }
165

```

166 \* 总结增删改

167 \* 一样地方获取连接, 释放资源, 获得PreparedStatement, 执行executeUpdate  
168 \* 不一样的地方, sql语句不一样, 设置参数也不一样

169 \* 总结查询单个

170 \* 一样地方获取连接, 释放资源, 获得PreparedStatement, executeQuery  
171 \* 不一样的地方, sql语句不一样, 设置参数也不一样, 返回值不一样

172 \* 总结查询列表

173 \* 一样地方获取连接, 释放资源, 获得PreparedStatement, executeQuery  
174 \* 不一样的地方, sql语句不一样, 设置参数也不一样, 返回值列表不一样

175

176 \* JdbcUtils的编写

```

177 public class JdbcUtils {
178
179     /**
180      * @param sql
181      * @param args
182      * @return
183      * CUD
184      */
185     public static boolean update(String sql, Object... args) {
186         Connection conn=ConnectionUtils.getConn();
187         PreparedStatement psmt=null;
188         int result=-1;
189         try {
190             psmt=conn.prepareStatement(sql);
191             int paramCount=psmt.getParameterMetaData().getParameterCount();

```

```

192         if(paramCount!=args.length) {
193             throw new IllegalArgumentException("expected is "+paramCount+",
194         }
195         for (int i = 0; i < args.length; i++) {
196             psmt.setObject(i+1, args[i]);
197         }
198         result=psmt.executeUpdate();
199     } catch (SQLException e) {
200         e.printStackTrace();
201     }finally {
202         ConnectionUtils.close(conn, psmt, null);
203     }
204     return result>0;
205 }
206
207 public static Object query(String sql,ResultSetHandler handler, Object... args) {
208     Connection conn=ConnectionUtils.getConnection();
209     PreparedStatement psmt=null;
210     ResultSet rs=null;
211     try {
212         psmt=conn.prepareStatement(sql);
213         int paramCount=psmt.getParameterMetaData().getParameterCount();
214         if(paramCount!=args.length) {
215             throw new IllegalArgumentException("expected is "+paramCount+",
216         }
217         for (int i = 0; i < args.length; i++) {
218             psmt.setObject(i+1, args[i]);
219         }
220         rs = psmt.executeQuery();
221         // 如何处理这个结果：有可能是Bean, List<Bean>,Map,List<Map>,1个值(
222         // 不能在处理这些结果，而是交给用户自己决定，
223         // 框架决定不了，此时可以定接口，让调用者实现
224         if(handler==null) {
225             throw new IllegalArgumentException("handler is null");
226         }
227         Object result=handler.handle(rs);
228         return result;
229     } catch (SQLException e) {
230         e.printStackTrace();
231     }finally {

```



```

232         ConnectionUtils.close(conn, psmt, rs);
233     }
234     return null;
235 }
236 }
237 * 定义ResultSetHandler
238 public interface ResultSetHandler {
239     Object handle(ResultSet rs);
240 }
241 * BeanHandler编写
242 public class BeanHandler implements ResultSetHandler {
243     private Class<?> clazz;
244
245     public BeanHandler(Class<?> clazz) {
246         this.clazz = clazz;
247     }
248
249     @Override
250     public Object handle(ResultSet rs) {
251         Object obj = null;
252         try {
253             if (rs.next()) {
254                 obj = this.clazz.newInstance();
255                 Map<String, Object> properties = new HashMap<String, Object>();
256                 ResultSetMetaData metaData = rs.getMetaData();
257                 int columnCount = metaData.getColumnCount();
258                 for (int i = 0; i < columnCount; i++) {
259                     String columnName = metaData洗getColumnName(i + 1);
260                     Object param = rs.getObject(i + 1);
261                     properties.put(columnName.toLowerCase(), param);
262                 }
263                 BeanUtils.populate(obj, properties);
264             }
265         } catch (Exception e) {
266             e.printStackTrace();
267         }
268         return obj;
269     }
270
271 }

```

```

272 * ListBeanHandler 编写
273 public class ListBeanHandler implements ResultSetHandler {
274     private Class<?> clazz;
275
276     public ListBeanHandler(Class<?> clazz) {
277         this.clazz = clazz;
278     }
279
280     @Override
281     public Object handle(ResultSet rs) {
282         List<Object> objs = new ArrayList<Object>();
283         try {
284             while (rs.next()) {
285                 Object obj = this.clazz.newInstance();
286                 Map<String, Object> properties = new HashMap<String, Object>();
287                 ResultSetMetaData metaData = rs.getMetaData();
288                 int columnCount = metaData.getColumnCount();
289                 for (int i = 0; i < columnCount; i++) {
290                     String columnName = metaData洗getColumnName(i + 1);
291                     Object param = rs.getObject(i + 1);
292                     properties.put(columnName.toLowerCase(), param);
293                 }
294                 BeanUtils.populate(obj, properties);
295                 objs.add(obj);
296             }
297         } catch (Exception e) {
298             e.printStackTrace();
299         }
300         return objs;
301     }
302 }
303
304 * copy StudentDaoImpl修改成StudentDaoImpl2
305 public class StudentDaoImpl2 implements StudentDao {
306
307     @Override
308     public boolean add(Student student) {
309         if (student == null) {
310             throw new IllegalArgumentException("student is null");
311         }

```

```
312     String sql = "insert into stu(sid,sname,age,gender) values(?,?,?,?)";
313     Object[] args = { student.getSid(), student.getSname(), student.getAge(), student.getGender() };
314     boolean result = JdbcUtils.update(sql, args);
315     return result;
316 }
317
318 @Override
319 public boolean update(String sid, String name) {
320     String sql = "update stu set sname=? where sid=?";
321     Object[] args = { name, sid };
322     boolean result = JdbcUtils.update(sql, args);
323     return result;
324 }
325
326 @Override
327 public boolean delete(String sid) {
328     String sql = "delete from stu where sid=?";
329     Object[] args = { sid };
330     boolean result = JdbcUtils.update(sql, args);
331     return result;
332 }
333
334 @Override
335 public Student queryById(String id) {
336     String sql = "select sid,sname,age,gender from stu where sid=?";
337     Object[] args = { id };
338     Student student=(Student) JdbcUtils.query(sql, new BeanHandler<Student>(Student.class));
339     return student;
340 }
341
342 @Override
343 public List<Student> queryAllStudents() {
344     String sql = "select sid,sname,age,gender from stu ";
345     @SuppressWarnings("unchecked")
346     List<Student> students=(List<Student>) JdbcUtils.query(sql, new ListBeanHandler<Student>());
347     return students;
348 }
349
350 // offset: row:多少行
351 @Override
```

```
352     public List<Student> queryStudents(int offset, int row) {
353         String sql = "select sid,sname,age,gender from (select rownum rn,sid,sr
354         Object[] args = {row,offset };
355         @SuppressWarnings("unchecked")
356         List<Student> students=(List<Student>) JdbcUtils.query(sql, new ListBea
357         return students;
358     }
359 }
360
361 * 测试
362 * studentDao=new StudentDaoImpl2();
363
```