1、概念

1、泛型是java中的一种类型

2、在java中所有的类型都可以用Type来代替

3、ParameterizedType该类型就是泛型（前面学的泛型就是学的它）

public interface ParameterizedType extends Type {

//得到实际类型的参数

Type[] getActualTypeArguments();

//得到rawType

Type getRawType();

//得到自己的类型

Type getOwnerType();

}

4、理解ActualTypeArguments、RawType

public class PersonService<E>{

}

上述的PersonService<E>由两部分组成

PersonService 叫做rawType

<E> 代表形参（实际类型） getActualTypeArguments()该方法告诉程序员E的实际类型 在编译阶段，把实参传递给形参

5、在泛型中传递参数的方式

1、public class ArrayList<E>{

}

声明了一个泛型

为ArrayList创建对象

ArrayList<Person> pList = new ArrayList<Person>();

2、public interface PersonDao<E>{

}

public interface P<E> extends PersonDao<E>{

}

public class PersonDaoImpl implements P<Person>{

}

传递过程： P<Person> ------》 P<E> ------》 PersonDao<E> ------》 PersonDao<E>

例如ArrayList

ArrayList<Person> ----> ArrayList<E> -----> AbstractList<E> ----> AbstractCollection<E> -----> Collection<E>

----> List<E> -----> Collection<E>

**所以泛型具有传递性**

2、**实际的应用：重构dao**

1、cn/itcast/shoa/dao/base/BaseDao

public interface BaseDao<E>{

/\*\*

\* 得到E代表的所有的实体对象

\* @return

\*/

public Collection<E> getAllEntry();

/\*\*

\* Serializable该类型可以接受所有的基本类型和String类型

\* @param id

\* @return

\*/

public E getEntryById(Serializable id);

public void saveEntry(E e);

public void deleteEntry(Serializable id);

public void updateEntry(E e);

}

2、实现类:cn/itcast/shoa/dao/base/impl/BaseDaoImpl

|  |
| --- |
| **package** cn.wang.oaA.dao.base.impl;  **import** java.io.Serializable;  **import** java.lang.reflect.ParameterizedType;  **import** java.util.Collection;  **import** javax.annotation.Resource;  **import** org.hibernate.metadata.ClassMetadata;  **import** org.springframework.orm.hibernate3.HibernateTemplate;  **import** cn.wang.oaA.dao.base.BaseDao;  /\*\*  \* BaseDao<E> 的泛型需要 BaseDaoImpl<E>来传递  \*/  **public** **class** BaseDaoImpl<E> **implements** BaseDao<E>{  /\*\*  \* 通过注解的方式引入hibernateTemplate  \*/  @Resource(name="hibernateTemplate")  **public** HibernateTemplate hibernateTemplate;    /\*\*  \* 定义实体的实际类型  \*/  **private** Class classt;    **public** BaseDaoImpl(){  /\*\*  \* 1、this可以代表子类或者本类  \* 2、不能把BaseDaoImpl在spring容器中实例化，因为如果在spring容器中实例化，则就不是泛型了  \* \*\*\*\*  \* 因为不能把BaseDaoImpl实例化  \* 所以 this 肯定代表的是 BaseDaoImpl的子类  \* \*\*\*\*  \* 3、所以根据以上两点可以得出this应该代表子类（例如PersonDaoImpl）  \* \*\*\*\*  \* this 代表的是 BaseDaoImpl<E>的子类 例如 PersonDaoImpl  \* this.getClass().getGenericSuperclass()获取的就是 BaseDaoImpl<E>本身的泛型  \* \*\*\*\*  \* 4、this.getClass().getGenericSuperclass()：就是当前类的泛型  \* \*\*\*\*  \* 通过getGenericSuperclass方法可以获取当前对象的直接超类的 Type  \* 只有当 BaseDaoImpl<E> 是一个泛型类的时候才能利用 getGenericSuperclass方法 得到泛型  \* 只有当 BaseDaoImpl<E> 创建实例对象的时候只有把实参传递给E ，BaseDaoImpl<E> 它才是泛型  \* \*\*\*\*  \* 5、注意：不能把BaseDaoImpl让spring容器实例化  \*/  ParameterizedType type = (ParameterizedType)**this**.getClass().getGenericSuperclass();  /\*\*  \* getActualTypeArguments()该方法告诉程序员E的实际类型  \* 返回值是一个数组 因为 泛型可以写多个  \*/  //因为将来E代表的是某一个持久化类，而该类型是class  **this**.classt = (Class)type.getActualTypeArguments()[0];  }    /\*\*  \* this.classt.getName() 代表持久化类  \*/  @Override  **public** Collection<E> getAllEntry() {  **return** **this**.hibernateTemplate.find("from "+**this**.classt.getName());  }  @Override  **public** E getEntryById(Serializable id) {  /\*\*  \* classMetadata是持久化类的数据字典  \* classMetadata.getIdentifierPropertyName() 得到标识符属性的名字  \* 根据主键查  \* 最后要强转成 E 当前的实体类  \*/  ClassMetadata classMetadata = **this**.hibernateTemplate.getSessionFactory().getClassMetadata(**this**.classt);  **return** (E)**this**.hibernateTemplate.find("from "+**this**.classt.getName() + " where " +classMetadata.getIdentifierPropertyName() +"=?",id).get(0);  }  @Override  **public** **void** saveEntry(E e) {  **this**.hibernateTemplate.save(e);  }  @Override  **public** **void** deleteEntry(Serializable id) {  E e = **this**.getEntryById(id);  **this**.hibernateTemplate.delete(e);  }  @Override  **public** **void** updateEntry(E e) {  **this**.hibernateTemplate.update(e);  }  } |