1.基础应用加强

1.1. 泛型基础加强

1. 如何理解泛型?
   1）参数化类型,是 JDK1.5 的新特性。（定义泛型时使用参数可以简单理解为形参）
   2）编译时的一种类型，此类型仅仅在编译阶段有效，运行时无效。

2. 为何使用泛型?
   3）提高编程时灵活性。
   4）提高程序运行时的性能。(在编译阶段解决一些运行时需要关注的问题，例如强转)

3. 泛型的应用类型?
   5）泛型类： class 类名<泛型>{}
   6）泛型接口：interface 接口名<泛型>{}
   7）泛型方法：访问修饰符 <泛型> 方法返回值类型 方法名(形参){}

4. 泛型的通配符? （这里的通配符可以看成是一种不确定的类型）
   8）泛型应用时有一个特殊符号”?”,可以代表一种任意参数类型，注释是实参。
   9）通配符泛型只能应用于变量的定义。

5. 泛型的上下界问题?
   10)指定泛型下界：<? super 类型>
   11)指定泛型上界：<? extends 类型>

   例如:
   ```
   List<? extends CharSequence> list1=new ArrayList<String>();
   List<? super Integer> list2=new ArrayList<Number>();
   ```

   说明：这种上下界一般会用于方法参数变量定义，方法返回值类型定义。

6. 泛型类型擦除?
   泛型是编译时的一种类型，在运行时无效，运行时候都会变成 Object 类型。

1.2. 序列化基础加强

1. 何为序列化&反序列化?
   12)序列化：将对象转换为字节的过程。
   13)反序列化：将字节转换为对象的过程。

2. 序列化的应用场景?
   1) 网络通讯
   2) 数据存储(例如文件，缓存)

3. Java 中的对象的序列化与反序列化?
   1) 对象要实现 Serializable 接口
   2) 添加序列化 id（为反序列化提供保障）
   3) 借助流对象实现序列化和反序列化?

4. Java 中的序列化存在安全问题如何解决?

   1) 添加 writeObject(ObjectOutpuStream out)方法
      对内容进行加密再执行序列化。
   1) 添加 readObject(ObjectInputStream in)方法对
      内容先进行反序列化然后在执行解密操作

5. Java 中序列化的粒度如何控制?

   1) Transient（当少量属性不需要序列化时，使用此关键字修饰）
   2) Externalizable（当只有少量属性需要序列化时实现此接口然后自己进行序列化操作，但是要序列化的对象必须时 public 修饰。）

7. Java 中序列化的性能问题及如何优化?

1.3. 注解应用基础加强

1. 如何理解注解(Annotation)?
   1) JDK1.5 推出的一种新的应用类型（特殊的 class）
   2) 元数据(Meta Data):一种描述性类型，例如@Override

2. 注解(Annotation)应用场景?
   1) 描述类及其成员(属性，方法)：例如@Overide

2）替换项目中 xml 方式对相关对象的描述(例如@Service,@Controller,...)

例如 spring 框架中相关注解?

1)@Configuration（描述配置类对象）

2)@Service(描述业务层对象)

3)@Controller(描述控制层对象)

4)@Responsitory(描述数据层对象)

5)@RestController(描述控制层对象)

6)@RestControllerAdvice（描述控制全局异常处理)

7)@Bean(描述 bean 对象，一般修饰方法将返回交给 spring 管理)

8)@Autowired（实现 bean 对象的自动装配)

9)…

3. 注解(Annotation)的定义?

可以借助@interface 关键字进行定义，例如 Override 注解的应用

```
@Target(value=METHOD)
@Retention(value=SOURCE)
public @interface Override{}
```

其中:

1)@Target 用于描述定义的注解能够修饰的对象。

2)@Retention 用于描述定义的注解何时有效。

4. 注解常用的生效范围?

1）编译时有效（例如@Retention(RetentionPolicy.SOURCE))

2）运行时有效（例如@Retention(RetentionPolicy.RUNTIME)

说明：我们自己定义的注解，包括框架中的很多注解基本都是运行时有效。

5. 注解应用案例分析实现?

1）与编译器结合实用(@Override)

2）与反射 API 结合使用(@RequiredLog,@Transaction,…)

例如:

1）通过反射或类上的注解

2）通过反射获取属性或方法上的注解。

1.4. 反射应用基础加强

---

1. 如何理解反射?

1）Java 中特有一种技术
2）JAVA 中自省特性的一种实现?(对象运行时动态发现对象成员)
3）是实现 JAVA 动态编程的基石?(例如 AOP,…)

2．反射的应用场景?
1）框架中对象的构建?（例如 mybatis 中的 resultType,resultMap,spring 中的 bean）
2）框架中方法的调用?（例如对象 set 方法，get 方法，spring mvc 控制层方法，..）

总之：反射不能预知未来，但可驾驭未来，通过反射可以更好构建一些编程框架，以实现通用性编程，从而达到简化代码编写。

3．反射的应用起点?

起点可以理解为反射应用的入口，在 java 中这个入口是字节码对象(Class 对象)。其获取方式如下：
1）类名.class
2）Class.forName(“包名.类名”)；最常用
3）实例对象.getClass();（获取已经存在的类对象）
说明：任意的一个类在同一个 JVM 内部,类对象是唯一的,此类对象会在第一次类加载时创建。

4．反射核心 API?
1）Constructor（构造方法对象类型）
2）Field（属性对象类型）
3）Method（方法对象类型）
4）Annotation(注解对象类型)
5）…

5．反射应用案例分析及实现?
1）基于反射构建类的实例：首先要基于类对象获取构造方法对象
2）基于反射获取对象属性，并为属性赋值。
3）基于反射获取对象方法，并执行对象方法。
4）基于反射获取描述对象的注解,并基于注解的含义执行下一步操作。
5）基于反射获取类上的泛型参数？（作业）