* 学习目标

* 能够理解反射的概述

  * 运行时，万物皆对象：Class,Field,Construcor,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

    * javabean：set（writeMethod），get（readMethdod）

    * InstroSpector,BeanInfo,PropertyDecriptor

* 能够掌握BeanUtils

    * BeanUtils.polulate(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

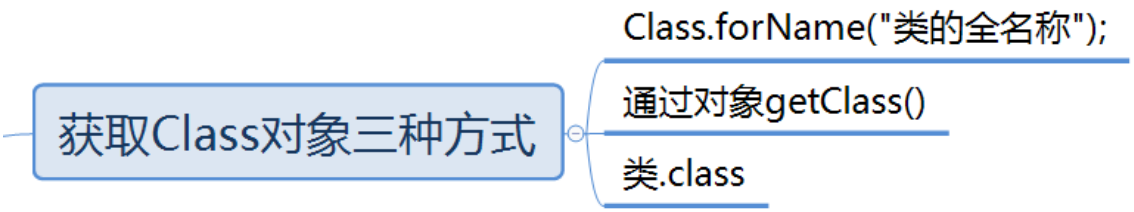
* 反射代码破坏了Java程序结构的抽象性

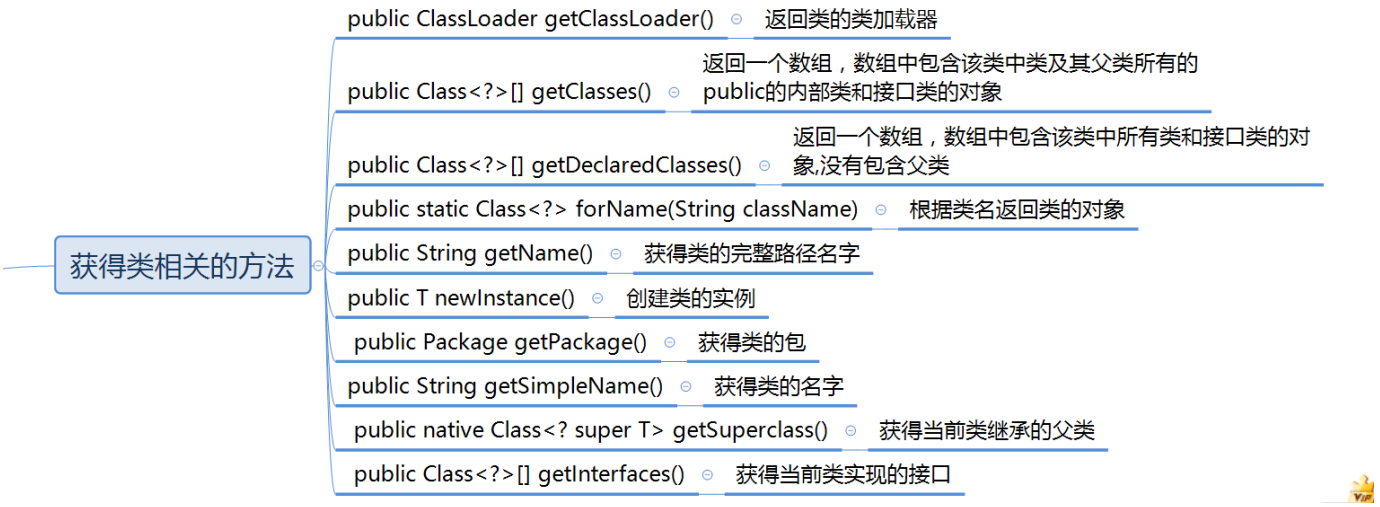
* 温馨提醒：

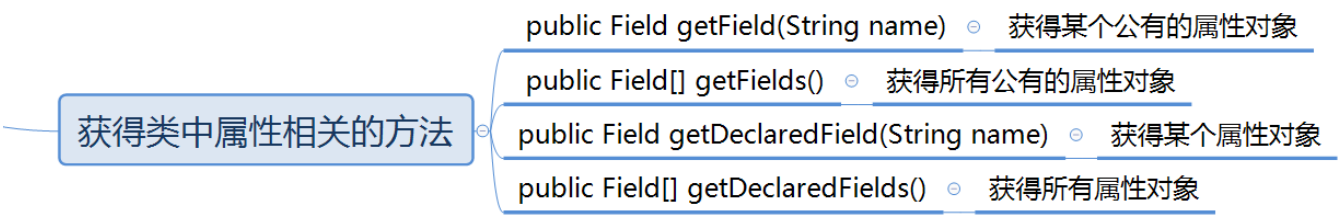
* 反射机制的功能非常强大，但不能滥用。在能不使用反射完成时，尽量不要使用

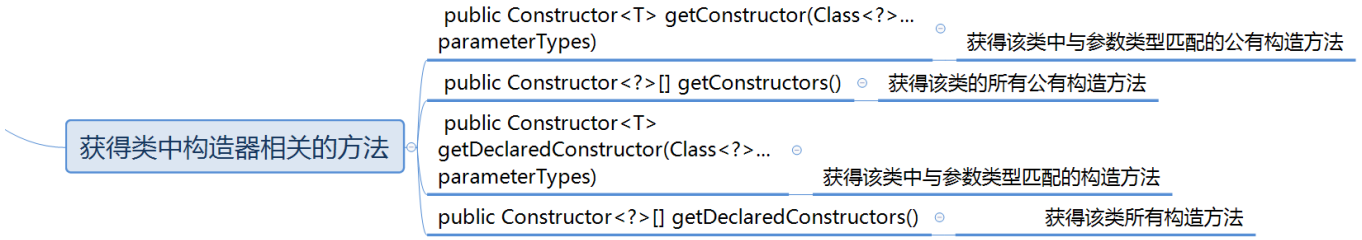
* 能够掌握常用反射的类

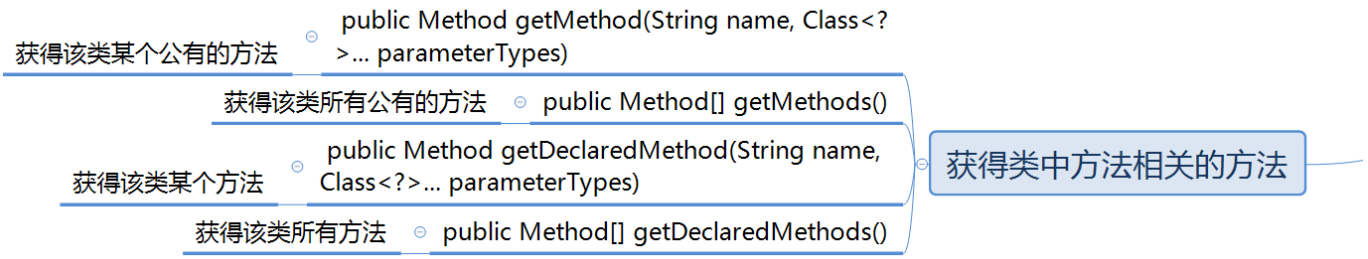
* Class

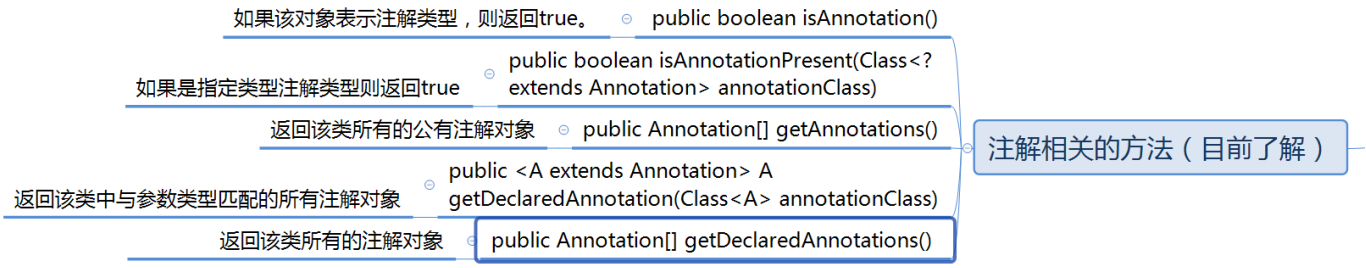




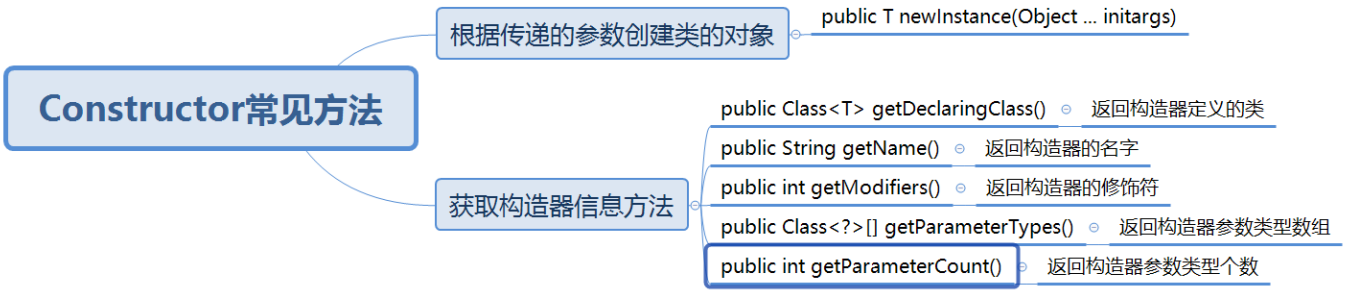




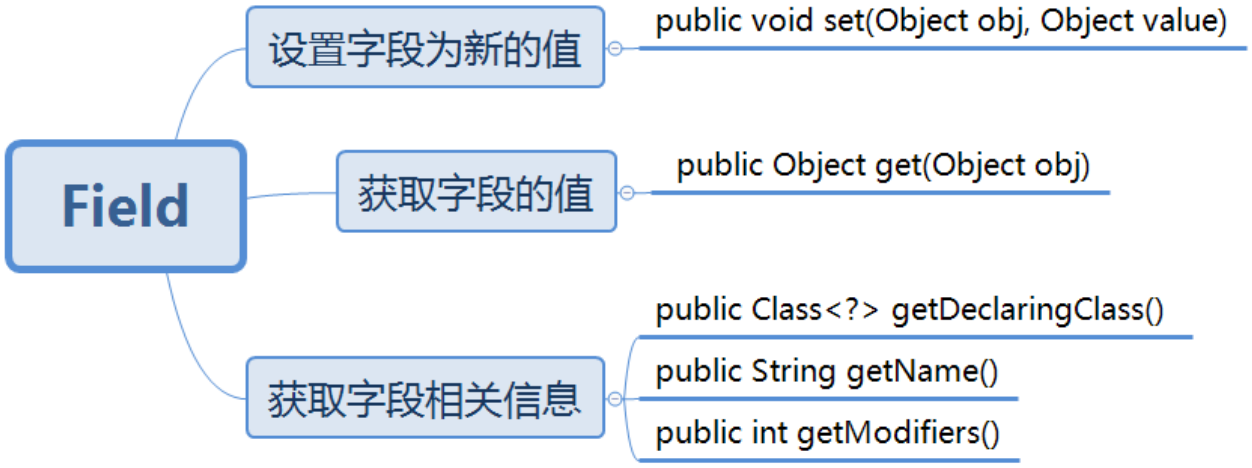




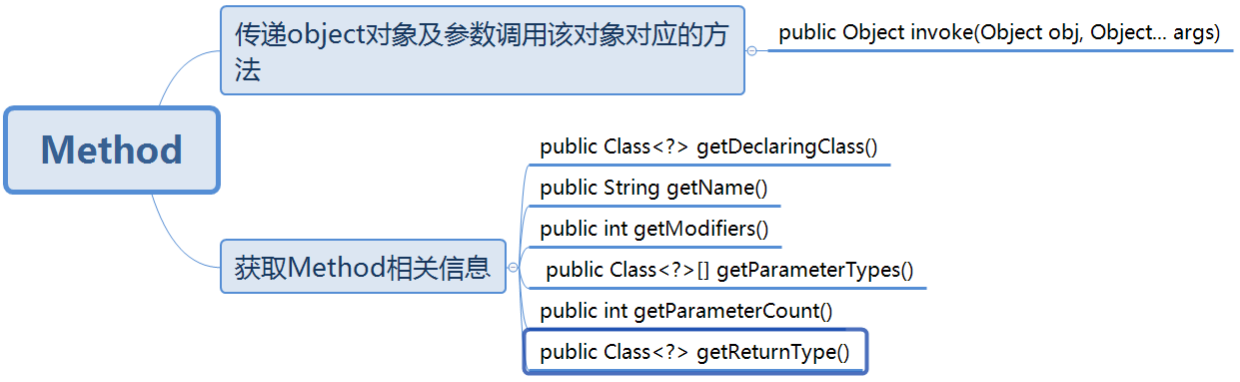
* Constructor



* Field



* Method




```
* 准备测试User的对象
public class User implements Comparable<User>{
```

```
    private int id;
    private String name;
    private String password;
    public User() {
        super();
    }
    public User(int id, String name, String password) {
        super();
        this.id = id;
        this.name = name;
        this.password = password;
    }
    //...
}
* 获取Class对象三种方式
@Test
    public void test1() throws ClassNotFoundException {
        // 获取Class对象三种方式
        Class<?> clazz=User.class;
        Class<?> clazz1=Class.forName("com.lg.bean.User");
        User user=new User();
        Class<?> clazz2=user.getClass();
        System.out.println(clazz);
        System.out.println(clazz==clazz1);
        System.out.println(clazz1==clazz2);
    }
* 结果
class com.lg.bean.User
