

HBase应用开发

www.huawei.com



目标

- 学完本课程后，您将能够：
 - 了解**HBase**应用开发适用场景
 - 熟悉**HBase**应用开发流程
 - 熟悉并使用**HBase**常用**API**
 - 理解业务表设计基本原则
 - 进行**HBase**应用开发



目录

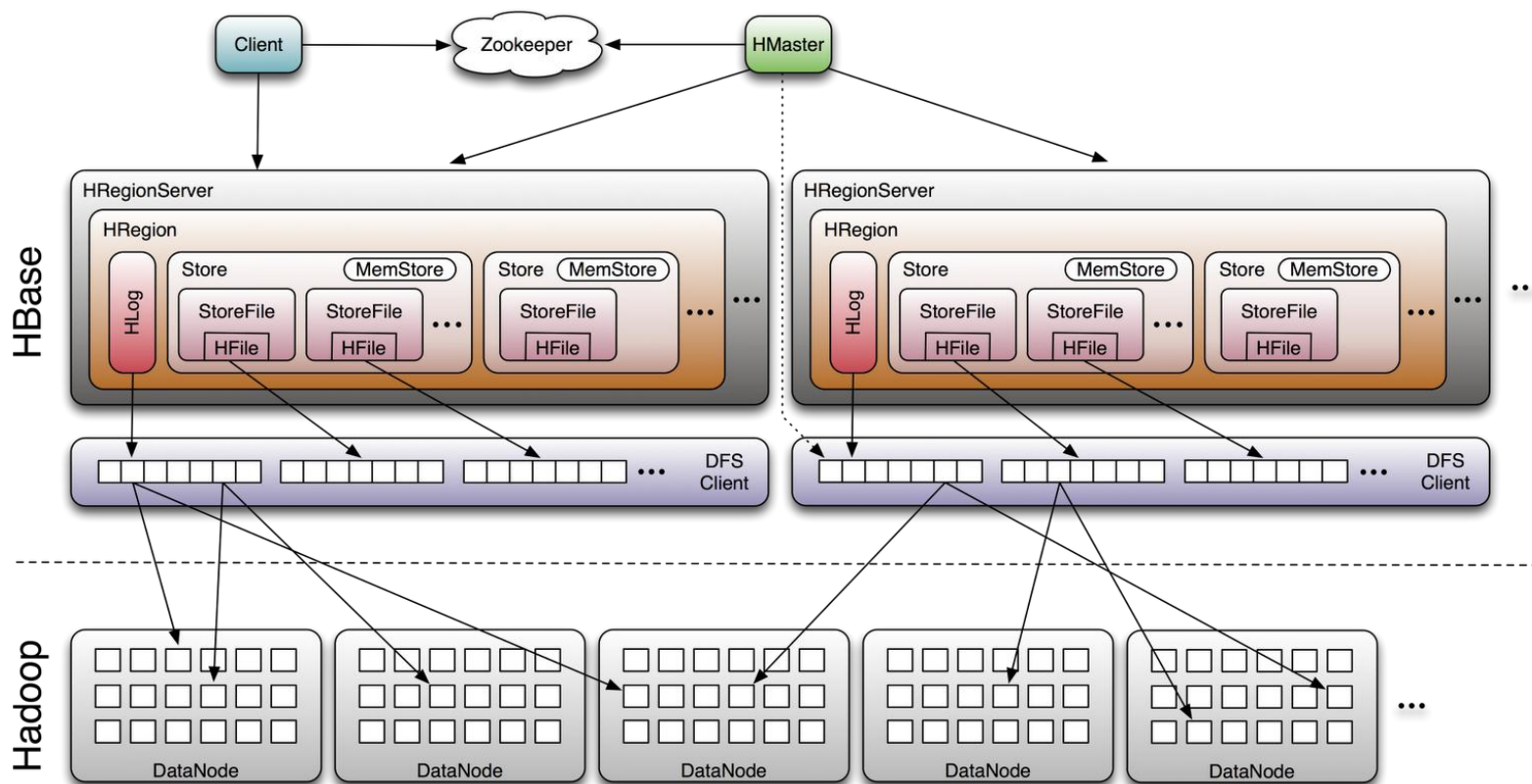
1. HBase应用场景
2. HBase应用开发流程
3. 应用开发案例分析
4. 表设计指导
5. 常用开发接口示例
6. 应用开发实践

