

**1 @RestController**

如果需要返回JSON，XML或自定义mediaType内容到页面，则需要在对应的方法上加上@ResponseBody注解。



另：



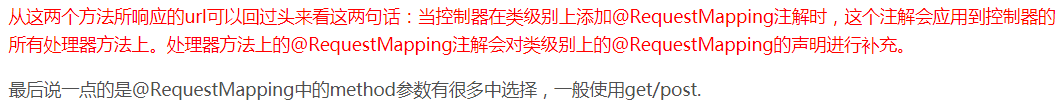
**2 @RequestMapping 配置url映射**

@RequestMapping此注解即可以作用在控制器的某个方法上，也可以作用在此控制器类上。

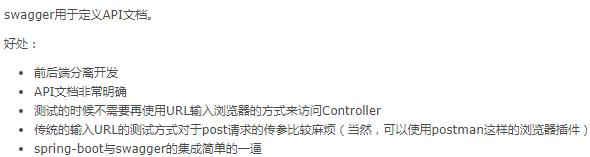
当控制器在类级别上添加@RequestMapping注解时，这个注解会应用到控制器的所有处理器方法上。处理器方法上的@RequestMapping注解会对类级别上的@RequestMapping的声明进行补充。







**3 @Api和@ApiOperation**







**4 @AllArgsConstructo**

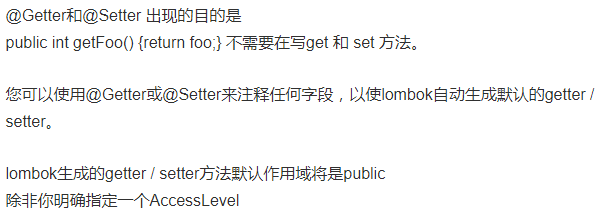
@AllArgsConstructor(onConstructor = @\_\_({@Autowired}))

