Ioc: Inversion of control,控制反转

控制反转是，关于一个对象如何获取他所依赖的对象的引用，这个责任的反转。

控制反转一般分为两种类型，依赖注入（Dependency Injection，简称DI）和依赖查找（Dependency Lookup）。依赖注入应用比较广泛。

DI即让***调用类对某一接口实现类***的**依赖关系**由**第三方（容器或协作类）注入**，以移除调用类对某一接口实现类的依赖（由通用容器解决对具体实现的依赖）。

第三方的容器，它帮助完成类的初始化与装配工作，让开发者从这些底层实现类的实例化、依赖关系装配等工作中脱离出来，专注于更有意义的业务逻辑开发工作。

IOC与工厂模式区别：

When using a factory your code is still actually responsible for creating objects. By DI you outsource that responsibility to another class or a framework, which is separate from your code.

**Ioc将对象的实现完全交给了独立于逻辑第三方的类实现。而工厂模式仍旧需要一个工厂类去管理格式类的实现**

依赖注入注入方法上看，主要可以划分为三种类型：**构造函数注入、属性注入和接口注入**，Spring支持构造函数注入和属性注入。

1. **public** **class** MoAttack {
2. **private** GeLi geli;
3. //①注入革离的具体扮演者
4. **public** MoAttack(GeLi geli){
5. **this**.geli = geli;
6. }
7. **public** **void** cityGateAsk(){
8. geli.responseAsk("墨者革离！");
9. }
10. }
11. **public** **class** Director {
12. **public** **void** direct(){
13. //①指定角色的扮演者
14. GeLi geli = **new** LiuDeHua();
16. //②注入具体扮演者到剧本中
17. MoAttack moAttack = **new** MoAttack(geli);
18. moAttack.cityGateAsk();
19. }
20. }

Director作为第三方容器，构造函数注入。

1. **public** **class** MoAttack {
2. **private** GeLi geli;
3. //①属性注入方法
4. **public** **void** setGeli(GeLi geli) {
5. **this**.geli = geli;
6. }
7. **public** **void** cityGateAsk() {
8. geli.responseAsk("墨者革离");
9. }
10. **public** **class** Director {
11. **public** **void** direct(){
12. GeLi geli = **new** LiuDeHua();
13. MoAttack moAttack = **new** MoAttack();
15. //①调用属性Setter方法注入
16. moAttack.setGeli(geli);
17. moAttack.cityGateAsk();
18. }
19. }

Director作为第三方，属性注入。

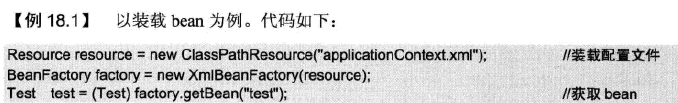
1. **public** **interface** ActorArrangable {
2. **void** injectGeli(GeLi geli);
3. }
4. **public** **class** MoAttack **implements** ActorArrangable {
5. **private** GeLi geli;
6. //①实现接口方法
7. **public** **void** injectGeli (GeLi geli) {
8. **this**.geli = geli;
9. }
10. **public** **void** cityGateAsk() {
11. geli.responseAsk("墨者革离");
12. }
13. }
14. **public** **class** Director {
15. **public** **void** direct(){
16. GeLi geli = **new** LiuDeHua();
17. MoAttack moAttack = **new** MoAttack();
18. moAttack. injectGeli (geli);
19. moAttack.cityGateAsk();
20. }
21. }

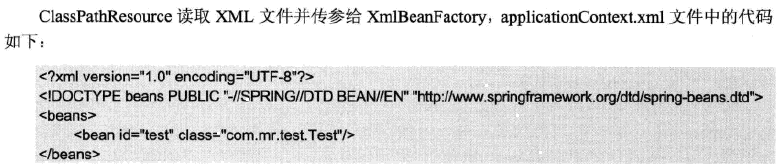
由于通过**接口注入**需要额外声明一个接口，增加了类的数目，而且它的效果和属性注入并无本质区别，因此我们不提倡采用这种方式。

Spring作为一个大容器通过xml控制类与类之间的装配关系。

[spring](http://lib.csdn.net/base/javaee)中的IoC的实现原理就是工厂模式加反射机制。

Bean factory





Spring Setter注入

