注解作用：每当你创建描述符性质的类或者接口时，一旦其中包含重复性的工作，就可以考虑使用注解来简化与自动化该过程。

Java提供了四种元注解，专门负责新注解的创建工作。

元注解

元注解的作用就是负责注解其他注解。Java5.0定义了4个标准的meta-annotation类型，它们被用来提供对其它 annotation类型作说明。

Java5.0定义的元注解：

1.@Target,

2.@Retention,

3.@Documented,

4.@Inherited

这些类型和它们所支持的类在java.lang.annotation包中可以找到。下面我们看一下每个元注解的作用和相应分参数的使用说明。

@Target

@Target说明了Annotation所修饰的对象范围：Annotation可被用于 packages、types（类、接口、枚举、Annotation类型）、类型成员（方法、构造方法、成员变量、枚举值）、方法参数和本地变量（如循环变量、catch参数）。在Annotation类型的声明中使用了target可更加明晰其修饰的目标。

作用：用于描述注解的使用范围（即：被描述的注解可以用在什么地方）

取值(ElementType)有：

1.CONSTRUCTOR:用于描述构造器

2.FIELD:用于描述域

3.LOCAL_VARIABLE:用于描述局部变量

4.METHOD:用于描述方法

5.PACKAGE:用于描述包

6.PARAMETER:用于描述参数

7.TYPE:用于描述类、接口(包括注解类型) 或enum声明

使用示例：

```
/***
 *
 * 实体注解接口
 */
@Target(value = {ElementType.TYPE})
@Retention(value = RetentionPolicy.RUNTIME)
public @interface Entity {
    /***
     * 实体默认firstLevelCache属性为false
     * @return boolean
     */
    boolean firstLevelCache() default false;
    /***
     * 实体默认secondLevelCache属性为false
     * @return boolean
     */
    boolean secondLevelCache() default true;
    /***
     * 表名默认为空
     * @return String
     */
    String tableName() default "";
    /***
     * 默认以""分割注解
     */
    String split() default "";
}
```

@Retention

@Retention定义了该Annotation被保留的时间长短：某些Annotation仅出现在源代码中，而被编译器丢弃；而另一些却被编译在class文件中；编译在class文件中的Annotation可能会被虚拟机忽略，而另一些在class被装载时将被读取（请注意并不影响class的执行，因为Annotation与class在使用

上是被分离的）。使用这个meta-Annotation可以对 Annotation的"生命周期"限制。

作用：表示需要在什么级别保存该注释信息，用于描述注解的生命周期（即：被描述的注解在什么范围内有效）

取值（RetentionPoicy）有：

1.SOURCE:在源文件中有效（即源文件保留）

2.CLASS:在class文件中有效（即class保留）

3.RUNTIME:在运行时有效（即运行时保留）

使用示例：

```
/***
 * 字段注解接口
 */
@Target(value = {ElementType.FIELD})//注解可以被添加在属性上
@Retention(value = RetentionPolicy.RUNTIME)//注解保存在JVM运行时刻,能够在运行时刻通过反射API来获取到注解的信息
public @interface Column {
    String name();//注解的name属性
}
```

Column注解的的RetentionPolicy的属性值是RUTIME,这样注解处理器可以通过反射，获取到该注解的属性值，从而去做一些运行时的逻辑处理

@Documented

@Documented用于描述其它类型的annotation应该被作为被标注的程序成员的公共API，因此可以被例如javadoc此类的工具文档化。Documented是一个标记注解，没有成员。

@Inherited

@Inherited 元注解是一个标记注解，@Inherited阐述了某个被标注的类型是被继承的。如果一个使用了@Inherited修饰的annotation类型被用于一个class，则这个annotation将被用于该class的子类。

注意：@Inherited annotation类型是被标注过的class的子类所继承。类并不从它所实现的接口继承annotation，方法并不从它所重载的方法继承annotation。

当@Inherited annotation类型标注的annotation的Retention是RetentionPolicy.RUNTIME，则反射API增强了这种继承性。如果我们使用java.lang.reflect去查询一个@Inherited annotation类型的annotation时，反射代码检查将展开工作：检查class和其父类，直到发现指定的annotation类型被发现，或者到达类继承结构的顶层。

自定义注解

使用@interface自定义注解时，自动继承了java.lang.annotation.Annotation接口，由编译程序自动完成其他细节。在定义注解时，不能继承其他的注解或接口。@interface用来声明一个注解，其中的每一个方法实际上是声明了一个配置参数。方法的名称就是参数的名称，返回值类型就是参数的类型（返回值类型只能是基本类型、Class、String、enum）。可以通过default来声明参数的默认值。

定义注解格式：

public @interface 注解名 {定义体}

注解参数的可支持数据类型：

1.所有基本数据类型(int,float,boolean,byte,double,char,long,short)