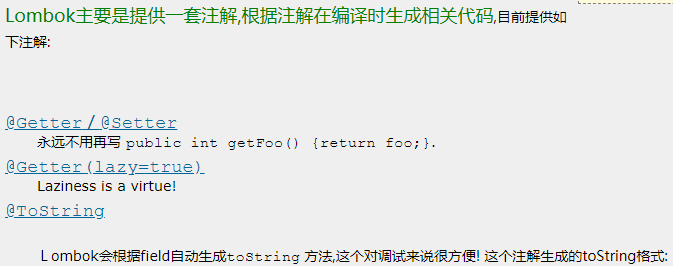
相当于



是为了解决注入的类太多的问题，不然每一个要注入的类都要加上一个@Autowired注解

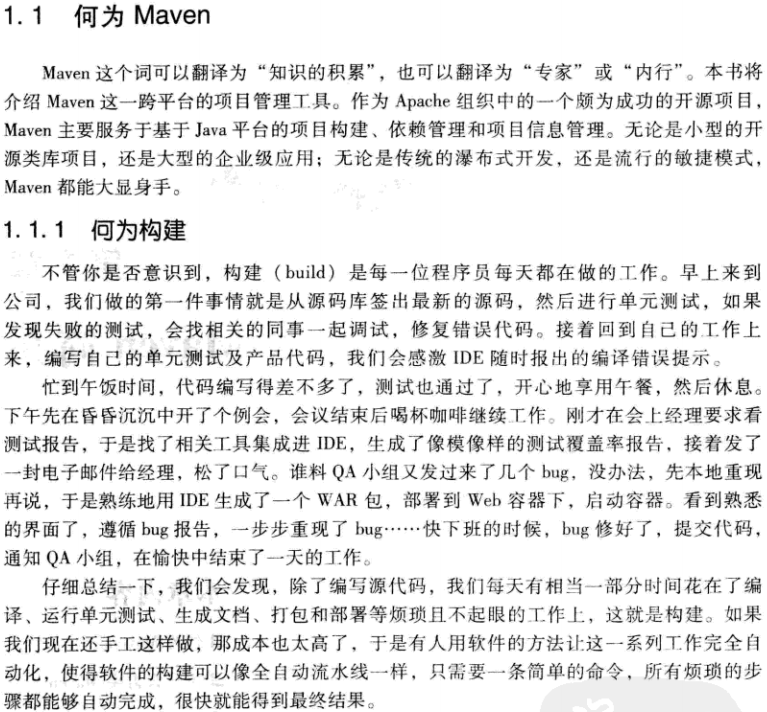
**5 @Getter、@Setter**

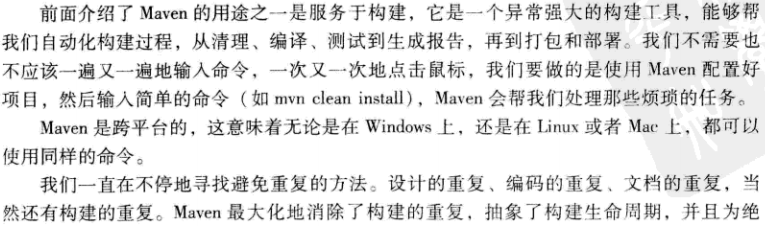


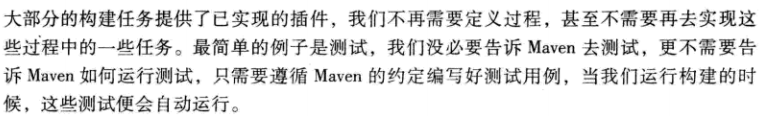


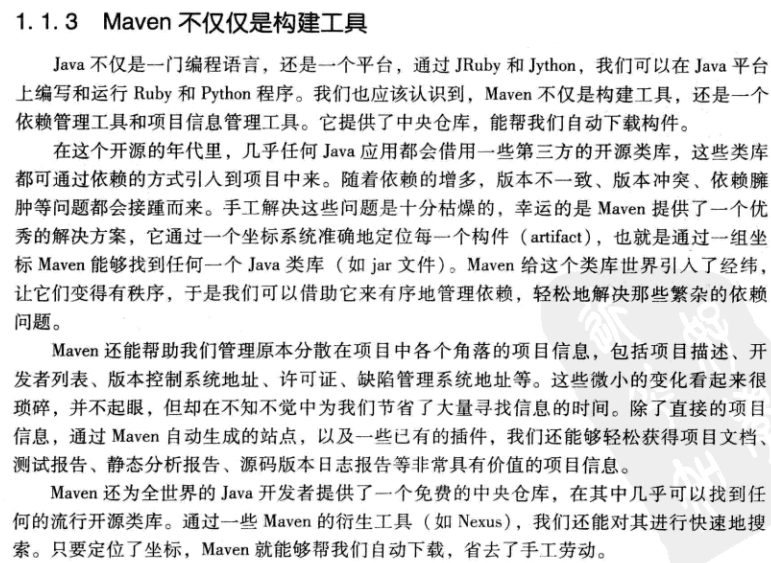
详见8 使用lombok简化代码

**6 Maven简介**









**7 Spring Data JPA**

JPA（Java Persistence API，Java持久化API），定义了对象关系映射（Object Relation Mapping，ORM）以及实体对象持久化的标准接口。

**JPA是一套接口规范**，JPA的实现主要包括 JBoss 的 Hibernate EntityManager、Oracle 捐献给 Eclipse 社区的 EclipseLink和Apache 的 OpenJPA 等。

ORM框架的出现使得我们能够以**面向对象的方式来操作关系数据库**，但是长久以来实现应用系统中的数据访问层依然是一件非常繁琐的事情，简单的查询也往往涉及大量重复代码。Spring Data JPA在则基于JPA进一步简化了数据访问层的实现，它提供了一种类似于**声明式编程**的方式，开发者只需要编写数据访问接口（称为Repository），Spring Data JPA就能基于接口中的方法命名自动地生成实现。

首先来实现一个Hello World应用

### 7.1 添加Spring Data JPA的依赖

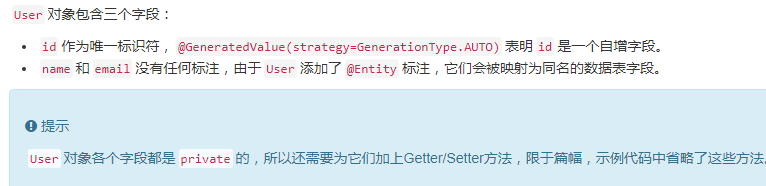


### 7.2 Spring Boot中数据源配置



### 7.3 定义简单的实体对象





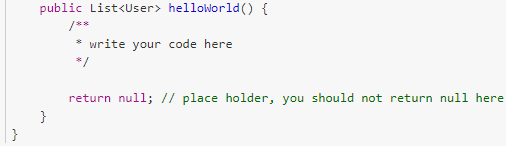
### 7.4 创建对象访问接口：Repository



### 7.5 使用Repository

创建一个DataInitialization类来使用UserRepository接口。





在DataInitialization的run方法中，我们通过UserRepository往数据库中插入了5条数据，并且使用repository.findAll()将数据库中的所有数据查询了出来。这样，我们就通过简单的3个类，完成了对User表的操作！