HBase的定义

HBase是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统。

- 适合于存储大表数据（表的规模可以达到数十亿行以及数百万列），并且对大表数据的读、写访问可以达到实时级别；
- 利用**Hadoop HDFS**（**Hadoop Distributed File System**）作为其文件存储系统，提供实时读写的数据库系统；
- 利用**ZooKeeper**作为协同服务。

HBase架构回顾



HBase的适用场景

- **HBase**适合具有如下需求的应用：
 - 海量数据（**TB**、**PB**）
 - 高吞吐量
 - 需要在海量数据中实现高效的随机读取
 - 需要很好的性能伸缩能力
 - 能够同时处理结构化和非结构化的数据
 - 不需要完全拥有传统关系型数据库所具备的**ACID**特性

成功应用场景1

- 项目背景

- 某银行仅支持查询最近一年的账户历史交易情况
- 超过一年的查询需要特殊申请，由专人进行人工查询

- 原因

- 1.传统数据库无法存储海量数据
- 2.大数据量下查询性能急剧下降

- HBase的优势

- 海量数据（**TB**、**PB**）：可由查询一年变为十年或更多
- 高效随机读取：查询超过一年的数据与查询最近数据同样高效

成功应用场景2

- 项目背景

- 某银行新增业务，希望提高账单分期的推广效率和收益。
- 计划利用海量的数据进行分析，挖掘出**3000**维客户特征，训练和执行信用卡账单分期预测模型，得到潜在账单分期客户列表。

- 项目特点与**HBase**的优势

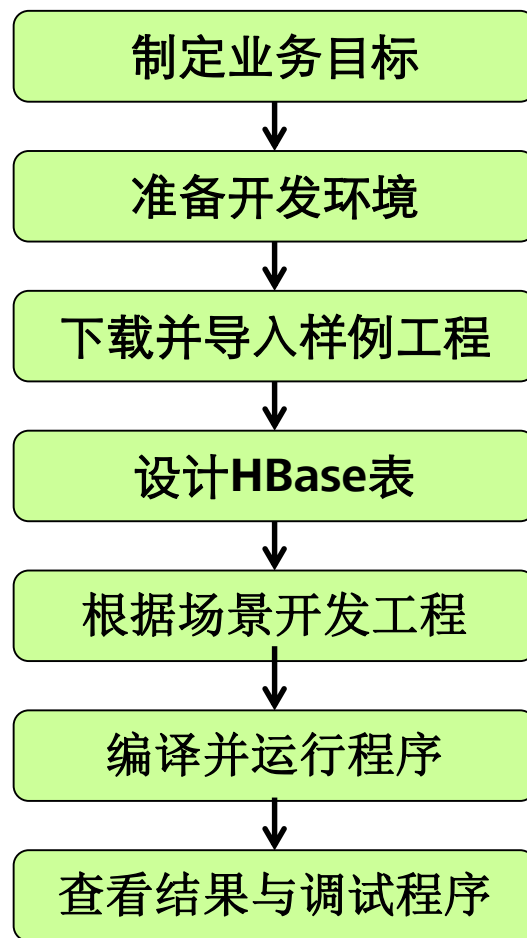
- 数据来源广，包含信用卡客户数据、交易数据、历史分期数据、其他来源数据。**(HBase表设计简单快捷)**
- 数据列非常多，并且非常稀疏。（支持列动态扩展）
- 数据源易变更，需动态扩展。（同时处理结构和非结构化数据）



目录

1. HBase应用场景
2. HBase应用开发流程
3. 应用开发案例分析
4. 表设计指导
5. 常用开发接口示例
6. 应用开发实践

HBase应用开发流程



制定业务目标

- 制定业务目标
 - 数据量?
 - 写入场景? 批量还是实时
 - 写入性能要求?
 - 查询场景? 影响**RowKey**设计和表设计
 - 查询性能要求?

