

## 年轻的架构师



**Pprun**

[quest.run@gmail.com](mailto:quest.run@gmail.com)

<http://hi.baidu.com/quest2run>

### 修订历史

版本	时间	描述
0.01	2009年10月01日	Start to compose the draft
0.02	2009年12月30日	Added data Model
0.1	2010年02月15日	Most content have been done: the document structure, etc.
0.5	2010年05月03日	Updated according to the latest implementation
0.99	2010年08月05日	Thoroughly updated and published to Internet
0.99.1	2010年10月02日	1. Typos fixing 2. Added 'Eclipse CVS' into the source check out section
1.0-beta	2011年02月02日	1. Introduced OAuth security solution for REST api (then deprecated Spring Security) 2. project has been moved to GoogleCode: <a href="http://code.google.com/p/hjpetstore/">http://code.google.com/p/hjpetstore/</a>



The following technologies are being used in the current implementation:

- [Maven based build architecture](#)
- [jQuery fisheye](#)
- [kaptcha](#)
- [OAuth](#)
- [Spring MVC 3](#)
- [Hibernate 3.5](#)
- [JBoss Cache 3 \(alternative infinispn\)](#)
- [JMS External Integration](#)
- [GlassFish 3 cluster](#)
- [Mysql fail-over and cluster](#)

- [Zabbix](#) / [Zapcat](#)

## 年轻的架构师

1	
修订历史.....	1
1 关于本文.....	5
2 获取 Hjpetstore 源代码.....	5
...2.1 CVS Client .....	6
...2.2 NetBeans CVS .....	6
...2.3 Eclipse CVS.....	8
3 Hjpetstore 配置与部署	
11	
4 需求说明书	
12	
...4.1 宠物商城.....	12
5 需求分析	
13	
...5.1 名词分析.....	13
...5.2 形容词分析.....	13
...5.3 动词分析.....	13
6 数据建模	
14	
...6.1 名词.....	14
...6.1.1 数据库设计范式.....	15
...6.2 形容词.....	17
...6.3 ERD.....	18
...6.3.1 ERD 图例.....	20
7 Use Case	
21	
...7.1 动词分析.....	21
...7.2 补上通用的 use case.....	22
...7.3 以 actor 为中心的 use case 设计.....	22
...7.3.1 unsigned user.....	22
...7.3.2 signed user.....	22
...7.3.3 supplier.....	23
...7.3.4 Hjpetstore.....	23
...7.4 将依赖的内部或外部组件视作黑盒.....	23
8 Hibernate 映射	
23	
...8.1 使用合成 ( synthetic identifier ) 或代理 ( surrogate ) 主键.....	24

...8.2 不要使用复合主键 ( composite primary key ) .....	24
...8.3 强烈建议引入 <version> 或者 <timestamp> .....	27
...8.4 使用中间表映射多对多关系.....	27
...8.5 尽量不要使用 class inheritance 映射.....	28
...8.6 永远不要使用 List.....	29
...8.7 EntityInterceptor.....	29
...8.8 不要指定任何 lazy=false.....	31
...8.8.1 Criteria fetch mode.....	32
...8.8.2 HQL join fetch.....	32
...8.9 显式指定外键名.....	32
...8.10 关于 equals() 和 hashCode() 方法.....	32
...8.10.1 相同与相等.....	33
...8.10.2 业务键(business key).....	33
...8.11 二级缓存.....	34
...8.11.1 二级缓存配置集中化.....	34
...8.11.2 ehcache.....	34
...8.11.3 Jboss cache 3.....	38
...8.11.4 infinispn.....	42
...8.11.5 查询缓存.....	44
...8.12 历史遗留数据.....	44
9 安全策略.....	
44.....	
...9.1 不要将密码存成明文.....	45
...9.2 使用 Kaptcha.....	46
...9.2.1 kaptcha 配置.....	47
...9.3 使用 https (SSL 通道).....	49
...9.4 密码算法及其参数的选择.....	49
...9.5 OAuth.....	50
...9.5.1 存在的安全标准.....	50
...9.5.2 基于数字签名(signature) 的安全方案.....	50
...9.5.3 OAuth.....	55
...9.5.4 应用举例.....	55
...9.6 RSA.....	60
...9.6.1 为什么要引入 RSA .....	60
...9.7 Spring Security 访问控制.....	62
...9.8 PCI (Payment Card Industry).....	62
10 事务.....	
62.....	
...10.1 事务的基本概念.....	62
...10.1.1 事务隔离级别.....	62
...10.1.2 事务传播行为.....	63
...10.1.3 只读提示.....	64
...10.1.4 事务超时周期.....	64
...10.2 Java EE 三种事务模型.....	64
...10.2.1 无事务.....	65
...10.2.2 编程式事务.....	65

...10.2.3 声明式事务.....	65
...10.3 Spring 四种事务策略.....	66
...10.3.1 前端事务策略.....	66
...10.3.2 Service 层事务策略.....	67
...10.3.3 高并发事务策略.....	67
...10.3.4 高性能涉外事务.....	67
11 部署架构	
68	
...11.1 物理拓扑.....	69
...11.2 负载均衡.....	69
...11.3 数据库失效转移.....	70
...11.3.1 Mysql JDBC 配置.....	70
...11.4 应用服务器集群.....	70
...11.4.1 GlassFish v3.....	70
...11.4.2 OpenMQ.....	70
12 SOA / ROA 应用架构	
71	
...12.1 数据层 DAO.....	71
...12.1.1 No Open Session in View pattern.....	71
...12.1.2 eagerly fetch mode.....	72
...12.2 JAXB.....	73
...12.2.1 RESTful controller 消息转换.....	74
...12.2.2 Batch xml 文件存储.....	75
...12.3 基于 Spring MVC 3 RESTful Controller.....	75
...12.3.1 基于 Annotation 声明式配置.....	75
...12.3.2 在 view 的定义中引用 jaxb2Marshaller.....	76
...12.3.3 定义视图解析器.....	76
...12.3.4 REST HTTP METHOD.....	77
13 集成	
81	
...13.1 内部组件集成.....	81
...13.1.1 Security REST Service.....	81
...13.2 外部集成.....	88
...13.2.1 Payment Partner 集成.....	88
...13.2.2 JMS 异步松散耦合: Supplier 集成.....	89
...13.2.3 JMS ObjectMessage.....	98
14 BI (商业智能) 与批处理	
100	
...14.1 定时启动.....	100
...14.2 任务 POJO.....	101
15 异常处理	
101	
...15.1 JSP/Servlet 表示层 error page.....	102

...15.1.1 首先定义 error code 映射.....	102
...15.1.2 Error page.....	102
...15.1.3 引用 Error page.....	102
...15.2 Controller MappingExceptionHandler.....	102
...15.2.1 MappingExceptionHandler.....	102
...15.2.2 AnnotationMethodHandlerExceptionHandler.....	103
...15.3 Service 层 RuntimeException.....	105
...15.4 Dao 层 DataAccessException.....	105
...15.4.1 使用 @Respository annotation.....	105
...15.4.2 DataAccessException 异常转换.....	105
16 测试	
106	
...16.1 基本的单元测试.....	106
...16.1.1 FileUtils 代码.....	106
...16.1.2 FileUtilsTest 代码.....	107
...16.2 集成测试.....	108
...16.2.1 Spring TestContext .....	108
...16.2.2 抽象测试基类.....	108
...16.2.3 Dao 测试.....	111
...16.2.4 事务支持.....	112
...16.2.5 Service 测试.....	112
...16.4 Mock 测试.....	112
...16.5 负载测试.....	112
17 调优与监控	
113	
...17.1 Netbeans Profile.....	113
...17.1.1 线程分析.....	113
...17.1.2 内存分析.....	115
...17.1.3 方法执行时间分析.....	116
...17.2 基于 JMX 的监控体系.....	117
...17.2.1 Zabbix .....	117
...17.2.2 Zapcat 作为 JMX agent 与 Zabbix 的桥梁.....	117
18 为什么写这篇文章	
120	

## 1 关于本文

- 本文是一篇 基于 Hibernate Spring 开源框架的 JAVA 企业应用的设计与实现说明书
- 本文采用纯开源技术: 从基础的 load balancer 到 Base64 工具性的代码
- 本文面象一线工程师, 内容布满源代码
- 本文不适合 JAVA 初学者, 起码要使用 Spring, Hibernate 写过至少一个实际的项目

- 本文不会是老练的设计师的菜肴
- 本文以 [hjpetstore2.0](#) 为例展开
- 文档最近更新见[我的博客](#)

## 2 获取 *Hjpetstore* 源代码

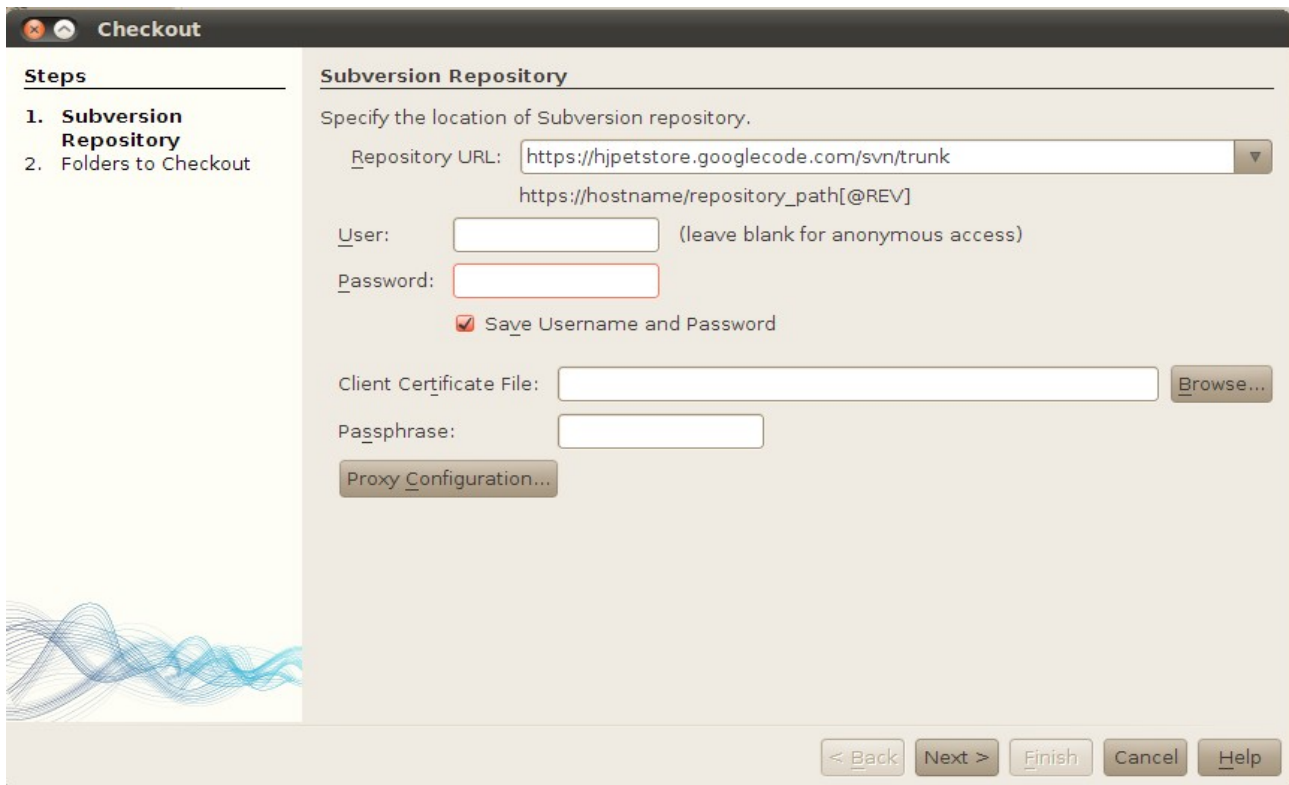
- SVN Client
- NetBeans SVN
- Eclipse SVN

### ...2.1 SVN Client

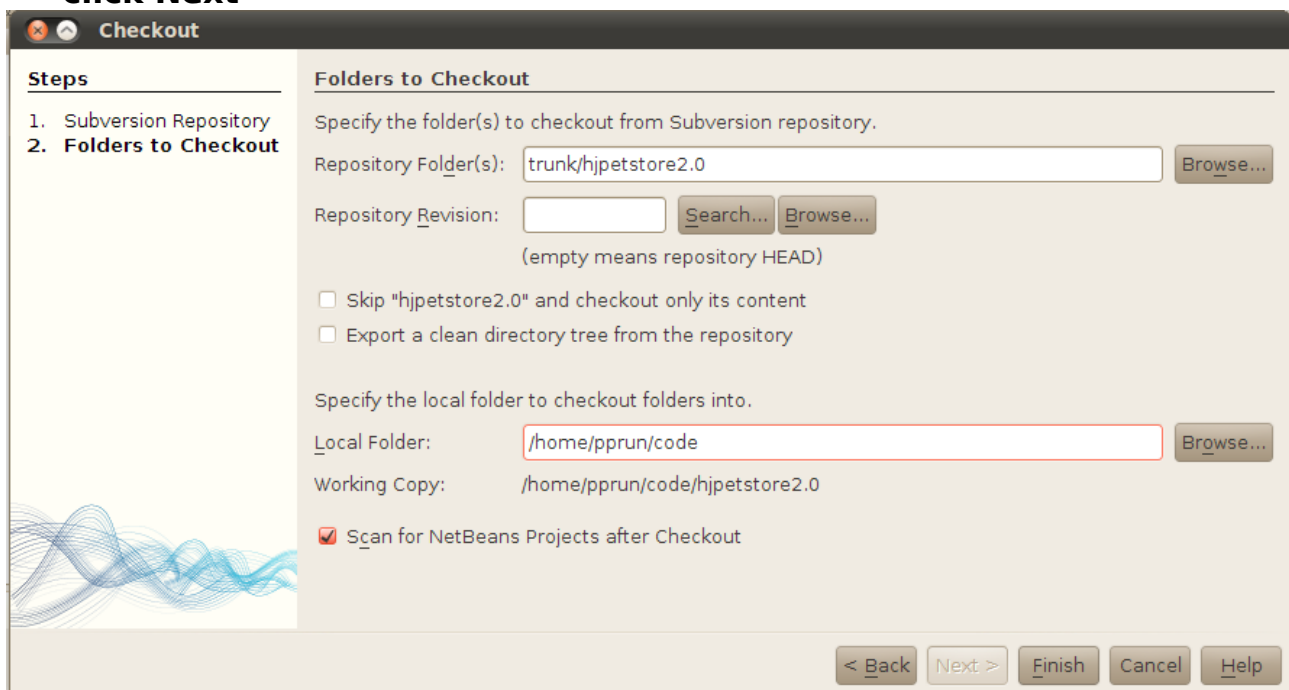
```
pprun@pprun-laptop:~$ mkdir sout
pprun@pprun-laptop:~$ cd svnout
pprun@pprun-laptop:~/svnout$ svn checkout
http://hjpetstore.googlecode.com/svn/trunk/ hjpetstore-read-only
```

### ...2.2 NetBeans Subversion

**Team | SubVersion | Checkout...**



**click Next**



**click Finish**

Once done, NetBeans will ask you to open the project. **Select Yes** to open

the project.

### ...2.3 Eclipse Subclipse

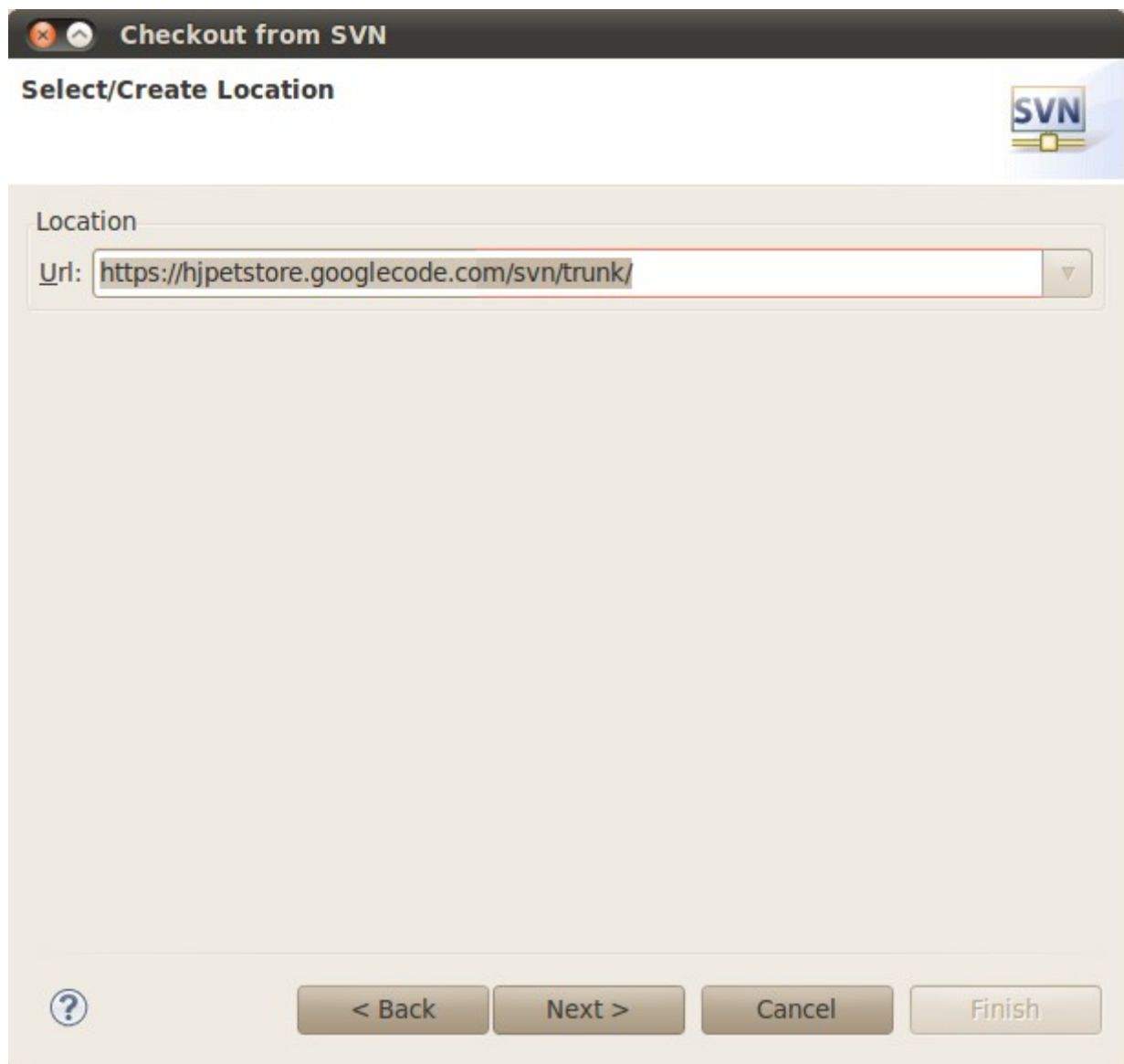
First of all, you need have the [Subclipse](#) and maven2 eclipse plugin installed (see [m2eclipse](#))

#### **File | New | Other...**

-> SVN | Checkout Projects from SVN

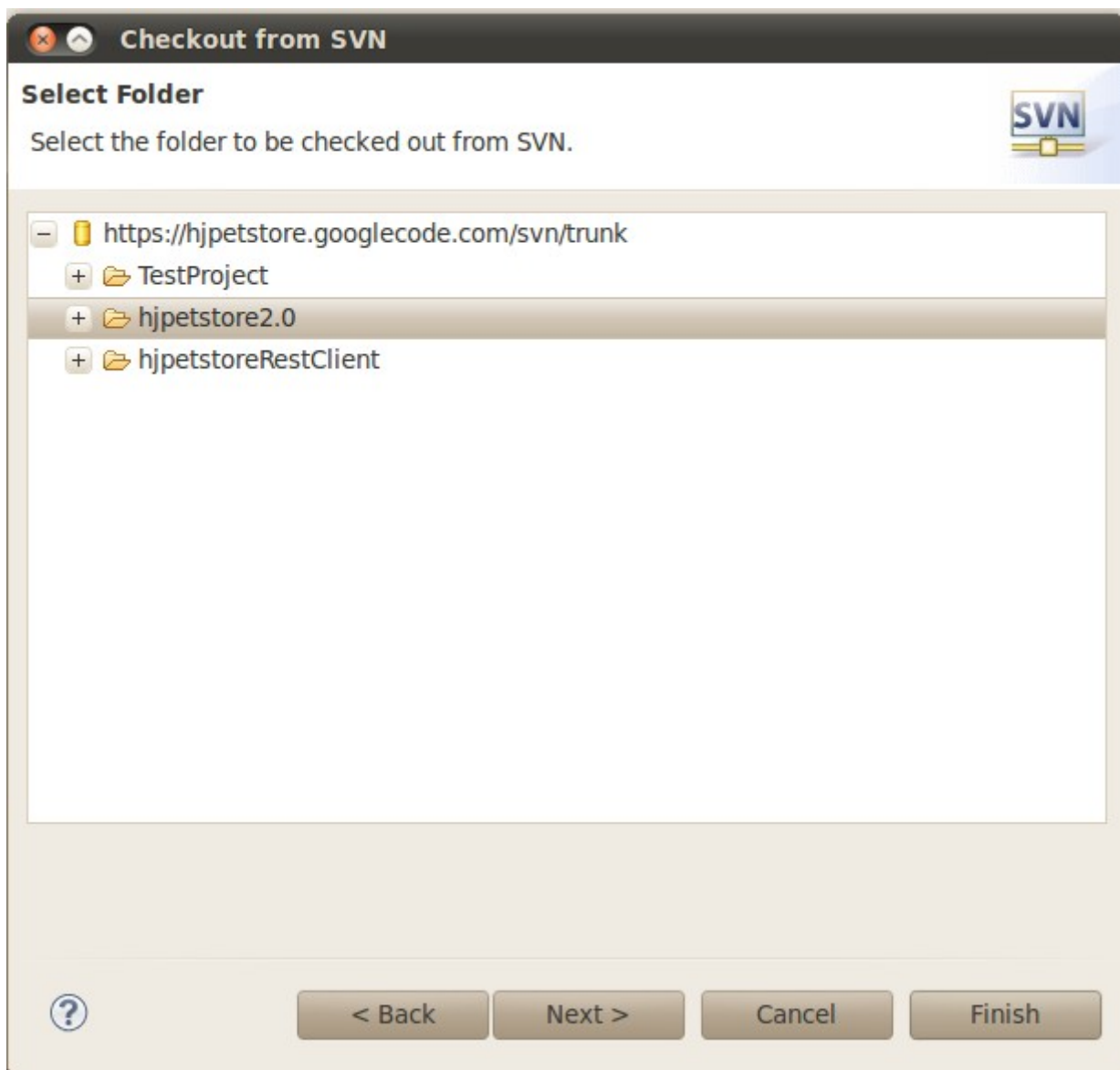
-> Create a new repository location (if

`https://hjpetstore.googlecode.com/svn/trunk/` does not exist in your eclipse setting)

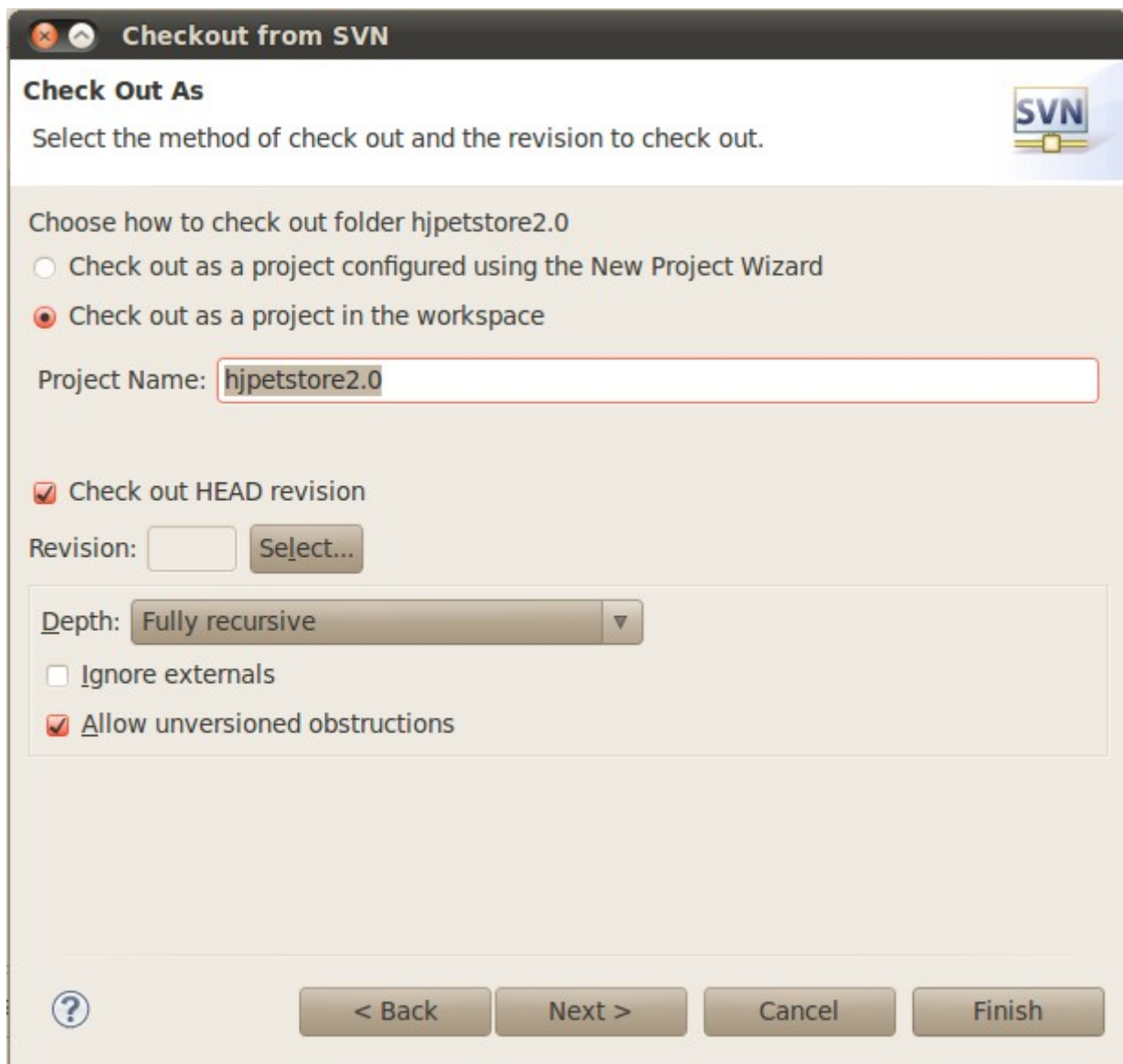




## Select Module

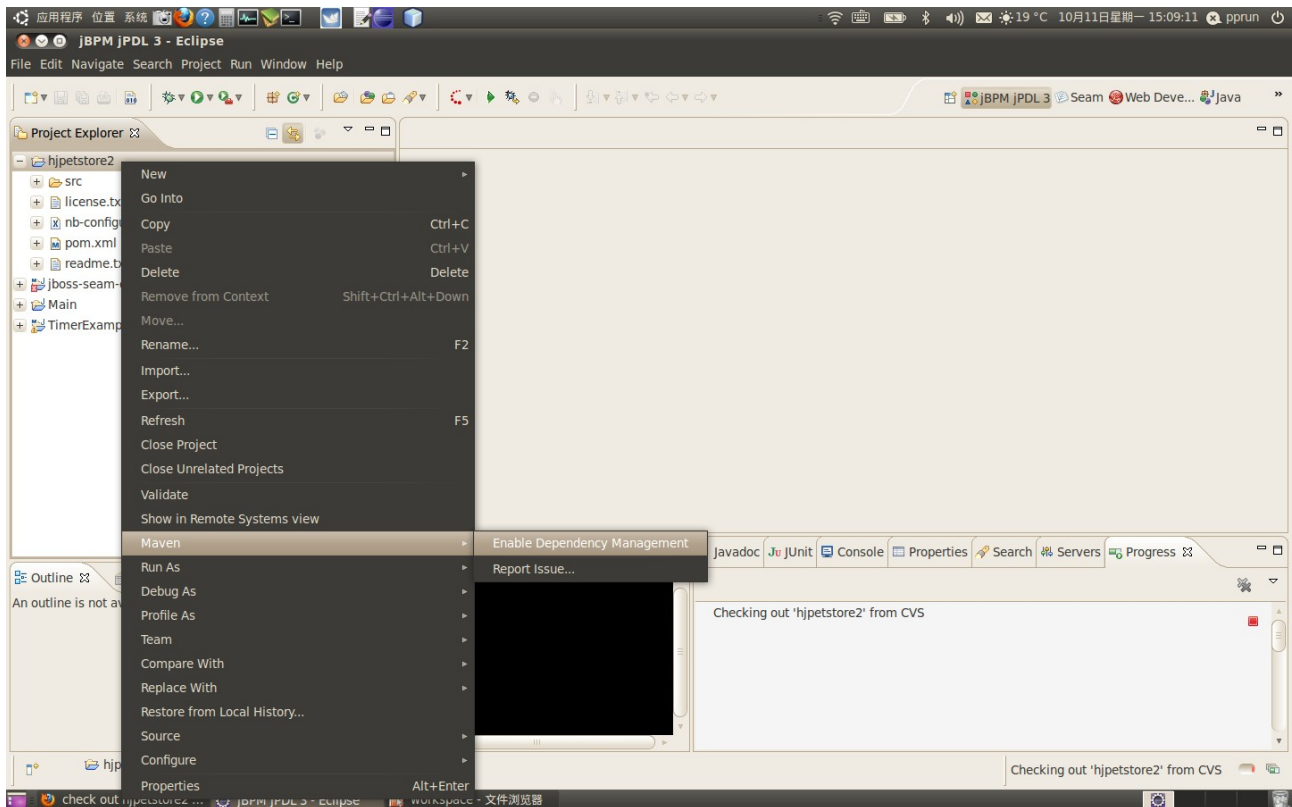


## Check Out As



## Enable Maven

(Optional, the setting of eclipse may make this step automatically)



### 3 *Hjpetstore* 配置与部署

- Assumptions
- Steps

#### **Assumptions:**

1. you have [check out hjpetstore2 source code](#)
2. you have installed maven
3. you have installed glassfish and set it up according to [Create GlassFish JNDI resource for hjpetstore 2.0](#)
4. you have completed the task described in `hjpetstore2.0/src/main/resources/conf/maven-readme.txt`  
(install two jar into local maven repository as they are not available in open community maven repositories)

#### **Steps:**

1. cd to the checkout folder  
`cd /home/pprun/cvsout/hjpetstore/hjpetstore2/`

2. `mvn clean package -Dmaven.test.skip=true`  
depends on the status of your network, it may take a while to finish at first time
3. start GlassFish default domain  
`/home/pprun/java/glassfish-3.0.1/bin/asadmin start-domain`
4. deploy it by:  
`pprun@pprun-laptop:/home/pprun/java/glassfish-3.0.1/bin/asadmin`  
`deploy target/hjpetstore.war`
5. access the application in browser: `http://localhost:8080/`