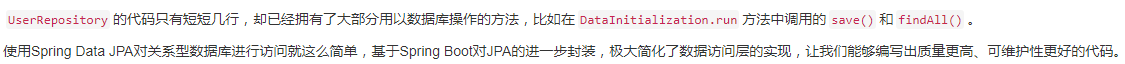
DataInitialization同时还实现了CommandLineRunner接口，该接口是Spring Boot为我们提供的。实现该接口的所有类都将在Spring Boot应用启动后被调用。也就是说，启动我们的应用后，DataInitialization.run就会被调用，这样我们就能在控制台看到相应的输出。

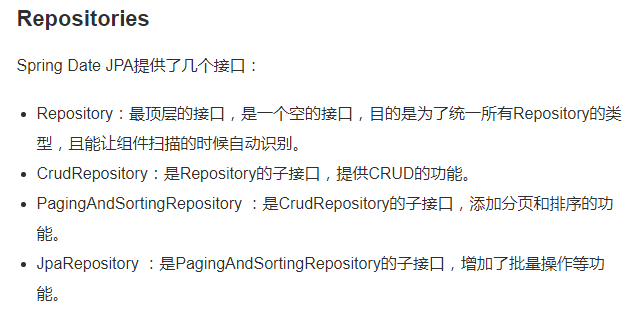
### 7.6 运行Spring Boot应用





### 7.7 总结

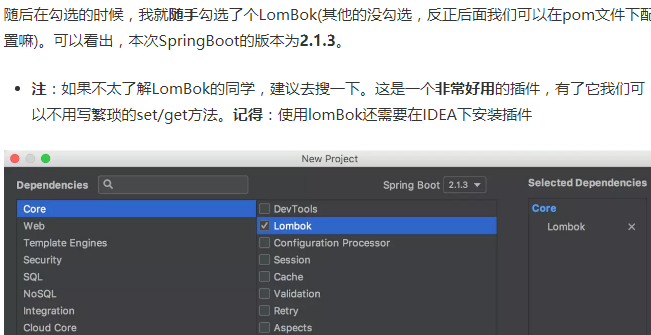




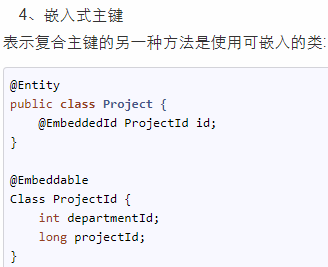
**8 使用lombok简化代码**

**详见：**

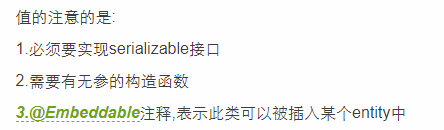
[**http://kriszhang.com/lombok/**](http://kriszhang.com/lombok/)

****

**9 关于复合主键：JPA的Embeddable注解**

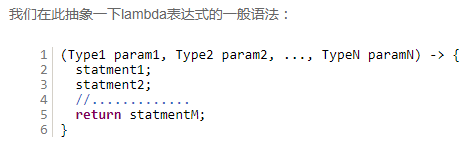


主键字段是在可嵌入类中定义的。该实体包含一个单独的主键字段，该字段用@EmbeddedId 注释，并包含一个可嵌入类的实例。当使用这个表单时，没有定义一个单独的ID类，因为可嵌入的类本身可以表示完整的主键值。



**10 Lambda表达式和Stream()**

定义：一个不用被绑定到一个标识符上，并且可能被调用的函数。这个解释还不够通俗，lambda表达式可以这样定义（不精确，自己的理解）：一段带有输入参数的可执行语句块。