true
true

* 获得类相关的方法
@Test
    public void test2() throws ClassNotFoundException, InstantiationException,
        Class<?> clazz=Class.forName("com.lg.bean.User");
        System.out.println("返回类的类加载器:"+clazz.getClassLoader());
        System.out.println("获得类的包:"+clazz.getPackage());
        System.out.println("创建类的实例:"+clazz.newInstance());
        System.out.println("获得类的完整路径名字:"+clazz.getName());
        System.out.println("获得类的名字:"+clazz.getSimpleName());
```

```
        System.out.println("获得当前类继承的父类:"+clazz.getSuperclass());
        System.out.println("获得当前类实现的接口:"+Arrays.toString(clazz.getInter
    }
* 结果
返回类的类加载器:sun.misc.Launcher$AppClassLoader@4e25154f
获得类的包:package com.lg.bean
创建类的实例:User [id=0, name=null, password=null]
获得类的完整路径名字:com.lg.bean.User
获得类的名字:User
获得当前类继承的父类:class java.lang.Object
获得当前类实现的接口:[interface java.lang.Comparable]

* 测试getClasses和getDeclaredClasses
 * 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
 * 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
public class A {
    private class B{
    }
    public class C{
    }
}
public class D extends A{
    private class E{
    }
    public class F{
    }
}
@Test
public void test3() throws ClassNotFoundException {
        Class<?> clazz=Class.forName("com.lg.bean.A");
        System.out.println("A:"+Arrays.toString(clazz.getClasses()));
        System.out.println("A:"+Arrays.toString(clazz.getDeclaredClasses()));
        clazz=Class.forName("com.lg.bean.D");
        System.out.println("D:"+Arrays.toString(clazz.getClasses()));
        System.out.println("D:"+Arrays.toString(clazz.getDeclaredClasses()));