2.String类型

3.Class类型

4.enum类型

5.Annotation类型

6.以上所有类型的数组

Annotation类型里面的参数该怎么设定：

第一,只能用public或默认(default)这两个访问权修饰.例如,String value();这里把方法设为defaul默认类型;

第二，参数成员只能用基本类型byte,short,char,int,long,float,double,boolean八种基本数据类型和 String,Enum,Class,annotations等数据类型，以及这一些类型的数组.例如，String value();这里的参数成员就为String;

第三，如果只有一个参数成员，最好把参数名称设为"value"，后加小括号.例：下面的例子FruitName注解就只有一个参数成员。

简单的自定义注解和使用注解实例：

示例1：

```
/***
 *主键注解接口
 */
@Target(value = {ElementType.FIELD})
@Retention(value = RetentionPolicy.RUNTIME)
public @interface Id {
}
```

示例2：

```
/**属性不需要被持久化注解**/
@Target(value = {ElementType.FIELD})
@Retention(value = RetentionPolicy.RUNTIME)
@Documented
public @interface Transient {
}
```

注解元素的默认值：

注解元素必须有确定的值，要么在定义注解的默认值中指定，要么在使用注解时指定，非基本类型的注解元素的值不可为null。因此， 使用空字符串或0作为默认值是一种常用的做法。这个约束使得处理器很难表现一个元素的存在或缺失的状态，因为每个注解的声明中，所有元素都存在，并且都具有相应的值，为了绕开这个约束，我们只能定义一些特殊的值，例如空字符串或者负数，一次表示某个元素不存在，在定义注解时，这已经成为一个习惯用法。

定义了注解，并在需要的时候给相关类，类属性加上注解信息，如果没有响应的注解信息处理流程，注解可以说是没有实用价值。如何让注解真真的发挥作用，

主要就在于注解处理方法，下一步我们将学习注解信息的获取和处理！

如果没有用来读取注解的方法和工作，那么注解也就不会比注释更有用处了。使用注解的过程中，很重要的一部分就是创建于使用注解处理器。Java SE5扩展了反射机制的API，以帮助程序员快速的构造自定义注解处理器。

注解处理器类库(java.lang.reflect.AnnotatedElement)：

Java使用Annotation接口来代表程序元素前面的注解，该接口是所有Annotation类型的父接口。除此之外，Java在java.lang.reflect 包下新增了AnnotatedElement接口，该接口代表程序中可以接受注解的程序元素，该接口主要有如下几个实现类：

Class：类定义

Constructor：构造器定义

Field：累的成员变量定义

Method：类的方法定义

Package：类的包定义

java.lang.reflect 包下主要包含一些实现反射功能的工具类，实际上，java.lang.reflect 包所有提供的反射API扩充了读取运行时Annotation信息的能力。当一个Annotation类型被定义为运行时的Annotation后，该注解才能是运行时可见，当class文件被装载时被保存在class文件中的Annotation才会被虚拟机读取。

AnnotatedElement 接口是所有程序元素（Class、Method和Constructor）的父接口，所以程序通过反射获取了某个类的AnnotatedElement对象之后，程序就可以调用该对象的如下四个个方法来访问Annotation信息：

方法1：<T extends Annotation> T getAnnotation(Class<T> annotationClass)：返回改程序元素上存在的、指定类型的注解，如果该类型注解不存在，则返回null。

方法2：Annotation[] getAnnotations()：返回该程序元素上存在的所有注解。

方法3：boolean is AnnotationPresent(Class<?extends Annotation> annotationClass)：判断该程序元素上是否包含指定类型的

注解，存在则返回true，否则返回false.

方法4：Annotation[] getDeclaredAnnotations()：返回直接存在于此元素上的所有注释。与此接口中的其他方法不同，该方法将忽略继承的注释。（如果没有注释直接存在于此元素上，则返回长度为零的一个数组。）该方法的调用者可以随意修改返回的数组；这不会对其他调用者返回的数组产生任何影响。

一个简单的注解处理器：

```
/***********注解声明***************/


/**
 * 水果名称注解
 * @author peida
 *
 */
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface FruitName {
    String value() default "";
}

/**
 * 水果颜色注解
 * @author peida
 *
 */
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface FruitColor {
    /**
     * 颜色枚举
     * @author peida
     *
     */
     public enum Color{ BULE,RED,GREEN};
```