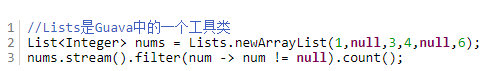


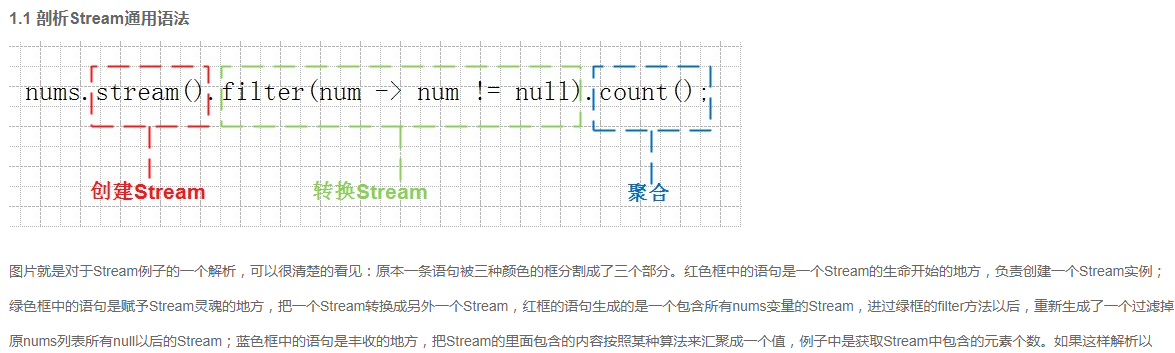


Stream概念:

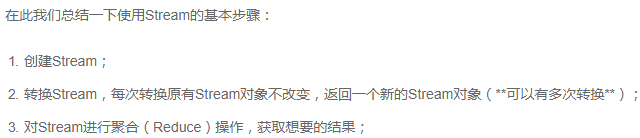
1. Stream是元素的集合，这点让Stream看起来用些类似Iterator；
2. 可以支持顺序和并行的对原Stream进行汇聚的操作；

**大家可以把Stream当成一个高级版本的Iterator。原始版本的Iterator，用户只能一个一个的遍历元素并对其执行某些操作；高级版本的Stream，用户只要给出需要对其包含的元素执行什么操作，比如“过滤掉长度大于10的字符串”、“获取每个字符串的首字母”等，具体这些操作如何应用到每个元素上，就给Stream就好了！**





图片就是对于Stream例子的一个解析，可以很清楚的看见：原本一条语句被三种颜色的框分割成了三个部分。红色框中的语句是一个Stream的生命开始的地方，负责创建一个Stream实例；绿色框中的语句是赋予Stream灵魂的地方，把一个Stream转换成另外一个Stream，红框的语句生成的是一个包含所有nums变量的Stream，进过绿框的filter方法以后，重新生成了一个过滤掉原nums列表所有null以后的Stream；蓝色框中的语句是丰收的地方，把Stream的里面包含的内容按照某种算法来汇聚成一个值，例子中是获取Stream中包含的元素个数。