BTW, to **undeploy** it:  
`pprun@pprun-laptop:asadmin undeploy`

## 4 需求说明书

- 不要期望需求说明书会很详细
- 不要期望需求说明书文档化了就不会改变
- 花上足够的时间，一字一句地推敲说明书

### ...4.1 宠物商城

1. 宠物商城计划在线交易 5 种宠物商品：鱼类，狗类，爬行类，猫类，鸟类，宠物由宠物提供商提供
2. 非注册用户可以浏览，搜寻商城内的所有宠物
3. 注册用户可以下订单购买商城内的宠物，采用与电子银行系统合作的方式进行在线支付
4. 同一订单可以同时包含多种不同种类的商品
5. 用户可查看自己的所有的定单历史记录
6. 宠物提供商负责宠物的配送，宠物提供商将跟踪订单状态并更新到系统中，以方便用户追踪
7. 配送地址可以和注册的地址不同
8. 当商品库存不足时，用户仍然可以下单购买，系统将通知所有的提供商，提供商将根据他们的状况更新系统的库存

## 5 需求分析

- 名词将形成系统的主体，即实体与对象
- 形容词构成了实体的多态性与对可变性的封装（以及业务流程的分支）

- 动词描述系统功能，以及决定系统的行为

### ...5.1 名词分析

名词分析是找出系统的对象与实体，以方便后续的建模设计

1. 宠物 商城目前计划在线交易 5 种宠物商品：鱼类，狗类，爬行类，猫类，鸟类，宠物由宠物提供商提供
2. 非注册用户可以浏览，搜寻商城内的所有宠物
3. 注册用户可以下订单购买商城内的宠物，采用与电子银行系统合作的方式进行在线支付
4. 交易成功后，系统将向用户注册邮箱发送通知邮件，同时向宠物提供商发送新订单通知
5. 同一订单可以同时包含多种不同种类的商品
6. 用户可查看自己的所有的定单历史记录
7. 宠物提供商负责宠物的配送，宠物提供商将跟踪订单状态并更新到系统中，以方便用户追踪
8. 配送地址可以和注册的地址不同
9. 当商品库存不足时，用户仍然可以下单购买，系统将通知所有的提供商，提供商将根据他们的状况更新系统的库存

### ...5.2 形容词分析

部分形容词将决定对象或实体的可变性或多态性，以及业务流程的分支

1. 宠物商城目前计划在线交易 5 种宠物商品：鱼类，狗类，爬行类，猫类，鸟类，宠物由宠物提供商提供
2. 非注册用户可以浏览，搜寻商城内的所有宠物
3. 注册用户可以下订单购买商城内的宠物，采用与电子银行系统合作的方式进行在线支付
4. 交易成功后，系统将向用户注册邮箱发送通知邮件，同时向宠物提供商发送新订单通知
5. 同一订单可以同时包含多种不同种类的商品
6. 用户可查看自己的所有的定单历史记录
7. 宠物提供商负责宠物的配送，宠物提供商将跟踪订单状态并更新到系统中，以方便用户追踪
8. 配送的地址可以和注册的地址不同
9. 当商品库存不足时，用户仍然可以下单购买，系统将通知所有的提供商，提供商将根据他们的状况更新系统的库存

### ...5.3 动词分析

动词与系统的 use case 息息相关，反映对象的行为以及系统所承载的功能

1. 宠物 商城目前计划在线交易 5 种宠物商品：鱼类，狗类，爬行类，猫类，鸟类，宠物由宠物提供商提供
2. 非注册用户可以浏览，搜寻商城内的所有宠物
3. 注册（注册）用户可以下订单购买商城内的宠物，采用与电子银行系统合作的方式进行

- 行在线支付
4. 交易成功后，系统将向用户注册邮箱发送通知邮件，同时向宠物提供商发送新订单通知
  5. 同一订单可以同时包含多种不同种类的商品
  6. 用户可查看自己的所有的定单历史记录
  7. 宠物提供商负责宠物的配送，宠物提供商将跟踪订单状态并更新到系统中，以方便用户追踪
  8. 配送的地址可以和注册的地址不同
  9. 当商品库存不足时，用户仍然可以下单购买，系统将通知所有的提供商，提供商将根据他们的状况更新系统的库存

## 6 数据建模

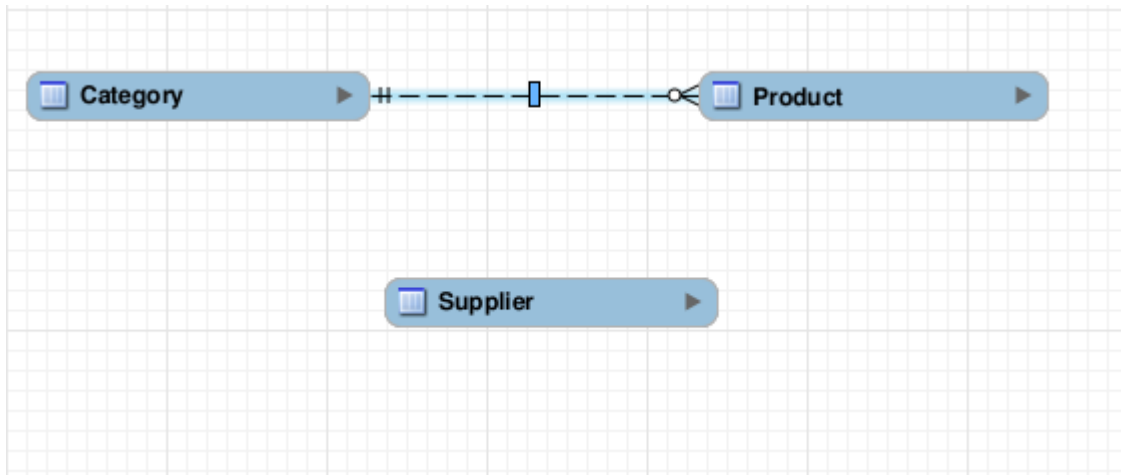
- 基于名词分析的结果进行数据建模
- 别忘了形容词
- 尽早与团队成员 review ERD

### ...6.1 名词

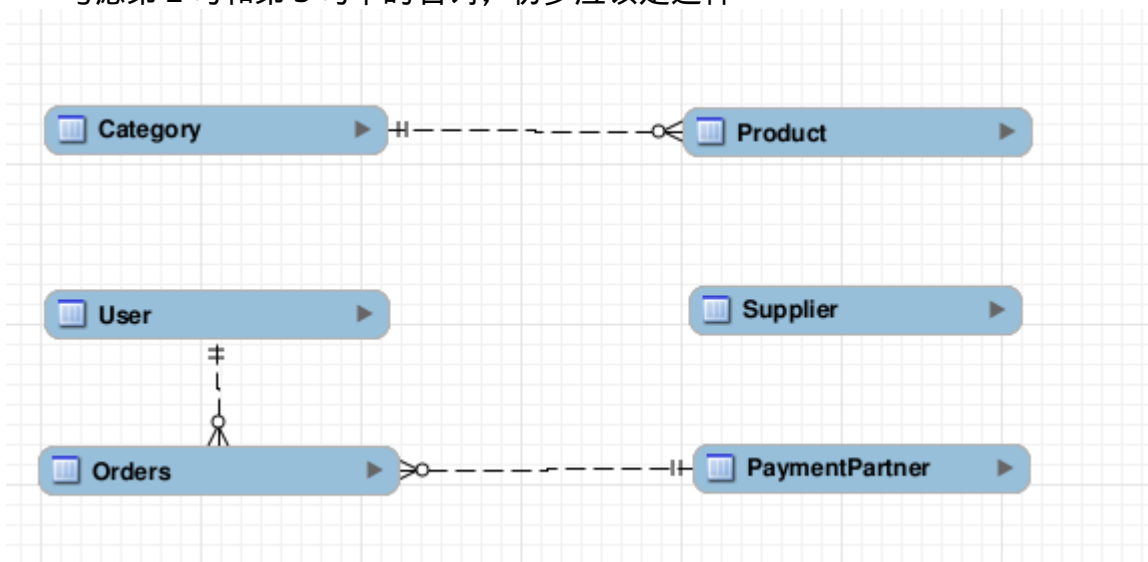
1. 宠物 商城目前计划在线交易 5 种宠物商品：鱼类，狗类，爬行类，猫类，鸟类，宠物由宠物提供商提供
2. 非注册用户可以浏览，搜寻商城内的所有宠物
3. 注册用户可以下订单购买商城内的宠物，采用与电子银行系统合作的方式进行在线支付
4. 交易成功后，系统将向用户注册邮箱发送通知邮件，同时向宠物提供商发送新订单通知
5. 同一订单可以同时包含多种不同种类的商品
6. 用户可查看自己的所有的定单历史记录
7. 宠物提供商负责宠物的配送，宠物提供商将跟踪订单状态并更新到系统中，以方便用户追踪
8. 配送地址可以和注册的地址不同
9. 当商品库存不足时，用户仍然可以下单购买，系统将通知所有的提供商，提供商将根据他们的状况更新系统的库存

由第 1 句话中的名词，可以简单地建立以下实体关系模型 (ERD)

- 一种种类可以包含零或多个商品，零是有可能一个种类初建，还没有加入商品
- 存在一个商品提供商实体



考虑第 2 句和第 3 句中的名词，初步应该是这样



但是，这个 Supplier 如果象这样孤立起来，它所包含的信息将与 User 的信息重复，这将违背数据库设计的第一范式 (First Normal Form )

以下回顾一下数据库的设计范式，第一范式和第二范式，注意，从第三范式开始，考虑到性能问题，对于大型的企业应用已经不适用了。

### ...6.1.1 数据库设计范式

#### ...6.1.1.1 First Normal Form

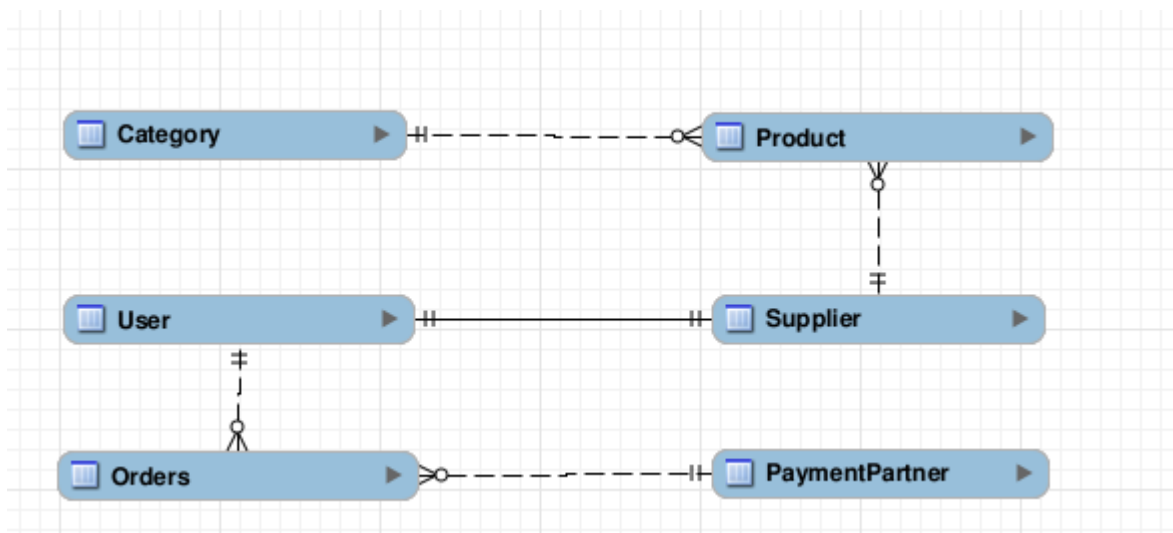
- Define the data items required, because they become the columns in a table. Place related data items in a table.
- Ensure that there are no repeating groups of data.
- Ensure that there is a primary key.

### ...6.1.1.2 Second Normal Form

- (Because primary key can consist of one or more columns) there must be no partial dependences of any of the columns on the primary key.

范式关注的是数据是否有冗余（也就是对存储的要求），但是对性能来讲，可能是个伤害。实际中，能保证到第二范式就相当不错了，尤其是引入了 OR/M 工具后。

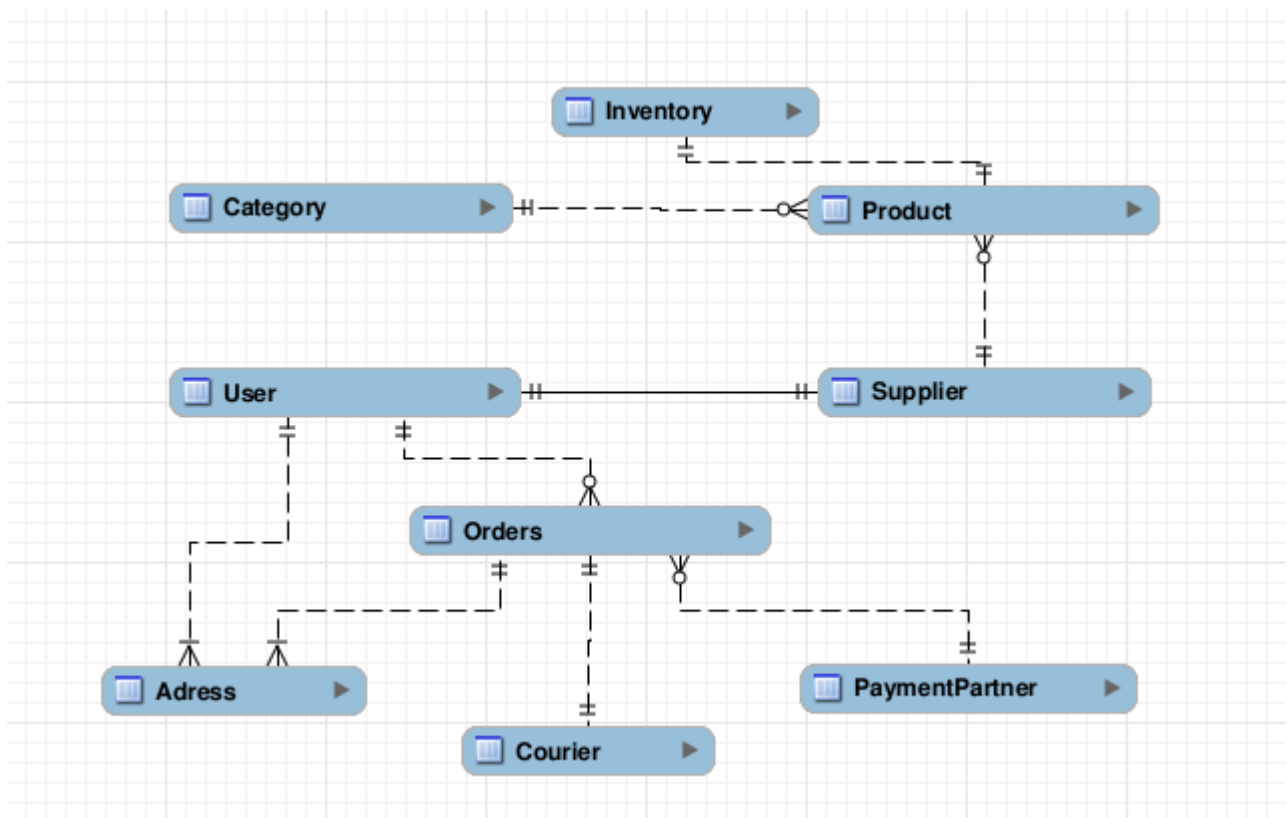
所以，将 Supplier 视为一个特殊的 User，除了 User 的所有特征外，它还可以外加一个提供商名字，以及由于业务需要，可以加入状态（有效，还是已经无效），以及，提供商是提供出售的商品的，所以，他应该跟商品关联，加上这些，进一步更新模型：



第 4 句中包含有新的实体邮箱和邮件，通知，邮箱是作为 用户信息的一部分，系统不记录邮件，通知将以 JMS 消息形式发出(如果是对象消息，将需要对象封装，但不一定是实体)，因此没有新的实体

5, 6 句没有新的实体，不用考虑，现在把后面 7, 8, 9 中的名词考虑进来，由于订单状态可以作为订单的一部分，不用考虑专门作为一个实体，因此加入配送方式，地址，库存：





## ...6.2 形容词

皮皮鲁看着初步出来的数据模型，然后回头看看需求说明书，名词分析基本一一对应了，但是当看形容词分析时，他发现了问题：

- 同一订单可以同时包含多种不同种类的商品

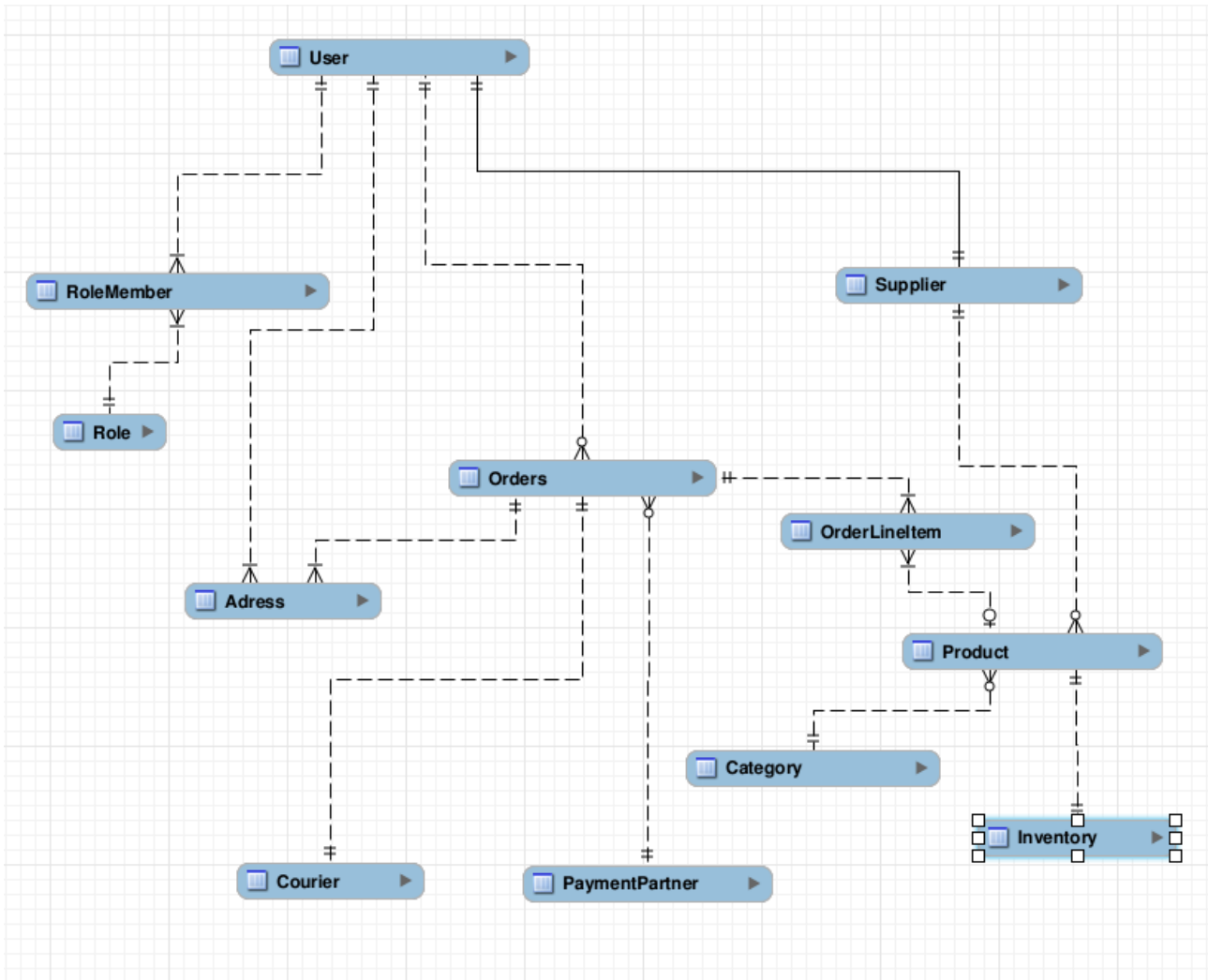
对于定单的考虑，皮皮鲁突然想起了经典的 order, orderItem 概念

- 用户可查看自己的所有的定单历史记录

用户应该有权限设置，一个用户可以是商城的最终用户，可以是管理员，可以是提供商，等等。这些应该建立起对就的模型，这就是用户角色，但是用户角色的设计存在两种主要的类型，继承与聚合。

- 继承  
角色的体系关系，比如管理员角色自动继承普通用户角色
- 聚合  
将角色以集合的形式分配给用户

皮皮鲁根据以往的经验，选择了聚合的方式，这就涉及到一个中间表来建立这种多对多的模型(many-to-many : User-to-role)。



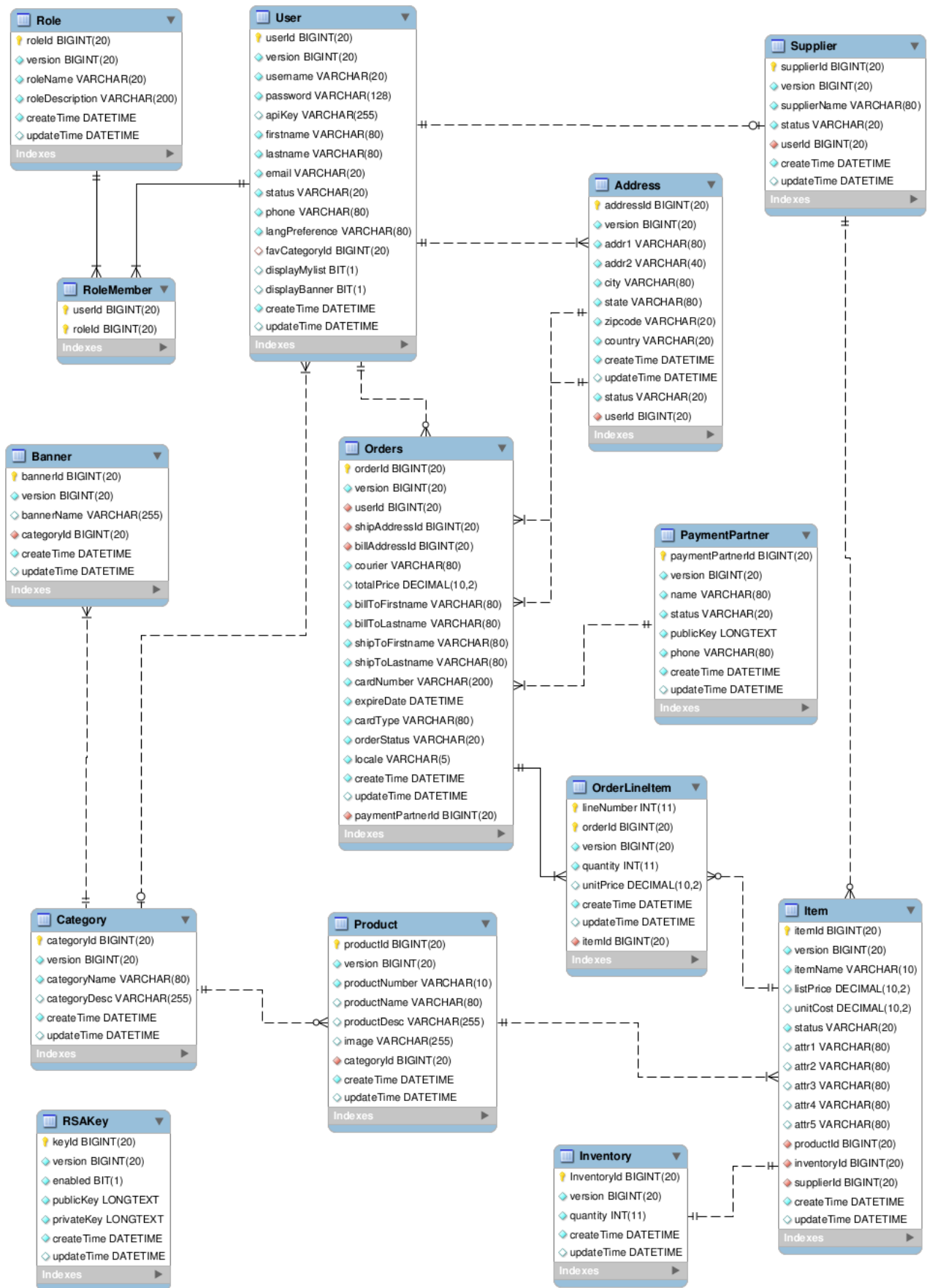
### ...6.3 ERD

数据模型设计 review 越早越好。让团队内所有的人员一起参与 review，就会减少后续阶段 ERD 的改动次数。

在 review 的过程中，队友根据以往的经验，提出以下建议：

- 定单应该有付款方地址(billingAddress)与送货地址(shipAddress)，因 order 与 Address 有两个多对一关系。
- 再有考虑进安全问题，需要系统保存非对称加密用的 key（有些系统将其存在文件系统中）。

最终的 ERD 如下：

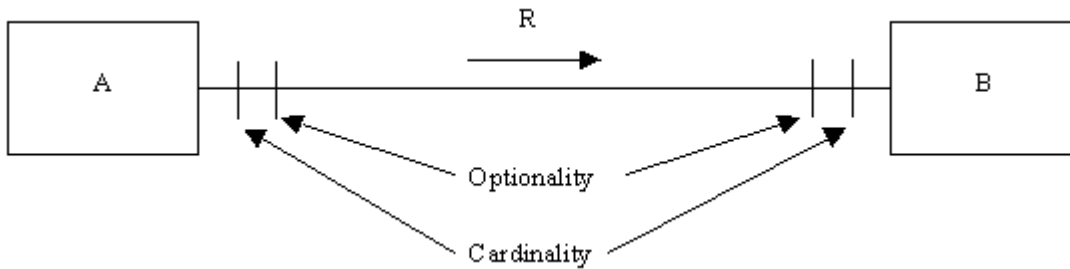


### ...6.3.1 ERD 图例

#### ...6.3.1.1 Cardinality and optionality

靠近方框的叫基数 (cardinality), 也就是最大值

远离方框的叫 *optionality* (不知道中文叫什么), 也就是最小值



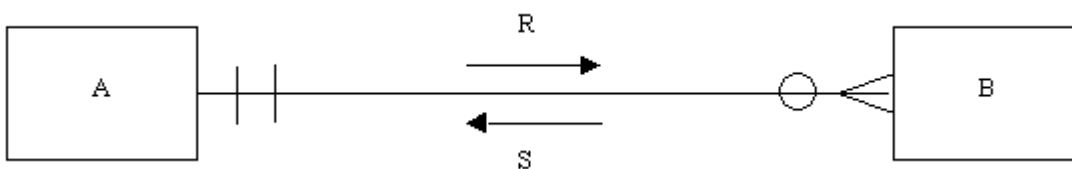
#### ...6.3.1.2 one-to-one

上图表示:

A 有且只有一个 B (1:1)

反过来, B 也有且只有一个 A (1:1)

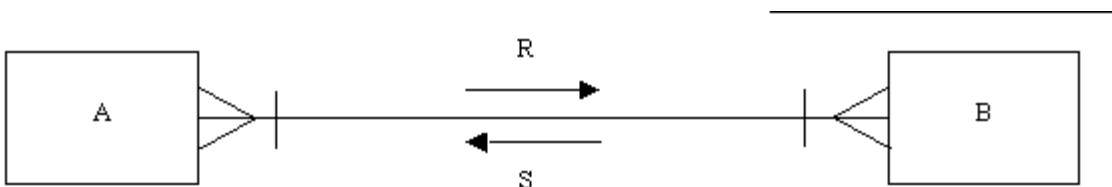
#### ...6.3.1.3 one-to-many



A 可以有零个或多个 B

B 有且只有一个 A

#### ...6.3.1.4 many-to-many



A 可以有 1 个或多个 B

B 可以有 1 个或多个 A

实际中, 对于可以为 0 个或多个的多对多模型, 通常是基于属性 mandatory 来决定, 如果非 mandatory 时, 则允许为 0

## 7 Use Case

- 基于动词分析设计到系统的 Use Case
- 补上通用的 user case
- 以 actor 为中心的 use case 设计
- 将依赖的内部或外部组件视作黑盒

### ...7.1 动词分析

分析每个动作的属主（即 actor），以表格的形式列出

1. 宠物 商城目前计划在线交易 5 种宠物商品：鱼类，狗类，爬行类，猫类，鸟类，宠物由宠物提供商提供
2. 非注册用户可以浏览，搜寻商城内的所有宠物
3. 注册（注册）用户可以下订单购买商城内的宠物，采用与电子银行系统合作的方式进行在线支付
4. 交易成功后，系统将向用户注册邮箱发送通知邮件，同时向宠物提供商发送新订单通知
5. 同一订单可以同时包含多种不同种类的商品
6. 用户可查看自己的所有的定单历史记录
7. 宠物提供商负责宠物的配送，宠物提供商将跟踪订单状态并更新到系统中，以方便用户追踪
8. 配送的地址可以和注册的地址不同
9. 当商品库存不足时，用户仍然可以下单购买，系统将通知所有的提供商，提供商将根据他们的状况更新系统的库存

Actor	Use case
Unsigned user	view
	search
	register
Signed user	view
	search
	signOn
	signOff
	placeOrder
	viewOrderStatus
supplier	Dispatch
	updateOrderStatus
hjpetstore	Send order email
	noticifySupplier

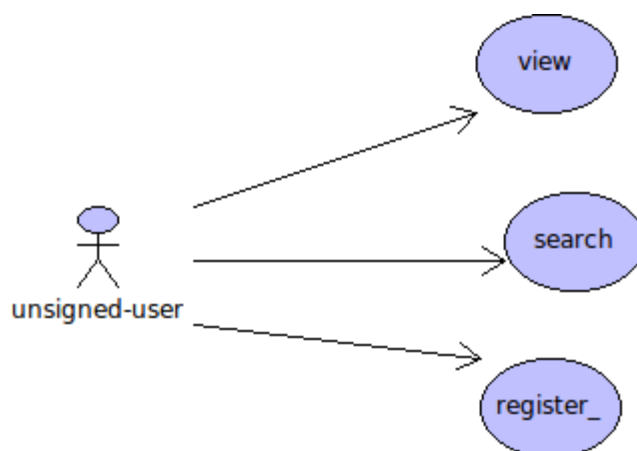
### ...7.2 补上通用的 **use case**

需求说明书通常会省略系统通用的功能，而设计人员需要捕捉到这一 gap，比如用户注销

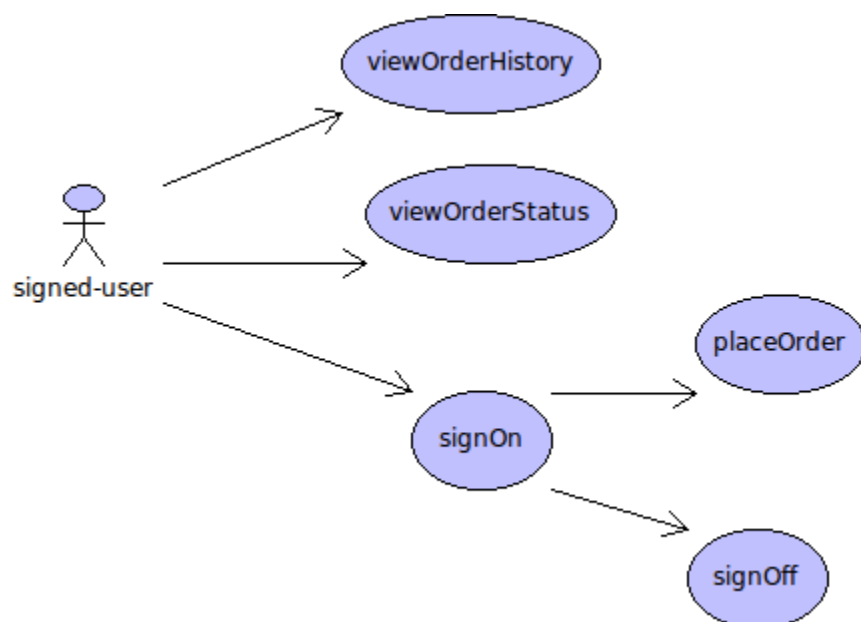
### ...7.3 以 **actor** 为中心的 **use case** 设计

Actor 在实体建模后就清晰了，use case 应该围绕着不同的 actor 进行设计

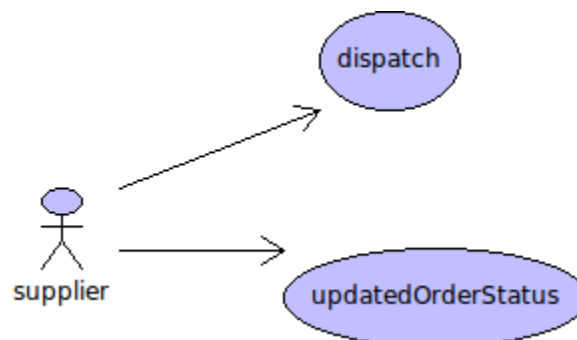
#### ...7.3.1 *unsigned user*



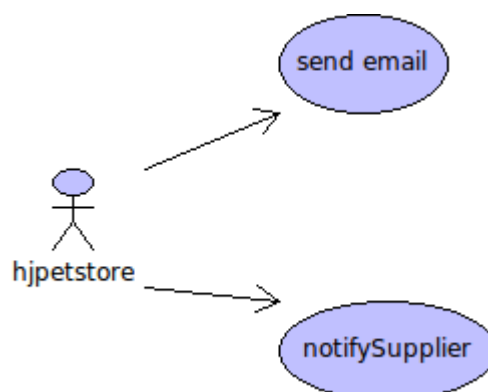
#### ...7.3.2 *signed user*



### ...7.3.3 *supplier*



### ...7.3.4 *Hjpetstore*



### ...7.4 将依赖的内部或外部组件视作黑盒

设计 use case 时，将 scope 限定在本系统功能范围之内，不需要考虑信赖的组件，否则 use case 会引起混乱

## 8 *Hibernate* 映射

- 使用合成 (synthetic identifier) 或代理主键 (surrogate key)
- 不要使用复合主键 (composite natural primary key)
- 强烈建议引入 <version> 或者 <timestamp>
- 使用中间表映射多对多关系
- 尽量不要使用 class inheritance 映射

- 永远不要使用 List
- EntityInterceptor
- 不要指定任何 lazy=false
- 显式指定外键名
- 关于 equals() 和 hashCode() 方法
- 二级缓存
- 历史遗留数据

### ...8.1 使用合成 ( **synthetic identifier** ) 或代理 ( **surrogate** ) 主键

这个合成或代理主键维护在系统内部，没有任务业务含义，不会被应用代码所引用，它的存在只为一个目的，为实体主键。

在遗留系统中的自然主键 ( natural ) 或多或少有业务含义，例如身份证号。大家知道我国的身份证号曾经发生过改变，从 15 位变成 18，竟然无法保证业务字段永远不会变，设计时就不要把它作为主键，主键的改动，以及涉及到的以其作外键的改动是令人挫败的。

```
protected Long id;
```

```
<id access="field" column="userId" name="id" type="long">
  <generator class="native"/>
</id>
```

### ...8.2 不要使用复合主键 ( **composite primary key** )

请不要使用复合主键，除了基于表的多对多映射关系，hibernate 自动产生的关联表的主键外。（那是 Hibernate 自动维护的，不需要指定）

理由如下：

- 首先，打破了第一条规则
- 再有，实现一个完善的 Id 主键类并不想象的那么简单，请看下面 equals 方法的注释
- 第三，复合关联主键中引用另外一个实体的主键，在新创建实体时，必须首先显式地 save，在实体处于 persisted 状态前，id 总是为 null

下面的 Id 为例，其中一个字段是 orderId，那么在插入一条 order 记录时，必须把其关联的所有 orderLineItem 全部插入，但因为 id.orderId 在 order 对象保存前为 null，所以这个关联关系需要手工维护，需要先 save order，再取出 orderId，手工设置到 orderLineItem 中去。

```
/**
 * It's critical that you implement equals() and hashCode() correctly,
 * because
 * Hibernate relies on these methods for cache lookups. Identifier classes
```



are also

- \* expected to implement Serializable.

