Compass概念：  
1：Compass相当于hb的SessionFactory  
2：CompassSession相当于hb的Session  
3：CompassTransaction相当于hb的transaction。

Compass 也是采用CompassConfiguration（装载配置和映射文件）进行创建的。创建Compass时将会链接已经存在的索引或者创建一个新的索 引。当Compass创建完后，就可以用compass得到compassSession。compassSession主要是起管理搜索引擎的数据。和 hb的SessionFactory一样，compass通常在系统启动时创建，在所有compassSession创建时使用。

当使用CompassSession查询数据时，将会返回CompassHits接口的实例。compassHits可以得到scores,resources和mapped objects.

Compass也提供了CompassTemplate和CompassCallback类处理会话和事务的处理。CompassTemplate template = new CompassTemplate(compass);

为了简化CompassConfiguration的建立，compass提供了CompassConfigurationFactory类来建立CompassConfiguration  
CompassConfiguration conf =CompassConfigurationFactory.newConfiguration();

除了通过xml文件设置外，可以通过CompassConfiguration.addXXX方法更改设置，也可以通过 CompassSetting来设置。compassSetting和java 的Properties相似。也可以通过CompassEnvironment和LuceneEnvironment类来设置。

Compass中必须设置的项包括：compass.engine.connection。

一个重要的设置方面是组设置。如下面设置一个test的转换器：  
org.compass.converter.test.type=eg.TestConverter  
org.compass.converter.test.param1=value1  
org.compass.converter.test.param2=value2

所有的compass的操作性设置都可以定义在一个配置文件中，文件的名字默认为：compass.cfg.xml，如果取名不一样这在初始化 CompassConfiguration时，使用CompassConfiguration.config(fileName)。

因为索引是事务性的。所以在进行操作的过程中，就存在锁的概念。可以设置锁文件的位置，默认为java.io.tmp，<transaction lockDir="/shared/index-lock" />

别名、资源和属性的概念：  
资源（Resource):资源表示属性的集合，相当于虚拟文档。一个资源通常和一个别名联系在一起，几个资源可以属于同一个别名。别名担当资源和映射定义的联系角色。属性是指一个键值对。

在OSEM/XSEM中，容易忽视的是资源在何处被使用，因为处理内容都被转换成应用程序的模型或者是xml的结构数据。资源很少被使用。

通过资源和属性，可以采用统一的方式访问相同语义的模型。比如应用程序中有两个模型：学生和教师。我们将学生和教师的名字都设置成相同语义的元数据：name（资源属性名），这样将会允许我们所有的name搜索显示结果在资源层次上。

分析器：该组件主要是预处理输入文本。用于搜索和索引的文本分析上。要求搜索和索引使用相同的分析器。  
Compass内置两个分析器名称：default和search。缺省分析器用户没有其它分析器配置时使用。search用于搜索查询的分析。  
配置定制的分析器的参数可以通过Setting方式置入：  
<analyzer name="deault" type="CustomAnalyzer" analyzerClass="eg.MyAnalyzer">  
<setting name="threshold">5</setting>  
</analyzer>

分析过滤器：  
分析过滤器能够被不同的分析器穿插使用。配置如下：  
<analyzer name="deafult" type="Standard" filters="test1, test2" />  
<analyzerFilter name="test1" type="eg.AnalyzerTokenFilterProvider1">  
<setting name="param1" value="value1" />  
</analyzerFilter>  
<analyzerFilter name="test2" type="eg.AnalyzerTokenFilterProvider2">  
<setting name="paramX" value="valueY" />  
</analyzerFilter>

同义处理：同义处理分析过滤器：返回给定词的同义词  
<analyzer name="deafult" type="Standard" filters="synonymFilter" />  
<analyzerFilter name="synonymFilter" type="synonym">  
<setting name="lookup" value="eg.MySynonymLookupProvider" />  
</analyzerFilter>

查询分析器：  
<queryParser name="test" type="eg.MyQueryParser">  
<setting name="param1" value="value1" />  
</queryParser>

索引文件的结构：

compass的子索引相当于lucene的一个索引。当compound设置为true时，lucene次采用一个segments文件存储所有索引内容。子索引对事务型操作尤为重要。

compass支持read\_committed和serializable、batch\_insert级别的事务

compass事务锁用在自索引级别上，这意味着脏操作只发生在各自的子索引上。

compass事务在脏操作（创建，保存，删除）时要求一个锁。搜索时应该只用read only事务。锁超时一般设置为10秒。  
<transaction lockTimeout="15" lockPollInterval="200" />