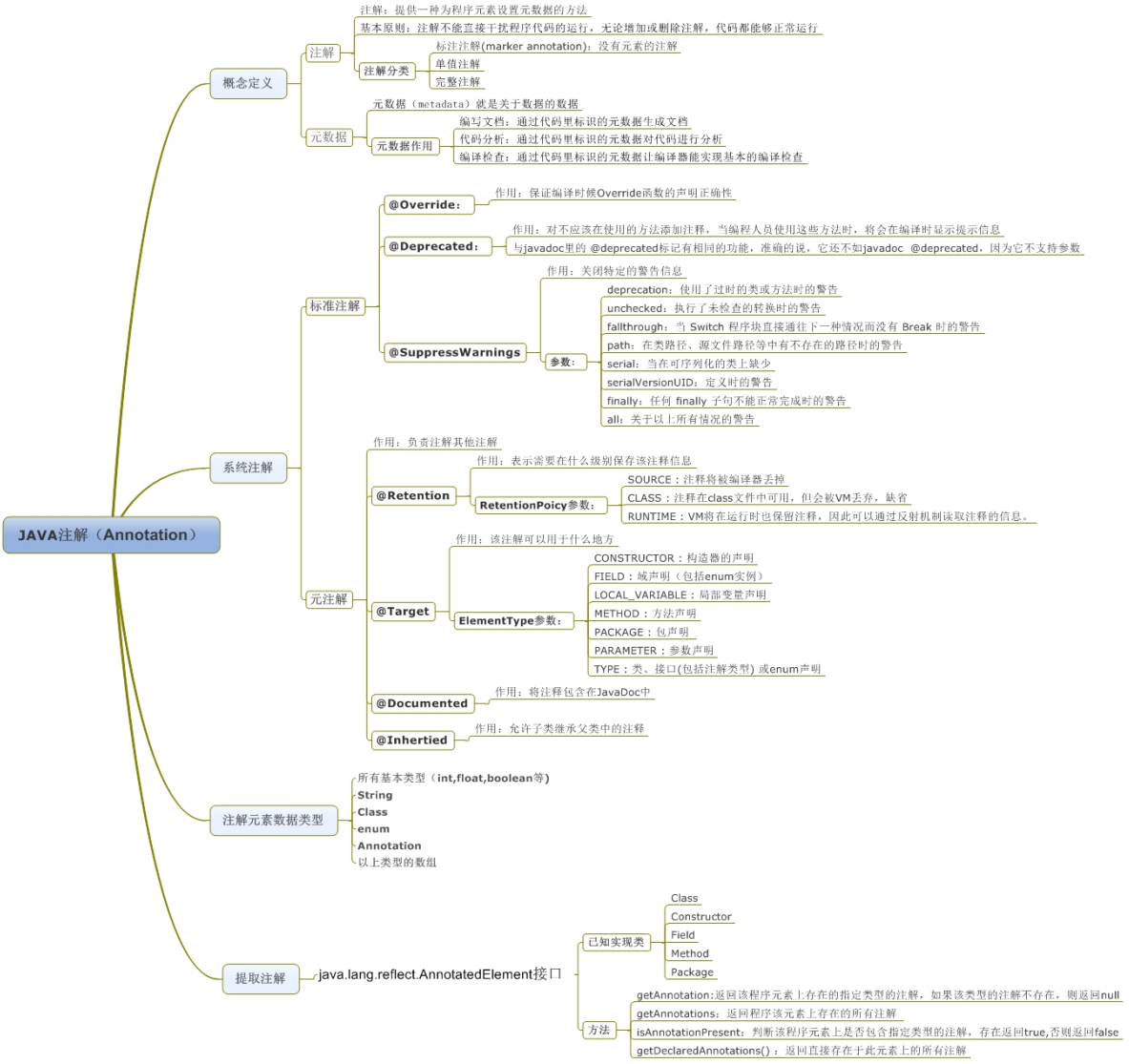
```
    /**
     * 颜色属性
     * @return
     */
    Color fruitColor() default Color.GREEN;

}

/**
 * 水果供应者注解
 * @author peida
 *
 */
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface FruitProvider {
    /**
     * 供应商编号
     * @return
     */
     public int id() default -1;

    /**
     * 供应商名称
     * @return
     */
     public String name() default "";

    /**
     * 供应商地址
     * @return
     */
     public String address() default "";
}

/***********注解使用***************/
```

```
public class Apple {

    @FruitName("Apple")
    private String appleName;

    @FruitColor(fruitColor=Color.RED)
    private String appleColor;

    @FruitProvider(id=1,name="陕西红富士集团",address="陕西省西安市延安路89号红富士大厦")
    private String appleProvider;

    public void setAppleColor(String appleColor) {
        this.appleColor = appleColor;
    }
    public String getAppleColor() {
        return appleColor;
    }

    public void setAppleName(String appleName) {
        this.appleName = appleName;
    }
    public String getAppleName() {
        return appleName;
    }

    public void setAppleProvider(String appleProvider) {
        this.appleProvider = appleProvider;
    }
    public String getAppleProvider() {
        return appleProvider;
    }

    public void displayName(){
        System.out.println("水果的名字是：苹果");
    }
```

```
}

/***********注解处理器***************/

public class FruitInfoUtil {
    public static void getFruitInfo(Class<?> clazz){

        String strFruitName=" 水果名称：";
        String strFruitColor=" 水果颜色：";
        String strFruitProvicer="供应商信息：";

        Field[] fields = clazz.getDeclaredFields();

        for(Field field :fields){
            if(field.isAnnotationPresent(FruitName.class)){
                FruitName fruitName = (FruitName)
field.getAnnotation(FruitName.class);
                strFruitName=strFruitName+fruitName.value();
                System.out.println(strFruitName);
            }
            else if(field.isAnnotationPresent(FruitColor.class)){
                FruitColor fruitColor= (FruitColor)
field.getAnnotation(FruitColor.class);

strFruitColor=strFruitColor+fruitColor.fruitColor().toString();
                System.out.println(strFruitColor);
            }
            else
if(field.isAnnotationPresent(FruitProvider.class)){
                FruitProvider fruitProvider= (FruitProvider)
field.getAnnotation(FruitProvider.class);
                strFruitProvicer=" 供应商编号："+fruitProvider.id()+" 供应
商名称："+fruitProvider.name()+" 供应商地址："+fruitProvider.address();
```

```
                System.out.println(strFruitProvicer);
            }
        }
    }
}

/***********输出结果***************/
public class FruitRun {

    /**
     * @param args
     */
     public static void main(String[] args) {

        FruitInfoUtil.getFruitInfo(Apple.class);

    }

}

=======================================
 水果名称：Apple
 水果颜色：RED
 供应商编号：1 供应商名称：陕西红富士集团 供应商地址：陕西省西安市延安路89号红富士大厦
```