- \*/

```
public static class Id implements Serializable {
```

```
    private int lineNumber;
```

```
    private long orderId;
```

```
    public Id() {
```

```
    }
```

```
    public Id(int lineNumber, long orderId) {
```

```
        this.lineNumber = lineNumber;
```

```
        this.orderId = orderId;
```

```
    }
```

```
    /**
```

```
     * @return the lineNumber
```

```
     */
```

```
    public int getLineNumber() {
```

```
        return lineNumber;
```

```
    }
```

```
    /**
```

```
     * @return the orderId
```

```
     */
```

```
    public long getOrderId() {
```

```
        return orderId;
```

```
    }
```

```
    /**
```

```
     * In general, we should not supply setter for composite-key,
```

when initOrder,

```
     * the orderId is still not persisted, then it is null.
```

```
     * We have to set it lately once the order persisted.
```

```
     * @param orderId
```

```
     */
```

```
    public void setOrderId(long orderId) {
```

```
        this.orderId = orderId;
```

```
    }
```

```
    /**
```

always

\* access the properties of the "other" object via the getter methods. This is

```

    * extremely important, because the object instance passed as other may
    be a proxy
    * object, not the actual instance that holds the persistent state. To
    initialize this
    * proxy to get the property value, you need to access it with a getter
    method. This is
    * one point where Hibernate isn't completely transparent. However, it's a
    good prac-
    * tice to use getter methods instead of direct instance variable access
    anyway.
    * @param o
    * @return
    */
    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }

        if (!(o instanceof Id)) {
            return false;
        }

        Id that = (Id) o;

        return (this.getLineNumber() == that.getLineNumber()) &&
        (this.getOrderId() == that.getOrderId());
    }

    @Override
    public int hashCode() {
        int result = 17;
        result = 31 * result + getLineNumber();
        result = 31 * result + (int) (getOrderId() ^ (getOrderId() >>> 32));
        return result;
    }

    @Override
    public String toString() {
        StringBuilder s = new StringBuilder();
        s.append("[");
        s.append("lineNumber=").append(getLineNumber());
        s.append(", ");
        s.append("orderId=").append(getOrderId());
        s.append("]");
        return s.toString();
    }

```

```
}  
}
```

另外，如果 Id 所包含的 field 是对象类型，在实现 equals, hashCode, toString 方法时，需要做非空保护。

### ...8.3 强烈建议引入 <version> 或者 <timestamp>

使用无业务意义的字段 version 或者 timestamp 以便 Hibernate 自动为并发控制使用 optimistic locking.

```
protected Long version;
```

```
<version access="field" name="version" type="long"/>
```

对实体状态的更新，捕获 javax.persistence.OptimisticLockException，对并发更新进行错误处理。

比较基于 version 与基于 timestamp 的实现，各自都存在潜在的问题：

- version 在达到数据类型的最大值后，目前 hibernate 的实现并不会重新归零，而是抛出异常，详细情况见 [Need help with versioning on optimistic locking mechanism](#)
- timestamp 的问题在于实现依赖于 JVM 以及 OS 的时间精度 (*In theory, a timestamp is slightly less safe, because two concurrent transactions may both load and update the same item in the same millisecond; in practice, this won't occur because a JVM usually doesn't have millisecond accuracy.*)

### ...8.4 使用中间表映射多对多关系

将 many-to-many 分解成熟悉的两个 many-to-one 将简化业务模型和映射关系。

Role : User = m2m

```
public class Role extends DomainObject {  
    private Set<User> users = new HashSet<User>();  
  
public class User extends DomainObject {  
    private Set<Role> roles = new HashSet<Role>();
```

```
<!--
    Mapping for the bidirectional many-to-many association for User : Role
    on a join table RoleMember
    You have enabled cascade="save-update" for both ends of the
collection.
    This isn't unreasonable, we suppose. On the other hand, the cascading
options
    all, delete, and delete-orphans aren't meaningful for many-to-many
associa-
tions.
    make sure the name of foreign-key is unique
-->
<set cascade="save-update" name="users" table="RoleMember">
  <key column="roleId" not-null="true"/>
  <many-to-many class="User" column="userId" foreign-
key="fk_role_userId"/>
</set>

<!--
    Mapping for the bidirectional many-to-many association for User : Role
    on a join table RoleMember

    You have enabled cascade="save-update" for both ends of the
collection.
    This isn't unreasonable, we suppose. On the other hand, the cascading
options
    all, delete, and delete-orphans aren't meaningful for many-to-many
associa-
tions.
    make sure the name of foreign-key is uniquee
-->
<set cascade="save-update" name="roles" table="RoleMember">
  <key column="userId" not-null="true"/>
  <many-to-many class="Role" column="roleId" foreign-
key="fk_user_roleId"/>
</set>
```

### ...8.5 尽量不要使用 **class inheritance** 映射

虽然在对象和关系都有 is a 和 has a 的概念, 但在 sql 的世界里却只有 has a。无论是 `<mapped-superclass>`, `<union-subclass>`, `<joined-subclass>` 还是 `discriminator` 都是不直观的概念, 在涉及到同时更新 parent-child 实体时, 都会使 dao 方法处理复杂化。

Domain 对象可是按照 inheritance 设计，所要做的是在 hbm.xml 中将父类的字段重复映射到子表。

### ...8.6 永远不要使用 List

在 Hibernate 的世界里，Set 比 List 更适合表示 collection 类型时，

在 Domain 对象中，对关联实体的数据类型永远不要使用 List。（*Please don't use List as the data type, which usually contain duplicates for eager collection fetching*, <http://opensource.atlassian.com/projects/hibernate/browse/HHH-1513>）

在不使用 open session in view 的模式中，通常的做法是在 DAO 中将在同一个用户会话 (conversation) 中的关联对象 fetch 出来，但很不幸，如果使用 List 类型，将撞上 Hibernate 的暗礁—List 并不排除重复的元素！而 Set 则自动排除重复的元素。

### ...8.7 EntityInterceptor

使用 SessionFactory EntityInterceptor 对稽查字段，例如 create\_time, update\_time, 实行自动插入值。

实现自动默认值有多种方式，但不同的数据库的方式有些不一致，在 trigger free 的方案里，在建库脚本中将些字段初始化为一个默认值，如 sysdate，但是 Mysql 有一个限制，在同一个表的定义中，只允许一个字段使用 now() 或者 sysdate() 函数 (MySQL has minor limitation on timestamps. Unfortunately you can create only one time stamp column that has *DEFAULT NOW()* value)

所以，这种情况更加迫切需要以程序可控的方式完成稽查字段的填充，entityInterceptor 即可完成这一任务：

#### DataContext-xxx.xml

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource" ref="dataSource"/>

  <!--
    externalized the hibernate configuration for more easier maintain and
    different strategy, such as unit test.
  -->
  <property name="configLocation" value="classpath:hibernate-
jta.cfg.xml" />

  <!-- Interceptor to set the createTime and updateTime fields
  automatically -->
  <property name="entityInterceptor">
    <bean
class="org.pprun.hjpetstore.persistence.AuditInterceptor" />
  </property>
```

```
<property name="eventListeners">
  <map>
    <entry key="merge">
      <bean
class="org.springframework.orm.hibernate3.support.IdTransferringMergeEvent
Listener"/>
    </entry>
  </map>
</property>
</bean>
```

#### AuditInterceptor.java

```
public class AuditInterceptor extends EmptyInterceptor {

    private static Log log = LogFactory.getLog(AuditInterceptor.class);
    public static final String CREATE_TIME_FIELD = "createTime";
    public static final String UPDATE_TIME_FIELD = "updateTime";

    /**
     * set updateTime.
     */
    @Override
    public boolean onFlushDirty(Object entity, Serializable id, Object[] currentState,
Object[] previousState, String[] propertyNames, Type[] types) {
        if (entity instanceof IAuditable) {
            for (int i = 0; i < propertyNames.length; i++) {
                if (UPDATE_TIME_FIELD.equals(propertyNames[i])) {
                    if (log.isDebugEnabled()) {
                        log.debug("AuditInterceptor:onFlushDirty:" + entity + ".set" +
propertyNames[i]);
                    }
                    currentState[i] = Calendar.getInstance();
                    return true;
                }
            }
        }
        return false;
    }

    /**
     * set createTime and updateTime.
     */
    @Override
    public boolean onSave(Object entity,
        Serializable id,
        Object[] state,
```

```
String[] propertyNames,
Type[] types) {
    boolean isModified = false;
    if (entity instanceof IAuditable) {
        for (int i = 0; i < propertyNames.length; i++) {
            if (UPDATE_TIME_FIELD.equals(propertyNames[i])) {
                if (log.isDebugEnabled()) {
                    log.debug("AuditInterceptor: onSave: " + entity + ".set" +
propertyNames[i]);
                }
                state[i] = Calendar.getInstance();
                isModified = true;
            } else if (CREATE_TIME_FIELD.equals(propertyNames[i])) {
                if (log.isDebugEnabled()) {
                    log.debug("AuditInterceptor: onSave: " + entity + ".set" +
propertyNames[i]);
                }
                state[i] = Calendar.getInstance();
                isModified = true;
            }
        }
    }
    return isModified;
}
```

当然，所有的 domain 对象都需要实现接口 **IAuditable**

lauditable.java

```
public interface IAuditable {

    public Calendar getCreateTime();

    public Calendar getUpdateTime();
}
```

### ...8.8 不要指定任何 **lazy=false**

在 hbm.xml 文件中，请不要指定任何 lazy=false，不要假定客户端一定需要某个关联的实体。一旦在 hbm.xml 中指定了，就无法在代码中补救，相反，如果 lazy=true，可以在代码中显式使用 eagerly fetch:

### ...8.8.1 Criteria fetch mode

```
public List<Order> getOrders(Calendar from, Calendar to) {  
    return getSession().createCriteria(Order.class)  
        .setFetchMode("shipAddress", FetchMode.JOIN)  
        .setFetchMode("billAddress", FetchMode.JOIN)  
        .setFetchMode("orderLineItems", FetchMode.JOIN)  
        .setFetchMode("orderLineItems.item", FetchMode.JOIN)  
        .add(Restrictions.between("createTime", from, to))  
        .addOrder(org.hibernate.criterion.Order.asc("orderStatus"))  
        .list();  
}
```

### ...8.8.2 HQL join fetch

```
String hql = "FROM Order AS o "  
    + "INNER JOIN FETCH o.shipAddress a "  
    + "INNER JOIN FETCH o.orderLineItems oli "  
    + "INNER JOIN FETCH o.paymentPartner pp "  
    + "LEFT JOIN FETCH oli.item i "  
    + "WHERE o.createTime BETWEEN :from AND :to "  
    + "ORDER BY o.orderStatus ASC ";
```

### ...8.9 显式指定外键名

如果在 hbm.xml 文件中没有显式地指定外键名, hibernate 将用 'FK\_随机数据' 作为外键名, 这样会使 D B A 发狂的。

```
<set cascade="save-update" name="roles" table="RoleMember">  
    <key column="userId" not-null="true"/>  
    <many-to-many class="Role" column="roleId" foreign-  
key="fk_user_roleId"/>  
</set>
```

另外, 注意的一点是, 外键名一定要在所有的实体中是唯一, 否则 hibernate hbm2ddl 会忽略掉重复的外键。

### ...8.10 关于 equals() 和 hashCode() 方法

如果同一个 domain 对象的多个实例将会在 detached 状态时存入 Set 中, 请实现基于 business key 的 equals() 和 hashCode()



### ...8.10.1 相同与相等

关于相同与相等，有以下概念：

- 对象相同( Object identical)是指它们占据同一块 JVM 内存, 此时 `== operator` 返回 `true`
- 对象相等(Object equal)是指它们拥有相同的值, 这个值是根据定义在 `equals` 方法含义来确定的 (并不是所有的字段都必须相同), 但是如果没有 override `java.lang.Object` 的 `equals` 方法的对象, 语义跟相同 ( identical) 同
- database identity, 对于被存到数据库中的对象, 相同意味着他们代表数据库表中的同一行 ( database identity )

正确地实现 `equals` 与 `hashCode` 方法将会避免

- 主键为 null 问题
- 相同对象在 Set 中只有一份的问题。。。 ( google it)

### ...8.10.2 业务键(*business key*)

Domain 对象的一个 field 或 field 组合, 它可以唯一地标记具有 database identity 属性的对象。如果没有使用代理主键 ( surrogate primary key ), 它就是那个不建议的自然主键 ( nat- ural key ), 但 business key 与 自然主键不同的是, business key 可以改变, 只要不经常变就行了。

以下代码将使用业务键(business key)实现这两个方法: 也就是在实现 `equals`, `hashCode` 方法中, 只包括那一个或几个可以足够标记对象是唯一的就可以了。例如在 `hjpetstore` 中, 用户的用户名是唯一性的, 所以它可以作为 `User` 的 business key

```
@Override
public boolean equals(Object other) {
    if (this == other) {
        return true;
    }
    if (!(other instanceof User)) {
        return false;
    }
    final User that = (User) other;
    return this.username.equals(that.getUsername());
}

@Override
public int hashCode() {

    return username.hashCode();
}
```

### ...8.11 二级缓存

- 设置 read-only 实体使用二级缓存
- 慎对 read-write 实体使用二级缓存
- 最好不要使用查询缓存

#### ...8.11.1 二级缓存配置集中化

Hibernate 支持将 cache 的配置设置在 hibernate.cfg.xml 文件中，而不是散落在各个 domain 对象 hbm.xml 中的 <cache> 元素中。集中化配置有利维护。

#### ...8.11.2 ehcache

ehcache 是简单，高效的，但是它的东家 Terracotta 没有把 cluster 的支持加入 free 版，所以只能在非 cluster 环境下使用它。

##### ...8.11.2.1 Enable 二级缓存

```
hibernate.cfg.xml

<!--
    use Ehcache in non-cluster environment:
http://ehcache.org/documentation/hibernate.html
-->
<property
name="hibernate.cache.region.factory_class">net.sf.ehcache.hibernate.EhCacheRegionFactory</property>
  <property
name="net.sf.ehcache.configurationResourceName">/ehcache.xml</property
>
  <property
name="hibernate.cache.use_second_level_cache">true</property>
```

##### ...8.11.2.2 二级缓存的实体及 collections

```
hibernate.cfg.xml

<!--
    You have to carefully evaluate the performance of a clustered cache with
full transaction
    isolation before using it in production. In many cases, you may be better
off disabling
    the second-level cache for a particular class if stale data isn't an option!
First
    benchmark your application with the second-level cache disabled. Enable
it for
```

```
    good candidate classes, one at a time, while continuously testing the
scalability of
    your system and evaluating concurrency strategies.
    -->
    <!-- cache setting for entities and collections -->
    <class-cache class="org.pprun.hjpetstore.domain.RSAKey" usage="read-
only"/>
    <class-cache class="org.pprun.hjpetstore.domain.PaymentPartner"
usage="read-only"/>
    <class-cache class="org.pprun.hjpetstore.domain.Banner" usage="read-
only"/>
    <class-cache class="org.pprun.hjpetstore.domain.Category" usage="read-
only"/>
    <class-cache class="org.pprun.hjpetstore.domain.Product" usage="read-
only"/>
    <class-cache class="org.pprun.hjpetstore.domain.Supplier" usage="read-
only"/>
    <class-cache class="org.pprun.hjpetstore.domain.Role" usage="read-
write"/>
    <collection-cache
collection="org.pprun.hjpetstore.domain.Category.products" usage="read-
only"/>
    <collection-cache collection="org.pprun.hjpetstore.domain.Product.items"
usage="read-only"/>
    <collection-cache collection="org.pprun.hjpetstore.domain.Supplier.items"
usage="read-only"/>
    <collection-cache collection="org.pprun.hjpetstore.domain.User.roles"
usage="read-write"/>
```

### ...8.11.2.3 定制 ehcache.xml 配置

每一个参数并不是可以胡乱地设置，需要根据业务需求，以及实体的变化频率来决定，必要时，作为 load testing 的一个 test case 来优化这些参数。

```
Ehcache.xml
<?xml version="1.0" encoding="UTF-8"?>

<!--
    This file will be used when the hibernate.cache.provider_class was specified
as
    org.hibernate.cache.EhCacheProvider (in non-cluster env.)
    EHCACHE documentation: http://ehcache.org/documentation/hibernate.html
-->

<ehcache>
```

<!-- Sets the path to the directory where cache .data files are created.

If the path is a Java System Property it is replaced by its value in the running VM.

The following properties are translated:

user.home - User's home directory

user.dir - User's current working directory

java.io.tmpdir - Default temp file path -->

-->

<diskStore path="java.io.tmpdir/ehcache"/>

<!--Default Cache configuration. These will applied to caches programmatically created through the CacheManager.

The following attributes are required:

maxElementsInMemory - Sets the maximum number of objects that will be created in memory

eternal - Sets whether elements are eternal. If eternal, timeouts are ignored and the element is never expired.

overflowToDisk - Sets whether elements can overflow to disk when the in-memory cache has reached the maxInMemory limit.

The following attributes are optional:

timeToIdleSeconds - Sets the time to idle for an element before it expires.

i.e. The maximum amount of time between accesses before an element expires

Is only used if the element is not eternal.

Optional attribute. A value of 0 means that an Element can idle for infinity.

The default value is 0.

timeToLiveSeconds - Sets the time to live for an element before it expires.

i.e. The maximum time between creation time and when an element expires.

Is only used if the element is not eternal.

Optional attribute. A value of 0 means that and Element can live for infinity.

The default value is 0.

diskPersistent - Whether the disk store persists between

restarts of the Virtual Machine.

The default value is false.

diskExpiryThreadIntervalSeconds- The number of seconds between runs of the disk expiry thread. The default value is 120 seconds.

-->

<!--

The timeToldleSeconds attribute defines the expiry time in seconds since an

element was last accessed in the cache. You must set a sensible value here, because

you don't want unused object to consume memory.

-->

<!--

Because readonly objects are immutable, you don't need them to be removed from the cache if they're being accessed regularly. Hence, timeToLive-

Seconds is set to a high number.

-->

<!--

Accessing data on the local disk is faster than accessing the database across a network if database is running on a different machine.

-->

```
<defaultCache maxElementsInMemory="500" eternal="false"
    timeToldleSeconds="1800" timeToLiveSeconds="1800"
    overflowToDisk="true" />
```

<!-- below two entries might not useful because we are not enabling query cache -->

```
<cache name="org.hibernate.cache.StandardQueryCache"
    maxElementsInMemory="1000" eternal="false"
timeToldleSeconds="43200"
    timeToLiveSeconds="43200" overflowToDisk="true" />
```

```
<cache name="org.hibernate.cache.UpdateTimestampsCache"
    maxElementsInMemory="1000" eternal="false"
timeToldleSeconds="43200"
    timeToLiveSeconds="43200" overflowToDisk="true" />
```

<!-- read only ones have longer live time -->

```
<cache name="org.pprun.hjpetstore.domain.RSAKey"
    maxElementsInMemory="5" eternal="false"
timeToldleSeconds="43200"
```

```
timeToLiveSeconds="43200" overflowToDisk="true" />

<!-- read-write ones have shorter live time -->
<cache name="org.pprun.hjpetstore.domain.Role"
      maxElementsInMemory="500" eternal="false"
timeToldleSeconds="120"
      timeToLiveSeconds="120" overflowToDisk="true" />

<!-- collections read only-->
<cache name="org.pprun.hjpetstore.domain.Category.products"
      maxElementsInMemory="5000" eternal="false"
timeToldleSeconds="43200"
      timeToLiveSeconds="43200" overflowToDisk="true" />

</ehcache>
```

### ...8.11.3 Jboss cache 3

对于 cluster 环境, Hibernate 社区内有多种可选方案, Jboss cache 3 (或者更先进的 infinispn)近年来表现最活跃。

#### ...8.11.3.1 Enable 二级缓存

Jboss 3 比 EnCache 要稍微复杂些, 因为它标称为 Transactional Cluster cache, 所以涉及到 JTA 相关的配置。

其中需要注意的是, hibernate 官方文档描述的是老版本的 [jboss cache v1 与 v2 的配置](#): 设置属性 `hibernate.cache.provider_class = org.hibernate.cache.TreeCacheProvider`

而对于 Jboss cache v3 应该设置属性 `hibernate.cache.region.factory_class = org.hibernate.cache.jbc.MultiplexedJBossCacheRegionFactory`

#### Hibernate-jta.cfg.xml

```
<!-- jboss cache -->
<!-- http://docs.jboss.org/jbossclustering/hibernate-caching/3.5/en-US/html_single/index.html -->
<!-- Make sure your Hibernate is configured to use JTA transactions -->
<property
name="hibernate.transaction.manager_lookup_class">org.hibernate.transacti
on.SunONETransactionManagerLookup</property>
<property
name="hibernate.transaction.factory_class">org.hibernate.transaction.JTATran
```

```
sactionFactory</property>
  <!-- <property
name="hibernate.current_session_context_class">jta</property>-->
  <!-- Do not set the legacy hibernate.cache.provider_class property when
using JBoss Cache 3.
    That is a legacy property from before Hibernate 3.3's redesign of second
level caching internals.
    It will not work with JBoss Cache 3.
    <property
name="hibernate.cache.provider_class">org.hibernate.cache.TreeCacheProvid
er</property>
    -->
    <property
name="hibernate.cache.use_second_level_cache">true</property>
    <!-- we won't want query cache
    <property name="hibernate.cache.use_query_cache">>false</property>
    <property name="hibernate.cache.jbc.query.localonly">>true</property>
    -->
    <property
name="hibernate.cache.region.factory_class">org.hibernate.cache.jbc.Mul
tiplexedJBossCacheRegionFactory</property>
    <!--
        Default is org/hibernate/cache/jbc2/builder/jbc2-configs.xml, a file
found in the hibernate-jbosscache2.jar.
    -->
    <property name="hibernate.cache.jbc.cfg.shared">hjpetstore-
treecache.xml</property>
    <!--
        When this setting is enabled, Hibernate adds an item to
the cache only after checking to ensure that the item isn't already
cached. This
        strategy performs better if cache writes (puts) are much more
expensive than
        cache reads (gets). This is the case for a replicated cache in a cluster,
but not for a
        local cache or a cache provider that relies on invalidation instead of
replication.
    -->
    <property name="hibernate.cache.use_minimal_puts">>true</property>
    <!-- default: jgroups-stacks.xml file found in the hibernate-jbosscache.jar
    <property name="hibernate.cache.jbc.cfg.multiplexer.stacks">jgroups-
stacks.xml</property>
    -->
```

### ...8.11.3.2 二级缓存的实体及 **collections**

注意到，对于 encache 中的 usage 为“read-write”的实体，这里需要指定为 “transactional”：

#### Hibernate-jta.cfg.xml

```
<class-cache class="org.pprun.hjpetstore.domain.RSAKey" usage="read-only"/>
<class-cache class="org.pprun.hjpetstore.domain.PaymentPartner" usage="read-only"/>
<class-cache class="org.pprun.hjpetstore.domain.Banner" usage="read-only"/>
<class-cache class="org.pprun.hjpetstore.domain.Category" usage="read-only"/>
<class-cache class="org.pprun.hjpetstore.domain.Product" usage="read-only"/>
<class-cache class="org.pprun.hjpetstore.domain.Supplier" usage="read-only"/>
<class-cache class="org.pprun.hjpetstore.domain.Role" usage="transactional"/>
<collection-cache collection="org.pprun.hjpetstore.domain.Category.products" usage="read-only"/>
<collection-cache collection="org.pprun.hjpetstore.domain.Product.items" usage="read-only"/>
<collection-cache collection="org.pprun.hjpetstore.domain.Supplier.items" usage="read-only"/>
<collection-cache collection="org.pprun.hjpetstore.domain.User.roles" usage="transactional"/>
```

### ...8.11.3.3 定制 **treecache.xml** 配置

如果对以下元素的含义感到陌生，建议在使用前，还是做一下功课：[Using JBoss Cache as a Hibernate Second Level Cache](#)

#### Treecache.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<cache-configs>
  <cache-config name="mvcc-entity">

    <!-- Node locking scheme -->
    <attribute name="NodeLockingScheme">MVCC</attribute>

    <!--
      READ_COMMITTED is as strong as necessary for most
      2nd Level Cache use cases.
    -->
```



```
-->
<attribute name="IsolationLevel">READ_COMMITTED</attribute>

<!-- Mode of communication with peer caches.

      INVALIDATION_SYNC is highly recommended as the mode for use
      with entity and collection caches.
-->
<attribute name="CacheMode">INVALIDATION_SYNC</attribute>

<!-- Name of cluster. Needs to be the same for all members, in order
      to find each other -->
<attribute name="ClusterName">mvcc-entity</attribute>

<!-- Use a UDP (multicast) based stack. A udp-sync stack might be
      slightly better (no JGroups FC) but we stick with udp to
      help ensure this cache and others like timestamps-cache
      that require FC can use the same underlying JGroups resources. -->
<attribute name="MultiplexerStack">udp</attribute>

<!-- Whether or not to fetch state on joining a cluster. -->
<attribute name="FetchInMemoryState">>false</attribute>

<!--
      The max amount of time (in milliseconds) we wait until the
      state (ie. the contents of the cache) are retrieved from
      existing members at startup. Ignored if FetchInMemoryState=false.
-->
<attribute name="StateRetrievalTimeout">20000</attribute>

<!--
      Number of milliseconds to wait until all responses for a
      synchronous call have been received.
-->
<attribute name="SyncReplTimeout">20000</attribute>

<!-- Max number of milliseconds to wait for a lock acquisition -->
<attribute name="LockAcquisitionTimeout">15000</attribute>

<!-- Lock Striping can lead to deadlocks -->
<attribute name="UseLockStriping">>false</attribute>

<!--
      Indicate whether to use marshalling or not. Set this to true if you
      are running under a scoped class loader, e.g., inside an application
      server.
-->
```

```
<attribute name="UseRegionBasedMarshalling">true</attribute>
<!-- Must match the value of "useRegionBasedMarshalling" -->
<attribute name="InactiveOnStartup">true</attribute>

<!-- For now. disable asynchronous RPC marshalling/sending -->
<attribute name="SerializationExecutorPoolSize">0</attribute>

<!-- Eviction policy configurations. -->
<attribute name="EvictionPolicyConfig">
  <config>
    <attribute name="wakeUpIntervalSeconds">5</attribute>
    <!-- Name of the DEFAULT eviction policy class. -->
    <attribute
name="policyClass">org.jboss.cache.eviction.LRUPolicy</attribute>
    <!-- Cache wide default -->
    <region name="/_default_">
      <!-- Evict LRU node once we have more than this number of
nodes -->
      <attribute name="maxNodes">10000</attribute>
      <!-- And, evict any node that hasn't been accessed in this many
seconds -->
      <attribute name="timeToLiveSeconds">1000</attribute>
      <!-- Don't evict a node that's been accessed within this many
seconds.
      Set this to a value greater than your max expected transaction
length. -->
      <attribute name="minTimeToLiveSeconds">120</attribute>
    </region>
    <!-- Don't ever evict modification timestamps -->
    <region name="/TS"
policyClass="org.jboss.cache.eviction.NullEvictionPolicy"/>
  </config>
</attribute>

</cache-config>
</cache-configs>
```

值得注意的是，Trecache 的内实现，依赖于 [jgroup](#) 作为多播通讯 (Multicast Communication)，如果目标部署平台是 ipV6 enabled，JDK 上存在 bug，见详细的 [workaround](#)

#### ...8.11.4 *infinispan*

[Infinispan](#) 是 Jboss cache 3 的继任者，今后的开发将集中在 infinispan 上。

#### ...8.11.4.1 Enable 二级缓存

配置基本上与 Jboss cache 3 相同, 除了

- hibernate.cache.region.factory\_class  
org.hibernate.cache.infinispan.InfinispanRegionFactory
- hibernate.cache.infinispan.cfg      使用默认值

##### Hibernate.cfg.xml

```
<!-- jboss cache -->
<!-- http://docs.jboss.org/jbossclustering/hibernate-caching/3.5/en-US/html_single/index.html -->
<!-- Make sure your Hibernate is configured to use JTA transactions -->
<property
name="hibernate.transaction.manager_lookup_class">org.hibernate.transaction.SunONETransactionManagerLookup</property>
<property
name="hibernate.transaction.factory_class">org.hibernate.transaction.JTATransactionFactory</property>

<property
name="hibernate.cache.use_second_level_cache">true</property>
<!-- we won't want query cache
<property name="hibernate.cache.use_query_cache">false</property>
<property
name="hibernate.cache.jdbc.query.localonly">true</property>-->
<property
name="hibernate.cache.region.factory_class">org.hibernate.cache.infinispan.InfinispanRegionFactory</property>
<!--
    When this setting is enabled, Hibernate adds an item to
    the cache only after checking to ensure that the item isn't already
    cached. This
    strategy performs better if cache writes (puts) are much more
    expensive than
    cache reads (gets). This is the case for a replicated cache in a cluster,
    but not for a
    local cache or a cache provider that relies on invalidation instead of
    replication.
-->
<property name="hibernate.cache.use_minimal_puts">true</property>
<property name="hibernate.cache.infinispan.statistics">true</property>
<!--
    Default is org/hibernate/cache/infinispan/builder/infinispan-configs.xml,
    a file found in the hibernate-infinispan.jar.

<property name="hibernate.cache.infinispan.cfg">hjpetstore-
```

```
treecache.xml</property>
-->
```

#### ...8.11.4.2 二级缓存的实体及 **collections**

同 二级缓存的实体及 collections 二级缓存的实体及 collections

#### ...8.11.4.3 定制 **infinispan-configs.xml** 配置

使用默认的配置: org/hibernate/cache/infinispan/builder/infinispan-configs.xml, a file found in the hibernate-infinispan.jar.