}

结果:
A:[class com.lg.bean.A$C]
A:[class com.lg.bean.A$B, class com.lg.bean.A$C]
```

```
D:[class com.lg.bean.D$F, class com.lg.bean.A$C]
D:[class com.lg.bean.D$E, class com.lg.bean.D$F]

* 测试获得Field的方法
public class Person {
    private int id;
    public String name;
    protected int age;
    String password;
}
@Test
public void test4() throws ClassNotFoundException, InstantiationException,
    IllegalAccessException, NoSuchFieldException, SecurityException {
        Class<?> clazz=Class.forName("com.lg.bean.Person");
        System.out.println("获得某个公有的属性对象:"+clazz.getField("name"));
        System.out.println("获得所有公有的属性对象:"+Arrays.toString(clazz.getFie
        System.out.println("获得某个属性对象:"+clazz.getDeclaredField("age"));
        System.out.println("获得所有属性对象:"+Arrays.toString(clazz.getDeclared
}

结果:
获得某个公有的属性对象:public java.lang.String com.lg.bean.Person.name
获得所有公有的属性对象:[public java.lang.String com.lg.bean.Person.name]
获得某个属性对象:protected int com.lg.bean.Person.age
获得所有属性对象:[private int com.lg.bean.Person.id,
          public java.lang.String com.lg.bean.Person.name,
          protected int com.lg.bean.Person.age,
          java.lang.String com.lg.bean.Person.password]

* 测试获取构造器的方法
public class Person {
    private int id;
    public String name;
    protected int age;
    String password;

    private Person(String name) {
        this.name=name;
    }
    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.cl
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", Strin
202     System.out.println("获得该类所有方法:"+Arrays.toString(clazz.getDeclaredMeth
```

