java枚举

**关键字enum可以将一组具名的值的有限集合创建为一种新的类型，而这些具有名的值程序的常规组件使用**

例一：

public enum Shrubery {  
 *GROUND*,*CRAWLING*,*HANGING*}

public class EnumClass {  
 public static void main(String[] args) {  
 for (Shrubery s : Shrubery.*values*()) {  
 System.*out*.println(s + ":" + s.ordinal());//枚举的顺序值  
 System.*out*.println(s.compareTo(Shrubery.*CRAWLING*));//返回的是两个枚举值的顺序之差  
 System.*out*.println(s.equals(Shrubery.*CRAWLING*));//是不是同一个枚举  
 System.*out*.println(s == Shrubery.*CRAWLING*);//同上==判断是否是同一个实例  
 System.*out*.println(s.getDeclaringClass());  
 System.*out*.println(s.name());  
 }  
 }  
}

values()方法返回enum包含的实例数组，而且有严格的顺序但是在Enum类中并没有values（）方法，values方法是由编译器添加的

**静态导入enum**

public enum Spiciness {  
 *NOT*,*MILD*,*MEDIUM*,*HOT*,*FLAMING*}

import static main.Enum.Spiciness.\*;  
public class Burrito {  
 Spiciness degree;  
 public Burrito(Spiciness degree) {  
 this.degree = degree;  
 }  
 @Override  
 public String toString() {  
 return "Burrito{" +  
 "degree=" + degree +  
 '}';  
 }  
 public static void main(String[] args) {  
 System.*out*.println(new Burrito(NOT));  
 System.*out*.println(new Burrito(MEDIUM));  
 System.*out*.println(new Burrito(HOT));  
 }  
}

**向enum中添加新的方法**

public enum OzWitch {  
 *WEST*("a"), *NORTH*("b"), *EAST*("c"), *SOUTH*("d");  
 private String description;  
  
 private OzWitch(String description) {  
 this.description = description;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public static void main(String[] args) {  
 for (OzWitch witch : OzWitch.*values*()) {  
 System.*out*.println(witch + ":" + witch.getDescription());  
 }  
 }  
}

enum除了不能继承以外基本就是一个常规类，所以可以向enum中添加新的方法甚至是main（）函数，enum和普通类没有太大区别除了有少许限制以外，都可以吧enum认为是一个普通类，从构造器private可以看出enum实际上在enum之外并不允许创建任何实例，就算不声明成private，也只能在enum内部使用其构造器构建enum实例，一旦enum定义结束，编译器不允许构造器构建任何实例

**switch语句中的enum**

在Java7之前，switch只能支持 byte、short、char、int或者其对应的封装类以及Enum类型

在Java7中加入了switch（string）

也可以利用enum实现switc（String）

public enum Animal {  
 *dog*, *cat*, *bear*;  
  
 public static Animal getAnimal(String animal) {  
 return *valueOf*(animal.toLowerCase());  
 }  
}

public class Client {  
 public void caseAnimal(String animal) {  
 switch (Animal.*getAnimal*(animal)) {  
 case *cat*:  
 System.*out*.println("this is a cat" + Animal.*cat*);  
 break;  
 case *dog*:  
 System.*out*.println("this is a dog");  
 break;  
 case *bear*:  
 System.*out*.println("this is a bear");  
 break;  
 }  
 }  
  
 /\*\*  
 \* @param args  
 \*/  
 public static void main(String[] args) {  
 Client client = new Client();  
 client.caseAnimal("cat");  
 }  
}

**enum不能继承其他的类，却可以实现一个或者多个接口，而且enum也可以像普通类一样做向上转型，enum是一个精简的类，可以消除很多重复的代码。也可以减少重复带来的麻烦**

public class Enums {  
 private static Random *rand* = new Random(47);  
  
 public static <T extends Enum<T>> T random(Class<T> ec) {  
 return *random*(ec.getEnumConstants());  
 }  
  
 public static <T> T random(T[] values) {  
 return values[*rand*.nextInt(values.length)];  
 }  
}

public enum Activity {  
 *SITTING*, *LYING*, *STANDING*, *HOPPING*, *RUNING*, *DODGING*, *JUMPING*, *FALLING*, *FLYING*;  
  
 public static void main(String[] args) {  
 for (int i = 0; i < 20; i++)  
 System.*out*.println(Enums.*random*(Activity.class));  
 }  
}