#### ...8.11.5 查询缓存

请阅读 hibernate reference [The Query Cache](#) 章节和来自 Ehcache/Terracotta 项目组的成员的文章 [Hibernate query cache consider harmful](#) (如果访问受限, 访问我的转载: [Hibernate query cache considered harmful?](#))

### ...8.12 历史遗留数据

对于遗留数据库, 在遵守以上原则的前提下, 提供数据移植策略远比通过 Hibernate 映射 trick 去迎合遗留数据结构来得简单。

竟然选择了 Hibernate, 就要将 Hibernate 的优势最大化, 避免使用 trick 和陷阱。请和 D B A 紧密合作。

Hibernate 是复杂的, 对于遗留数据库, 如果是业务复杂的模型, 不建议在不允许修改模型的情况下使用 Hibernate, 同时, 如果遗留数据库被企业内部多个产品共享, 请慎重地进行数据迁移, 一个最惨的结果是: 与花在 regression test 上的努力, 时间相比, Hibernate 的好处并不明显。

## 9 安全策略

- 不要将密码存成明文
- 使用 Kaptcha
- 使用 https (SSL 通道)
- 密码算法及其参数的选择
- SOA / ROA public api key
- RSA
- Spring Security 访问控制
- PCI (Payment Card Industry)

### ...9.1 不要将密码存成明文

明文的密码，只要记下，写下或存储在计算机中，就不再是密码，它是明文。

企业应用中，密码在数据库需要采用 HASH 算法进行加密，通常采用 MD5 加密算法。JAVA 通过 [JCE](#) 对密码领域提供全面支持

```
public static String md5(byte[] plainText) throws
NoSuchAlgorithmException, UnsupportedEncodingException {
    MessageDigest messageDigest = MessageDigest.getInstance("MD5");

    String md5 = new BigInteger(1,
messageDigest.digest(plainText)).toString(16);
    if (md5.length() < 32) {
        md5 = "0" + md5;
    }

    messageDigest.reset();

    return md5;
}
```

HASH 算法理论上保证不可逆性，即从一个 **HASH** 值，理论上很难，至少“很不容易”得到原文。说“很不容易”，是因为黑客们会想尽一切可能的办法的，比如 HASH 字典攻击：他们将收集一个字典库来保存常用的密码的 HASH 值。

另一个问题是，如果两个人使用相同的密码，如果皮皮鲁使用'haoboy5258'，假设另外一个小伙也喜欢上这个密码，并且用在一个低安全性的系统中。

'haoboy5258' 的 HASH 值可以通过以下命令获得（windows 的同学可以使用 [online tools](#)）：

```
pprun@pprun-laptop:~$ echo -n 'j2eej2ee' | md5sum
b6dcb0189da62b6b849903dcd57f84be -
```

HASH 保证对于同一个输入一定会返回同一个输出，这也是 HASH 的本意，用于保证内容的确没有被串改。

对于一个低安全的系统，黑客们很容易攻破并获得密码的 HASH 值，有了它，通过查找 HASH 字典，便可以得到相应的明文了。

这样一来打破了 HASH 算法理论上保证不可逆性，即通过 HASH 值不能知道明文。（因为黑客有一个字典，存着他所掌握的 明文<->HASH 值 的字典）

为了使黑客更累些，通常的做法是在生成对应的 HASH 值时再带上一个该用户特有的属性，比如 create\_time 或者，如果用户名是唯一的，可以用 username，这个值就是通过所说的 **salt**，就是在生成 **hash** 时，撒上把盐，呵呵。

```
public static String md5(byte[] plainText, byte[] salt) throws
```

```
NoSuchAlgorithmException, UnsupportedEncodingException {
    MessageDigest messageDigest = MessageDigest.getInstance("MD5");
    messageDigest.update(salt);
    String md5 = new BigInteger(1,
messageDigest.digest(plainText)).toString(16);
    if (md5.length() < 32) {
        md5 = "0" + md5;
    }

    messageDigest.reset();

    if (log.isDebugEnabled()) {
        log.debug(new String(plainText, "UTF8") + "[salt=" + new String(salt,
"UTF8") + "]" 's MD5: " + md5);
    }

    return md5;
}
```

这样一来，数据库中保存的是这个 md5 值作为密码，在用户 login 时比对的是 用这个值和用他们输入的用户和密码生成的 md5 值，如果相等，则是系统有效用户：

#### UserServiceImpl.java

```
@Override
public User getUser(String username, String password) throws
ServiceException {
    String passwordMd5 =
MessageDigestUtil.md5(password.getBytes("UTF8"),
username.getBytes("UTF8"));

    return this.userDao.getUser(username, passwordMd5);
}
```

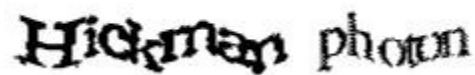
## ...9.2 使用 Kaptcha

DoS(Denial of Service) 恶劣的家伙们最常使用的方式之一， 注册，登录，搜索（可选）需要 CAPTCHA(全自动区分计算机和人类的图灵测试) 来应对这种无聊的攻击手段。

系统需要安全，但是不能丧失用户友好性。

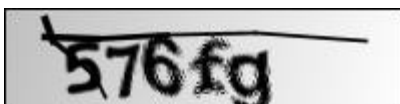
使用开源的或者自制的 captcha 已经很常见了，但希望这种 captcha 不要把系统的真正用户惹烦了。

目前开源的大多数实现生成的 captcha 的可读性太差，以下所谓的 TOP10 IT 公司他们的注册流程中采用的 captcha：



你能在 10 秒内看清楚，并且猜对吗？我猜大部分人都有这样的经历，一遍，两遍，三遍... 总是“**你所输入的字符与图片显示的不相符 ...**”

还好 [kaptcha](#) 的作者意识到了这一点，以下是 kaptcha 生成的图片：



### ...9.2.1 kaptcha 配置

#### ...9.2.1.1 KaptchaServlet

在 web.xml 中配置 servlet

#### web.xml

```
<servlet>
  <servlet-name>Kaptcha</servlet-name>
  <servlet-
class>com.google.code.kaptcha.servlet.KaptchaServlet</servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Kaptcha</servlet-name>
  <url-pattern>/kaptcha.jpg</url-pattern>
</servlet-mapping>
```

### ...9.2.1.2 KAPTCHA\_SESSION\_KEY

对于 Spring MVC controller 可以以下代码进行 Session 属性 KAPTCHA\_SESSION\_KEY 进行校验,

#### UserFormController.java

```
private boolean validateCaptcha(HttpServletRequest request) {
    String kaptchaExpected = (String)
request.getSession().getAttribute(com.google.code.kaptcha.Constants.KAPTCH
A_SESSION_KEY);
    //String kaptchaReceived = userForm.getKaptcha();
    String kaptchaReceived = request.getParameter("kaptcha");

    if (log.isDebugEnabled()) {
        log.debug("Received kaptcha: " + kaptchaReceived + " is comparing
with Expected kaptcha: " + kaptchaExpected + "...");
    }

    if (kaptchaReceived == null || !
kaptchaReceived.equalsIgnoreCase(kaptchaExpected)) {
        log.error("Received kaptcha: " + kaptchaReceived + " is comparing
with Expected kaptcha: " + kaptchaExpected + "...");
        return false;
    }

    return true;
}
```



### ...9.3 使用 https (SSL 通道)

用户敏感信息提交页面需要 https (SSL 通道), 没有 ssl 的表单提交没有任何安全可言, 如果你的系统在用户提交敏感信息时没有使用 https, 请立即 file 一个 p0 的 bug。

事实上采用 https 是非常简单的, 因为没有那个 web 服务器不支持这一协议的。

1. 在页面中 hardcode https url 跳转, 然后在处理完成后 hardcode http url 跳回
2. 采用 filter 或者 interceptor 将所有需要跳转的请求统一在同一个地方处理, 然后,
  - 在整个 session 中继续使用 https
  - 或者在请求完成时 hardcode 跳转回 http url, 如果工作流不是很复杂的话, 也同样可以使用 filter 或者 interceptor 完成返回的逻辑

#### HttpsUrlRewritingInterceptor.java

```
public class HttpsUrlRewritingInterceptor extends HandlerInterceptorAdapter {

    private int sslPort;

    public void setSslPort(int sslPort) {
        this.sslPort = sslPort;
    }

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        if ("https".equals(request.getScheme()) == false) {
            // if it is already in https, bypass
            StringBuilder sslUrl = new StringBuilder();
            sslUrl.append("https://").append(request.getServerName()).append(":")
.append(sslPort).append(request.getRequestURI());

            response.sendRedirect(sslUrl.toString());
        }

        return true;
    }
}
```

### ...9.4 密码算法及其参数的选择

密码算法或其参数的选择本着够用就行, 但要支持可扩展。科学是无止境的, 同样, 黑客利用技术的手段也一样。再有, 过去可靠的算法, 并不意味着目前或将来一定可靠。如同 [王小云](#) 破解 MD5 和 SHA-1 算法引起美国人恐慌一样。

密码本身是性能的杀手, 是人类“自找麻烦”, 就象我们给家里安防盗门, 公司的配保险柜一样。选择应该本着够用就行, 但一定要保留扩展的余地, 这会涉及到

- 密码算法

- 算法参数
- 数据库字段长度的定义

例如：目前 RSA 算法 key 的长度为 1024 已经足矣就会一般性的攻击了。

#### RSAUtil.java

```
private static final String ALGORITHM = "RSA";
private static final int keyLength = 1024;

public static KeyPair generateKeyPair() throws NoSuchAlgorithmException {
    KeyPairGenerator keyGen = KeyPairGenerator.getInstance(ALGORITHM);
    keyGen.initialize(keyLength);
    KeyPair keyPair = keyGen.generateKeyPair();
    return keyPair;
}
```

## ...9.5 OAuth

### ...9.5.1 存在的安全标准

1. *Basic Authentication* 离开了 HTTPS，毫无安全可言；HTTPS 对于分布式网络架构不友好，且性能消耗大。
2. *Digest Authentication* 概念复杂，工业支持不充分
3. *Certificate (mutual)* 昂贵的证书签发费，自签证书不可信任；必须基于 HTTPS 握手相互认证过程；

### ...9.5.2 基于数字签名(**signature**)的安全方案


用户只需要一个登录用户管理系统的用户名 / 密码对，这个密码不会用作 服务身份证，如：REST API. 用户只需登录并到 Profile 中取出 API Key 和 Secret Key.

1. 密钥初始化 并关联到用户数据

```
${access key} = MD5(RSA KeyPair.public)
```

```
${secret key} = MD5(RSA KeyPair.private)
```

**MD5** 的目的是使 **Key** 的长度缩短，以方便用于构造 **URI**

Profile Information	
Language Preference:	<input type="text" value="汉语"/>
Favourite Category:	<input type="text" value="DOGS"/>
<input checked="" type="checkbox"/> Enable My Favorite Product List	
<input checked="" type="checkbox"/> Enable My Banner	
API Key:	e4fae4f09fd3b2e6201b7b213d4deae7
Secret Key:	59573390586f8694ebe597c607e9c754
<i>Your Secret Access Key is a secret and should be known only by you and Hjpetstore. It is important to keep it confidential to protect your account. Never include it in your requests to Hjpetstore REST and never e-mail it to anyone.</i>	
<div> <input type="text"/></div>	
<input type="button" value="submit"/>	

## 2. 客户端构造请求

HTTP HEADER:

`${date}` =

Date: `EEE yyyy-MM-dd HH:mm:ss zzz`

因为 *http 1.1* 并没有规范化 *HTTP HEADER* 的字符集，从兼容的目的，大多数 *SERVER* 采用 *us-ascii*。所以格式化日期时，使用 *TIME\_ZONE\_UTC*, *Locale.US*

`${api key}` = `URLEncoder.encode(access key, CommonUtil.UTF8);`

`${secretKey}` = 通过登录从用户信息中获得

`${signature}` =

```
URLEncoder.encode (
rfc2104HmacSha512(
httpMethod\n
date\n
resourcePath\n),
secretKey,
CommonUtil.UTF8);
```

Request =

`http://host:port/resourcePath?apiKey=${api key}&signature=${signature}`

例如:

```
pprun@pprun-laptop:~$ curl -H 'Content-Type: application/xml' -H 'Accept: application/02-11 19:10:46 UTC' 'http://localhost:8080/hjpetstore/rest/products/dog?apiKey=e4fae4f09fd3b2e6201b7b213d4deae7&signature=zVe5WbWJuJuPQpFLJVpJ4XNweeKvGSLBBJTace4BayNH07x3poa8gHsIxLkIpFXsd5OQ%3D%3D&page=1&max=100'
```

### 3. 服务端验证

```
${access key} = URLDecoder.decode(api key, CommonUtil.UTF8);
```

```
${secret key} = get from user data by ${access key}
```

```
${date} = Get from HTTP HEADER
```

```
${httpMethod} = HTTP METHOD
```

```
${resourcePath} = Request URI
```

```
${signature} = URLDecoder.decode(signature passed in URI, CommonUtil.UTF8);
```

```
${calculated signature} =
```

```
URLEncoder.encode (
```

```
rfc2104HmacSha512(
```

```
httpMethod\n
```

```
date\n
```

```
resourcePath\n),
```

```
secretKey,
```

```
CommonUtil.UTF8);
```

```
if (calculatedSignature.equals(signature) == true) authenticate successful
```

```
else failed
```

```
OauthFilter.java
```

```
private void validateSignature(HttpServletRequest  
httpServletRequest) {
```

```
String apiKey =
HttpServletRequest.getParameter("apiKey");

    if (apiKey == null || apiKey.trim().isEmpty()) {
        throw new
CodedValidationException(ErrorCodeConstant.ERROR_CODE_INVALID_API
_KEY, "apiKey can not be null");
    }

    if (log.isDebugEnabled()) {
        log.debug("apiKey before decode: " + apiKey);
    }

    // decode back
    try {
        apiKey = URLDecoder.decode(apiKey, CommonUtil.UTF8);
    } catch (UnsupportedEncodingException ex) {
        throw new
SignatureException("UnsupportedEncodingException when
URLDecoder.decode: " + CommonUtil.UTF8, ex);
    }

    if (log.isDebugEnabled()) {
        log.debug("apiKey after decode: " + apiKey);
    }

    String secretKey =
userService.getUserSecretKeyForApiKey(apiKey);
    if (secretKey == null) {
        throw new
CodedValidationException(ErrorCodeConstant.ERROR_CODE_INVALID_API
_KEY, "no secretKey can be found for the apiKey: " + apiKey);
    }

    String signature =
HttpServletRequest.getParameter("signature");

    if (signature == null || signature.isEmpty()) {
        throw new
CodedValidationException(ErrorCodeConstant.ERROR_CODE_INVALID_SIG
NATURE, "signature can not be null");
    }

    if (log.isDebugEnabled()) {
        log.debug("signature before decode: " + signature);
    }

    // decode back
```

```
        try {
            signature = URLDecoder.decode(signature,
CommonUtil.UTF8);
        } catch (UnsupportedEncodingException ex) {
            throw new
SignatureException("UnsupportedEncodingException when
URLDecoder.decode: " + CommonUtil.UTF8, ex);
        }
        if (log.isDebugEnabled()) {
            log.debug("signature after decode: " + signature);
        }

        String date = httpRequest.getHeader("Date");
        if (date == null || date.isEmpty()) {
            throw new
CodedValidationException(ErrorCodeConstant.ERROR_CODE_NO_FOUND_DA
TE_HEADER, "Date header MUST be set in the request");
        }
        if (log.isDebugEnabled()) {
            log.debug("header Date: " + date);
        }

        // String has been Signed = HTTP-Method + "\n" + Date +
"\n" + resourcePath + "\n";
        String httpMethod = httpRequest.getMethod();
        if (log.isDebugEnabled()) {
            log.debug("httpMethod: " + httpMethod);
        }

        String resourcePath = httpRequest.getRequestURI();
        if (log.isDebugEnabled()) {
            log.debug("resourcePath: " + resourcePath);
        }
        String calculatedSignature =
MessageDigestUtil.calculateSignature(httpMethod, date,
resourcePath, secretKey);

        if (log.isDebugEnabled()) {
            log.debug("calculateSignature = " +
calculatedSignature);
        }

        if (calculatedSignature.equals(signature) == false) {
            throw new SignatureException("Invalid signature: the
calculated signature (" + calculatedSignature +
") does not match with the signature passed
in (" + signature + ").");
        }
    }
}
```

```
}  
}
```

### ...9.5.3 OAuth

[OAuth](#) 被国际巨头所采纳: Twitter, Yahoo, Google, Amazon

[OAuth 2](#) 正在被 [IETF OAuth WG](#) 采纳, 期待早日标准化。

以上实现是基于 OAuth 思想, 简化其 3-leggs 成 2-leggs, 去掉 Token, 且不支持 revoke 功能



### ...9.5.4 应用举例

基于 SOA/ROA 架构, 如果 web service 面象的是大众, 是一个 public API, 安全策略将是怎样呢?

SOA / ROA 公共 API 的访问, 用户需要通过注册获取一个 public/private 密钥对, public api key, private Key 只用于加密和用户数字签名(signature)的过程, 然后 public api key 和 用户数字签名(signature)作为请求参数传入。象 Google/Yahoo/Amazon 的 public web service 的方式那样。

系统或许并不想因为这个“自找麻烦”的安全问题失去潜在的用户, 但同时也要保证系统的安全以及可控性, 比如非 VIP 用户, 每天 < 1000 requests

这个 public api key, 根据需求, 可以是 md5, sha-1(或许这个不应该在考虑之列了, 因为王小云的原因), sha-256, sha-512, 等等:

hjpetstore 提供一个 REST 服务: 通过关键字, 查找宠物店里的出售的宠物列表:

```
HjpetstoreController.java
```

```
/**  
 * RESTful match path '/products/{keyword}' with apikey as request
```

```
parameter.  
*  
* <p>  
* For example: user pprun <br />  
* {@code  
* curl -H 'Content-Type: application/xml' -H 'Accept: application/xml' -H  
'Date: Fri 2011-02-11 19:10:46 UTC'  
'http://localhost:8080/hjpetstore/rest/products/dog?  
apiKey=e4fae4f09fd3b2e6201b7b213d4deae7&signature=zVe5WbWJuJuPQpF  
LJVpJ4XMYTThdROf5iaU76zdWLweeKvGSLBBJTace4BayNH07x3poa8gHslxLklpF  
Xsd5OQ%3D%3D&page=1&max=100'  
* }  
*  
* @param apiKey  
* @param keyword  
* @return  
*/  
@RequestMapping(value = "/products/{keyword}", method =  
RequestMethod.GET)  
public ModelAndView getProductsByKeyword(  
    @RequestParam("page") int page,  
    @RequestParam("max") int max,  
    @PathVariable("keyword") String keyword) {  
  
    if (log.isDebugEnabled()) {  
        log.debug("HjpetstoreController is processing request for keyword: " +  
keyword);  
    }  
    Products products = hjpetstoreService.searchProductList(keyword, page,  
max);  
  
    ModelAndView mav = new ModelAndView("products");  
    mav.addObject(products);  
    return mav;  
}
```

请求认证是通过 OAuthFilter 来担当。

下面是 Client 的代码：

#### HjpetstoreRestClient Project - SearchProductListRest.java

```
/**  
* Swing Client using RestTemplate to call public service.  
*  
* @author pprun  
*/
```



```
public class SearchProductListRest {
    private static Log log = LogFactory.getLog(SearchProductListRest.class);

    private static final String ZONE_DATE_FORMAT = "EEE, yyyy-MM-dd
HH:mm:ss zzz";
    private RestOperations restTemplate;
    private XPathOperations xpathTemplate;
    private String apiKey;
    private String secretKey;

    @Required
    public void setApiKey(String apiKey) {
        this.apiKey = apiKey;
    }

    @Required
    public void setSecretKey(String secretKey) {
        this.secretKey = secretKey;
    }

    public SearchProductListRest(RestOperations restTemplate, XPathOperations
xpathTemplate) {
        this.restTemplate = restTemplate;
        this.xpathTemplate = xpathTemplate;
    }

    public void searchProductList(String keyword, String apikey, int page, int
max) {
        List<BufferedImage> images = searchProductImage(keyword, apikey,
page, max);
        displayImage(keyword, images);
    }

    @SuppressWarnings("unchecked")
    private List<BufferedImage> searchProductImage(String keyword, String
apikey, int page, int max) {

        final String restUri = "http://localhost:8080/hjpetstore/rest/products/
{keyword}?page={page}&max={max}";
        //the server need authenticate, we have set the http header

        HttpHeaders requestHeaders = createHttpHeader();

        String url = buildURL(HttpMethod.GET, requestHeaders, restUri, keyword,
page, max);
        HttpEntity<String> requestEntity = new HttpEntity(requestHeaders);
        HttpEntity<Source> response = restTemplate.exchange(url,
```

```
        HttpMethod.GET,
        requestEntity,
        Source.class,
        keyword, page, max);
    Source product = response.getBody();

    //Source product = restTemplate.getForObject(url, Source.class, keyword,
    page, max);

    final String imageUrl = "http://localhost:8080/hjpetstore/images/
    {imageName}";

    List<BufferedImage> images = (List<BufferedImage>)
    xpathTemplate.evaluate("//image", product, new NodeMapper() {

        @Override
        public Object mapNode(Node node, int i) throws DOMException {
            //Element image = (Element) node;
            org.jdom.Element image = new DOMBuilder().build((Element) node);
            String imageName = image.getValue();
            //String imageName = image.getFirstChild().getNodeValue();

            return restTemplate.getForObject(imageUrl, BufferedImage.class,
            imageName);
        }
    });

    return images;
}

private String buildURL(HttpMethod httpVerb, HttpHeaders requestHeaders,
String path, Object... urlVariables) {
    // construct the url
    String originPath = path; // make a copy because the process still need
    parametered path outside this method,
    if (urlVariables != null) {
        // if the path contains variable, expand it
        // http://localhost:8080/hjpetstore/rest/paymentpartner/{name}

        UriTemplate uriTemplate = new UriTemplate(path);
        URI expanded = uriTemplate.expand(urlVariables);
        path = expanded.getRawPath();
        if (log.isDebugEnabled()) {
            log.debug("expanded path = " + path);
        }
    }
}
```

```
String httpMethod = httpVerb.toString();
String date = requestHeaders.getFirst("Date");
String signature = MessageDigestUtil.calculateSignature(httpMethod,
date, path, secretKey);
try {
    StringBuilder s = new StringBuilder();
    s.append(originPath);
    s.append("&apiKey=");
    s.append(URLEncoder.encode(apiKey, "UTF-8"));
    s.append("&signature=");
    s.append(URLEncoder.encode(signature, "UTF-8"));
    if (log.isDebugEnabled()) {
        log.debug(s.toString());
    }

    return s.toString();
} catch (UnsupportedEncodingException ex) {
    throw new IllegalStateException("UnsupportedEncodingException
when URLEncoder.encode", ex);
}
}

/**
 * help method create HTTP headers.
 * @return
 */
private HttpHeaders createHttpHeader() {
    HttpHeaders requestHeaders = new HttpHeaders();
    // Accept
    requestHeaders.set("Accept", "application/xml");
    requestHeaders.set("Accept-Charset", "UTF-8");

    // Date
    String date = getDateStringWithZone(Calendar.getInstance(),
ZONE_DATE_FORMAT, TimeZone.getTimeZone("UTC"), Locale.US);
    requestHeaders.set("Date", date);

    return requestHeaders;
}
// now use OAuth, instead of Basic Authentication
// /**
//  * If the server need authenticate, we have set the http header.
//  * @return
//  */
// private HttpEntity<?> setHttpHeader() {
//     HttpHeaders requestHeaders = new HttpHeaders();
```

```
//      // Authorization
//      String userPassword = "pprun" + ":" + "pprunpprun";
//      String encoding = new
String(Base64.encodeBase64(userPassword.getBytes()),
Charset.forName("UTF-8"));
//      requestHeaders.set("Authorization", "Basic " + encoding);
//      // Accept
//      requestHeaders.set("Accept", "application/xml");
//      requestHeaders.set("Accept-Charset", "UTF-8");
//
//      HttpEntity<?> requestEntity = new HttpEntity(requestHeaders);
//      return requestEntity;
//  }

private void displayImage(String keyword, List<BufferedImage> imageList)
{
    JFrame frame = new JFrame("Hjpetstore " + keyword + " images");
    frame.setLayout(new GridLayout(3, imageList.size() / 3));
    for (BufferedImage image : imageList) {
        frame.add(new JLabel(new ImageIcon(image)));
    }
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
}

public static String getDateStringWithZone(Calendar cal, String format,
TimeZone timeZone, Locale locale) {
    if (cal == null) {
        return null;
    }

    SimpleDateFormat sdf = new SimpleDateFormat(format, locale);
    sdf.setTimeZone(timeZone);
    return sdf.format(cal.getTime());
}
}
```

## ...9.6 RSA

### ...9.6.1 为什么要引入 **RSA**

假设皮皮鲁和鲁西西分别就职于高安全的企业，由于业务往来，皮皮鲁需要给鲁西西发送加密邮件，他们可能会采用如下方案：

### ...9.6.1.1 对称加密

- 皮皮鲁要给鲁西西发邮件，首先得到或者由鲁西西传递给皮皮鲁一个鲁西西的密码
- 然后皮皮鲁用鲁西西的密码加密邮件然后传递给鲁西西
- 鲁西西用同样的密码解密邮件得到明文

这个流程里有一个重要的问题，密码的传送方式：邮件，电话，短信都是不安全的，而且就算是安全的，双方还得记住这个密码，明文保存起来违反了 不要将密码存成明文 原则。这就是为什么非对称加密诞生的原因，RSA 是最常用的非对称加密算法。

### ...9.6.1.2 非对称加密

- 皮皮鲁要给鲁西西发邮件，首先得到或者由鲁西西传递给皮皮鲁一个鲁西西的公钥
- 然后皮皮鲁用鲁西西的公钥加密邮件然后传递给鲁西西
- 鲁西西用自己的私钥解密邮件得到明文

但是这个流程里有一个重要的问题没有澄清，首先对邮件加密本身没有问题，但是会被另一种攻击，就是有黑客在中间截获了邮件后，然后自己用鲁西西的公钥加密一个邮件传给鲁西西。鲁西西怎么能区别她收到的加密邮件真的是皮皮鲁发出来的呢？所以这样就需要对身份认证的一个机制了，CA 认证服务器就是通过给系统内的用户发证书的形式来做身份认证。但是 CA 是需要付费的，人家凭什么免费保证你的信用呢，如果免费，所有可以去申签，保不准干坏事，CA 也会失去信任的，怎么办？

### ...9.6.1.3 非对称加密 and 消息摘要

简单的办法就前面讲到的 HASH 啦，消息摘要。流程如下：

- 皮皮鲁要给鲁西西发邮件，先对邮件用不可逆加密算法（如 md5 / sha-512 算法）对邮件生成一个消息摘要
- 然后用鲁西西的公钥对邮件本身加密
- 最后将密文连同消息摘要一起发给鲁西西
- 鲁西西通过自己的私钥去解密邮件，然后通过 md5 / sha-512 算法生成消息摘要，如果和前面的消息摘要一样，则可证明这个邮件是皮皮鲁发的

大家可能还有疑问，那个黑客不会也按照皮皮鲁的步骤生成消息摘要，一并传给鲁西西吗？  
是的，完全可能。但是消息摘要可以随便公布的，正如大家在下载软件时可以看到：

```
ebf7ad055bc39634065daa10de980d7e *ubuntu-7.10-alternate-amd64.iso
9a4ae3cfd68911a861d094ec834c9b48 *ubuntu-7.10-alternate-i386.iso
61c87943a92bc7bf519da4e2555d6e86 *ubuntu-7.10-desktop-amd64.iso
d2334dbba7313e9abc8c7c072d2af09c *ubuntu-7.10-desktop-i386.iso
43ff753b260729b12c7d21d3a6db8c73 *ubuntu-7.10-server-amd64.iso
7d88cd87df509a740d9f47b9bbf1375e *ubuntu-7.10-server-i386.iso
5308a79f5e652edba5be84644ee14b09 *ubuntu-7.10-server-sparc.iso
```

皮皮鲁当然也可以公布自己的消息摘要：

To 鲁西西:

皮皮鲁(SHA512 checksum:  
adace8b3f847066f8231304134d320f2cf416fc5ed9525b6594f1f8d45015fc2c9791b67b059f894f5127df6e  
157c7075cb965356b7d61d94ee6a8c838726359)

Hjpetstore 利用这种安全策略与 payment partner 集成，见 Payment Partner 集成 Payment Partner 集成

## ...9.7 Spring Security 访问控制

见 Spring Security 访问控制 Spring Security 访问控制

## ...9.8 PCI (Payment Card Industry)

对于高安全系统，例如 PCI (Payment Card Industry 支付卡行业数据安全标准)，安全专家少不了。

如果你身处这样的一个安全系统，首先恭喜你！

但同时，你要知道身上的责任，公司的信用是建立在系统的安全之上的。同时信用卡安全事故折射这一现实的世界——对于这样的系统，安全不再是“自找麻烦”。反而有时要把自己当成“黑客”，通过写测试模拟各种攻击场景。