事务隔离：  
1：read\_committed：当开始该事务时，是不需要锁的。因此速度会快。  
2：serializable：和上面一样。只是当事务开始时，对所有的自索引有一个锁。性能降低。  
3：batch\_insert：使用了lucene提供的快速的批量索引的功能。这个事务操作只支持create操作。如果已经有同名的别名和ids的资源已经存在，那么将会在一个索引中出现两个资源。这种事务是不能回滚的。

FS Transaction Log：存储许多事务数据到文件系统中。  
<transaction isolation="read\_committed">  
<readCommittedSettings>  
<fsTransLog path="/tmp" readBufferSize="32" writeBufferSize="4098" />  
</readCommittedSettings>  
</transaction>

常量子索引hash:  
影射别名到子索引的最简单办法是将某个别名的所有的可搜索内容索引到相同的子索引里面。如：  
<compass-core-mapping>  
<[mapping] alias="test-alias" sub-index="test-subindex">  
<!-- ... -->  
</[mapping]>  
</compass-core-mapping>  
test-alias将会影射所有的实例到test-subindex子索引中。如果sub-index没有定义，则将缺省为alias。

Modulo Sub Index Hashing：允许将一个别名代表的实例索引到不同的子索引中。根据给定的大小对索引进行分割。文件名是给定的前缀+“\_"+数字。  
<compass-core-mapping>  
<[mapping] alias="A">  
<sub-index-hash type="org.compass.core.engine.subindex.ModuloSubIndexHash">  
<setting name="prefix" value="test" />  
<setting name="size" value="2" />  
</sub-index-hash>  
<!-- ... -->  
</[mapping]>  
</compass-core-mapping>  
会产生[test\_0]和[test\_1]两个子索引文件。

Custom Sub Index Hashing：  
ConstantSubIndexHash 和 ModuloSubIndexHash都实现了compass的SubIndexHash接口。定制子索引hash必须实现getSubIndexes和 mapSubIndex(String alias,Property[] ids)两个方法。

Optimizers:优化器，每个脏操作提交成功都会在各自的子索引中产生另一个segment，子索引中的segment越多，搜索操作就越慢，因此保持索引优化，控制segment的数量很重要。要做的就是合并小segment到大的segment 。  
索引优化器在子索引级别执行。在优化的过程中，优化器将锁定子索引，以便于脏操作。

调度优化：compass的每一个优化器都能包装为调度方式执行。  
<optimizer scheduleInterval="90" schedule="true" />

Aggressive Optimizer：通过设置segments的大小，当达到指定的大小时，将所有的segement合并到一个segment。这样搜索的效率最高。  
Adaptive Optimizer：和Aggressive Optimizer不同的是，该优化器只合并新的小segment。  
Null Optimizer：不做任何优化。当做batch\_insert事务时，离线创建索引或已经全部优化索引，一般使用它。

直接访问Lucene：compass提供了LuceneHelper类，该类可以直接访问lucene的api。

索引的对象遵循以下原则：  
实现默认的无参数构造器，不要是public的。便于compass采用Constants.newInstance()  
提供identifier，  
提供访问和设置方法  
建议重载equals和hashcode方法。建议以业务主键为参考。

alias:每一个影射定义都注册了一个别名。这个别名用来联系类的osem定义和类本身。  
Root：在compass中有两类可搜索的类：root searchable和non-root searchable 类。root searchable类最好定义作为hits结果返回的类。non-root searchable类不要求定义id影射。

子索引：默认情况下，每一个root searchable类都有自己的子索引，名称缺省为alias。子索引的名称也可以自由控制。允许几个root searchable类索引到相同的子索引中。或者用子索引hash功能。

searchable id不要求定义搜索的元数据，如果没有定义，compass自动创建内部的元数据id。如果searchable id不需要被搜索，那么需要为它定义一个可搜索元数据。注意下面的元数据定义方式：

@Searchable  
public class Author {  
@SearchableId(name = "id")  
private Long id;  
// ...  
}

@Searchable  
public class Author {  
@SearchableId  
@SearchableMetaData(name = "id")  
private Long id;  
//

Searchable Constant：允许对一个类定义一系列的的常量数据。对于添加静态元数据是非常有用的。  
<constant>  
<meta-data>type</meta-data>  
<meta-data-value>person</meta-data-value>  
<meta-data-value>author</meta-data-value>  
</constant>

Searchable Dynamic Meta Data：允许将表达式的结果保存到搜索引擎中。该影射不能影射任何类属性。动态元数据的值是根据动态转换器计算表达式得到的。compass内建了比如 el，jexl，velocity，ognl，groovy等转换器。当定义表达式后。root 类被注册为data key下的值。  
<dynamic-meta-data name="test" converter="jexl">  
data.value + data.value2  
</dynamic-meta-data>