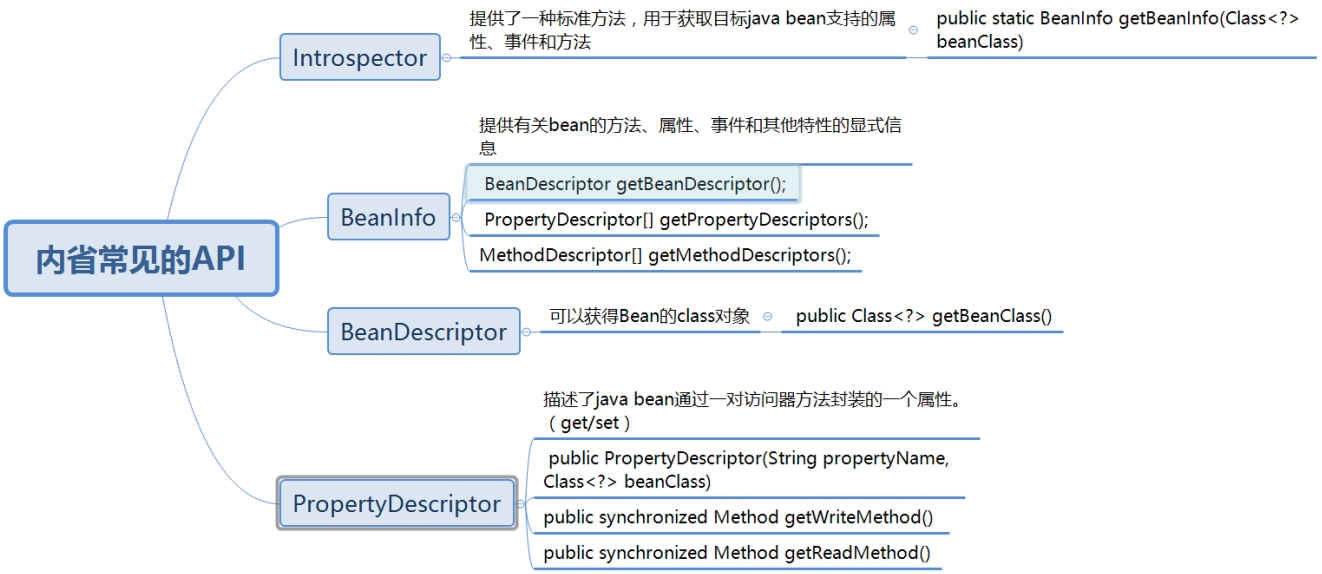
```
    }

结果：
获得该类某个公有的方法:public java.lang.String com.lg.bean.Person.say3(java.lang.
获得该类所有公有的方法:[public java.lang.String com.lg.bean.Person.say3(java.lang
             public static java.lang.String com.lg.bean.Person.say31(java.lang.
             public final void java.lang.Object.wait() throws java.lang.Interru
             public final void java.lang.Object.wait(long,int) throws java.lang
             public final native void java.lang.Object.wait(long) throws java.l
             public boolean java.lang.Object.equals(java.lang.Object),
             public java.lang.String java.lang.Object.toString(),
             public native int java.lang.Object.hashCode(),
             public final native java.lang.Class java.lang.Object.getClass(),
             public final native void java.lang.Object.notify(),
             public final native void java.lang.Object.notifyAll()]
获得该类某个方法:private void com.lg.bean.Person.say1(java.lang.String)
获得该类所有方法:[private void com.lg.bean.Person.say1(java.lang.String),
           public java.lang.String com.lg.bean.Person.say3(java.lang.String,int
           boolean com.lg.bean.Person.isOk(boolean),
           protected java.lang.String com.lg.bean.Person.say2(java.lang.String)
           private static void com.lg.bean.Person.say12(java.lang.String),
           static boolean com.lg.bean.Person.isOk1(boolean),
           public static java.lang.String com.lg.bean.Person.say31(java.lang.St
           protected static java.lang.String com.lg.bean.Person.say21(java.lang


* 构造器的测试
@Test
    @Test
    public void test8() throws ClassNotFoundException,
                    InstantiationException,
                    IllegalAccessException,
                    NoSuchFieldException, 、
                    SecurityException{
        Class<?> clazz=Class.forName("com.lg.bean.User");
        // 无参数构造
        Object obj= clazz.newInstance();// 通常使用这种，很多框架都使用这种，
                                        // 所以JavaBean有个要求要提供无参数的构造
        System.out.println(obj);
        Field id = clazz.getDeclaredField("id");
```

```
        Field name = clazz.getDeclaredField("name");
        Field password = clazz.getDeclaredField("password");
        // java.lang.IllegalAccessException:
        // Class com.lg.test.Test1 can not access a member of class
        // com.lg.bean.User with modifiers "private"
        // 暴力反射
        id.setAccessible(true);
        name.setAccessible(true);