准备开发环境

- 准备开发环境

准备项	说明
操作系统	Windows 系统，推荐 Windows 7 以上版本。
安装 JDK	开发环境的基本配置。版本要求： 1.7 或者 1.8 。
安装和配置 Eclipse	用于开发 HBase 应用程序的工具。
网络	确保客户端与 HBase 服务主机在网络上互通。

下载并导入HBase样例工程

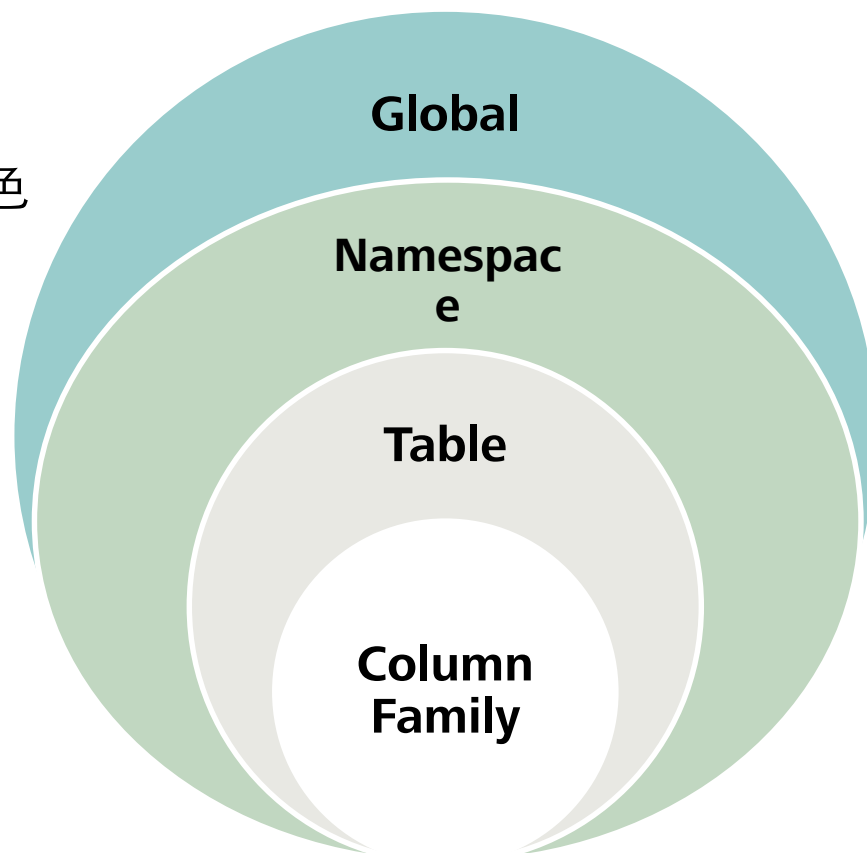
- 下载并导入**HBase**样例工程
 - 1、下载并解压**HBase**客户端压缩包
 - 2、在**FusionInsight Manager**页面新建用户，用于认证与操作
 - 3、下载用户的认证凭据文件
 - 4、配置认证凭据文件到**HBase**客户端样例工程
 - 5、执行样例工程自动配置脚本（已完成配置文件和**jar**的拷贝）
 - 6、导入样例工程到**Eclipse**开发环境并学习样例代码