作为工程师，必须与安全专家紧密合作，不能有任何冒险的心理。此外，所有安全数据必须跟普通的数据隔离，否则将遭到 PCI 委员会的严厉处罚，一个月的处罚金足以 pay 一个安全顾问一年。

Hjpetstore 已经预备了这一方案，虽然对卡片及用户数据没有与普通数据分隔开，但是，SecurityService 设计成一个 REST 组件，可以单独部署。

## 10 事务

- 事务的基本概念
- Java EE 三种事务模型
- Spring 四种事务策略

### ...10.1 事务的基本概念

#### ...10.1.1 事务隔离级别

在 java.sql.Connection 类的 [javadoc](#) 中声明了五个事务隔离级别常量成员：

- **TRANSACTION\_NONE**

没有事务支持，不可能用在真正的应用中

- **TRANSACTION\_READ\_UNCOMMITTED**

允许“脏”读，不可再现重读和影子读：具体表现为，允许数据库的一行由一个事务单元 A 进行改变的同时，允许由另一个事务单元 B 读取这个被事务 A 改变后的同一行，而此时事务 A 还没提交它的改变，因为数据正处在改变状态还未将改变提交到数据库中（术语“脏”的来历），如果事务后来又回滚了它的改变，那么事务 B 读取的信息将是无效的，而这无效只有等到事务 B 提交时才能察觉到（以失败告终）。此级别提高了事务的并发性，但同时导致了无效的读增多。此方案在只读事务单元时有用

- **TRANSACTION\_READ\_COMMITTED**

此级别不允许“脏读”，但未可再现重读和影子读还是允许的，此级别阻止了读取未提交的改变。

- **TRANSACTION\_REPEATABLE\_READ**

此级别不允许“脏读”，也不允许不可再现的重读，但影子读还是允许的。所谓的“不可再现的重读”为：一个事务单元 A 读取一行，紧接着第二个事务单元 B 改变了这行，但未提交更改，事务单元 A 再接着读取这行，如果允许事务单元 A 两次读取的内容不同，则叫做不可再现的重读，如果数据库保证了两次读取的内容必须相同，则为可再现重读。

- **TRANSACTION\_SERIALIZABLE**

此级别指示“脏读”，不可再现重读及影子读都被禁止。本级别除了 TRANSACTION\_REPEATABLE\_READ

中的限制之外，还得满足未影子读。所谓影子读为：如果一个事务单元 A 通过 Where

条件限制了取回的结果集，而第二事务单元 B 插入了一行且这行满足 Where

条件，但未提交更改，当事务单元 A 再次通过同样的条件进行查询时，如果可以获得刚刚由事务单元 B 插入的记录，则表示影子读，如果不可能获得则叫禁止影子读

### ...10.1.2 事务传播行为

除了上面已经介绍了的隔离级别，另一个重要的概念则是事务传播行为(Propagation behavior)：

- **PROPAGATION\_MANDATORY**

指示方法执行必须具有事务支持，否则异常将会抛出。

- **PROPAGATION\_NESTED**

指示如果当前已经存在一个事务，该方法将在自己的事务单元中执行，但该事务单元将嵌套在已存在的事务单元之中，而且这个事务单元可单独于外包的事务单元进行提交、回滚。如果没有事务单元存在，它的行为与 PROPAGATION\_REQUIRED 一样，一个新的事务单元将被初始化出来。但是每个数据库厂间对这个传播行为的支持不大一样。请留意。

- **PROPAGATION\_NEVER**

指示方法不应运行在一个事务单元中，如果已经有一个事务单元存在了，将抛出异常。

- **PROPAGATION\_NOT\_SUPPORTED**

指示方法不应运行在一个事务单元中，但如果已经存了一个事务单元，在方法的执行过程中将会将事务功能挂起。

- **PROPAGATION\_REQUIRED**

指示方法必须运行在事务单元中，如果已经存了一个事务单元，则方法将合并到该事务单元中去，如果没有存在的事务单元，则将创建出一个新的事务单元。

- **PROPAGATION\_REQUIRES\_NEW**

指示此方法必须运行在自己的事务单元中，如果一个事务单元已经存在的话，一个新的事务单元将创建且原先存在的事务单元在这个方法的执行其间将被挂起。

- **PROPAGATION\_SUPPORTS**

指示方法不必一定要运行在事务单元中，但是如果有已经存在的事务单元的话，它也乐意在事务单元中运行。

### ...10.1.3 只读提示

此属性只对 PROPAGATION\_REQUIRED, PROPAGATION\_REQUIRES\_NEW, and PROPAGATION\_NESTED 传播行为有意义，因为对于其它没有事务单元的传播行为是没有必要的。只读提示给予数据库有机会作出更大胆的优化。

如果是使用 Hibernate 作为 O/R mapping 的话，Hibernate 可以使用 FLUSH\_NEVER 模式避免不必要的数据同步消耗，因为是吹读，数据同步是根本没有必要的。

#### **ORACLE Tip:**

*Read-only connections are supported by the Oracle server, but not by the Oracle JDBC drivers. For transactions, the Oracle server supports only the TRANSACTION\_READ\_COMMITTED and TRANSACTION\_SERIALIZABLE transaction isolation levels.*

*The default is TRANSACTION\_READ\_COMMITTED.*

### ...10.1.4 事务超时周期

对于长时间运行的事务操作，必需估计操作的最坏情形（也就可能的最大运行时间），由于事务会大量使用数据库锁，将相关的数据库列，或数据库行或整个表锁住。长时间的锁将会使数据库长时间不可用。通过设置超时时间，超过了这个时间，整个事务会回滚。

## ...10.2 Java EE 三种事务模型

模型指的是那些成型了的概念，所采用的平台中可以利用的方式。Java EE 支持三种事务模型：

- 无事务
- 编程式事务
- 声明式事务



### ...10.2.1 无事务

呵呵，无事务竟然也是一种模型？

因为 JDBC connection 默认属性 `autoCommit` 是 `true`, 所以, 在纯 JDBC 代码或者调用存储过程的代码中, 你完全可以不用关心事务 (当然前提是你的业务模型足够简单)。

可是在引入 OR/M 框架, 如 Hibernate 后, 即使是 `readonly` 的操作也要包围在事务访问中。所以事务的概念才渐渐被重视。

此外, 在使用声明式事务模型时, 由于使用不当或者配置失误, 也会沦为无事务模型。这种情况, 在高并发的企业应用中, 会导致 `database connection` 或 `hibernate session time out`。原因是 `connection` 或 `session` 的打开与释放并不是程序控制, 而是由底层的 JDBC 控制。依赖于所使用的 `connection factory` 或者 `connection pool`, 在 `idle` 以及相关的 `timeToLive` 时间到时, 才会将连接释放。

### ...10.2.2 编程式事务

对于编程式事务, 本文不讨论, 因为出了考虑的范围。简单地讲:

- 在 EJB 或其他标准的 Java EE 使用 `UserTransaction`
- 在 Spring 的世界里, 使用它的 `TransactionTemplate` / `TransactionManager`

### ...10.2.3 声明式事务

以 Spring 声明式事务模型为例, 来消化一下这一大箩筐概念。但是, 还有关于 AOP 的新的概念:

#### ...10.2.3.1 advice

基于方法命名 `pattern` 的 `advice` 配置。

##### ApplicationContext.xml

```
<!--
Transaction advice definition, based on method name patterns.
Defaults propagation="REQUIRED" for all methods
and with read-only hint for all methods which only retrieve data.

NOTE: it is important to comply with the team internal name convention
for this contract, and we DO need '*' pattern to match all methods which
is not enumerated in the list.
-->
<tx:advice id="txAdvice">
  <tx:attributes>
    <tx:method name="get*" read-only="true"/>
    <tx:method name="is*" read-only="true"/>
    <tx:method name="search*" read-only="true"/>
    <tx:method name="find*" read-only="true"/>
    <tx:method name="insert*" />
    <tx:method name="save*" />
    <tx:method name="update*" />
    <tx:method name="*" />
  </tx:attributes>
</tx:advice>
```

值得注意的是, 虽然这里考虑到了大多数常见的方法命名, 但是, 如果团队里头没有一个代码规范, 可能没法阻止新来的成员将 `getxxx` 方法命名成 `retrievexxx`, 这样一来, 根据上面的定义, 这个方法将被归到

<tx:method name="\*" /> 这一 fallback 项中去，并没有利用到 read-only 特性。所以代码命名规范很重要，岗前的 coach 也很重要。

### ...10.2.3.2 Advisor

Advisor 是将前面定义的事务 advice 绑定到 pointcut 上，也就是将事务属性作用相应的 joint point（这里是方法的执行）

#### ApplicationContext.xml

```
<!-- transaction management -->
<aop:config>
  <!--
    This definition creates auto-proxy infrastructure based on the given pointcut,
    expressed in AspectJ pointcut language.
    We might simply this configuration by making use of the powerful pointcut expression,
    but if express clearly, it is useful and important for new team member to
    understand the whole Transacion weaving/configuration, transaction is vital for
    business.

    Spring recommends that you only annotate concrete classes (and methods of concrete
    classes) with
    the @Transactional annotation, as opposed to annotating interfaces.
  -->
  <aop:advisor pointcut="execution(public *
org.pprun.hjpetstore.service.CategoryServiceImpl.*(..))" advice-ref="txAdvice"/>
  <aop:advisor pointcut="execution(public *
org.pprun.hjpetstore.service.ItemServiceImpl.*(..))" advice-ref="txAdvice"/>
  <aop:advisor pointcut="execution(public *
org.pprun.hjpetstore.service.OrderServiceImpl.*(..))" advice-ref="txAdvice"/>
  <aop:advisor pointcut="execution(public *
org.pprun.hjpetstore.service.ProductServiceImpl.*(..))" advice-ref="txAdvice"/>
  <aop:advisor pointcut="execution(public *
org.pprun.hjpetstore.service.UserServiceImpl.*(..))" advice-ref="txAdvice"/>

</aop:config>
```

## ...10.3 Spring 四种事务策略

### ...10.3.1 前端事务策略

Service 层不考虑事务问题，由前端 Web Controller 或者 Web Service 客户来控制事务。

应用此类事务策略的原因是，Service 已存在的接口满足不了客户端的需求，太过于细粒度化。即从客户端的角度看来，一个完整的事务被拆分成了两个或两个以上的 Service 方法，如果事务设置于 Service 层，保证不了客户端的原子性。

举个经典的例子：皮皮鲁要出游巴厘岛，他选择了一家网上旅游公司，他们除了提供机票预定外，还有旅店预定。很显然，如果皮皮鲁光订上了机票，可是在旅游公司的系统在跟旅店的系统合作时出现了问题，结果没能成功订上旅店。如果整个交易被分拆成两个单独的事务，皮皮鲁需要为定上的机票付款吗？没有住处，皮皮鲁去得成巴厘岛吗？

### ...10.3.2 Service 层事务策略

SOA / ROA 架构最常用的事务模型，配置见 AdvisorAdvisor

### ...10.3.3 高并发事务策略

从 Service 层事务策略演化而来，原因是 Service 层的 API 过于粗粒度，复杂的逻辑处理掺杂很多 DAO api 调用，导致负载能力不足，即能够支持并发的用户数达不到指标。

由于事务的开启意味着对资源上锁或 MVCC，如果事务开启的时间过长，要么本次交易由于 transaction time out 导致交易失败，要么，由于并发访问同一资源，但由于长时间的 transaction 锁，导致，其它并行的交易“饿死”，而以 session time 失败。

因此希望将锁的时间拆分：

将事务的划分往底层推，即把需要访问的所有 DAO 做出一个 Transaction Wrapper，将对应的 Service 层的事务剥离。

#### ...10.3.3.1 存在的风险

- 只读事务有可能读到过期的数据

也就是在数据取出后，由于出了 DAO 就释放了锁，另一个事务有可能更新了这一数据，而当前交易却在使用过期的数据进行计算

- 写事务存在乐观锁问题

同样由于出了 DAO 锁就已经释放，在 DAO 外的计算中对 detached domain 对象进行更新，然后在后续的 DAO 操作中提交更改，可是这时却发现，其他并行的 transaction 已经对实体进行了更改。如果是基于 hibernate <version> 的乐观锁并发访问控制，由于提交更新时发现，version 值不是实体当前的 version + 1, (由于每次的更新，都将 version 递增)，因此 javax.persistence.OptimisticLockException 抛出，这是期待的结果。交易回滚，前端提示用户重试。

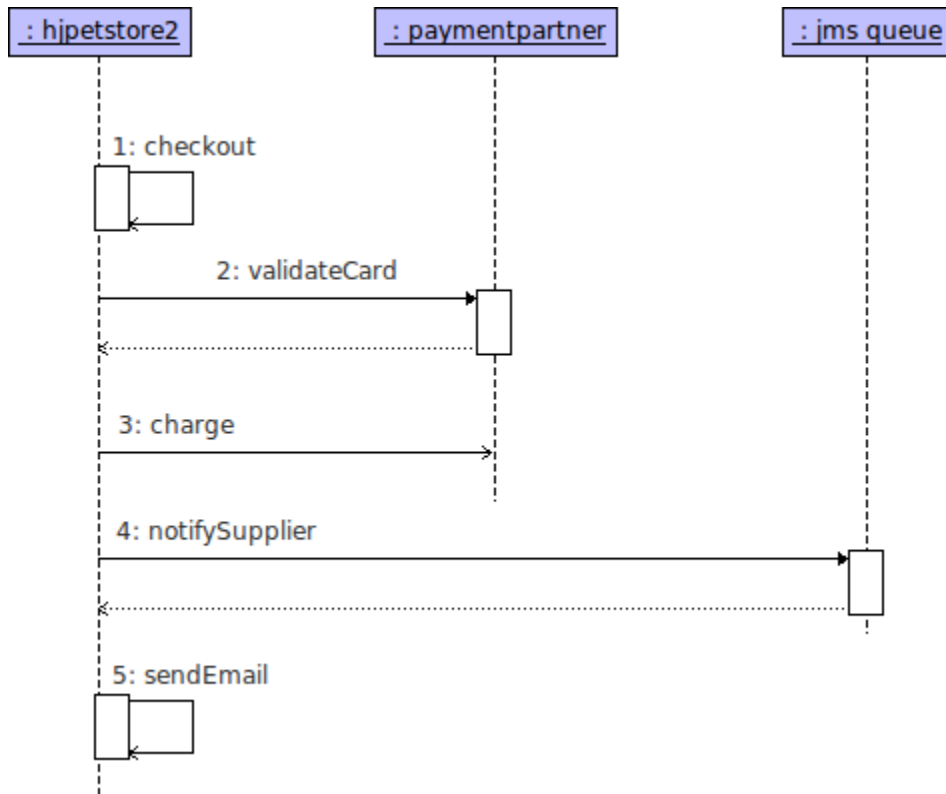
但如果 javax.persistence.OptimisticLockException 过于频繁，说明这一事务策略并不适合。或许需要 PessimisticLock

```
Order o = (Order) session.get(Order.class, 1, LockMode.UPGRADE);
```

### ...10.3.4 高性能涉外事务

首先得指出的是，这是个很难实现的策略，在考虑这一策略前，请三思系统的整体架构问题。

典型的例子是 order charge (即银行卡收钱问题)，正常的业务逻辑如下：



大部分 ecommerce 系统都是通过和支付网关打交道来完成这一流程，支付网关往往提供 WebService API（不考虑简单的跳转到支付系统的情况）。问题是：

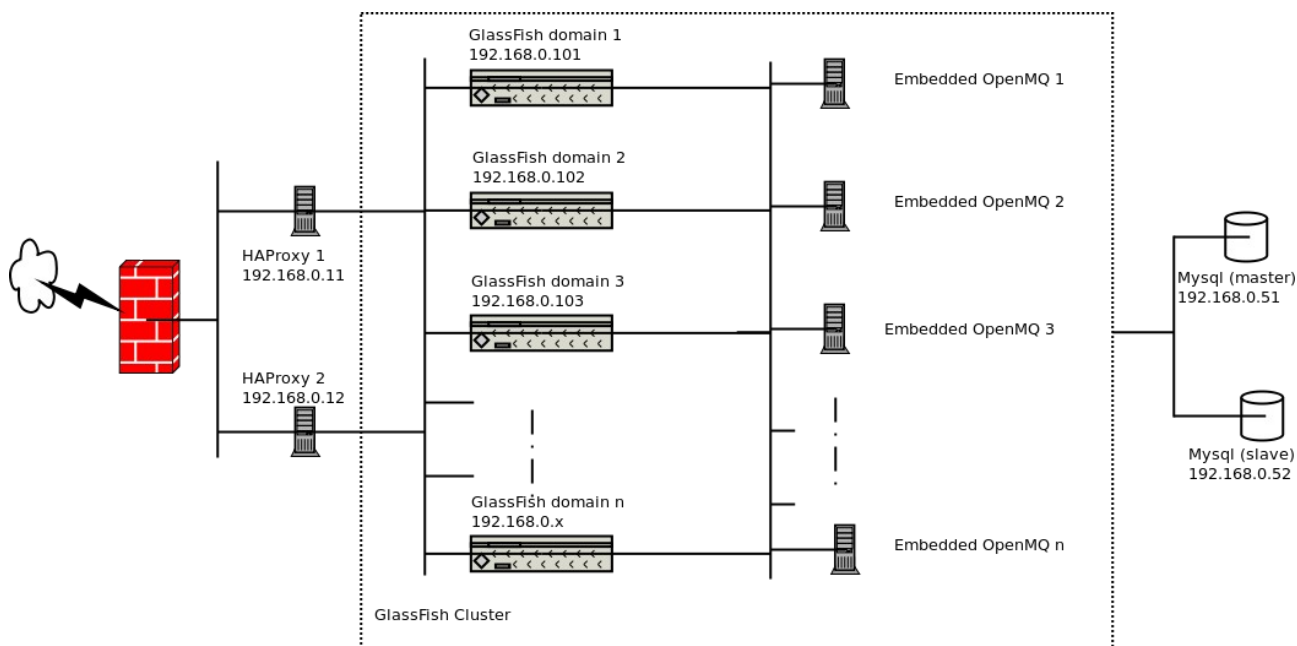
1. 如果支付网关暂时不可用，业务要不要往下进行？
2. 回答这一问题涉及到和支付网关的集成是同步还是异步模式？
3. 如果采用同步模式，对外部服务的调用要不要包含在声明式的事务作用域中？
  - 如果包含在事务中  
那么就形成了不折不扣的 long time transaction and long time locking，它往往就是系统的瓶颈。（想想，收钱可是系统最关键的功能）
  - 如果不包含在事务中  
怎么来保证 ACID？复杂的方案是 [Compensating\\_transaction](#)（中文叫冲正），采用复杂的 Try Cancel/Confirm (TCC) 策略

以上序列图是采用异步 retry 的基于 JMS 的模式避免以上复杂的事务问题，只能成功 charge 的 order 才会继续 step 4, 5（见以上序列图）。

## 11 部署架构

- 物理拓扑
- 负载均衡
- 数据库失效转移
- 应用服务器集群

## ...11.1 物理拓扑



## ...11.2 负载均衡

采用 HAProxy load balancer，其竞争对手还有：

- [Pen](#) 是一个非常简单易用的 load balancer，但是功能上有些限制，比如并发连接数问题
- [nginx](#) 是一个非常不错且采纳率非常高的 load balancer

但是，对于旗鼓相当的竞争对手，我一般更青睐于后起之秀，所以采用 [HAProxy](#)

### haproxy.cfg

```
global
maxconn 40000
nbproc 4
daemon
quiet

defaults
mode http
clitimeout 120000
srvtimeout 30000
contimeout 4000
balance roundrobin

listen glassfish_cluster 0.0.0.0:8080
mode http
```

```
stats enable
stats uri /ha?stats
balance roundrobin
server glassfish_d1 192.168.0.101:8080 weight 1 maxconn 10000 check
server glassfish_d2 192.168.0.102:8080 weight 1 maxconn 10000 check
server glassfish_d3 192.168.0.103:8080 weight 1 maxconn 10000 check
server glassfish_d4 192.168.0.104:8080 weight 1 maxconn 10000 check
```

### ...11.3 数据库失效转移

基于 Mysql master/slave 模型的数据库失效转移策略

TODO: add install instruction

#### ...11.3.1 Mysql JDBC 配置

Jdbc.properties

```
jdbc.url=jdbc:mysql://192.168.1.6:3306,192.168.1.8:3308/hjpetstore?
useUnicode=true&characterEncoding=UTF-
8&autoReconnect=true&failOverReadOnly=false
```

### ...11.4 应用服务器集群

#### ...11.4.1 GlassFish v3

GlassFish cluster 的支持并不算完美，但还在进一步演化。

集群采用 n + 1 节点模式，1 台 管理节点 GMS，3 台 工作节点

Manager node	192.168.0.101
Node1	192.168.0.102
...	...

(TODO: add instructions and the admin console screenshots)

#### ...11.4.2 OpenMQ

采用内嵌的 OpenMQ，如果业务消息量巨大的话，应该考虑单独的 OpenMQ 实例。



## 12 SOA / ROA 应用架构

- 数据层 DAO
- JAXB
- Spring MVC 3 RESTful Controller

### ...12.1 数据层 DAO

对于企业应用，整个业务都是围绕数据展开，因此，数据访问层设计与实现的好坏直接影响到整个系统的质量。基于 Hibernate Spring 开源框架的 DAO 层给实现带来了以下好处：

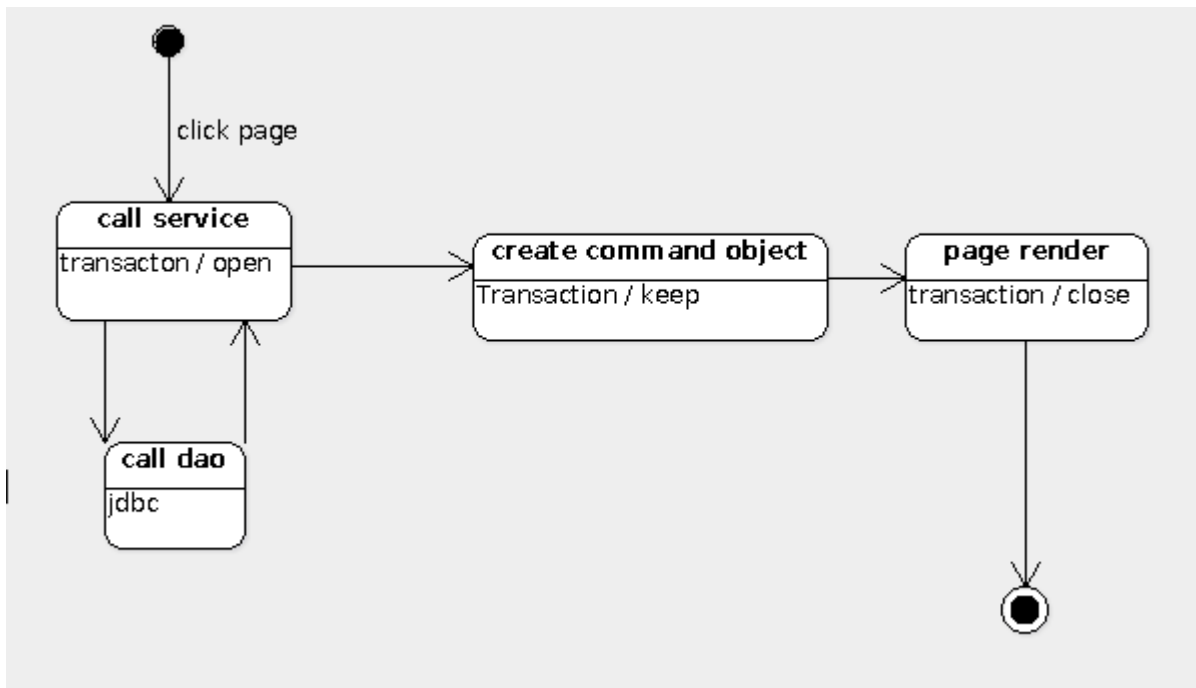
1. 减少了重复的模板代码
  - 实例创建工厂
  - 连接的获取与释放
  - 事务管理
  - try/catch/finally
2. 规范了异常体系与处理逻辑

与此同时，框架的集成也存在一反模式，“Open Session in View”就是其中之一。

#### ...12.1.1 No Open Session in View pattern

Open Session in View pattern 被 Web 应用用来实现 conversation 的便利方法，

但同时它被视为性能与负载的杀手，它将一个本该早早释放的数据库资源锁延长到整个用户会话（user conversation）才释放：



Transaction 的状态跨越了整个应用的层次结构：dao->service->controller-jsp, 直到 page render 成功后，transaction 才关闭。

Transaction 的打开意味着数据使用 lock 或者 multiversion concurrency control (MVCC) 机制来控制对数据的访问。时间越长，如果是非 read-only 交易，这一 Transaction 将阻止任何其它对同一数据并发访问操作。

这也是大多数 Java EE 应用失败的根源，假死，翻多页后出现空白页，可支持的并发用户数性能低下。

怎么办？可是如果不使用 Open Session In View，映射文件中又不许指定了 lazy = true，我怎么来应对 Hibernate 丑名昭著的 LazyInitializationException？

### ...12.1.2 eagerly fetch mode

通过一次在 Dao 方法中将所有将要用到的数据一次性取出，将事务时间最小化，以提高系统的并发和负载能力：

HibernateOrderDao.java

```
@Override
public List<Order> getOrders(Calendar from, Calendar to) {
    return getSession().createCriteria(Order.class)
        .setFetchMode("shipAddress", FetchMode.JOIN)
        .setFetchMode("billAddress", FetchMode.JOIN)
        .setFetchMode("orderLineItems", FetchMode.JOIN)
        .setFetchMode("orderLineItems.item", FetchMode.JOIN)
        .add(Restrictions.between("createTime", from, to))
}
```



```
.addOrder(org.hibernate.criterion.Order.asc("orderStatus"))
.list();
}
```

如果涉及到的关联对象太多，且只有少部分关联对象的属性才会使用到，这种情况下，将数据全部取出并穿越紧缺的网络资源，然后，只使用其中的小部分后就丢弃，显然是需要优化的。

设计师需要仔细分析哪些属性在同一个 conversation 中的后续的步骤要用到，然后定义出一个 DTO 封装所有需要用到的关联对象的属性集合。

可以采用 HQL 动态实例化(dynamic instantiation) 或者 Criteria ResultTransformer 来进行这一优化。

#### ...12.1.2.1 HQL 动态实例化(dynamic instantiation )

```
SELECT NEW OrderSummary (...) from Order o, Address a, Address b,
OrderLineItem oli, Item i where ...
```

#### ...12.1.2.2 Criteria ResultTransformer

```
criteria ResultTransformer
getSession().createCriteria(Order.class)
    .setFetchMode("shipAddress", FetchMode.JOIN)
    .setFetchMode("billAddress", FetchMode.JOIN)
    .setFetchMode("orderLineItems", FetchMode.JOIN)
    .setFetchMode("orderLineItems.item", FetchMode.JOIN)
    .add(Restrictions.between("createTime", from, to))
    .addOrder(org.hibernate.criterion.Order.asc("orderStatus"))
    .setProjection(Projections.projectionList().add( ) ...)
    .setResultTransformer(
        new AliasToBeanResultTransformer(OrderSummary.class)
    );
.list();
```

### ...12.2 JAXB

[JAXB](#) 已经包含在标准的 Java SE 6 中了，在 hjpetstore 2 中，jaxb 作为主要的 marshaller 机制，用于

- RESTful controller 消息转换
- batch 的 xml 存储

### ...12.2.1 **RESTful controller** 消息转换

在 SOA / ROA 架构中，web service 的请求与响应（还是说 request and response 吧，呵呵）往往被封装成对象，而不是简单的文本或请求参数（当然，如果业务逻辑足够简单也无妨），而且，往往需要对请求对象做 validation。Jaxb 即在这一背景下诞生的。

Hjpetstore 2 中基于 Spring MVC 3 的 controller 采用 jaxb Marshaller 机制：

1. 首先使用 jaxb annotation 标记要绑定的 bean

Products.java

```
@XmlRootElement(name="products")
public class Products {

    private List<ProductSummary> products;

    @XmlElement(name="product")
    public List<ProductSummary> getProductList() {
        if (products == null) {
            products = new ArrayList<ProductSummary>();
        }

        return products;
    }
}
```

2. 然后，在 Spring 配置文件中定义 Jaxb2Marshaller

ApplicationContext.xml

```
<bean id="jaxb2Marshaller"
class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
    <property name="classesToBeBound">
        <list>
            <value>org.pprun.hjpetstore.persistence.jaxb.Products</value>
            ...
        </list>
    </property>
    <!-- we can depend on the xsd file for automatically validation -->
    <!-- <property name="schema"
value="classpath:org/springframework/oxm/schema.xsd"/>-->
</bean>
```

3. 将在基于 Spring MVC 3 RESTful 中介绍在 controller 中使用 jaxb，见在 view 的定义中引用 jaxb2Marshaller 在 view 的定义中引用 jaxb2Marshaller

### ...12.2.2 **Batch xml** 文件存储

不再需要调用 dom api 去构造 xml 文档，这一切都归功于 JAXB

```
OrderIncomeExport.java
```

```
this.jaxb2Marshaller.marshall(biViewOrderList, new StreamResult(os));
```

### ...12.3 基于 **Spring MVC 3 RESTful Controller**

一切只因为简单，[REST](#) 根本没有规范，因为它太简单。

这些年从事于 SOA / ROA 架构系统，跟随 Web Service 一路走来，记忆中的 Apache SOAP, Apache Axis, XFire, 标准化的 JAX-RPC, JAX-WS, 到后来半路杀出的 Spring-WS。一切刚刚开始，一切又一去不复返，短命是纯技术的特征。

RPC 的平台局限性，SOAP 的类型 bind 的易脆性，JAX-WS 工具的信赖度，Spring-WS contract first 使一个小时可以完成的工作（甚至在 IDE 中十分钟可以完成），要花上一整天，还要到看到 JUnit 的 **green bar** 才算数。

已经标准化的 JAX-RS，证明 RESTful 已经成为 SOA / ROA 加架的主流，以至于 Spring-WS 停止在了他的 1.59 版本，SpringSource 放弃了这个市场么？

(Spring reference)

#### 2.5.6.1 Comprehensive REST support

*Server-side support for building RESTful applications has been provided as an extension of the existing annotation driven MVC web framework. Client-side support is provided by the RestTemplate class in the spirit of other template classes such as JdbcTemplate and JmsTemplate. Both server and client side REST functionality make use of HttpConverters to facilitate the conversion between objects and their representation in HTTP requests and responses.*

#### ...12.3.1 基于 **Annotation** 声明式配置

首先需要枚举出 AnnotationMethodHandlerAdapter 使用的 messageConverters

```
Hjpetstore-servlet.xml
```

```
<context:component-scan base-package="org.pprun.hjpetstore.web.rest"/>
```

```
<bean  
class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHa  
ndlerMapping"/>
```

```
<bean  
class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHa  
ndlerAdapter">
```

```
<!-- 19.9.2 HTTP Message Conversion  
several main media type converters have been registered,
```

but if we overwrite this property, we have to list all our need-->

```
<property name="messageConverters">
  <list>
    <bean
class="org.springframework.http.converter.xml.MarshallingHttpMessageConve
rter">
      <property name="supportedMediaTypes">
        <list>
          <value>application/xml</value>
          <value>text/xml</value>
          <!-- curl set this type automatically -->
          <value>application/x-www-form-urlencoded</value>
        </list>
      </property>
      <property name="marshaller" ref="jaxb2Marshaller" />
      <property name="unmarshaller" ref="jaxb2Marshaller" />
    </bean>
    <bean
class="org.springframework.http.converter.StringHttpMessageConverter"/>
  </list>
</property>
</bean>
```