        password.setAccessible(true);
        id.set(obj, 18);
        name.set(obj, "xiaohei");
        password.set(obj, "123");
        System.out.println(id.get(obj));
        System.out.println(name.get(obj));
        System.out.println(password.get(obj));
        System.out.println(obj);
    }
 结果:
User [id=0, name=null, password=null]
18
xiaohei
123
User [id=18, name=xiaohei, password=123]


* 测试方法
@Test
    public void test9() throws ClassNotFoundException, InstantiationException,
        Class<?> clazz=Class.forName("com.lg.bean.User");
        // 无参数构造
        Object obj= clazz.newInstance();// 通常使用这种，很多框架都使用这种，
                                        // 所以JavaBean有个要求要提供无参数的构造
        System.out.println(obj);
        Method setIdMethod = clazz.getDeclaredMethod("setId", int.class);
        Method getIdMethod = clazz.getDeclaredMethod("getId");
        Method setNameMethod = clazz.getDeclaredMethod("setName", String.class)
        Method getNameMethod = clazz.getDeclaredMethod("getName");
        Method setPasswordMethod = clazz.getDeclaredMethod("setPassword", Strir
        Method getPasswordMethod = clazz.getDeclaredMethod("getPassword");
        setIdMethod.invoke(obj, 18);
```

```
        setNameMethod.invoke(obj, "xiaohei");
        setPasswordMethod.invoke(obj, "123");
        System.out.println(getIdMethod.invoke(obj));
        System.out.println(getNameMethod.invoke(obj));
        System.out.println(getPasswordMethod.invoke(obj));
        System.out.println(obj);
    }
结果:
User [id=0, name=null, password=null]
18
xiaohei
123
User [id=18, name=xiaohei, password=123]

* 通用型的编写
@Test
    public void test10() throws Exception {
        Class<?> clazz = Class.forName("com.lg.bean.User");
        // 调用无参数构造
        Object obj = clazz.newInstance();
        Map<String,Object> map=new HashMap<String,Object>();
        map.put("id", 18);
        map.put("name", "xiaohei");
        map.put("password","123456");
        Field[] fields = clazz.getDeclaredFields();
        for (int i = 0; i < fields.length; i++) {
            String fieldName = fields[i].getName();
            Object arg = map.get(fieldName);
            String methodName="set"+fieldName.substring(0,1).toUpperCase()+fiel
            Class<?> type=fields[i].getType();
            Method method = clazz.getMethod(methodName,type);
            method.invoke(obj, arg);
        }
        System.out.println(obj);
    }
* 结果
User [id=0, name=null, password=null]
18
xiaohei
123
```