Searchable Reference：映射root类和其它类之间的关系。在marshals的过程中，只marshal参考对象的id。但是在unmarshal过程中根据id装载参考的对象。  
compass在参考对象上不执行任何级联操作，也不提供延迟加载。  
<class name="A" alias="a">  
<id name="id" />  
OSEM - Object/Search Engine Mapping  
Compass - Java Search Engine 40  
<reference name="b" />  
<!-- ... -->  
</class>  
<class name="B" alias="b">  
<id name="id" />  
<!-- ... -->  
</class>

Searchable Component：嵌入一个可搜索类到本身的搜索类中。组件参考搜索类能够设置为 Root。为Root的组件一般都是具有id属性。比如人员和姓名组件（non-root)，人员和帐户（root)。  
<class name="A" alias="a">  
<id name="id" />  
<component name="b" />  
<!-- ... -->  
</class>  
<class name="B" alias="b" root="false">  
<!-- ... -->  
</class>

继承处理：  
<class name="A" alias="a">  
<id name="id" />  
<property name="aValue">  
<meta-data>aValue</meta-data>  
</property>  
</class>  
<class name="B" alias="b" extends="a">  
<property name="bValue">  
<meta-data>aValue</meta-data>  
</property>  
</class>

Root 类有自己的索引，而依赖Root类的非Root类不需要索引。  
Class mapping能够继承其它class mapping（可以超过一个）。也能够继承contract mapping。

contract：相当于java语言中的接口。如果有几个相同的类具有相似的属性。就可以定义一个contract。然后在子类中extend该contract。

通用元数据提供了将元数据名称和别名定义从osem文件提取到外面的方式。当你的应用程序有大量的域模型时尤其有用。另外一个优势就是添加额外的信息倒元数据中，不如描述。也能制定元数据定义的格式，这样就不用在osem 文件中定义了 。  
通过集中话元数据，其它工具也能更好地利用这些信息。  
OSEM文件引用通用元数据的方式是采用${}.

query syntax：  
jack ：缺省的查询域中包括jack字段。  
jack london：缺省的查询域中包括 jack 或 london, 或者2者都有。  
+jack +london： 缺省的查询域中必须包括jack和london。  
name:jack：name字段中包括jack。  
name:jack -city:london ：name字段中包括jack但是city字段中不包括london。  
name:"jack london" ：name字段中包括jack london短语。  
name:"jack london"~5 ：name字段包括至少5次jack and london短语  
jack\* 包含以jack开头的词条。  
jack~ 包括以jack结尾的词条。  
birthday:[1870/01/01 TO 1920/01/01] birthday从1870-01-01到1920-01-01。

CompassHits, CompassDetachedHits & CompassHitsOperations：  
compassHits：所有的搜索结果都是通过该接口访问。只能用在事务上下文中。如果脱离上下文，则需要detached。compassHits和compassDetachedHits都共享相同的操作接口：compassHitsOperation。

getLength() or length() ：得到搜索资源的长度  
score(n) 第n个搜索资源的分值。  
resource(n) 第n个搜索资源  
data(n) 第n个对象实例。

CompassQuery and CompassQueryBuilder：  
CompassQueryBuilder提供了程序创建compassQuery的功能。compassQuery能够用来添加排序和执行查询。  
CompassHits hits = session.createQueryBuilder()  
.queryString("+name:jack +familyName:london")  
.setAnalyzer("an1") // use a different analyzer  
.toQuery()  
.addSort("familyName", CompassQuery.SortPropertyType.STRING)  
.addSort("birthdate", CompassQuery.SortPropertyType.INT)  
.hits();

CompassQueryBuilder queryBuilder = session.createQueryBuilder();  
CompassHits hits = queryBuilder.bool()  
.addMust( queryBuilder.term("name", "jack") )  
.addMustNot( queryBuilder.term("familyName", "london") )  
.toQuery()  
.addSort("familyName", CompassQuery.SortPropertyType.STRING)  
.addSort("birthdate", CompassQuery.SortPropertyType.INT)  
.hits();

注意排序的属性必须是un\_tokenized。

OSEM映射文件：  
<class name="eg.A" alias="a">  
<id name="id" />  
<property name="familyName">  
<meta-data>family-name</meta-data>  
</property>  
<property name="date">  
<meta-data converter-param="YYYYMMDD">date-sem</meta-data>  
</property>  
Working with objects  
Compass - Java Search Engine 78  
</class>  
查询方式：采用compassQueryBuilder，许多查询可以直接工作在mapping的层次上。  
CompassQueryBuilder queryBuilder = session.createQueryBuilder();  
CompassHits hits = queryBuilder.term("a.familyName.family-name", "london").hits();  
// 采用类属性的元数据id, 在上面的例子中将采用第一个元数据.  
CompassHits hits = queryBuilder.term("a.familyName", "london").hits();  
//查询编辑器将会采用相应的转化器转换数据。  
CompassHits hits = queryBuilder.term("a.date.date-sem", new Date()).hits();  
CompassHits hits = queryBuilder.bool()  
.addMust( queryBuilder.alias("a") )  
.addMust( queryBuilder.term("a.familyName", "london") )  
.toQuery().hits();