### ...12.3.2 在 **view** 的定义中引用 **jaxb2Marshaller**

Hjpetstore-servlet.xml

```
<!-- searchProducts rest GET -->
<bean name="products"
class="org.springframework.web.servlet.view.xml.MarshallingView">
  <constructor-arg ref="jaxb2Marshaller" />
</bean>
```

### ...12.3.3 定义视图解析器

ContentNegotiatingViewResolver 对多视图的支持

Hjpetstore-servlet.xml

```
<!-- view resolver -->
<bean
class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver"
">
  <property name="mediaTypes">
    <map>
```

```

        <entry key="xml" value="application/xml"/>
        <entry key="html" value="text/html"/>
    </map>
</property>
<property name="viewResolvers">
    <list>
        <bean
class="org.springframework.web.servlet.view.BeanNameViewResolver"
p:order="1"/>

    </list>
</property>
</bean>

```

以上是基本元素，最终这些都是在 controller 中使用的,如下。

#### ...12.3.4 REST HTTP METHOD

Resource sample	GET	POST	PUT	DELETE
<a href="http://localhost:8080/hjpetstore/rest/products/">http://localhost:8080/hjpetstore/rest/products/</a>	List the collection of resource or Retrieve the specified resource			Delete the resource
<a href="http://localhost:8080/hjpetstore/rest/encrypt">http://localhost:8080/hjpetstore/rest/encrypt</a>		Update or replace the resource		
<a href="http://localhost:8080/hjpetstore/rest/supplier/response">http://localhost:8080/hjpetstore/rest/supplier/response</a>			Create resource	
<a href="http://localhost:8080/hjpetstore/rest/order/1">http://localhost:8080/hjpetstore/rest/order/1</a>				Delete the order specified by id

##### ...12.3.4.1 GET

HjpetstoreController.java

@Controller

```
public class HjpetstoreController extends BaseController {

    @RequestMapping(value = "/products/{keyword}", method =
RequestMethod.GET)
    public ModelAndView getProductsByKeyword(
        @RequestParam("apikey") String apiKey,
        @RequestParam("page") int page,
        @RequestParam("max") int max,
        @PathVariable("keyword") String keyword) {

        if (log.isDebugEnabled()) {
            log.debug("HjpetstoreController is processing request for keyword: " +
keyword);
        }
        Products products = hjpetstoreService.searchProductList(apiKey,
keyword, page, max);

        ModelAndView mav = new ModelAndView("products");
        mav.addObject(products);
        return mav;
    }
}
```

#### CURL 测试:

```
pprun@pprun-laptop:~$ curl -u pprun:pprunpprun -H 'Content-Type:
application/xml' -H 'Accept: application/xml'
'http://localhost:8080/hjpetstore/rest/products/dog?
apikey=bc7163dab8eb79a9867b4604b46b0328e9ace555ef5d9526e1fcd748f
9864bf85d59e97c044a2d9795736753c2b0d77cd085eb05d854e5849f42f37f8
5851220&page=1&max=100'
```

#### Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?
><products><product><image>dog2.jpg</image><productDesc>Friendly
dog from
England</productDesc><productName>Bulldog</productName></product>
<product><image>dog6.jpg</image><productDesc>Cute dog from
France</productDesc><productName>Poodle</productName></product>...
</products>
```

### ...12.3.4.2 POST

SecurityController.java

```
@RequestMapping(value = "/encrypt", method = RequestMethod.POST)
public ModelAndView encryptCardNumber(@RequestBody
DecryptCardNumber cardNumber) {
    log.info("encrypt card number: " + cardNumber.getCardNumber());

    String encryptCardNumber =
securityService.encryptCardNumber(cardNumber.getCardNumber());

    EncryptCardNumber ecn = new EncryptCardNumber();
    ecn.setCardNumber(encryptCardNumber);

    ModelAndView mav = new ModelAndView("encryptCardNumber");
    mav.addObject(ecn);

    return mav;
}
```

#### CURL 测试:

```
pprun@pprun-laptop:~$ curl -u pprun:pprunpprun -H 'Content-Type:
application/xml' -H 'Accept: application/xml' -H 'Accept-Charset: UTF-8' -d '<?
xml version="1.0" encoding="UTF-8" standalone="yes"?
><DecryptCardNumber><cardNumber>0987654321</cardNumber></Decry
ptCardNumber>' 'http://localhost:8080/hjpetstore/rest/encrypt'
```

#### Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?
><EncryptCardNumber><cardNumber>MMCoArvbu44suYeCx9M/1KGHIE0tIm
meE1eXiXrASm3DpkAkiwSxMnU6ExUhWZd8syQuvLttY11r8l68xFudlsd1X0xF/P
+0SI29LvB1d03eGylufODqgvkhuJ9itmybbUm1R9wqiwwTVyKfj03o/132UPRTDA
MTKSec/a4Q3GI=</cardNumber></EncryptCardNumber>
```

### ...12.3.4.3 PUT

#### SecurityController.java

```
@RequestMapping(value = "/supplier/response", method =
RequestMethod.PUT)
public ModelAndView responseOrderStatus(
    @RequestBody SupplierOrderStatusRequest
supplierOrderStatusRequest) {

    long orderId = supplierOrderStatusRequest.getOrderId();
    String status = supplierOrderStatusRequest.getStatus();
    String supplier = supplierOrderStatusRequest.getSupplier();
}
```

```
        log.info("SupplierResponseController is processing request: orderId = " +
orderId
        + ", status= " + status
        + ", supplier= " + supplier);

        SupplierOrderStatusResponse.Result result =
SupplierOrderStatusResponse.Result.SUCCESS;

        try {
            supplierResponseQueueSender.responseOrderStatus(supplierOrderStat
usRequest);
        } catch (Throwable t) {
            log.error("SupplierResponseController - failed to send
responseOrderStatus");
            result = SupplierOrderStatusResponse.Result.FAILED;
        }

        SupplierOrderStatusResponse response = new
SupplierOrderStatusResponse();

        response.setResult(result);

        ModelAndView mav = new ModelAndView("supplierResponse");
        mav.addObject(response);
        return mav;
    }
```

#### CURL 测试:

```
pprun@pprun-laptop:~$ curl -u pprun:pprunpprun -H 'Content-Type:
application/xml' -H 'Accept: application/xml' -H 'Accept-Charset: UTF-8' -d '<?
xml version="1.0" encoding="UTF-8" standalone="yes"?
><SupplierOrderStatusRequest><orderId>1</orderId><status>IN_TRANSIT<
/status><supplier>Beijing Little Pet
Supplier</supplier></SupplierOrderStatusRequest>' -X PUT
'http://localhost:8080/hjpetstore/rest/supplier/response'
```

#### Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?
><SupplierOrderStatusResponse><result>SUCCESS</result></SupplierOrder
StatusResponse>
```

### ...12.3.4.4 DELETE

Currently, in hjpetstore 2.0, we don't have DELETE method.



## 13 集成

- 内部组件集成
- 外部集成

### ...13.1 内部组件集成

企业内部为什么还要集成，为什么不直接丢一个包(jar 文件)给信赖的组件？

- 松耦合使组件可以并行进行演化
- 不存在二机抽兼容问题
- 只需要保证接口兼容
- 出于安全因素考虑

本文不打算展开讨论，只关心 hjpetstore 中内部组件集成：出于安全考虑。

#### ...13.1.1 Security REST Service

按照 [PCI](#) 的规定，用户敏感数据以及卡片数据应该与通常的数据分开，采用更加严格的访问控制，提供给前端或外部的是经过加密处理后的数据。

Hjpetstore 只是一个 sample, 对卡片及用户数据没有与普通数据分隔开，但是，SecurityService 设计成一个 REST 组件，可以单独部署，并且利用 Spring security 访问控制，只允许 ROLE\_OPERATOR 的角色的用户访问。

##### ...13.1.1.1 Service API:

###### SecurityService.java

```
public interface SecurityService {

    PaymentPartner getPaymentPartner(String name);

    RSAKey getEnabledRSAKey();

    /**
     * User system wide public key encrypt the card number.
     *
     * @param plainCardNumber the plain text of card number
     * @return the encrypt card number
     */
    String encryptCardNumber(String plainCardNumber) throws
    SecurityServiceException;

    /**
     * User system wide public key decrypt the card number.
```

```
*
* @param encryptCardNumber the encrypt card number
* @return the decrypt card number
*/
String decryptCardNumber(String encryptCardNumber) throws
SecurityServiceException;

/**
 * Encrypt the cardnumber for the specify the payment partner by using its
 * publicKey assigned to us.
 *
 * @param partnerName the payment partner name
 * @param plainCardNumber the card number to encrypted
 * @return the encrypt card number
 * @throws ServiceException
 */
String encryptCardNumberForPartner(String partnerName, String
plainCardNumber) throws SecurityServiceException;
}
```

### ...13.1.1.2 REST Controller

Service API 的实现被包装为 RESTful 与外部通信

SecurityController.java

#### @Controller

```
public class SecurityController extends BaseController {
```

```
/**
 * Restful POST cardNumber to encrypt.
 * RequestBody will use the registered messageConverter -
 * MarshallingHttpMessageConverter, which is set up in
 * AnnotationMethodHandlerAdapter.
 * <p>
 * For example: user pprun <br />
 * {@code
 * curl -u pprun:pprunpprun -H 'Content-Type: application/xml' -H 'Accept:
 * application/xml' -H 'Accept-Charset: UTF-8' -d '<?xml version="1.0"
 * encoding="UTF-8" standalone="yes"?
 * ><DecryptCardNumber><cardNumber>0987654321</cardNumber></Decry
 * ptCardNumber>' 'http://localhost:8080/hjpetstore/rest/encrypt'
 * }
 *
 * @param cardNumber the plain/decrypt card number
 */
@RequestMapping(value = "/encrypt", method = RequestMethod.POST)
public ModelAndView encryptCardNumber(@RequestBody
```

```
DecryptCardNumber cardNumber) {
    log.info("encrypt card number: " + cardNumber.getCardNumber());

    String encryptCardNumber =
securityService.encryptCardNumber(cardNumber.getCardNumber());

    EncryptCardNumber ecn = new EncryptCardNumber();
    ecn.setCardNumber(encryptCardNumber);

    ModelAndView mav = new ModelAndView("encryptCardNumber");
    mav.addObject(ecn);

    return mav;
}
...
}
```

### ...13.1.1.3 Spring REST client: RestTemplate

如大家所期望的，Spring 提供 RestTemplate 封装 REST client，如同熟悉的 JmsTemplate, JDBCTemplate, HibernateTemplate 一样。

#### ...13.1.1.3.1 RestTemplate 配置

RestTemplate 封装了一个 ClientHttpRequestFactory，Spring3 提供两种实现：

- SimpleClientHttpRequestFactory 采用 JDK URLConnection
- CommonsClientHttpRequestFactory 采用 commons/httpclient 3.x (注意不是 4)

Hjpetstore 采用 CommonsClientHttpRequestFactory，通过它定义了 MultiThreadedHttpConnectionManager 用于并发访问控制，以下参数应该根据 [SLA](#) (service level agreement) 来确立：

- soTimeout
- connectionTimeout
- defaultMaxConnectionsPerHost
- maxTotalConnections
- staleCheckingEnabled

applicationContext.xml

```
<bean id="securityServiceRestClient"
class="org.pprun.hjpetstore.web.rest.client.SecurityServiceRestClientImpl">
    <property name="restTemplate" ref="restTemplate"/>
    <property name="username" value="${rest.securityService.username}"
```

```
/>
    <property name="password" value="$
{rest.securityService.password}" />

    <property name="encryptUrl" value="$
{rest.securityService.encrypt.url}" />
</bean>

<!-- the template can be used to call home-grown or external RESTful
service -->
    <bean id="restTemplate"
class="org.springframework.web.client.RestTemplate">
    <property name="requestFactory">
    <bean
class="org.springframework.http.client.CommonsClientHttpRequestFactory">
    <constructor-arg>
    <bean class="org.apache.commons.httpclient.HttpClient">
    <constructor-arg index="0">
    <bean id="httpClientParams"
class="org.apache.commons.httpclient.params.HttpClientParams">
    <property name="authenticationPreemptive"
value="true"/>
    </bean>
    </constructor-arg>
    <constructor-arg index="1">
    <bean
class="org.apache.commons.httpclient.MultiThreadedHttpConnectionMan
ager">
    <property name="params">
    <bean
class="org.apache.commons.httpclient.params.HttpConnectionManagerParam
s">
    <property name="soTimeout" value="$
{rest.service.timeout}" />
    <property name="connectionTimeout" value="$
{rest.service.connectionTimeout}" />
    <property
name="defaultMaxConnectionsPerHost" value="$
{rest.service.maxConnections}" />
    <property name="maxTotalConnections"
value="$ {rest.service.maxTotalConnections}" />
    <property name="staleCheckingEnabled"
value="$ {rest.service.staleCheckEnabled}" />
    </bean>
    </property>
    </bean>
    </constructor-arg>
```

```
        </bean>
    </constructor-arg>
</bean>
</property>
</bean>
```

### ...13.1.1.3.2 RestTemplate's REST call

#### a HTTP HEADER

**createHttpHeader** 方法设置 http header:

- Accept
- Accept-Charset

#### SecurityServiceRestClientImpl.java

```
/**
 * help method create HTTP headers.
 * @return
 */
private HttpHeaders createHttpHeader() {
    HttpHeaders requestHeaders = new HttpHeaders();
    // Accept
    requestHeaders.set("Accept", "application/xml");
    requestHeaders.set("Accept-Charset", CommonUtil.UTF8);

    // Date
    String date =
    CalendarUtil.getDateStringWithZone(Calendar.getInstance(),
    CalendarUtil.ZONE_DATE_FORMAT,
    TimeZone.getTimeZone(CommonUtil.TIME_ZONE_UTC), Locale.US);
    requestHeaders.set("Date", date);

    return requestHeaders;
}
```

#### b HTTP 请求

通过调用 RestTemplate.**postForObject** 发起一个 http POST 请求

#### SecurityServiceRestClientImpl.java

```
public class SecurityServiceRestClientImpl implements
SecurityServiceRestClient {
```

```
@Override
public EncryptCardNumber encryptCardNumber(String cardNumber) {
    HttpHeaders requestHeaders = createHttpHeader();
    DecryptCardNumber decryptCardNumber = new DecryptCardNumber();
    decryptCardNumber.setCardNumber(cardNumber);
    HttpEntity<String> entity = new HttpEntity(decryptCardNumber,
requestHeaders);

    HttpEntity<EncryptCardNumber> response =
restTemplate.exchange(encryptUrl,
    HttpMethod.POST,
    entity,
    EncryptCardNumber.class);

    EncryptCardNumber encryptCardNumber = response.getBody();
    return encryptCardNumber;
}

private String buildURL(HttpMethod httpVerb, HttpHeaders requestHeaders,
String path, Object... urlVariables) throws ServiceException {
    // construct the url
    String originPath = path; // make a copy because the process still need
parametered path outside this method,

    if (log.isDebugEnabled()) {
        log.debug("originPath path = " + path);
    }
    if (urlVariables != null) {
        // if the path contains variable, expand it
        // http://localhost:8080/hjpetstore/rest/paymentpartner/{name}

        UriTemplate uriTemplate = new UriTemplate(path);
        URI expanded = uriTemplate.expand(urlVariables);
        path = expanded.getRawPath();
        if (log.isDebugEnabled()) {
            log.debug("expanded path = " + path);
        }
    }

    String httpMethod = httpVerb.toString();
    String date = requestHeaders.getFirst("Date");
    String secretKey = userService.getUserSecretKeyForApiKey(apiKey);
    String signature = MessageDigestUtil.calculateSignature(httpMethod,
date, path, secretKey);
    try {
```

```
        StringBuilder s = new StringBuilder();
        s.append(originPath);
        s.append("?apiKey=");
        s.append(URLEncoder.encode(apiKey, CommonUtil.UTF8));
        s.append("&signature=");
        s.append(URLEncoder.encode(signature, CommonUtil.UTF8));
        if (log.isDebugEnabled()) {
            log.debug(s.toString());
        }

        return s.toString();
    } catch (UnsupportedEncodingException ex) {
        throw new ServiceException("UnsupportedEncodingException when
URLEncoder.encode: " + CommonUtil.UTF8, ex);
    }
}
...
}
```

#### ...13.1.1.4 Spring Security 访问控制

(This section is deprecated by OAuthFilter)

- 拦截所有的 '/rest/' url pattern
- HTTP Basic Authentication
- 基于 md5 的 basic 认证

#### ApplicationContext-security.xml

```
<http auto-config="true">
    <intercept-url pattern="/rest/**" access="ROLE_OPERATOR" requires-
channel="any"/>

</http>

<!--
    We have username as the salt for password's MD5, the
Usernames/Passwords are:
    pprun/pprunpprun
-->
<authentication-manager>
    <authentication-provider>
        <password-encoder hash="md5"/>
        <user-service>
            <user name="pprun"
password="ca27a6128e8c68d15d5ef9e2b7cfa903"
```

```
authorities="ROLE_ADMIN,ROLE_USER,ROLE_OPERATOR" />
    </user-service>
    </authentication-provider>
</authentication-manager>
```

#### Web.xml

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

### ...13.2 外部集成

不与外部集成的系统是信息的孤岛，SOA / ROA 的最终目的是快速集成，而 REST 又成为这一领域的后起之秀。

#### ...13.2.1 *Payment Partner* 集成

Hjpetstore 包含一个 简单实现的 payment REST service  
PaymentController.java。

与 payment partner 的集成步骤和 内部的 SecurityService 基本一样。不同的是，采用了 RSA 非对称加密 and 消息摘要安全策略：

- 利用 payment partner 的公钥加密卡号
- 采用 sha-512 HASH 算法加密卡号，以避免中途串改
- 然后将以上两个加密后的值连同其他参数一起作为请求 body 发起 REST 调用。

#### PaymentRestClientImpl.java

```
// create Charge request
PaymentChargeRequest paymentChargeRequest = new
PaymentChargeRequest();
    paymentChargeRequest.setCardNumber(encryptCardNumber.getCardN
umber());
    paymentChargeRequest.setMerchantName(merchantName);
    paymentChargeRequest.setHash(cardNumberHash);
    paymentChargeRequest.setAmount(amount);

    HttpHeaders requestHeaders = createHttpHeader();
    HttpEntity<String> entity = new HttpEntity(paymentChargeRequest,
requestHeaders);
```



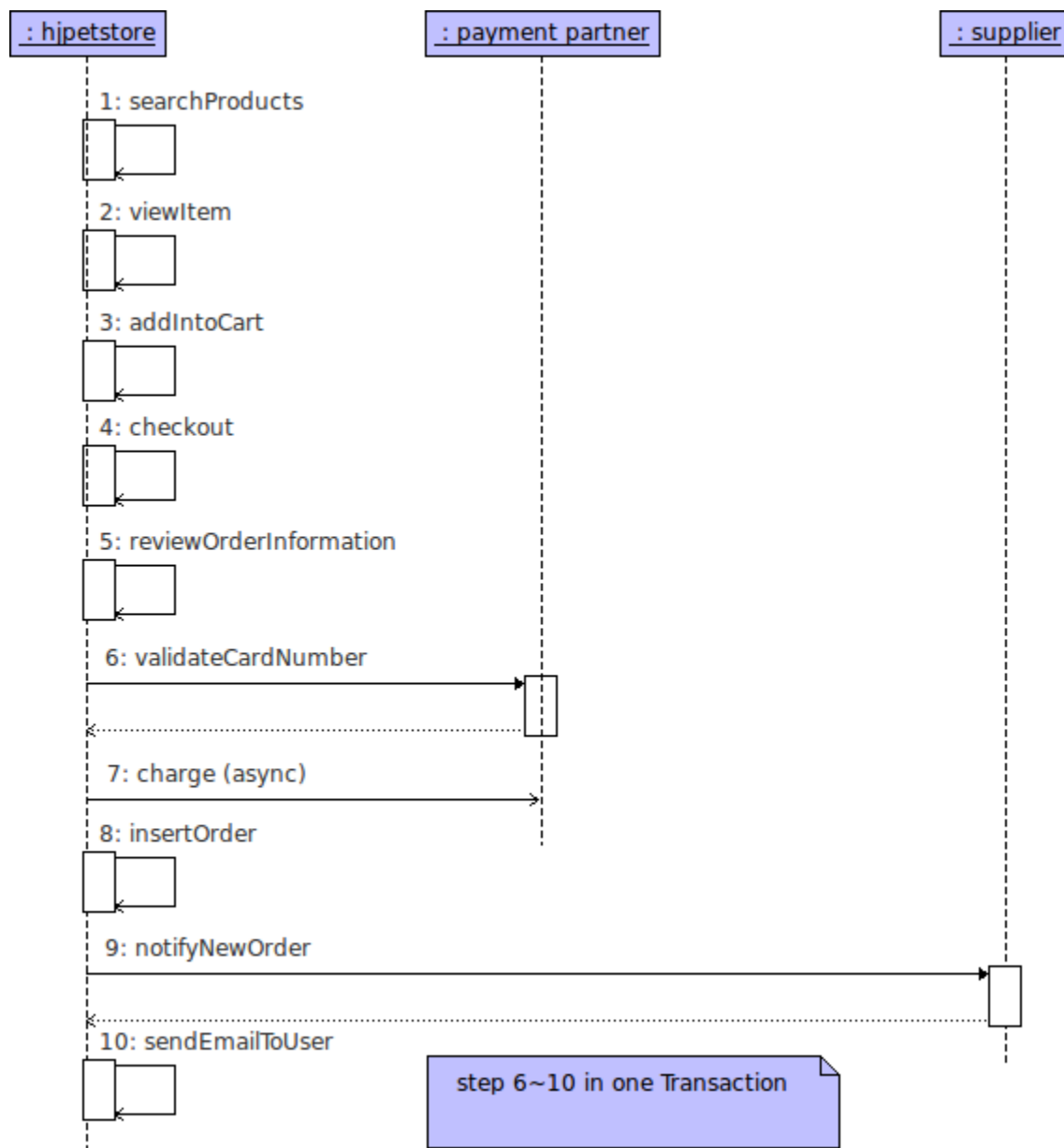
```
PaymentChargeResponse paymentChargeResponse =  
restTemplate.postForObject(chargeUrl, entity,  
PaymentChargeResponse.class);
```

### ...13.2.2 *JMS* 异步松散耦合: *Supplier* 集成

JMS 异步模型被 [Enterprise Integration Patterns](#) 喻为 远程通信 的 Best Strategy, 攻克 远程通信固有的 延时, 不可靠等缺陷。

#### ...13.2.2.1 同步模型的缺点

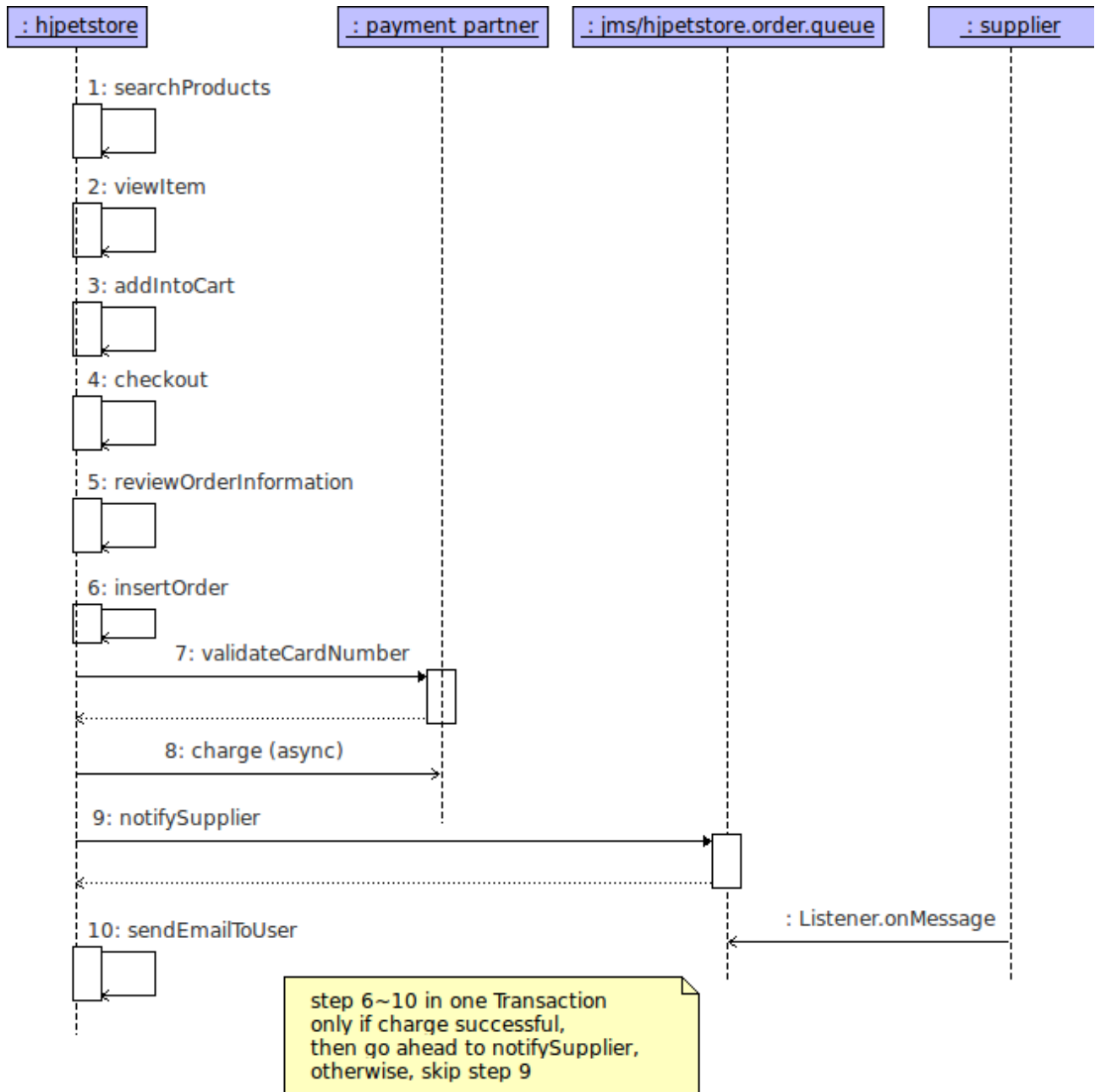
皮皮鲁 浏览到 hjpetstore, 看到一条漂亮的可爱的小狗, 决定买下来。



如果采用的是同步模型，如果图所示第 9 步 `notifyNewOrder`, 只有等到 `supplier` 的 `response` 后才进行完成整个交易，如果 `Supplier` 的系统暂时不可达，维护或升级等原因。皮皮鲁的交易以失败告终，hjpetstore 损失的不仅仅一次订单，我打赌皮皮鲁再也不会来 hjpetstore 了。

### ...13.2.2.2 异步模型

下面看看采用 JMS 异步模式的情况：



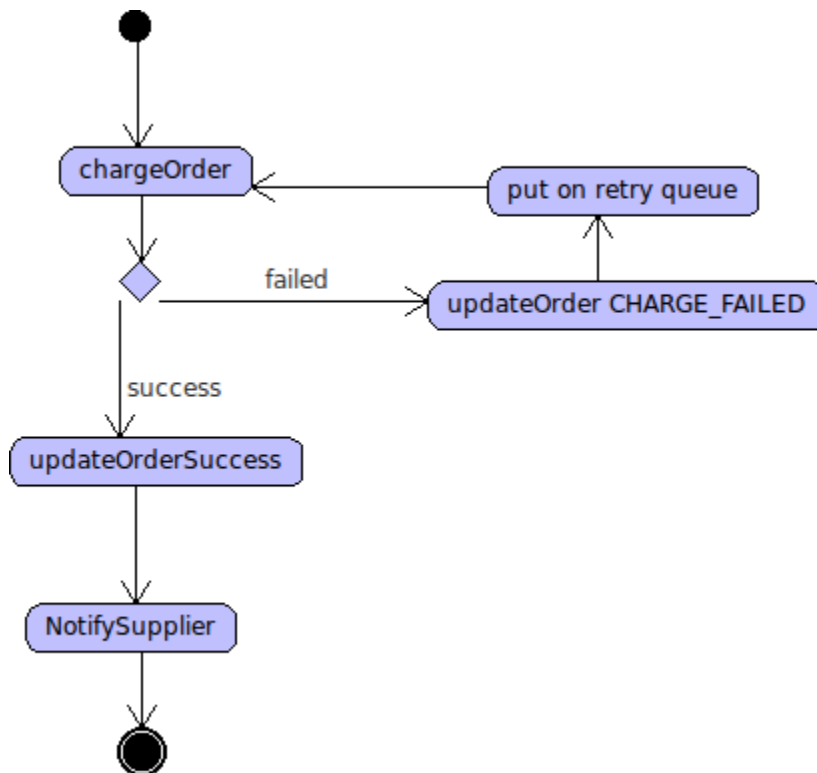
在第九步把消息发到 JMS 队列上 'jms/hjpetstore.order.queue'，Supplier 监听队列上的消息。消息的完整性与事务全由 JMS 系统处理，不信赖通信的双方的状态。

同时请注意 charge，这里是使用的 异步模式，但是为什么 validateCardNumber 是同步的呢？这里有个原则，并不是所有与外部的集成都采用异步模型才算合理。

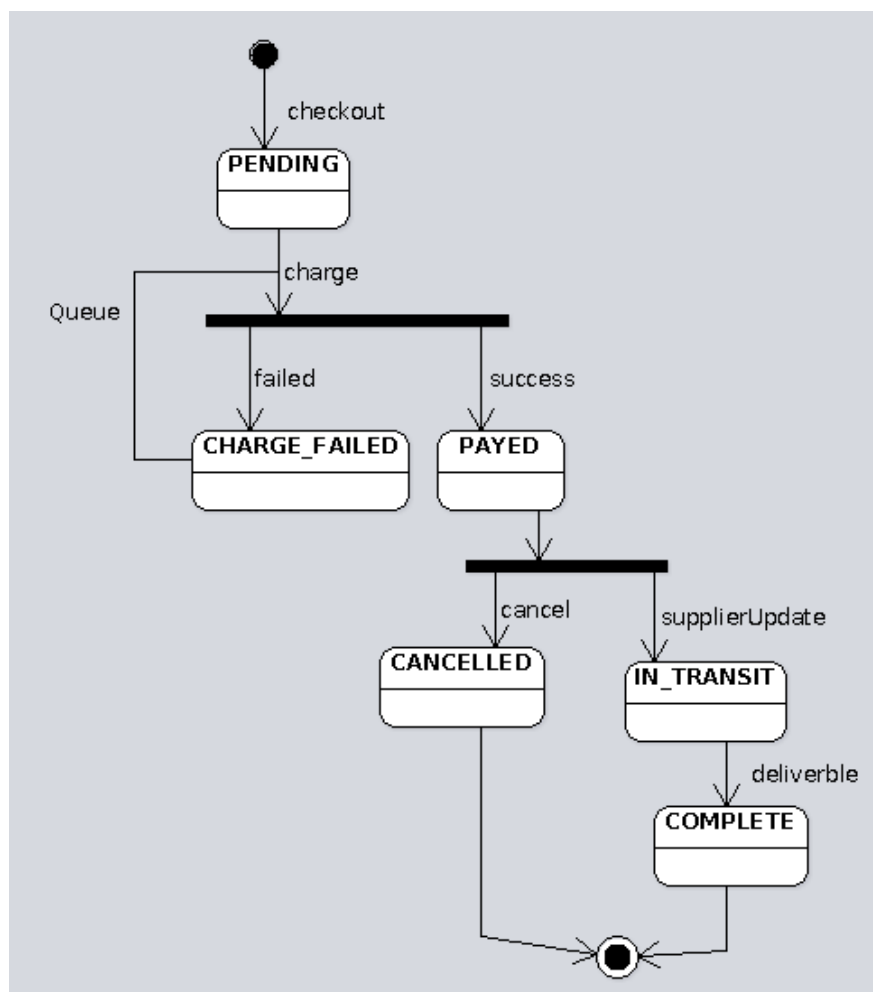
对于卡号无效，可不能掉以轻心，可以直接终止整个交易，所以再长时间也得等，这没办法。

但对于扣款 (charge) 为什么可以呢，因为 hjpetstore 采用 retry，如果失败，将 charge 放入 'jms/hjpetstore.chargeOrder.retry.queue' 直到成功为止，一旦成功，紧接着 notifySupplier:

### ...13.2.2.2.1 Charge 流程



### ...13.2.2.2.2 Order 的状态图



### ...13.2.2.3 基于 Spring JMS 的实现

采用了 Spring JMS, 所有的 producer 和 listener (consumer)类可以是 Message-Driven POJO (MDP) 不必信赖任何框架, hjpetstore 保留了接口 `MessageListener`, 是为了将来演示 MDB (message drive bean)之用。

#### ...13.2.2.3.1 服务器配置

首先, 需要在 GlassFish 中设置需要的 JNDI 资源, 参见 [Create GlassFish JNDI resource for hjpetstore 2.0](#)

#### ...13.2.2.3.2 jmsConnectionFactory

Jms.xml
<pre>&lt;jee:jndi-lookup id="jmsConnectionFactory" jndi-name="jms/hjpetstore.connectionFactory"/&gt;</pre>
Jms 连接工厂, 配置了通用的事务等相关属性

### ...13.2.2.3.3 Queue

#### **hjpetstoreOrderQueue**

Jms.xml
<pre>&lt;jee:jndi-lookup id="hjpetstoreOrderQueue" jndi-name="jms/hjpetstore.order.queue"/&gt;</pre>
Jms 队列，用于通知 supplier 新的定单

#### **chargeOrderRetryQueue**

Jms.xml
<pre>&lt;jee:jndi-lookup id="chargeOrderRetryQueue" jndi-name="jms/hjpetstore.chargeOrder.retry.queue"/&gt;</pre>
Jms 队列，用于 charge 重试

#### **supplierResponseQueue**

Jms.xml
<pre>&lt;jee:jndi-lookup id="supplierResponseQueue" jndi-name="jms/hjpetstore.supplier.response.queue"/&gt;</pre>
Jms 队列，用于 supplier 更新 order 的状态

### ...13.2.2.3.4 Topic

为什么这个需要使用 publish/subscribe 而不是 点对点 (point-to-point)模式呢？因为系统引入竞争，不依赖于任何一个商品提供商，响应快的提供商将有更多机会。

#### **hjpetstoreBackOrderedTopic**

Jms.xml
<pre>&lt;jee:jndi-lookup id="hjpetstoreBackOrderedTopic" jndi-name="jms/hjpetstore.backOrdered.topic"/&gt;</pre>
Jms Topic，用于向所有的 supplier 通知，hjpetstore 有商品缺货了

### ...13.2.2.3.5 Spring JmsTemplate

Jms.xml
<pre>&lt;!-- the producer side --&gt; &lt;bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"</pre>