Java注解的基础知识点（见下面导图）基本都过了一遍，下一篇我们通过设计一个基于注解的简单的ORM框架，来综合应用和进一步加深对注解的各个知识点的理解和运用。


JAVA注解（Annotation）
概念定义
注解
注解：提供一种为程序元素设置元数据的方法
基本原则：注解不能直接干扰程序代码的运行，无论增加或删除注解，代码都能够正常运行
注解分类
标注注解(marker annotation)：没有元素的注解
单值注解
完整注解
元数据
元数据（metadata）就是关于数据的数据
元数据作用
编写文档：通过代码里标识的元数据生成文档
代码分析：通过代码里标识的元数据对代码进行分析
编译检查：通过代码里标识的元数据让编译器能实现基本的编译检查
系统注解
标准注解
@Override：
作用：保证编译时候Override函数的声明正确性
@Deprecated：
作用：对不应该在使用的方法添加注释，当编程人员使用这些方法时，将会在编译时显示提示信息
与javadoc里的 @deprecated标记有相同的功能，准确的说，它还不如javadoc @deprecated，因为它不支持参数
@SuppressWarnings
作用：关闭特定的警告信息
参数：
deprecation：使用了过时的类或方法时的警告
unchecked：执行了未检查的转换时的警告
fallthrough：当 Switch 程序块直接通往下一种情况而没有 Break 时的警告
path：在类路径、源文件路径等中有不存在的路径时的警告
serial：当在可序列化的类上缺少
serialVersionUID：定义时的警告
finally：任何 finally 子句不能正常完成时的警告
all：关于以上所有情况的警告
元注解
作用：负责注解其他注解
@Retention
作用：表示需要在什么级别保存该注释信息
RetentionPoicy参数：
SOURCE：注释将被编译器丢掉
CLASS：注释在class文件中可用，但会被VM丢弃，缺省
RUNTIME：VM将在运行时也保留注释，因此可以通过反射机制读取注释的信息。
@Target
作用：该注解可以用于什么地方
ElementType参数：
CONSTRUCTOR：构造器的声明
FIELD：域声明（包括enum实例）
LOCAL_VARIABLE：局部变量声明
METHOD：方法声明
PACKAGE：包声明
PARAMETER：参数声明
TYPE：类、接口(包括注解类型) 或enum声明
@Documented
作用：将注释包含在JavaDoc中
@Inhertied
作用：允许子类继承父类中的注释
注解元素数据类型
所有基本类型（int,float,boolean等）
String
Class
enum
Annotation
以上类型的数组
提取注解
java.lang.reflect.AnnotatedElement接口
已知实现类
Class
Constructor
Field
Method
Package
方法
getAnnotation:返回该程序元素上存在的指定类型的注解，如果该类型的注解不存在，则返回null
getAnnotations：返回程序该元素上存在的所有注解
isAnnotationPresent：判断该程序元素上是否包含指定类型的注解，存在返回true,否则返回false
getDeclaredAnnotations()：返回直接存在于此元素上的所有注解