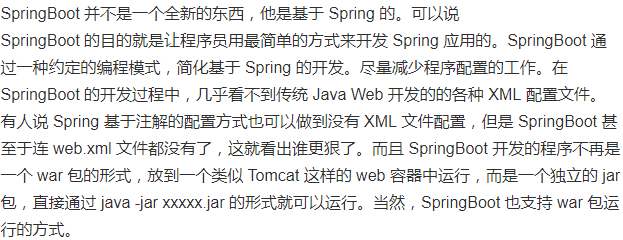


**11 整合Spring Data JPA与Spring MVC: 分页和排序**

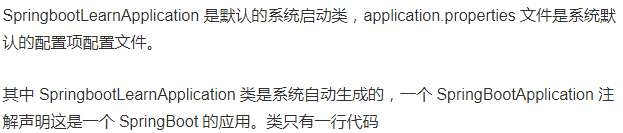




**12 spring boot**









**13 jpa Optional<T>**



**14 drools规则引擎**

详见博客：

<http://www.ityouknow.com/drools/2017/08/07/drools-started.html>

### 1 drool概述



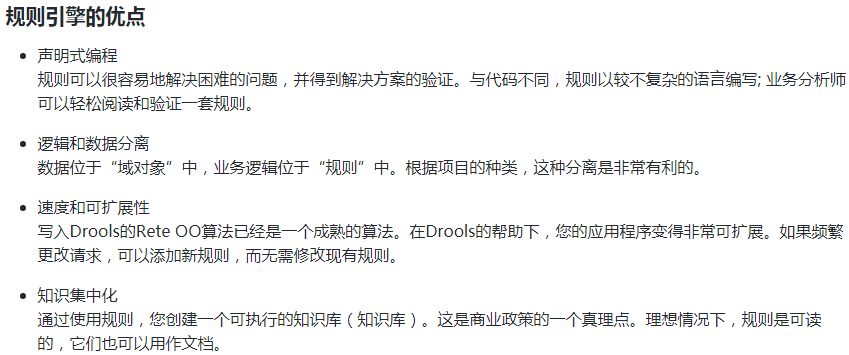
规则引擎起源于基于规则的专家系统，而基于规则的专家系统又是专家系统的其中一个分支。专家系统属于人工智能的范畴，它模仿人类的推理方式，使用试探性的方法进行推理，并使用人类能理解的术语解释和证明它的推理结论。

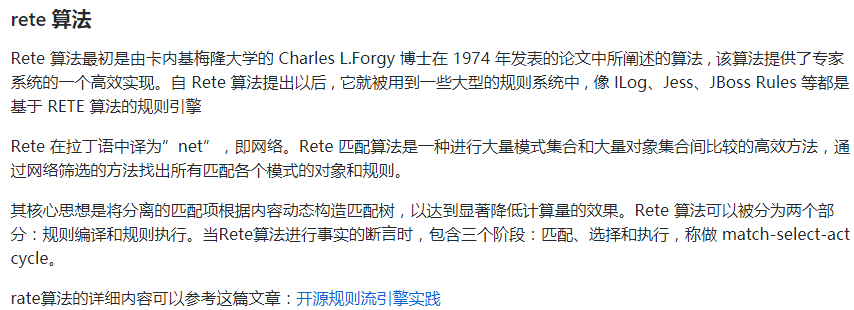
利用它就可以在应用系统中分离商业决策者的商业决策逻辑和应用开发者的技术决策，并把这些商业决策放在中心数据库或其他统一的地方，让它们能在运行时可以动态地管理和修改，从而为企业保持灵活性和竞争力提供有效的技术支持。

在需求里面我们往往把约束，完整性，校验，分支流等都可以算到业务规则里面。在规则引擎里面谈的业务规则重点是谈当满足什么样的条件的时候，需要执行什么样的操作。因此一个完整的业务规则包括了条件和触发操作两部分内容。而引擎是事物内部的重要的运行机制，规则引擎即重点是解决规则如何描述，如何执行，如何监控等一系列问题。

规则引擎由推理引擎发展而来，是一种嵌入在应用程序中的组件，实现了将业务决策从应用程序代码中分离出来，并使用预定义的语义模块编写业务决策。**接受数据输入，解释业务规则，并根据业务规则做出业务决策。**

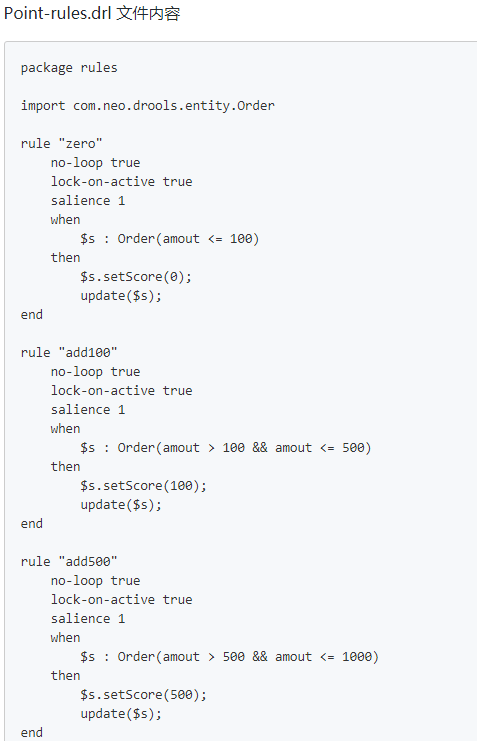
java开源的规则引擎有：Drools、Easy Rules、Mandarax、IBM ILOG。使用最为广泛并且开源的是Drools