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

* 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




```
* 测试
    @Test
    public void test9() throws Exception {
        Class<?> clazz = Class.forName("com.lg.bean.User");
        // 调用无参数构造
        Object obj = clazz.newInstance();
        Map<String,Object> map=new HashMap<String,Object>();
        map.put("id", 18);
        map.put("name", "xiaohei");
        map.put("password","123456");
        BeanInfo beanInfo = Introspector.getBeanInfo(clazz);
        PropertyDescriptor[] propertyDescriptors = beanInfo.getPropertyDescript
        for (PropertyDescriptor pd : propertyDescriptors) {
            Object arg = map.get(pd.getName());
            if(arg==null) {
```

```
                continue;
            }
            Method writeMethod = pd.getWriteMethod();
            writeMethod.invoke(obj, arg);
        }
        System.out.println(obj);
    }

 * 结果
 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

```
* BeanUtils开发
  * 导入jar
    * commons-beanutils-1.9.3.jar
    * commons-logging-1.2-sources.jar
  * 代码
@Test
    public void test10() throws Exception {
        Class<?> clazz = Class.forName("com.lg.bean.User");
        // 调用无参数构造
        Object obj = clazz.newInstance();
        Map<String,Object> map=new HashMap<String,Object>();
        map.put("id", 18);
        map.put("name", "xiaohei");
        map.put("password","123456");
        BeanUtils.populate(obj, map);
        System.out.println(obj);
    }
  * 结果
    User [id=18, name=xiaohei, password=123456]
```

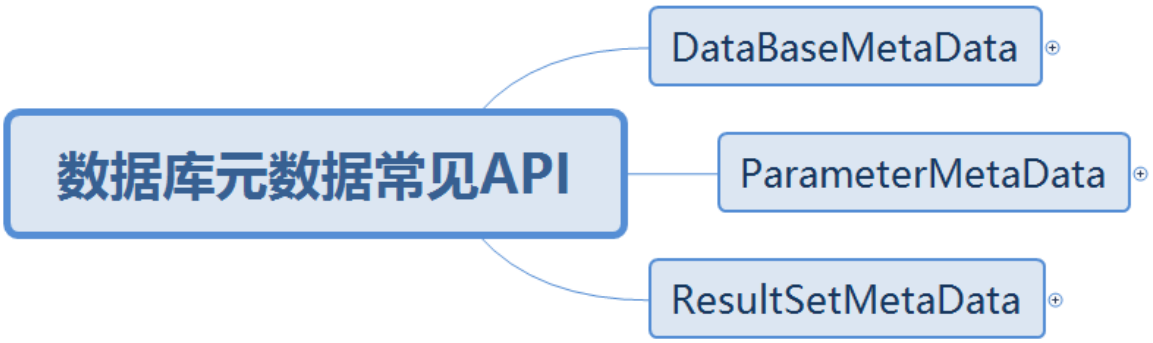
* 能够掌握数据库的元数据

* 元数据的概述

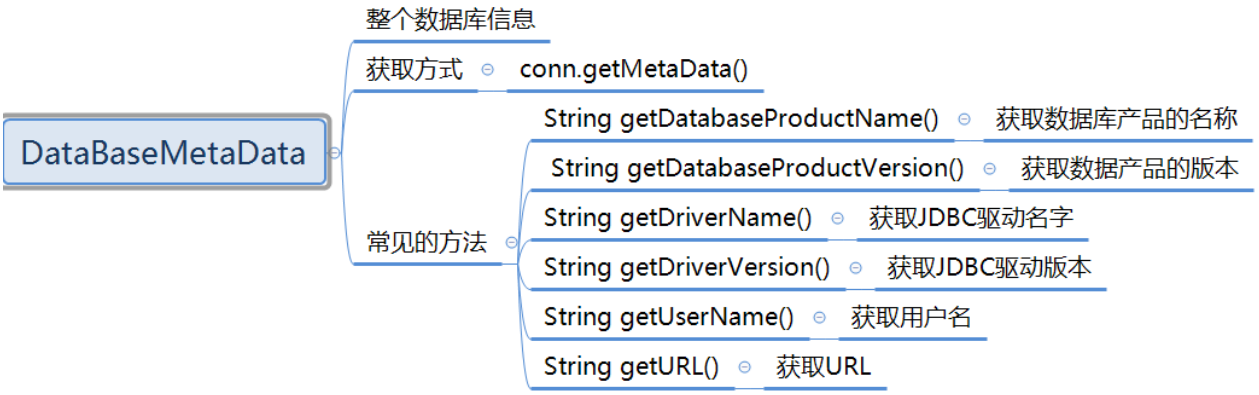
* 元数据(MetaData)，即定义数据的数据。

* 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

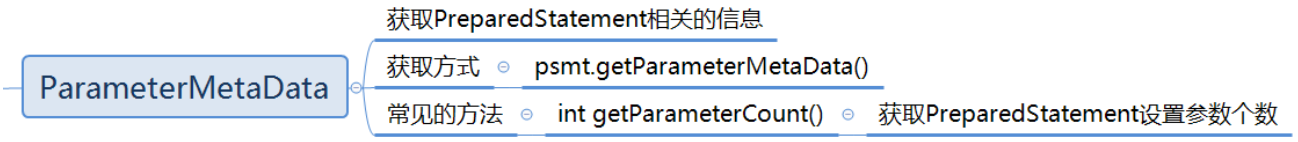
* 数据库元数据常见的API



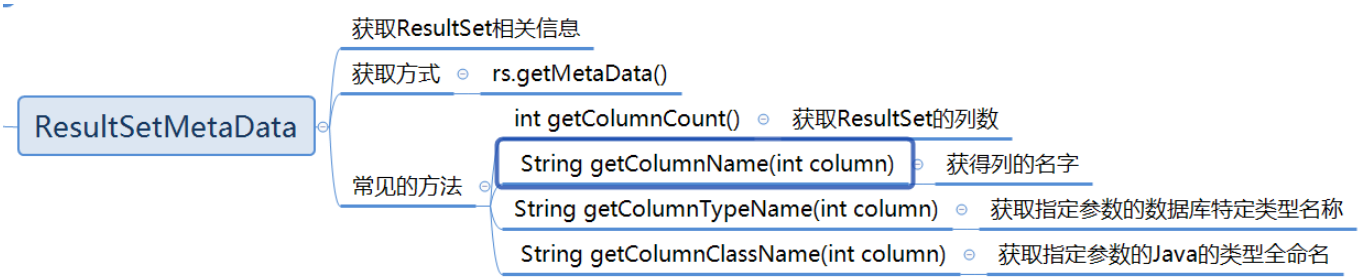
* DataBaseMetaData



* ParameterMetaData



* ResultSetMetaData