CompassHighlighter：提供高亮度匹配搜索的文字字段。  
CompassHits hits = session.find("london");  
String fragment = hits.highlighter(0).fragment("description");  
高亮度只能用于CompassHits，只能用在事务上下文中。

在detachedHits中使用高亮度：  
CompassHits hits = session.find("london");  
//在事务上下文中处理高亮度。  
for (int i = 0.; i < 10; i++) {  
hits.highlighter(i).fragment("description"); // this will cache the highlighted fragment  
}  
CompassHit[] detachedHits = hits.detach(0, 10).getHits();  
// outside of a transaction (maybe in a view technology)  
for (int i = 0; i < detachedHits.length; i++) {  
// this will return the first fragment  
detachedHits[i].getHighlightedText().getHighlightedText();  
// this will return the description fragment, note that the implementation  
// implements the Map interface, which allows it to be used simply in JSTL env and others  
detachedHits[i].getHighlightedText().getHighlightedText("description");  
}

GPS通过2个概念提供了整合不同的可索引的数据源：CompassGps和CompassGpsDevice。Device可以结合任何类型的可 索引数据来源，它提供索引数据、搜索数据、敏感数据变化的能力。 GPS建立在Compass基础之上。利用Compass的特征，如：事务、OSEM以及 API等。

CompassGps是GPS的主要接口，它拥有一系列的CompassGpsDevices，并且管理他们的生命周期。

Compass提供了两个Gps的实现：  
SingleCompassGps：拥有一个compass实例。这个compass实例用来做索引和镜像操作。  
DualCompassGps：拥有两个Compass实例。indexCompass和mirrorCompass。主要处理两个事务级别。 indexCompass一般采用 batch\_insert隔离级别，而mirrorCompass采用read\_committed事务级别。

hibernate Gps Device

hb3新的基于时间的机制提供了实时数据改变的镜像。数据的传送顺序为：Database -- Hibernate -- Objects -- Compass::Gps --Compass::Core (Search Engine).  
在hb3中，程序配置：  
Compass compass = ... // set compass instance  
CompassGps gps = new SingleCompassGps(compass);  
CompassGpsDevice hibernateDevice =  
new Hibernate3GpsDevice("hibernate", sessionFactory);  
gps.addDevice(hibernateDevice);  
.... // configure other devices  
gps.start();

hibernate device提供了一个fetchCount参数，这个参数控制索引一个类的分页进程。

实时数据镜像：在hb3中是基于新的时间机制。数据改变将会反射给compass index。配置时要注意的是系统和compass必须使用相同的SessionFactory。如果使用hb3和spring，请使用 SpringHibernate3GpsDevice。

Compass与spring结合：  
1：支持compass级别的工厂bean。采用Spring ioc的配置选项。  
2：提供Compass DAO级别的支持。采用事务整合和DAO支持类。  
3：扩展GPS Hiberante3 device。  
4：提供OJB的整合支持。  
5：提供Spring MVC的搜索控制器和索引控制器。

spring中配置compass bean:  
<bean id="compass"  
class="org.compass.spring.LocalCompassBean">  
<property name="resourceLocations">  
<list>  
<value>classpath:org/compass/spring/test/A.cpm.xml</value>  
</list>  
</property>  
<property name="compassSettings">  
<props>  
<prop key="compass.engine.connection">  
target/testindex  
</prop>  
<!-- This is the default transaction handling  
(just explicitly setting it) -->  
<prop key="compass.transaction.factory">  
org.compass.core.transaction.LocalTransactionFactory  
</prop>  
</props>  
</property>  
</bean>

compass提供CompassDaoSupport类来执行Dao操作。

public class LibraryCompassDao extends CompassDaoSupport {  
public int getNumberOfHits(final String query) {  
Integer numberOfHits = (Integer)getCompassTemplate().execute(  
new CompassCallback() {  
public Object doInCompass(CompassSession session) {  
CompassHits hits = session.find(query);  
return new Integer(hits.getLength());  
}  
}  
);  
}  
return numberOfHits.intValue();  
}

然后配置该DAO类使用的Compass实例：  
<beans>  
<bean id="libraryCompass" class="LibraryCompassDao">  
<property name="compass">  
<ref local="compass" />  
</property>  
</bean>  
</beans>

SpringSyncTransaction：compass将同步spring本身的事务.  
SpringHibernate3GpsDevice：对Hibernate3GpsDevice的扩展，能够处理spring带来的sessionFactory以便使用时间机制监听实时数据改变。