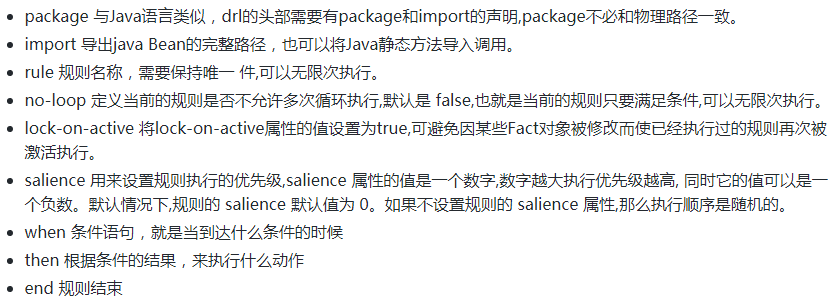






drools有专门的规则语法drl，就是专门描述活动的规则是如何执行的





这个规则文件就是描述了，当符合什么条件的时候，应该去做什么事情，每当规则有变动的时候，我们只需要修改规则文件，然后重新加载即可生效。





### 2 drools－kie API 和kmodule.xml 文件详解

详见：

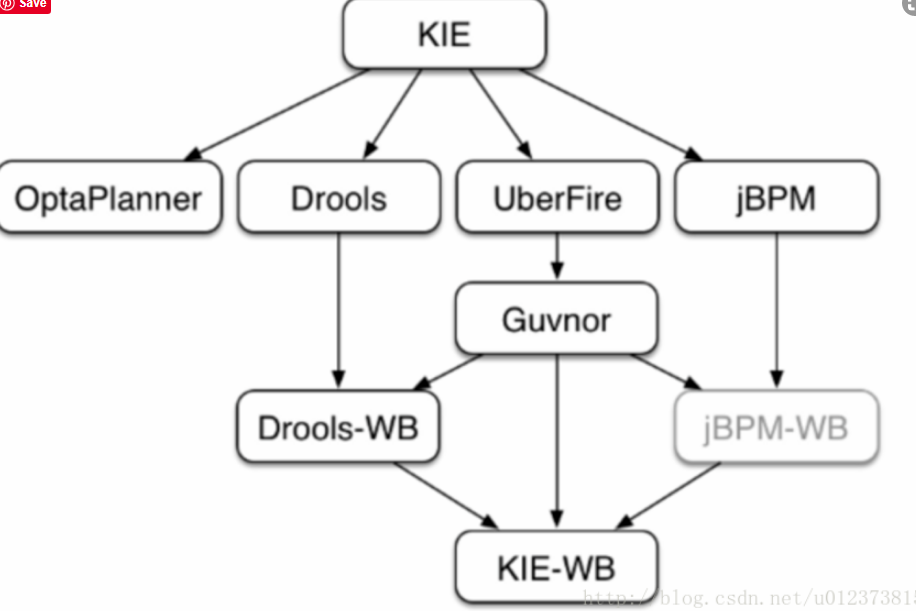
<https://blog.csdn.net/u012373815/article/details/53907340>

#### 2.1 什么是KIE

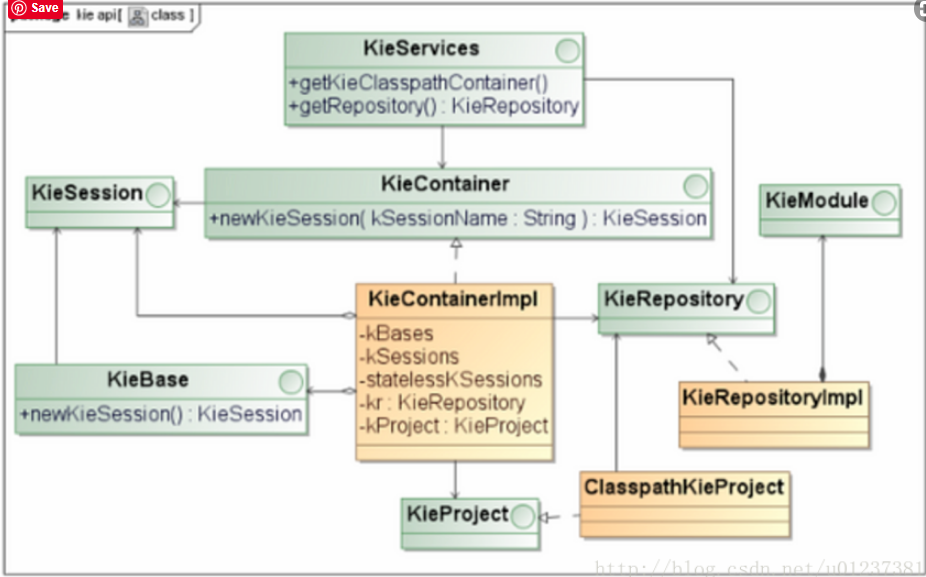
KIE是jBoss里面一些相关项目的统称，下图就是KIE代表的一些项目，其中我们比较熟悉的就有jBPM和Drools。