```
        p:connectionFactory-ref="jmsConnectionFactory"
        p:destinationResolver-ref="jmsDestResolver"
p:sessionTransacted="true"/>

    <!-- notify the supplier the order initiated -->
    <bean id="hjpetstoreOrderQueueSender"
class="org.pprun.hjpetstore.service.jms.HjpetstoreOrderQueueSender" >
        <property name="jmsTemplate" ref="jmsTemplate"/>
        <property name="hjpetstoreOrderQueue" ref="hjpetstoreOrderQueue"/>
    </bean>
...

```

### ...13.2.2.3.6 Message-Driven POJO

**a**            消息发送者（或生产者）

HjpetstoreOrderQueueSender.java

```
public class HjpetstoreOrderQueueSender {

    private static final Log log =
LogFactory.getLog(HjpetstoreOrderQueueSender.class);
    private JmsTemplate jmsTemplate;
    private Queue hjpetstoreOrderQueue;

    @Required
    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    @Required
    public void setHjpetstoreOrderQueue(Queue hjpetstoreOrderQueue) {
        this.hjpetstoreOrderQueue = hjpetstoreOrderQueue;
    }

    public void notifySupplier(final OrderLineItem orderLineItem) {
        this.jmsTemplate.send(this.hjpetstoreOrderQueue, new MessageCreator()
    {

        @Override
        public Message createMessage(Session session) throws JMSException
    {
        MapMessage message = session.createMapMessage();
    }
    }
    }
}

```

```
        long orderId = orderLineItem.getOrder().getId();
        String itemName = orderLineItem.getItem().getItemName();
        int quantity = orderLineItem.getQuantity();
        String supplierName =
orderLineItem.getItem().getSupplier().getSupplierName();

        message.setLong("order.id", orderId);
        message.setString("item.name", itemName);
        message.setInt("quantity", quantity);

        // the supplier should use message selector to filter the message
        message.setStringProperty("supplier", supplierName);

        log.info("HjpetstoreOrderQueueSender is sending jms message: "
            + "[order.id=" + orderId
            + ", item.name=" + itemName
            + ", quantity=" + quantity
            + ", supplierName=" + supplierName
            + "]");

        return message;
    }
});
}
}
```

**b**      消息消费者(或接收者)

由于涉及到多线程并发问题, 这里使用 [Pooling target sources](#) 来应对多线程问题

Jms.xml

```
<!-- listen to supplier order response status -->
<bean id="jmsSupplierResponseContainer"
class="org.springframework.jms.listener.DefaultMessageListenerContainer">
    <property name="connectionFactory" ref="jmsConnectionFactory"/>
    <property name="destination" ref="supplierResponseQueue"/>
    <property name="messageListener"
ref="supplierResponseMessageListener"/>
    <property name="transactionManager" ref="transactionManager"/>
    <property name="sessionTransacted" value="true"/>
    <property name="concurrency" value="1-10"/>
</bean>

<bean id="supplierResponseMessageListenerTarget"
    class="org.pprun.hjpetstore.service.jms.SupplierResponseMessageListen
```



```
er"
    scope="prototype">
        <description>Create a prototype supplierResponseMessageListener with
its own copy of orderService</description>
        <property name="orderService" ref="orderService"/>
    </bean>

    <bean id="supplierResponseMessageListenerPoolTargetSource"
class="org.springframework.aop.target.CommonsPoolTargetSource">
        <property name="targetBeanName"
value="supplierResponseMessageListenerTarget"/>

        <property name="maxSize" value="25"/>
    </bean>

    <bean id="supplierResponseMessageListener"
class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="targetSource"
ref="supplierResponseMessageListenerPoolTargetSource"/>
    </bean>
```

### SupplierResponseMessageListener

```
public class SupplierResponseMessageListener implements MessageListener
{

    private static final Log log =
LogFactory.getLog(SupplierResponseMessageListener.class);
    private OrderService orderService;

    @Autowired
    @Required
    public void setOrderService(OrderService orderService) {
        this.orderService = orderService;
    }

    public void onMessage(Message message) {
        try {
            if (message instanceof MapMessage) {
                MapMessage msg = (MapMessage) message;

                Long orderId = msg.getLong("order.id");
                String status = msg.getString("deliver.status");

                log.info("----- SupplierResponseMessageListener -----");
                log.info("orderId: " + orderId);
            }
        }
    }
}
```

```
        log.info("status: " + status);
        log.info("-----");

        // update the order status
        if (Order.OrderStatus.IN_TRANSIT.toString().equals(status)) {
            if (log.isDebugEnabled()) {
                log.debug("update orderStatus to: " +
Order.OrderStatus.IN_TRANSIT);
            }
            orderService.updateOrderStatus(orderId,
Order.OrderStatus.IN_TRANSIT);
        } else if (Order.OrderStatus.COMPLETE.toString().equals(status)) {
            if (log.isDebugEnabled()) {
                log.debug("update orderStatus to: " +
Order.OrderStatus.COMPLETE);
            }

            orderService.updateOrderStatus(orderId,
Order.OrderStatus.COMPLETE);
        } else {
            // log warn and ignore it
            log.warn("supplier sent back a un-recognized staus of the order");
        }

    } else {
        throw new UnsupportedOperationException("Message Type has not
supported yet.");
    }
} catch (JMSEException ex) {
    log.error("cannot convert the message", ex);
    throw new MessageConversionException("cannot convert the
message", ex);
}
}
```

### **...13.2.3 JMS ObjectMessage**

#### **...13.2.3.1 JMS 规范定义的消息类型**

#### **Message Type Body Contains**

TextMessage      A java.lang.String object (for example, the contents of an XML

file).

MapMessage	A set of name-value pairs, with names as String objects and values as primitive types in the Java programming language. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined.
BytesMessage	A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.
StreamMessage	A stream of primitive values in the Java programming language, filled and read sequentially.
ObjectMessage	A <b>Serializable</b> object in the Java programming language.
Message	Nothing. Composed of header fields and properties only. This message type is useful when a message body is not required.

除了 ObjectMessage 消息外，其他消息类型都跨平台的，也就是说，不管消息中间件是何种语言实现的。ObjectMessage 由于使用了 java.io.Serializable 机制，所以限制了它只能用在 Java 的世界里。

### ...13.2.3.2 ChargeOrderRetryMessageListener

由于 order charge retry 机制不需要与外部系统集成，所以很自然可以使用 ObjectMessage，而不用考虑跨平台问题，对象正是采用 JAXB Marshaller:

#### ChargeOrderRetryMessageListener.java

```
@Override
public void onMessage(Message message) {
    try {
        if (message instanceof ObjectMessage) {
            ObjectMessage msg = (ObjectMessage) message;
            OrderRetryDto dto = (OrderRetryDto) msg.getObject();

            log.info("----- ChargeOrderRetryMessageListener -----");
            log.info("received objectMessage: " + dto);
            log.info("-----");

            log.info("try to call orderService.chargeOrder again");
            // try to call orderService.chargeOrder again
        }
    }
}
```

```
        orderService.chargeOrder(dto.getOrder(),
dto.getPlainCardNumber());

        } else {
            throw new UnsupportedOperationException("Message Type has not
supported yet.");
        }
    } catch (JMSEException ex) {
        log.error("cannot convert the message", ex);
        throw new MessageConversionException("cannot convert the
message", ex);
    }
}
```

## 14 BI (商业智能) 与批处理

- Scheduler(定时启动)
- Task (任务)

对于真实的企业应用，批处理(Batch)一般用来对数据进行智能分析，与支付系统结算，收导出。由于数据量大，计算密集，通常这种应用需要单独部署。并且为了减少数据库的压力，通常是晚上运行。

Hjpetstore 简单地模拟了这一需求，收入导出。真实的应用，可能会涉及到将导出的数据文件上传到服务器，比如 ERP，或将分析数据结果调用 CRM 接口。

### ...14.1 定时启动

Spring 3 已经自产了定时器或 scheduler 功能，不必信赖于 OpenSymphony Quartz 了。

#### bi.xml

```
<task:scheduled-tasks scheduler="orderIncomeExportScheduler">
    <task:scheduled ref="orderIncomeExport" method="exportOrderIncome"
cron="{order.income.export.schedule}"/>
</task:scheduled-tasks>

<task:scheduler id="orderIncomeExportScheduler" pool-size="$
{order.income.export.pool.size}"/>
```

#### bi.properties

```
# 1:00 am every day
order.income.export.schedule=0 0 1 * * ?
```

```
order.income.export.pool.size=4
```

## ...14.2 任务 POJO

Task 对象，同样是纯 POJO, OrderIncomeExport 采用了 JAXB2Marshaller，将定制的 Order DTO 对象输出为 xml

bi.xml

```
<bean id="orderIncomeExport"
class="org.pprun.hjpetstore.service.bi.OrderIncomeExport" >
  <property name="jaxb2Marshaller" ref="jaxb2Marshaller" />
  <property name="exportFolder" ref="exportFolder" />
  <property name="orderService" ref="orderService" />
</bean>
```

OrderIncomeExport.java

```
public class OrderIncomeExport {

    private Marshaller jaxb2Marshaller;

    @Required
    public void setJaxb2Marshaller(Marshaller jaxb2Marshaller) {
        this.jaxb2Marshaller = jaxb2Marshaller;
    }

    public void exportOrderIncome() throws Exception {
        // heavy task for work horse here
    }
}
```

## 15 异常处理

既然程序出现异常是不可避免，那异常处理应当纳入设计的组成部分。对于 B/S 结构的应用，异常处理通常涉及到以下层次结构：

- JSP/Servlet 表示层 error page
- Controller MappingExceptionResolver
- Service 层 定制的 RuntimeException
- DAO 层 DataAccessException

## ...15.1 JSP/Servlet 表示层 **error page**

Servlet / JSP 规范了 ErrorCode 及 Error Page 的处理。

### ...15.1.1 首先定义 **error code** 映射

Web.xml

```
<error-page>
  <error-code>400</error-code>
  <location>/WEB-INF/jsp/shop/ErrorCode.jsp</location>
</error-page>
<error-page>
  <error-code>404</error-code>
  <location>/WEB-INF/jsp/shop/ErrorCode.jsp</location>
</error-page>
```

...

### ...15.1.2 **Error page**

需要一个 Error code page 来显示出错信息，对于最终用户界面，不应该把异常栈直接显示给用户。

ErrorCode.jsp

```
<%@ page isErrorPage="true" contentType="text/html; charset=UTF-8" %>
```

...

### ...15.1.3 引用 **Error page**

最后在所有非 error page 的页面中，指定 `errorPage` 属性，可以将其置于 \*.jspx 页面中，然后所有的 jsp 使用 '`<%@ include`' 指令将其包含进来：

\*.jspx

```
<%@ page errorPage="/jsp/ErrorCode.jsp"
contentType="text/html; charset=UTF-8" %>
```

## ...15.2 Controller MappingExceptionResolver

在早先的基于 Spring 的应用中，最常见的是使用 MappingExceptionHandlerResolver，而 Spring MVC 3 引入了 AnnotationMethodHandlerExceptionHandlerResolver。

### ...15.2.1 **MappingExceptionHandlerResolver**

使用 MappingExceptionHandlerResolver 来做异常映射到相应的 view:

Hjpetstore-servlet.xml

```
<bean
```

```
class="org.springframework.web.servlet.handler.SimpleMappingExceptionRes  
olver">  
    <property name="exceptionMappings">  
        <props>  
            <prop  
key="org.springframework.dao.DataAccessException">dataAccessFailure</pr  
op>  
            </props>  
        </property>  
    </bean>
```

### ...15.2.2 *AnnotationMethodHandlerExceptionHandler*

在 Spring MVC 3 中，可以使用基于 annotation 的异常处理模式 AnnotationMethodHandlerExceptionHandler，但要注意的是，这种处理是以 controller 为中心的，普通的 jsp 的 error code 不在其的 scope 内，也就是说，基于这种方式的异常处理，只管 controller 抛出的异常，而处理不了通常的 Servlet 的异常。

BaseController 作所有 controller 的基类，包含了大部分常见异常的处理：

#### BaseController.java

```
public abstract class BaseController {  
  
    private static final Log log = LogFactory.getLog(BaseController.class);  
    private static final String COMMON_ERROR = "CommonError";  
    private static final String COMMON_ERROR_ATTRIBUTE = "commonError";  
  
    @ExceptionHandler({NoSuchRequestHandlingMethodException.class})  
    public ModelAndView handleNotFound(Exception ex, HttpServletRequest  
request) {  
        String path = request.getPathInfo();  
  
        log.warn("The page not found: " + path, ex);  
  
        ModelAndView mav = composeModelAndView(path, ex);  
        return mav;  
    }  
  
    ...  
  
    @ExceptionHandler({ServiceException.class})  
    public ModelAndView handleServiceException(Exception ex,  
HttpServletRequest request) {  
        String path = request.getPathInfo();  
  
        log.warn("Can't accomplish the request because of service layer error
```

```
when access to: " + path, ex);
    ModelAndView mav = composeModelAndView(path, ex);
    return mav;
}

@ExceptionHandler({SecurityServiceException.class})
public ModelAndView handleSecurityServiceException(Exception ex,
HttpServletRequest request) {
    String path = request.getPathInfo();

    log.warn("Can't accomplish the request because of SecurityService
Exception when access to: " + path, ex);
    ModelAndView mav = composeModelAndView(path, ex);
    return mav;
}

@ExceptionHandler({DataAccessException.class})
public ModelAndView handleDataAccessException(Exception ex,
HttpServletRequest request) {
    String path = request.getPathInfo();

    log.warn("Can't accomplish the request because of data layer error when
access to: " + path, ex);
    ModelAndView mav = composeModelAndView(path, ex);
    return mav;
}

/**
 * This is the fallback for all other UncaughtException.
 * @param ex
 * @param request
 * @return
 */
@ExceptionHandler({Exception.class})
public ModelAndView handleUncaughtException(Exception ex,
HttpServletRequest request) {
    String path = request.getPathInfo();

    log.warn("Can't accomplish the request because an unexpected error
when access to: " + path, ex);
    ModelAndView mav = composeModelAndView(path, ex);
    return mav;
}

private ModelAndView composeModelAndView(String path, Exception ex) {
    ModelAndView mav = new ModelAndView(COMMON_ERROR);
}
```



```
CommonError commonError = new CommonError();
commonError.setPath(path);
// only output exception information when debug
if (log.isDebugEnabled()) {
    commonError.setException(ex.toString());
}

mav.addObject(COMMON_ERROR_ATTRIBUTE, commonError);
return mav;
}
}
```

### ...15.3 Service 层 RuntimeException

对于以往的基于 Spring 的应用，通常按照业务定制相应的 RuntimeException，在 service 层抛出，然后在 controller 里捕获对异常进行处理。

有了 Spring MVC 3 AnnotationMethodHandlerExceptionResolver，在 BaseController 中集中处理所有的业务异常：ServiceException

### ...15.4 Dao 层 DataAccessException

通常在 Dao 层将所有异常都转接到 Spring 的 RuntimeException 体系中来 — DataAccessException

#### ...15.4.1 使用 @Repository annotation

在 Spring 3 中，可以使用@Repository 标记 DAO

#### \*.DAO (实现类)

```
/**
 *
 * {@literal DAO} annotated {@code @Repository} is eligible for Spring
 * DataAccessException translation.
 * @see Repository
 * @author <a href="mailto:quest.run@gmail.com">prrun</a>
 */
@Repository
public class HibernateOrderDao extends HibernateDaoSupport implements
OrderDao {
```

#### ...15.4.2 DataAccessException 异常转换

然后在配置文件中定义，这样做的好处是，不需要重复地 throw new DataAccessException 将异常进行包装了。

DataAccessContext-jta.xml

```
<!-- {@literal DAO} annotated {@code @Repository} is eligible for Spring  
DataAccessException translation. -->  
<bean  
class="org.springframework.dao.annotation.PersistenceExceptionTranslationPo  
stProcessor" />
```

## 16 测试

- 没有测试的代码是不完整的代码
- 没有测试的项目是未完成的项目。
- 不写测试的程序员是不合格的程序员（其中尤其是那种会写但偷懒不写是不可饶恕的。跑题了!）

因为再仔细的人也有疏忽的时候，越是简单的逻辑，可是却越有可能存在 BUG。因此，测试代码的 cover 率以及质量都就应该是 code review 的一部分。

### ...16.1 基本的单元测试

基本单元测试是对项目的基础类，无信赖或无太多外部信赖，因此不涉及到 Spring IoC 的 POJO，对于这些类的测试只需要 Junit 就行了。

以下测试方法是用来测试这样一个功能：

在指定的目录下生成一个新的文件，如果文件已经存在，则在文件名后加下划线并追加一个数据，从数字 1 开始，同时考虑有无扩展名的情况， 例如：

- 有扩展名

/tmp/a.txt

如果再次要生成一个 a.txt, 则文件名为 a\_1.txt, 再次请求生成时 a\_2.txt

- 无扩展名

/tmp/test

test\_1, test\_2

#### ...16.1.1 FileUtils 代码

FileUtils.java

```
public static File getNewFile(String path, String fileName, String extension) {  
  
    if (extension != null && extension.isEmpty() == false) {  
        extension = "." + extension;  
    } else {  
        extension = "";  
    }  
  
    File file = new File(path + File.separator + fileName + extension);  
    int i = 0;
```

```
while (file.exists()) {
    // existing? append a increasing number
    file = new File(path + File.separator + fileName + "_" + ++i +
extension);
}

return file;
}
```

### ...16.1.2 **FileUtilsTest** 代码

生成模板式的测试代码在大部分 IDE 中，只需要点几下鼠标即可，要注意的是，任何计算结果都要 Assert:

#### FileUtilsTest.java

```
@Test
public void testGetNewFile() throws IOException {
    System.out.println("createNewFile");
    String path = System.getProperty("java.io.tmpdir");
    log.info("file path: " + path);

    // no extension
    String fileName = "test";
    String extension = null;
    File result1 = FileUtils.getNewFile(path, fileName, extension);
    assertEquals("test", result1.getName());

    // write something in it to let it create
    result1.deleteOnExit(); // we can run this test again
    new FileOutputStream(result1).write("This is a
test".getBytes(Charset.forName("UTF-8")));

    // once again
    fileName = "test";
    File result2 = FileUtils.getNewFile(path, fileName, extension);
    assertEquals("test_1", result2.getName());
    // write something in it to let it create
    result2.deleteOnExit(); // we can run this test again
    new FileOutputStream(result2).write("This is a
test".getBytes(Charset.forName("UTF-8")));

    // test file with extension
    fileName = "test";
    extension = ".txt";
    File result3 = FileUtils.getNewFile(path, fileName, extension);
    assertEquals("test.txt", result3.getName());
}
```

```
// write something in it to let it create
result3.deleteOnExit(); // we can run this test again
new FileOutputStream(result3).write("This is a
test".getBytes(Charset.forName("UTF-8")));

// once again
fileName = "test";
extension = "txt";
File result4 = FileUtils.getNewFile(path, fileName, extension);
assertEquals("test_1.txt", result4.getName());
// write something in it to let it create
result4.deleteOnExit(); // we can run this test again
new FileOutputStream(result4).write("This is a
test".getBytes(Charset.forName("UTF-8")));
}
```

## ...16.2 集成测试

### ...16.2.1 *Spring TestContext*

如果因为不会写测试而没写，那么 Spring TestContext 使写测试变得简单。对于业务复杂的系统，没有 Spring 测试框架支持的话，写集成测试的确令人生畏，Spring 的作者们完全理解这一点，因此他们也在不断改进，在 Spring 2.5 重写了部分测试框架。

下面例子是以一个 Dao 测试为例展开的。

### ...16.2.2 抽象测试基类

很显然项目中会有多个 Dao，甚至会有基于多种不同框架的实现，因此需要一个抽象基类来定义通用的部分。

AbstractDaoTest 从便利的 AbstractTransactionalJunit4SpringContextTests 派生，并且定义了公共的 Hibernate sessionFactory，以及一些公共的方法：

AbstractDaoTest.java

```
@ContextConfiguration(locations = {"hibernate-dao-local-test.xml"})
public abstract class AbstractDaoTest extends
AbstractTransactionalJunit4SpringContextTests {

    // common fields and method can put here
    @Resource(name = "sessionFactory")
    protected SessionFactory sessionFactory;

    /**
     * Manual flush is required to avoid false positive in test when using ORM
     framework
    */
}
```

```
*/  
public void flush() {  
    sessionFactory.getCurrentSession().flush();  
}  
}
```

#### ...16.2.2.1 配置文件

注意到在上面的类中显式地写出：

**@ContextConfiguration(locations = {"hibernate-dao-local-test.xml"})**

这一点很重要，不要过多地使用框架 default，Spring TestContext 会自动在 classpath 中查找以类名开始，以'-context.xml' 结束的配置文件。

如果不明确地指定，会使新手或组内新成员产生更多迷惑和加大学习曲线，而且一不小心改了文件名的话，就不 work 了。

这个配置文件指定了如下几点：

- 基于 annotation
- 定义数据源
- 定义事务管理以及 事务 Advisor

#### hibernate-dao-local-test.xml

```
<context:property-placeholder location="classpath:test-jdbc.properties"/>  
  
<context:annotation-config/>  
  
<bean id="testDataSource"  
class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-  
method="close">  
    <property name="driverClass" value="${jdbc.driverClassName}"/>  
    <property name="jdbcUrl" value="${jdbc.url}"/>  
    <property name="user" value="${jdbc.username}"/>  
    <property name="password" value="${jdbc.password}"/>  
    <property name="minPoolSize" value="${jdbc.minPoolSize}"/>  
    <property name="initialPoolSize" value="${jdbc.initialPoolSize}"/>  
    <property name="maxPoolSize" value="${jdbc.maxPoolSize}"/>  
    <property name="maxStatements" value="${jdbc.maxStatements}"/>  
    <property name="maxIdleTime" value="${jdbc.maxIdleTime}"/>  
    <property name="checkoutTimeout" value="${  
{jdbc.checkoutTimeout}"/>  
</bean>  
  
<!-- Local Hibernate SessionFactory -->  
<bean id="sessionFactory"
```

```
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource" ref="testDataSource"/>

  <property name="configLocation"
value="classpath:hibernate.cfg.xml" />

  <!-- Interceptor to set the createTime and updateTime fields
automatically -->
  <property name="entityInterceptor">
    <bean class="org.pprun.hjpetstore.persistence.AuditInterceptor" />
  </property>
  <property name="eventListeners">
    <map>
      <entry key="merge">
        <bean
class="org.springframework.orm.hibernate3.support.IdTransferringMergeEvent
Listener"/>
      </entry>
    </map>
  </property>
</bean>

  <bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
  </bean>

  <aop:config>
    <aop:advisor pointcut="execution(public *
org.pprun.hjpetstore.dao.hibernate.*(..))" advice-ref="txAdvice"/>
  </aop:config>

  <tx:advice id="txAdvice" >
    <tx:attributes>
      <tx:method name="get*" read-only="true"/>
      <tx:method name="is*" read-only="true"/>
      <tx:method name="search*" read-only="true"/>
      <tx:method name="find*" read-only="true"/>
      <tx:method name="insert*" />
      <tx:method name="save*" />
      <tx:method name="update*" />
      <tx:method name="*" />
    </tx:attributes>
  </tx:advice>
```

```
<!-- {@literal DAO} annotated {@code @Repository} is eligible for Spring
DataAccessException translation. -->
<bean
class="org.springframework.dao.annotation.PersistenceExceptionTranslationPo
stProcessor" />
</beans>
```

### ...16.2.3 Dao 测试

有了这些基于设施，新添加一个基于 Hibernate 的 DAO 的测试代码是很容易：

#### ...16.2.3.1 配置文件

##### HibernateCategoryDaoTest-context.xml

```
<bean id="testCategoryDao"
class="org.pprun.hjpetstore.dao.hibernate.HibernateCategoryDao">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

#### ...16.2.3.2 代码

##### HibernateCategoryDaoTest.xml

**@ContextConfiguration(locations = {"HibernateCategoryDaoTest-context.xml"})**

```
public class HibernateCategoryDaoTest extends AbstractDaoTest {
```

```
    @Resource(name="testCategoryDao")
    private CategoryDao categoryDao;
```

```
    /**
```

```
     * Test of getCategoryList method, of class HibernateCategoryDao.
```

```
     */
```

```
    @Test
```

```
    public void testGetCategoryList() {
        System.out.println("getCategoryList");
```

```
        List<Category> categoryList = categoryDao.getCategoryList();
```

```
        // Use the inherited countRowsInTable() convenience method from
        // AbstractTransactionalUnit4SpringContextTests to verify the results.
```

```
        assertEquals("Hibernate query must return the same number of vets",
super.countRowsInTable("Category"), categoryList.size());
```

```
    }
```

```
/**
 * Test of getCategory method, of class HibernateCategoryDao.
 */
@Test
public void testGetCategory() {
    System.out.println("getCategory");
    String categoryName = "DOGS";

    Category result = categoryDao.getCategory(categoryName);

    assertNotNull("The result set should not be null", result);

    assertEquals(categoryName, result.getCategoryName());
}
}
```

#### ...16.2.4 事务支持

AbstractTransactionalJUnit4SpringContextTests 从名字来看,就知道它具备事务支持,默认情况 rollback = true,也就是只要退出了测试方法,其实对数据库的修改全部回退测试之前的状态,这一点重要:

- 测试代码不必实现 delete / delete cascade 代码,同时避免向 DBA 申请表或实体的 delete 权限
- 测试库的数据不会不停地上涨