下载并导入HBase样例工程-用户权限

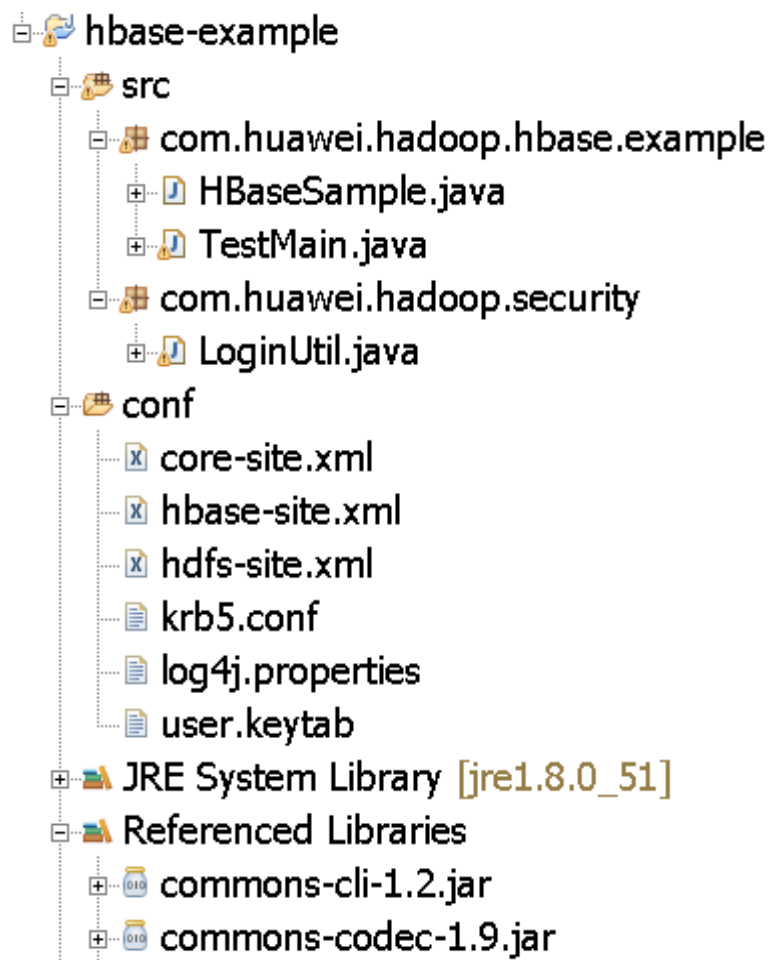
- **HBase**权限传递关系

- 权限控制级别分为四层角色

- Global
 - Namespace
 - Table
 - Column Family



下载并导入HBase样例工程-样例代码1



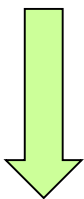
样例源码

配置文件

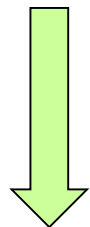
依赖jar

下载并导入HBase样例工程-样例代码2

获取
配置



安全
认证



```
Configuration conf = HBaseConfiguration.create();
String userdir = System.getProperty("user.dir") +
File.separator + "conf" + File.separator;
conf.addResource(new Path(userdir + "core-site.xml"));
conf.addResource(new Path(userdir + "hdfs-site.xml"));
conf.addResource(new Path(userdir + "HBase-site.xml"));
```

```
if (User.isHBaseSecurityEnabled(conf)) {
    userName = "HBaseDeveloper";
    userKeytabFile = userdir + "user.keytab";
    krb5File = userdir + "krb5.conf";

    LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME,
        userName, userKeytabFile);

    LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL
        _KEY,
        ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
    LoginUtil.login(userName, userKeytabFile, krb5File, conf);
}
```


下载并导入HBase样例工程-样例代码2

获得
Admin

```
Connection conn =ConnectionFactory.createConnection(conf);  
Admin admin = conn.getAdmin();
```



设置表
属性

```
TableName tableName =  
    TableName.valueOf("HBase_sample_table");  
HTableDescriptor htd = new HTableDescriptor(tableName);  
HColumnDescriptor hcd = new HColumnDescriptor("info");  
htd.addFamily(hcd);
```



调用建表
API

```
admin.createTable(htd);
```

设计HBase表

- 设计**HBase**表
 - 根据业务表的关系设计表与**Family**
 - 根据查询和写入场景设计**RowKey**

根据场景开发工程

- 根据场景开发工程
 - 梳理业务场景流程
 - 设计各模块接口
 - 如果使用的是安全集群，需要进行安全认证
 - 熟悉**HBase**提供的相应**API**
 - 调用业务需要的**API**实现各功能

编译并运行程序

- 编译并运行程序
 - 在开发环境**Eclipse**中，右击**TestMain.java**，单击“**Run as > Java Application**”运行对应的应用程序工程。

查看结果与调试程序

- 查看结果与调试程序
 - 查看**HBase API**返回结果是否符合预期。
 - 若有运行**HBase shell**或者访问**Web** 页面权限，可以通过**shell**操作命令或者浏览**HBase**原生页面去查看程序运行结果。（如建表是否成功、数据是否已写入）。