**使用接口组织枚举，因为无法使用enum继承子类。如果要想扩展enum中的元素，在一个接口内部创建枚举，以此进行元素分组，嵌套**

public enum SecurityCategory {  
 *STOCK*(Security.Stock.class), *BOND*(Security.Bond.class);  
 Security[] values;  
  
 SecurityCategory(Class<? extends Security> kind) {  
 values = kind.getEnumConstants();  
 }  
  
 interface Security {  
 enum Stock implements Security {  
 *SHORT*, *LONG*, *MARGIN* }  
  
 enum Bond implements Security {  
 *MUNICIPAL*, *JUNK* }  
 }  
  
 public Security randomSelection() {  
 return Enums.*random*(values);  
 }  
  
 public static void main(String[] args) {  
 for (int i = 0; i < 10; i++) {  
 SecurityCategory securityCategory = Enums.*random*(SecurityCategory.class);  
 System.*out*.println(securityCategory + ":" + securityCategory.randomSelection());  
 }}}

**使用EnumSet高性能enum集合**

public enum AlarmPoint {  
 *START1*, *START2*, *LOBBY*, *OFFICE1*, *OFFICE2*, *OFFICE3*, *OFFICE4*,  
 *BATHROOM*, *UTILITY*, *KITCHEN*}

public class EnumSets {  
 public static void main(String[] args) {  
 EnumSet<AlarmPoint> points = EnumSet.*noneOf*(AlarmPoint.class);  
 System.*out*.println("新建" + points);  
 points.addAll(EnumSet.*of*(*START1*, *START2*, *KITCHEN*));  
 System.*out*.println("加入元素" + points);  
 points = EnumSet.*allOf*(AlarmPoint.class);  
 points.removeAll(EnumSet.*of*(*START1*, *START2*, *KITCHEN*));  
 System.*out*.println("移除部分元素" + points);  
 points.removeAll(EnumSet.*range*(*OFFICE1*, *OFFICE4*));  
 System.*out*.println("移除部分元素" + points);  
 points = EnumSet.*complementOf*(points);  
 System.*out*.println("加入全部元素" +points);  
 }  
}

**EnumMap要求key必须来自enum，内部用数组实现所有速度快，只能用enum的实例作为键来使用put（），其他和map用法一致**

**常量相关的方法**

public enum ConstandSpacialMethod {  
 *DATA\_TIME* {  
 String getInfo() {  
 return DateFormat.*getDateTimeInstance*().format(new Date());  
 }  
 },  
 *CLASS\_PATH* {  
 String getInfo() {  
 return System.*getenv*("CLASSPATH");  
 }  
 },  
 *VERSION* {  
 String getInfo() {  
 return System.*getProperty*("java.version");  
 }  
 };  
  
 abstract String getInfo();  
  
 public static void main(String[] args) {  
 for (ConstandSpacialMethod csm : *values*()) {  
 System.*out*.println(csm.getInfo());  
 }  
 }  
  