```
* 测试DataBaseMetaData
@Test
    public void test1() throws SQLException {
        Connection conn = ConnectionUtils.getConn();
        DatabaseMetaData metaData = conn.getMetaData();
        String productName=metaData.getDatabaseProductName();
        String productVersion=metaData.getDatabaseProductVersion();
        String driverName=metaData.getDriverName();
        String driverVersion=metaData.getDriverVersion();
        String url=metaData.getURL();
        String username=metaData.getUserName();
        System.out.println(productName);
        System.out.println(productVersion);
        System.out.println(driverName);
        System.out.println(driverVersion);
        System.out.println(url);
        System.out.println(username);
        ConnectionUtils.close(conn, null,null);
    }
* 结果
Oracle
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Oracle JDBC driver
11.2.0.1.0
jdbc:oracle:thin:@192.168.1.121:1521/orcl
SCOTT

* 测试ParameterMetaData
@Test
```

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

* 测试ResultSetMetaData
@Test
    public void test3() throws SQLException {
        Connection conn = ConnectionUtils.getConn();
        String sql = "select * from t_user where id=?";
        PreparedStatement psmt = conn.prepareStatement(sql);
        psmt.setInt(1, 5);
        ResultSet rs = psmt.executeQuery();
        ResultSetMetaData metaData = rs.getMetaData();
        int columnCount = metaData.getColumnCount();
        System.out.println("columnCount:" + columnCount);
        for (int i = 0; i < columnCount; i++) {
            String columnName = metaData.getColumnName(i + 1);
            String columnTypeName = metaData.getColumnTypeName(i + 1);
            System.out.println("columnName:" + columnName);
            System.out.println("columnTypeName:" + columnTypeName);
        }
        ConnectionUtils.close(conn, psmt, rs);
    }

结果:
columnCount:3
columnName:ID
columnTypeName:NUMBER
columnName:NAME
columnTypeName:VARCHAR2
```

```
columnName:PASSWORD
columnTypeName:VARCHAR2
```

* 能够自定义JDBC的框架

```
* Student
  public class Student {
    private String sid;
    private String sname;
    private int age;
    private String gender;
 }
* 观察StudentDaoImpl
public class StudentDaoImpl implements StudentDao {

    @Override
    public boolean add(Student student) {
        if (student == null) {
            throw new IllegalArgumentException("student is null");
        }
        Connection con = ConnectionUtils.getConn();
        PreparedStatement st = null;
        int result = 0;
        try {
            String sql = "insert into stu(sid,sname,age,gender) values(?,?,?,?)
            st = con.prepareStatement(sql);
            st.setString(1, student.getSid());
            st.setString(2, student.getSname());
            st.setInt(3, student.getAge());
            st.setString(4, student.getGender());
            result = st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConnectionUtils.close(con, st, null);
        }
```

```
        return result > 0;
    }

    @Override
    public boolean update(String sid, String name) {
        Connection con = ConnectionUtils.getConn();
        PreparedStatement st = null;
        int result = 0;
        try {
            String sql = "update stu set sname=? where sid=?";
            st = con.prepareStatement(sql);
            st.setString(1, name);
            st.setString(2, sid);
            result = st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConnectionUtils.close(con, st, null);
        }
        return result > 0;
    }

    @Override
    public boolean delete(String sid) {
        Connection con = ConnectionUtils.getConn();
        PreparedStatement st = null;
        int result = 0;
        try {
            String sql = "delete from stu where sid=?";
            st = con.prepareStatement(sql);
            st.setString(1, sid);
            result = st.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConnectionUtils.close(con, st, null);
        }
        return result > 0;
    }

