**spring4的泛型依赖注入是什么原理？**

* Dao

public class BaseDao<T> {

public void add(){}

}

@Repository

public class EmpDao extends BaseDao<Employee>{ }

* Service

public class BaseService<T> {

@Autowired

protected BaseDao<T> baseDao;

}

@Service

public class EmpService extends BaseService<Employee>{

public void add(){

System.out.println(baseDao)

}

}

* 测试

ApplicationContext ctx = new ClassPathXmlApplicationContext("spring-config6.xml");

EmpService empService = ctx.getBean(EmpService.class);

empService.add();

//输出 dao.impl.EmpDao@214b199c

**获取EmpService spring自动匹配到了 EmpDao**

**但是！如果再添加一个 泛型 类型为Employee的bean，问题就来了。**

@Repository

public class EmpDao2 extends BaseDao<Employee>{ }

**添加第二个BaseDao的子类 泛型也为 Employee，再次测试出了如下异常:**

expected single matching bean but found 2: empDao,empDao2

**可以肯定的一点是，Spring 是通过 泛型参数 来匹配EmpDao 与 EmpService的。但是sping是如何动态获取泛型类型的？泛型为何没有被擦除？**

|  |
| --- |
| 很有深度的问题，如果不自己做个实验，可能都不会注意到。  如果用Pom等机制，可以把依赖的源代码下载下来，然后可以直接在 spring 的库中设置断点，来调试 spring 代码，如果足够耐心，逐行执行一次，就可以初见端倪了。  代码编写完了，command ＋ o 找到 DefaultListableBeanFactory，他在 spring bean 这个包中  定位到 findAutowireCandidates，可以在这里卡一个断点，然后debug运行你的程序。该方法负责找到匹配的 bean 注入到 service中。其中有如下代码负责过滤掉最开始找到的 candidates  for (String candidate : candidateNames) {  if (!isSelfReference(beanName, candidate) && isAutowireCandidate(candidate, descriptor)) {  addCandidateEntry(result, candidate, descriptor, requiredType);  }  }  其中函数 isAutowireCandidate 往里面找  public boolean isAutowireCandidate(BeanDefinitionHolder bdHolder, DependencyDescriptor descriptor) {  if (!super.isAutowireCandidate(bdHolder, descriptor)) {  // If explicitly false, do not proceed with any other checks...  return false;  }  return checkGenericTypeMatch(bdHolder, descriptor);  }  看到这里有范型检查相关的内容，再往深入找，会看到这段代码  if (checkGenerics) {  // Recursively check each generic  ResolvableType[] ourGenerics = getGenerics();  ResolvableType[] typeGenerics = other.as(ourResolved).getGenerics();  if (ourGenerics.length != typeGenerics.length) {  return false;  }  if (matchedBefore == null) {  matchedBefore = new IdentityHashMap<>(1);  }  matchedBefore.put(this.type, other.type);  for (int i = 0; i < ourGenerics.length; i++) {  if (!ourGenerics[i].isAssignableFrom(typeGenerics[i], matchedBefore)) {  return false;  }  }  }  所以 spring 其实是利用反射机制，获取类型的范型的，然后做了比较返回了合适的 bean 进行注入的。 |