}

**虽然使用enum 有一些限制，但是我们还是把它看作类**

**使用enum的职责链**

职责链模式处理多过程结果确定的任务，比如邮件处理，excel处理。enum使其结构清晰，便于理解

class Mail {  
 enum GeneralDelivery {  
 *YES*, *NO1*, *NO2*, *NO3*, *NO4*, *NO5* }  
  
 enum Scannability {  
 *UNSCANNABLE*, *YES1*, *YES2*, *YES3*, *YES4*, *YES5* }  
  
 enum Readability {  
 *ILLEGIBLE*, *YES1*, *YES2*, *YES3*, *YES4*, *YES5* }  
  
 enum Address {  
 *INCRRECT*, *OK1*, *OK2*, *OK3*, *OK4*, *OK5*, *OK6* }  
  
 enum ReturnAddress {  
 *MISSING*, *OK1*, *OK2*, *OK3*, *OK4*, *OK5* }  
  
 GeneralDelivery generalDelivery;  
 Scannability scannability;  
 Readability readability;  
 Address address;  
 ReturnAddress returnAddress;  
 static long *counter*;  
 long id = *counter*++;  
  
 @Override  
 public String toString() {  
 return "Mail:" + id;  
 }  
  
  
 public String details() {  
 return "Mail{" +  
 "generalDelivery=" + generalDelivery +  
 ", scannability=" + scannability +  
 ", readability=" + readability +  
 ", address=" + address +  
 ", returnAddress=" + returnAddress +  
 ", id=" + id +  
 '}';  
 }  
  
 public static Mail randomMail() {  
 Mail mail = new Mail();  
 mail.generalDelivery = Enums.*random*(GeneralDelivery.class);  
 mail.scannability = Enums.*random*(Scannability.class);  
 mail.readability = Enums.*random*(Readability.class);  
 mail.address = Enums.*random*(Address.class);  
 mail.returnAddress = Enums.*random*(ReturnAddress.class);  
 return mail;  
 }  
  
 public static Iterable<Mail> generator(final int count) {  
 return new Iterable<Mail>() {  
 int n = count;  
  
 @Override  
 public Iterator<Mail> iterator() {  
 return new Iterator<Mail>() {  
 @Override  
 public boolean hasNext() {  
 return n-- > 0;  
 }  
  
 @Override  
 public Mail next() {  
 return *randomMail*();  
 }  
  
 @Override  
 public void remove() {  
 throw new UnsupportedOperationException();  
 }  
 };  
 }  
  
 };  
 }  
  
}  
  
public class postOffice {  
 enum MailHandler {  
 *GENERAL\_DELIVERY* {  
 @Override  
 boolean handle(Mail mail) {  
 switch (mail.generalDelivery) {  
 case *YES*:  
 System.*out*.println("Using general deliver for " + mail);  
 return true;  
 default:  
 return false;  
 }  
 }  
 },  
 *MACHINE\_SCAN* {  
 @Override  
 boolean handle(Mail mail) {  
 switch (mail.scannability) {  
 case *UNSCANNABLE*:  
 return false;  
 default:  
 switch (mail.address) {  
 case *INCRRECT*:  
 return false;  
 default:  
 System.*out*.println("Deliver " + mail + "automatically");  
 return true;  
 }  
 }  
 }  
 },  
 *VISUAL\_INSPECTION* {  
 @Override  
 boolean handle(Mail mail) {  
 switch (mail.readability) {  
 case *ILLEGIBLE*:  
 return false;  
 default:  
 switch (mail.address) {  
 case *INCRRECT*:  
 return false;  
 default:  
 System.*out*.println("Deliver " + mail + "normally");  
 return true;  
 }  
 }  
 }  
 },  
 *RETURN\_TO\_SENDER* {  
 @Override  
 boolean handle(Mail mail) {  
 switch (mail.returnAddress)  
 {  
 case *MISSING*:return false;  
 default:  
 System.*out*.println("return to sender ");  
 return true;  
 }  
 }  
 };  
  
 abstract boolean handle(Mail mail);  
 }  
 static void handle(Mail m)  
 {  
 for(MailHandler mailHandler : MailHandler.*values*())  
 {  
 if(mailHandler.handle(m))  
 return;  
 System.*out*.println("dead letter");  
 }  
 }  
  
 public static void main(String[] args) {  
 for(Mail mail:Mail.*generator*(10))  
 {  
 System.*out*.println(mail.details());  
 *handle*(mail);  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 }  
 }  
  
  
}

**enu与状态机**