```

```
72      @Override
73      public Student queryById(String id) {
74          String sql = "select sid,sname,age,gender from stu where sid=?";
75          Connection con = ConnectionUtils.getConn();
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.getConn();
107         PreparedStatement st = null;
108         ResultSet rs = null;
109         List<Student> students = new ArrayList<Student>();
110         try {
111             st = con.prepareStatement(sql);
```

```
            rs = st.executeQuery();
            while (rs.next()) {
                String sid = rs.getString("sid");
                String sname = rs.getString("sname");
                String gender = rs.getString("gender");
                try {
                    int age = Integer.parseInt(rs.getString("age"));
                    students.add(new Student(sid, sname, age, gender));
                } catch (NumberFormatException e) {
                    students.add(new Student(sid, sname, 0, gender));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConnectionUtils.close(con, st, rs);
        }
        return students;
    }

    // offset: row:多少行
    @Override
    public List<Student> queryStudents(int offset, int row) {
        String sql = "select sid,sname,age,gender from (select rownum rn,sid,sr
        Connection con = ConnectionUtils.getConn();
        PreparedStatement st = null;
        ResultSet rs = null;
        List<Student> students = new ArrayList<Student>();
        try {
            st = con.prepareStatement(sql);
            st.setInt(1, row);
            st.setInt(2, offset);
            rs = st.executeQuery();
            while (rs.next()) {
                String sid = rs.getString("sid");
                String sname = rs.getString("sname");
                String gender = rs.getString("gender");
                try {
                    int age = Integer.parseInt(rs.getString("age"));
                    students.add(new Student(sid, sname, age, gender));
```

```
                } catch (NumberFormatException e) {
                    students.add(new Student(sid, sname, 0, gender));
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            ConnectionUtils.close(con, st, rs);
        }
        return students;
    }

}

* 总结增删改
   * 一样地方获取连接，释放资源，获得PreparedStatement，执行executeUpdate
   * 不一样的地方，sql语句不一样，设置参数也不一样
* 总结查询单个
   * 一样地方获取连接，释放资源，获得PreparedStatement，executeQuery
   * 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
* 总结查询列表
    * 一样地方获取连接，释放资源，获得PreparedStatement，executeQuery
    * 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

* JdbcUtils的编写
public class JdbcUtils {

    /**
     * @param sql
     * @param args
     * @return
     * CUD
     */
    public static boolean update(String sql,Object... args) {
        Connection conn=ConnectionUtils.getConn();
        PreparedStatement psmt=null;
        int result=-1;
        try {
            psmt=conn.prepareStatement(sql);
            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... a
208         Connection conn=ConnectionUtils.getConn();
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(cour
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 {
```

```
            ConnectionUtils.close(conn, psmt, rs);
        }
        return null;
    }
}
* 定义ResultSetHandler
public interface ResultSetHandler {
    Object handle(ResultSet rs);
}
* BeanHandler编写
public class BeanHandler implements ResultSetHandler {
    private Class<?> clazz;

    public BeanHandler(Class<?> clazz) {
        this.clazz = clazz;
    }

    @Override
    public Object handle(ResultSet rs) {
        Object obj = null;
        try {
            if (rs.next()) {
                obj = this.clazz.newInstance();
                Map<String, Object> properties = new HashMap<String, Object>();
                ResultSetMetaData metaData = rs.getMetaData();
                int columnCount = metaData.getColumnCount();
                for (int i = 0; i < columnCount; i++) {
                    String columnName = metaData.getColumnName(i + 1);
                    Object param = rs.getObject(i + 1);
                    properties.put(columnName.toLowerCase(), param);
                }
                BeanUtils.populate(obj, properties);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return obj;
    }

}
```

```
* ListBeanHandler 编写
public class ListBeanHandler implements ResultSetHandler {
    private Class<?> clazz;

    public ListBeanHandler(Class<?> clazz) {
        this.clazz = clazz;
    }

    @Override
    public Object handle(ResultSet rs) {
        List<Object> objs = new ArrayList<Object>();
        try {
            while (rs.next()) {
                Object obj = this.clazz.newInstance();
                Map<String, Object> properties = new HashMap<String, Object>();
                ResultSetMetaData metaData = rs.getMetaData();
                int columnCount = metaData.getColumnCount();
                for (int i = 0; i < columnCount; i++) {
                    String columnName = metaData.getColumnName(i + 1);
                    Object param = rs.getObject(i + 1);
                    properties.put(columnName.toLowerCase(), param);
                }
                BeanUtils.populate(obj, properties);
                objs.add(obj);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return objs;
    }
}

* copy StudentDaoImpl修改成StudentDaoImpl2
public class StudentDaoImpl2 implements StudentDao {

    @Override
    public boolean add(Student student) {
        if (student == null) {
            throw new IllegalArgumentException("student is null");
        }
```

```
312         String sql = "insert into stu(sid,sname,age,gender) values(?,?,?,?)";
313         Object[] args = { student.getSid(), student.getSname(), student.getAge(
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.
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 ListBea
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,sn
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 
```