这些项目都有一定的关联关系，并且存在一些通用的API，比如说涉及到构建（building）、部署（deploying）和加载（loading）等方面的，这些API就都会以KIE作为前缀来表示这些是通用的API。前面看到的一些KieServices、KieContainer、KieSession类就都是KIE的公共API。

总的来说，就是jBoss通过KIE将jBPM和Drools等相关项目进行了一个整合，统一了他们的使用方式。像KieServices这些KIE类就是整合后的结果，在Drools中这样使用，在jBPM里面也是这样使用。



#### 2.2 kieServices和kieContainer



通过KieServices对象得到一个KieContainer，利用kieContainer对象创建一个新的KieSession，

创建session的时候我们传入了一个name：“ksession-name”，这个字符串很眼熟吧，这个就是我们定义的kmodule.xml文件中定义的ksession的name。kieContainer根据kmodule.xml定义的ksession的名称找到KieSession的定义，然后创建一个KieSession的实例。

KieSession就是一个到规则引擎的链接，通过它就可以跟规则引擎通讯，并且**发起执行规则**的操作。即**KieSession就是应用程序跟规则引擎进行交互的会话通道。**

**通过kSession.insert方法来将事实（Fact）插入到引擎中，也就是Working Memory中。然后通过kSession.fireAllRules方法来通知规则引擎执行规则。**



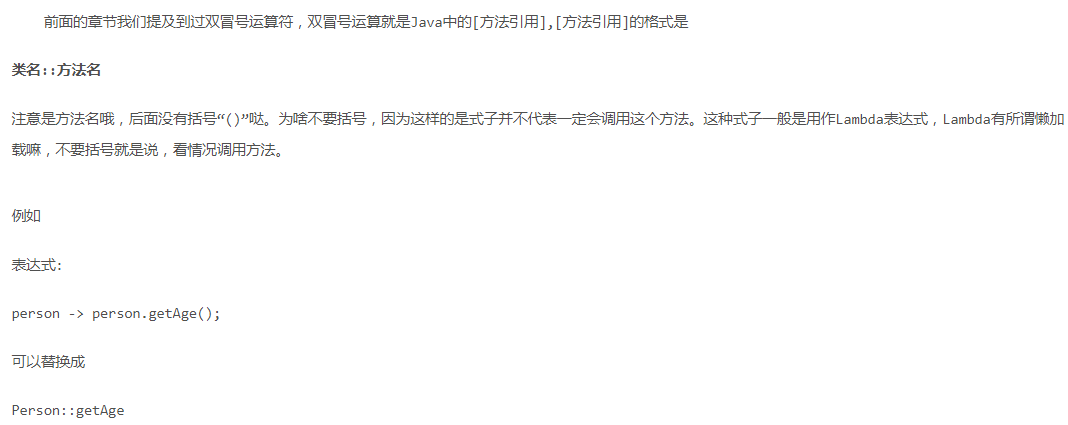
**KieSession:**

KieSession就是一个跟Drools引擎打交道的会话，其基于KieBase创建，它会包含运行时数据，包含“事实 Fact”，并对运行时数据事实进行规则运算。我们通过KieContainer创建KieSession是一种较为方便的做法，其实他本质上是从KieBase中创建出来。的。

KieSession就是应用程序跟规则引擎进行交互的会话通道。

创建KieBase是一个成本非常高的事情，KieBase会建立知识（规则、流程）仓库，而创建KieSession则是一个成本非常低的事情，所以KieBase会建立缓存，而KieSession则不必

**15 双冒号::**



**16 Array.asList()用法 ：**