##### ...16.2.4.1 自动回滚@rollback

当需要显式指定时,可以直接在 @Test 上面加上 @Rollback(true),这样一来,不管 default 是什么值,此测试方法一定是自动回滚的。

#### ...16.2.5 Service 测试

TODO

#### ...16.3

#### ...16.4 Mock 测试

TODO

#### ...16.5 负载测试

TODO



## 17 调优与监控

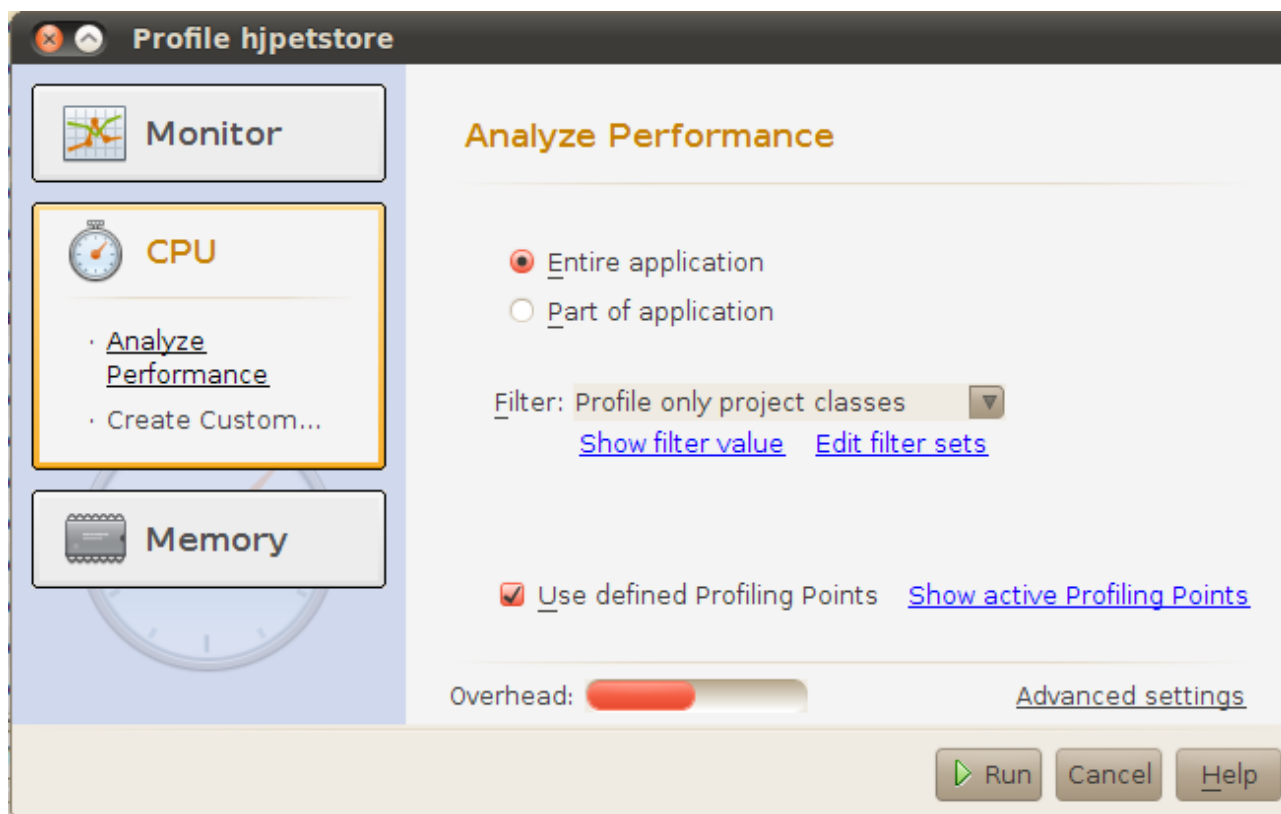
在工程 deliver 给 QA 进行测试前，需要对应用进行调优：

- 观察 JVM Heap 以及 GC 情况，避免内存泄漏
- 监视线程情况
- 检查方法执行时间

### ...17.1 Netbeans Profile

市场上有很多优秀的工具，可以完成以上任务。其实，不用花钱，NetBeans Profile 已经非常好用，它提供：

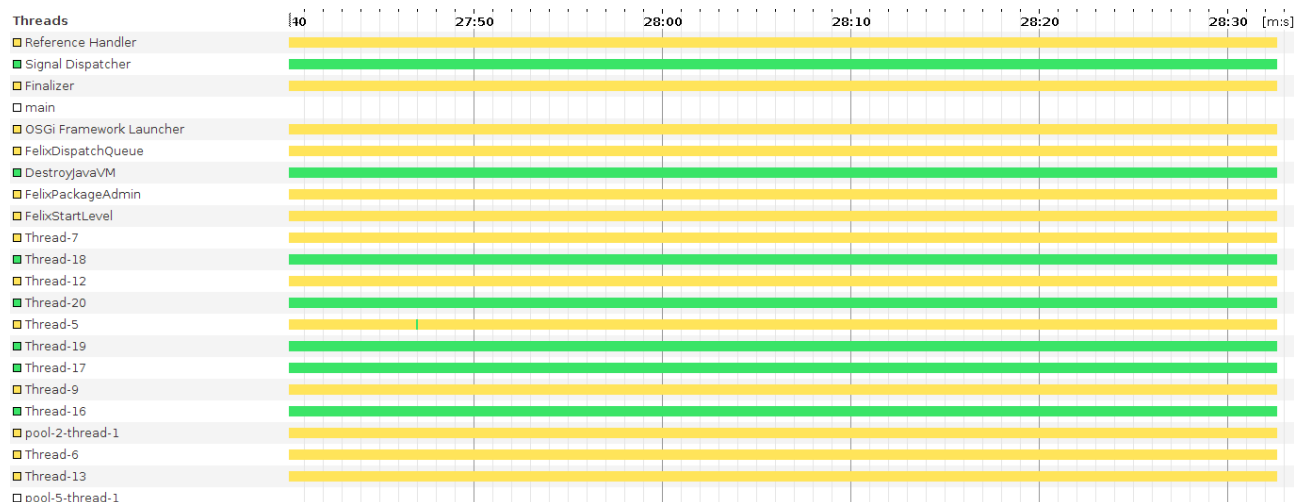
- Monitor 提供对线程数据的监控
- CPU 是对方法执行时间的性能分析
- Memory 是分析 类加载，虚拟机堆内存，以及 GC 时间



在分析数据之前，最好跑一遍单元测试，然后手工把系统的所有功能都走一遍，以模拟所有场景。

#### ...17.1.1 线程分析

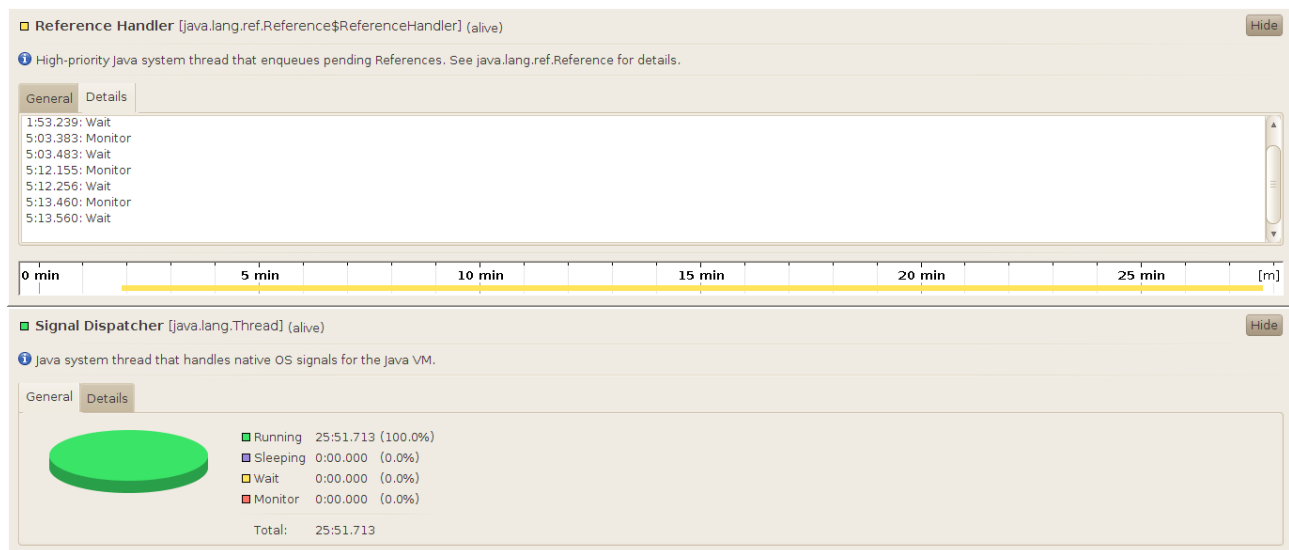
区别显示 active, waiting, sleep, stop 线程



## 线程执行时间列表

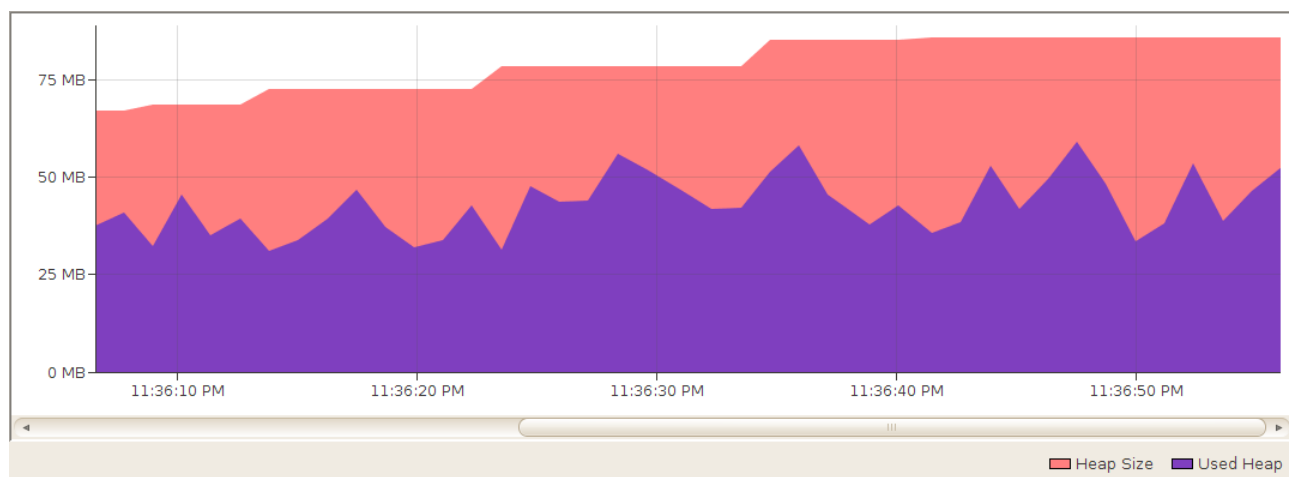
Thread	Running ▾	Sleeping	Wait	Monitor	Total
Signal Dispatcher	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
DestroyJavaVM	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
Thread-18	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
Thread-20	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
Thread-19	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
Thread-16	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
UDP mcast,udp,127.0.1.1:43186	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
DiagnosticsHandler,udp,127.0.1.1:43186	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
FD_SOCKET server socket acceptor,null,null	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
UDP ucast,udp,127.0.1.1:43186	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
Zabbix-agent	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
RMI TCP Accept-0	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
RMI TCP Accept-7776	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
jms_ACCEPT	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
admin_ACCEPT	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
RMI TCP Accept-8686	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
telnetconsole.Listener	26:49.135 (100.0%)	0.0 (0.0%)	0.0 (0.0%)	0.0 (0.0%)	26:49.135
Thread-17	26:49.034 (99.9%)	0.0 (0.0%)	0.0 (0.0%)	0.101 (0.0%)	26:49.135
RMI TCP Connection(5)-127.0.1.1	30.049 (26.3%)	0.0 (0.0%)	1:24.103 (73.6%)	0.0 (0.0%)	1:54.152
RMI TCP Connection(11)-127.0.1.1	29.921 (26.2%)	0.0 (0.0%)	1:24.256 (73.7%)	0.0 (0.0%)	1:54.177
RMI TCP Connection(9)-127.0.1.1	29.920 (26.2%)	0.0 (0.0%)	1:24.236 (73.7%)	0.0 (0.0%)	1:54.156
RMI TCP Connection(13)-127.0.1.1	29.913 (26.2%)	0.0 (0.0%)	1:24.234 (73.7%)	0.0 (0.0%)	1:54.147
RMI TCP Connection(7)-127.0.1.1	29.912 (26.2%)	0.0 (0.0%)	1:24.223 (73.7%)	0.0 (0.0%)	1:54.135
http-thread-pool-8080-(1)	10.776 (0.7%)	0.0 (0.0%)	23:28.335 (99.1%)	1.105 (0.0%)	23:40.216
http-thread-pool-8181-(3)	10.392 (0.8%)	0.0 (0.0%)	19:37.064 (98.9%)	2.658 (0.2%)	19:50.114
http-thread-pool-8080-(2)	7.546 (0.5%)	0.0 (0.0%)	23:24.935 (99.4%)	0.300 (0.0%)	23:32.781

## 每个线程的饼状图显示详细信息



### ...17.1.2 内存分析

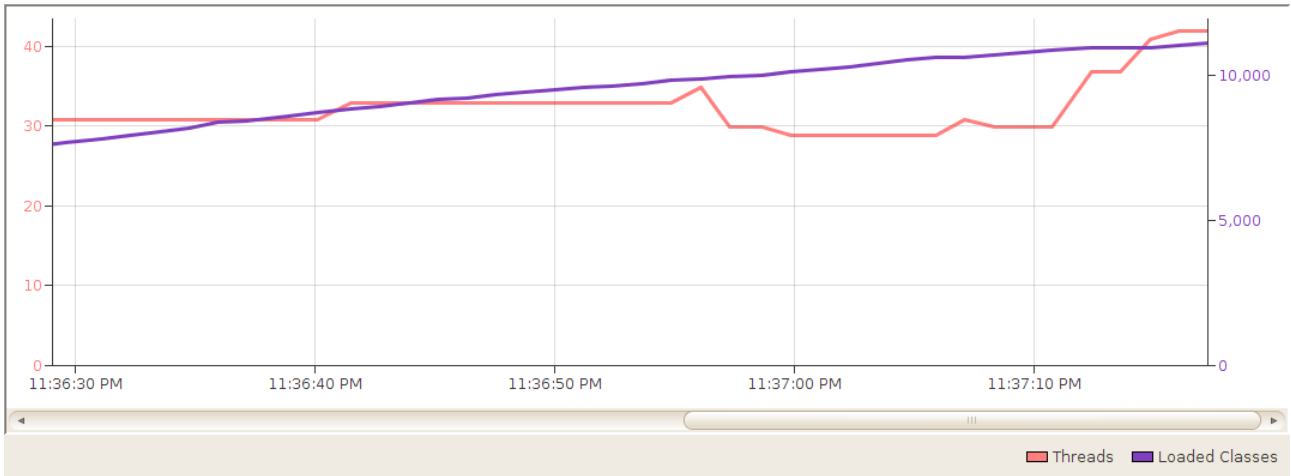
下图是取自我的 laptop 部署 hjpetstore 时的状态，在应用运行阶段，我期望 Heap 的 size 趋于平衡，如果不停的波动，则说明 -Xms, -Xmx 参数有问题



如果发现某个对象分配在运行趋于平衡时段还不停地增长，则有可能产生内存泄漏：

Class Name - Live Allocated Objects	Live Bytes	Live Bytes	Live Objects ▾	Allocated ...	Avg. Age	Generations
org.pprun.common.monitor. <b>MethodInvocation</b>		96 B (0%)	3 (0%)	3	19.7	3
org.pprun.hjpetstore.domain. <b>Banner</b>		64 B (0%)	2 (0%)	3	2.5	2
org.pprun.hjpetstore.web.rest.supplier. <b>SupplierRespon...</b>		16 B (0%)	1 (0%)	1	72.0	1
org.pprun.hjpetstore.web.rest.partner.client. <b>PaymentRe...</b>		32 B (0%)	1 (0%)	1	94.0	1
org.pprun.hjpetstore.web.rest.partner. <b>PaymentController</b>		16 B (0%)	1 (0%)	1	73.0	1
org.pprun.hjpetstore.web.rest.client. <b>SecurityServiceRe...</b>		40 B (0%)	1 (0%)	1	100.0	1
org.pprun.hjpetstore.web.rest. <b>SecurityController</b>		16 B (0%)	1 (0%)	1	72.0	1
org.pprun.hjpetstore.web.rest. <b>HjpetstoreController</b>		16 B (0%)	1 (0%)	1	72.0	1
org.pprun.hjpetstore.web. <b>ViewProductController</b>		16 B (0%)	1 (0%)	1	68.0	1
org.pprun.hjpetstore.web. <b>ViewOrderController</b>		16 B (0%)	1 (0%)	1	67.0	1
org.pprun.hjpetstore.web. <b>ViewItemController</b>		16 B (0%)	1 (0%)	1	68.0	1
org.pprun.hjpetstore.web. <b>ViewCategoryController</b>		16 B (0%)	1 (0%)	1	69.0	1
org.pprun.hjpetstore.web. <b>ViewCartController</b>		16 B (0%)	1 (0%)	1	68.0	1
org.pprun.hjpetstore.web. <b>UserSession</b>		16 B (0%)	1 (0%)	1	17.0	1
org.pprun.hjpetstore.web. <b>UserFormController</b>		104 B (0%)	1 (0%)	1	67.0	1

类加载在运行期望，也同样期望平稳



...17.1.3 方法执行时间分析

按总执行时间长短排列，在解决性能问题时，应该遵循 20/80 原则，即将主要精力放在占去执行总执行时间 80% 的那些代码上。通过这个图，可以清楚地看到，除却 monitor 相关的代码，Dao 方法占据了大部分时间，对于企业应用来说，这是可预见的，因为企业应用是建立在以数据为中心的。优化时应该将主要精力放在排在前 10/20 的方法代码上。

Hot Spots - Method	Self time [%] ▼	Self time	Invocations
org.pprun.hjpetstore.dao.hibernate.HibernateProductDao.getProductListByCategory (String, int...		4487 ms (15.7%)	9
org.pprun.hjpetstore.dao.hibernate.HibernateUserDao.getUserAndFetch (String)		3874 ms (13.6%)	8
org.pprun.common.monitor.MethodInvocationInterceptor.invoke (org.aopalliance.intercept.MethodI...		3122 ms (10.9%)	66
org.pprun.hjpetstore.web.rest.client.SecurityServiceRestClientImpl.decryptCardNumber (String)		2427 ms (8.5%)	6
org.pprun.hjpetstore.web.rest.client.SecurityServiceRestClientImpl.getPaymentPartner (String)		2226 ms (7.8%)	3
org.pprun.hjpetstore.dao.hibernate.HibernateCategoryDao.getCategoryList ()		1823 ms (6.4%)	5
org.pprun.hjpetstore.dao.hibernate.HibernateUserDao.getUser (String, String)		1819 ms (6.4%)	2
org.pprun.hjpetstore.web.rest.partner.client.PaymentRestClientImpl.validateCardNumber (String,...		1120 ms (3.9%)	1
org.pprun.hjpetstore.dao.hibernate.HibernateRSAKeyDao.getEnabledRSAKey ()		1049 ms (3.7%)	7
org.pprun.hjpetstore.dao.hibernate.HibernateProductDao.searchProductList (String, int, int)		1010 ms (3.5%)	3
org.pprun.hjpetstore.dao.hibernate.HibernateOrderDao.getOrderById (long)		875 ms (3.1%)	2
org.pprun.hjpetstore.dao.hibernate.HibernateOrderDao.getOrdersByUsername (String, int, int)		752 ms (2.6%)	1

以下是按调用次数排列，可以显示哪些方法调用最频繁，见最后一列：

Hot Spots - Method	Self time [%]	Self time	Invocations ▾
org.pprun.hjpetstore.persistence.DomainObject.<init> ()		12.6 ms (0%)	431
org.pprun.hjpetstore.persistence.StringEnumUserType.deepCopy (Object)		0.358 ms (0%)	114
org.pprun.hjpetstore.persistence.StringEnumUserType.nullSafeGet (java.sql.ResultSet, String[], Object)		1.70 ms (0%)	103
org.pprun.common.monitor.MethodInvocationInterceptor.invoke (org.aopalliance.intercept.MethodInvocation)		1800 ms (6%)	86
org.pprun.common.monitor.MethodInvocationInterceptor.getStatisticName (org.aopalliance.intercept.MethodInvocation)		1.70 ms (0%)	86
org.pprun.common.monitor.MethodInvocationMonitorMXBeanImpl.addStatistic (String, long, boolean)		1.6 ms (0%)	86
org.pprun.common.monitor.MethodInvocation.onExecute (long, boolean)		0.568 ms (0%)	86
org.pprun.hjpetstore.domain.Category.<init> ()		0.558 ms (0%)	86
org.pprun.hjpetstore.domain.Category.\$\$jvassinst_9.getHibernateLazyInitializer ()		0.747 ms (0%)	85
org.pprun.hjpetstore.domain.Product.<init> ()		0.447 ms (0%)	72
org.pprun.hjpetstore.persistence.jaxb.ProductSummary\$jaxbAccessorM_getProductName_setProductName_java_lang_String.get (Object)		0.163 ms (0%)	63
org.pprun.hjpetstore.persistence.jaxb.ProductSummary\$jaxbAccessorM_getProductDesc_setProductDesc_java_lang_String.get (Object)		0.158 ms (0%)	63
org.pprun.hjpetstore.persistence.jaxb.ProductSummary\$jaxbAccessorM_getImage_setImage_java_lang_String.get (Object)		0.170 ms (0%)	63
org.pprun.hjpetstore.domain.Banner.<init> ()		0.438 ms (0%)	62
org.pprun.hjpetstore.domain.Address.<init> ()		0.772 ms (0%)	44
org.pprun.hjpetstore.persistence.StringEnumUserType.setParameterValues (java.util.Properties)		0.672 ms (0%)	42
org.pprun.hjpetstore.persistence.StringEnumUserType.sqlTypes ()		0.135 ms (0%)	42
org.pprun.hjpetstore.persistence.jaxb.EncryptCardNumber\$jaxbAccessorM_getCardNumber_setCardNumber_java_lang_String.get (Object)		0.128 ms (0%)	42
org.pprun.hjpetstore.persistence.StringEnumUserType.<init> ()		0.140 ms (0%)	42
org.pprun.hjpetstore.persistence.jaxb.DecryptCardNumber\$jaxbAccessorM_getCardNumber_setCardNumber_java_lang_String.get (Object)		0.112 ms (0%)	39
org.pprun.hjpetstore.persistence.StringEnumUserType.equals (Object, Object)		0.098 ms (0%)	38
org.pprun.hjpetstore.domain.Category.toString ()		0.413 ms (0%)	33
org.pprun.hjpetstore.domain.User.<init> ()		0.425 ms (0%)	30
org.apache.jsp.WEB_002dINF.jsp.shop.EditUserForm_jsp._jspx_meth_c_out_36 (javax.servlet.jsp.tagext.JspTag, javax.servlet.jsp.PageContext, int[])		1.36 ms (0%)	30
org.apache.jsp.WEB_002dINF.jsp.shop.EditUserForm_jsp._jspx_meth_c_if_9 (javax.servlet.jsp.tagext.JspTag, javax.servlet.jsp.PageContext, int[])		1.30 ms (0%)	30
org.apache.jsp.WEB_002dINF.jsp.shop.EditUserForm_jsp._jspx_meth_c_out_35 (javax.servlet.jsp.tagext.JspTag, javax.servlet.jsp.PageContext, int[])		0.708 ms (0%)	30

## ...17.2 基于 JMX 的监控体系

当系统在运行状态时，通常希望了解系统的运行状态：

- 当前线程总数
- 内存使用情况
- Gc 参数是否合理

同时也希望系统在出问题或出现不正常的情况能及时通知维护人员，这些情况包括：

- 机器掉电或系统宕机
- Network 不可达
- 外部依赖的服务不可达
- 磁盘空间由于 log 文件不停地累积而告急
- 应用无响应 / 假死导致在一段时间内无 traffic
- 处理某个请求出错率超过预期
- Batch 处理发现记录数低于正常值
- Batch 运行失败

### ...17.2.1 Zabbix

在 Free 的产品中，Zabbix 在这方面已经可以满足所有以上需求，使用之前，所要做的是做一下功课，学习一下 [Zabbix Manual](#)

对于自定义的监视项，可以通过 Zabbix 的 UserParameters 机制来实现，但是，如果应用已经把监视数据暴露给了 JMX agent 的话，一种更方便的选择是 [Zapcat](#)

### ...17.2.2 Zapcat 作为 JMX agent 与 Zabbix 的桥梁

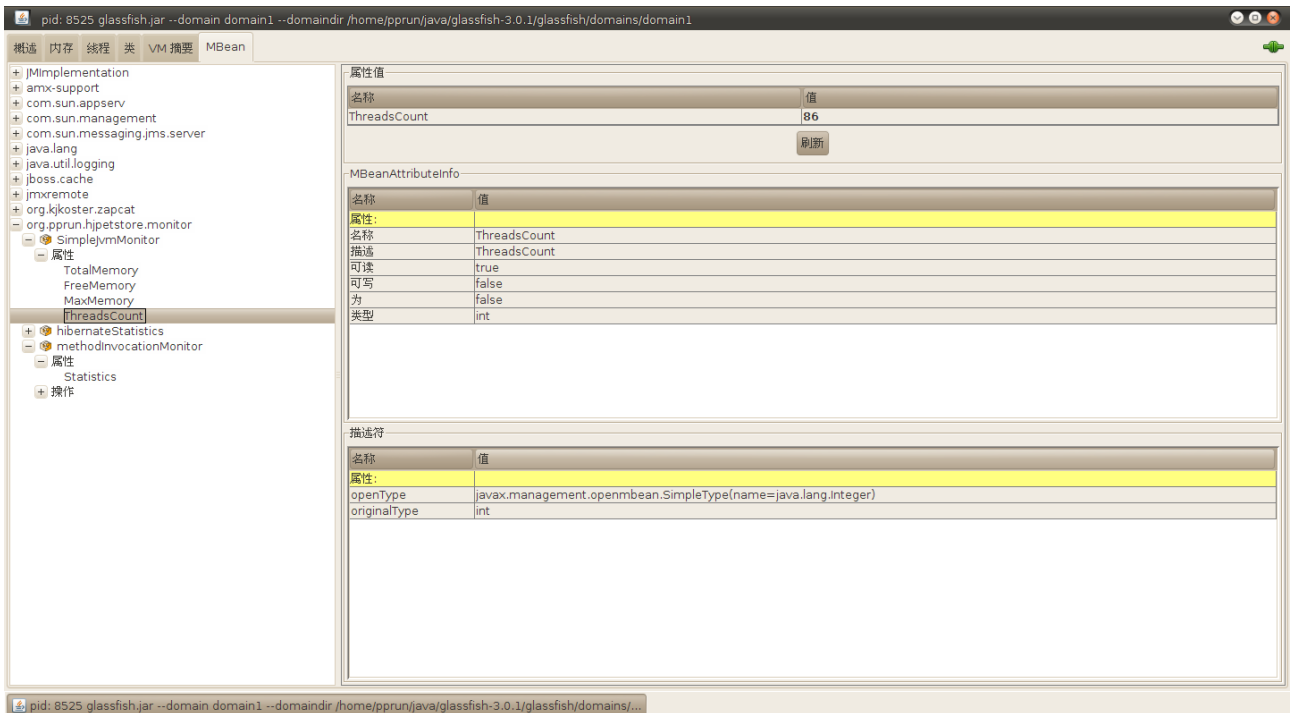
Hjpetstore 实现了以下 JMX 监控

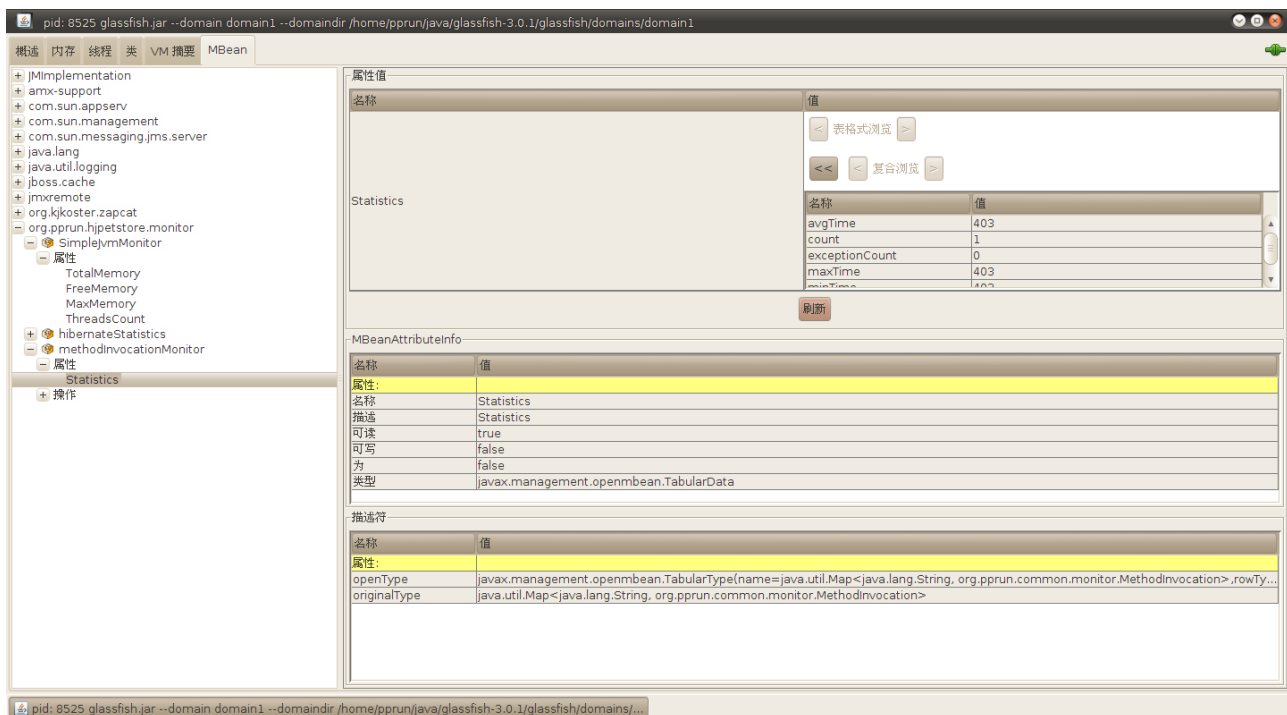
- C3P0 连接池的监控（如果是非 JTA 应用服务器环境的话）

- JVM 堆内存以及线程的监控（JDK 默认有了对 HEAP 及 GC 相关的项，但是因为这些项与选择的垃圾收集器紧密相关的，并不通用）
- service 方法调用统计：调用次数，平均执行时间，最大，最小执行时间以及出现异常的次数

### ...17.2.2.1 使用 jconsole

在 jconsole 中看到所有暴露的 MBean (c3p0 除外，因为使用 jta)





### ...17.2.2.2 zapcat 配置

Hjpetstore 将 zapcat 作为 spring bean 在应用部署时就监听在 10052(可配置)端口上:

#### Monitor.xml

```
<!-- zapcat, the JMX bridge and Java agent for zabbix -->
<bean id="zabbixAgent" class="org.kjkoster.zapcat.zabbix.ZabbixAgent"
destroy-method="stop" lazy-init="false">
  <!-- 'null' to listen on any available address -->
  <constructor-arg index="0"><null/></constructor-arg>
  <!-- the port listen on, default is '10052' -->
  <constructor-arg index="1" value="${zapcat.port}"/>
</bean>
```

```
# zapcat port
zapcat.port=10052
```

### ...17.2.2.3 Zabbix 配置

所有在 jconsole 里头可以查看的内容, 都可以通过 zapcat 在 zabbix 中显示出来, 所要做的就是通过 zabbix php front end 配置 item key

Mbean 属性	Key of Zabbix item jmx[<object name>][<property name>]
org.pprun.hjpetstore.monitor:name=SimpleJvmMonitor.TotalMemory	jmx[org.pprun.hjpetstore.monitor:name=SimpleJvmMonitor][TotalMemory]
org.pprun.hjpetstore.monitor:name=SimpleJvmMonitor.FreeMemory	jmx[org.pprun.hjpetstore.monitor:name=SimpleJvmMonitor][FreeMemory]
org.pprun.hjpetstore.monitor:name=SimpleJvmMonitor.MaxMemory	jmx[org.pprun.hjpetstore.monitor:name=SimpleJvmMonitor][MaxMemory]
org.pprun.hjpetstore.monitor:name=SimpleJvmMonitor.ThreadsCount	jmx[org.pprun.hjpetstore.monitor:name=SimpleJvmMonitor][ThreadsCount]
org.pprun.hjpetstore.monitor:name=methodInvocationMonitor.Statistics	jmx[org.pprun.hjpetstore.monitor:name=methodInvocationMonitor][Statistics]

## 18 为什么写这篇文章

- 皮皮鲁找到了一个机会对过去进行总结，同时将新的技术与思想应用到现实存在的例子当中，因为对于已经上线了的产品，是没有多大的自由来供一线的工程做实验的。
- 作为一线的工程师，在言听计从的背后，如果发现现有的设计的缺陷，可是由于等级关系 ... ， 应该有一个空间去证明自己的想法。
- 希望这篇文章对于成长中的 JAVA EE 工程师有用。
- 年轻的架构师，年轻并不代表 junior，实践出真知。