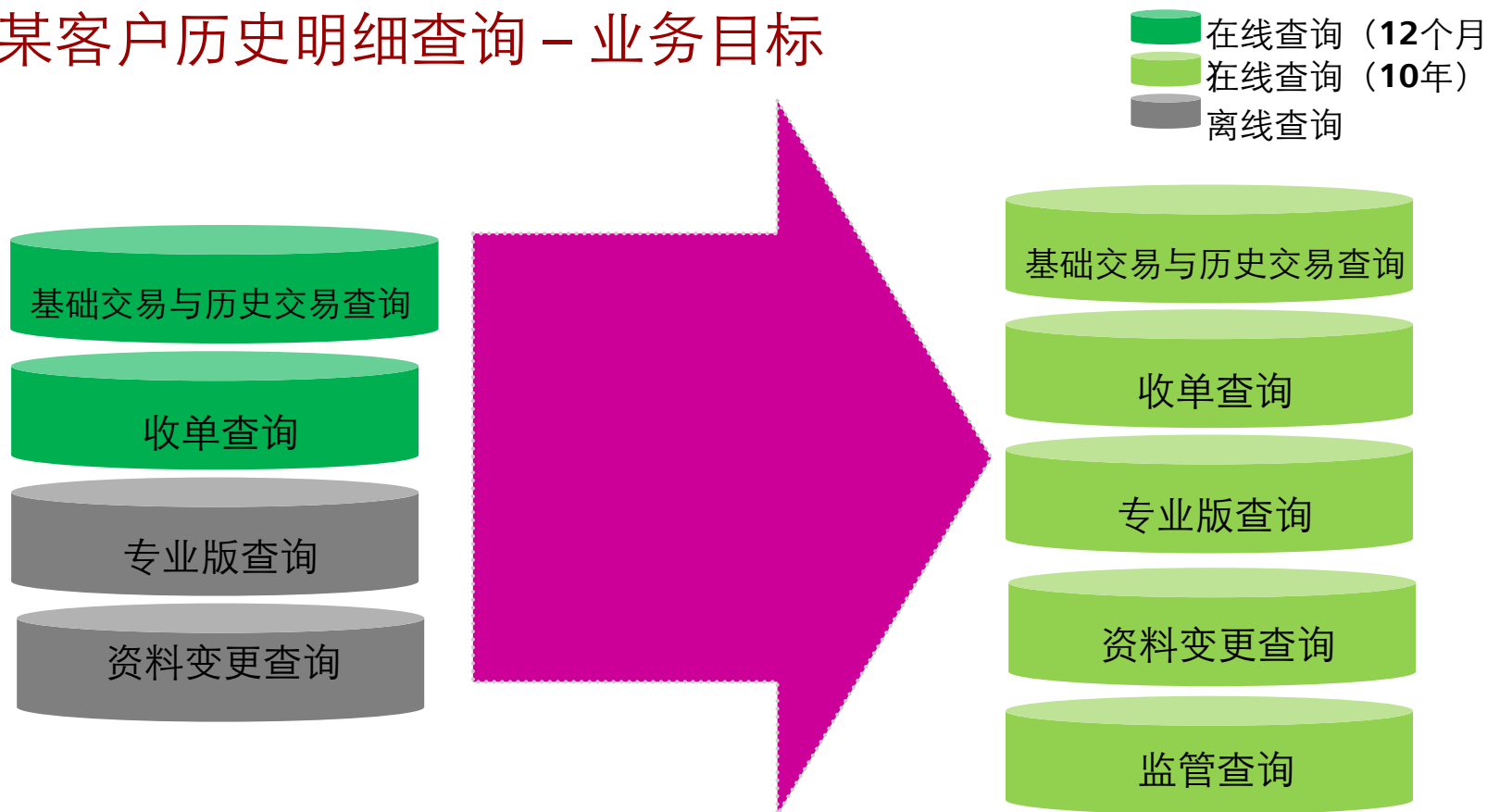


目录

1. HBase应用场景
2. HBase应用开发流程
3. 应用开发案例分析
4. 表设计指导
5. 常用开发接口示例
6. 应用开发实践

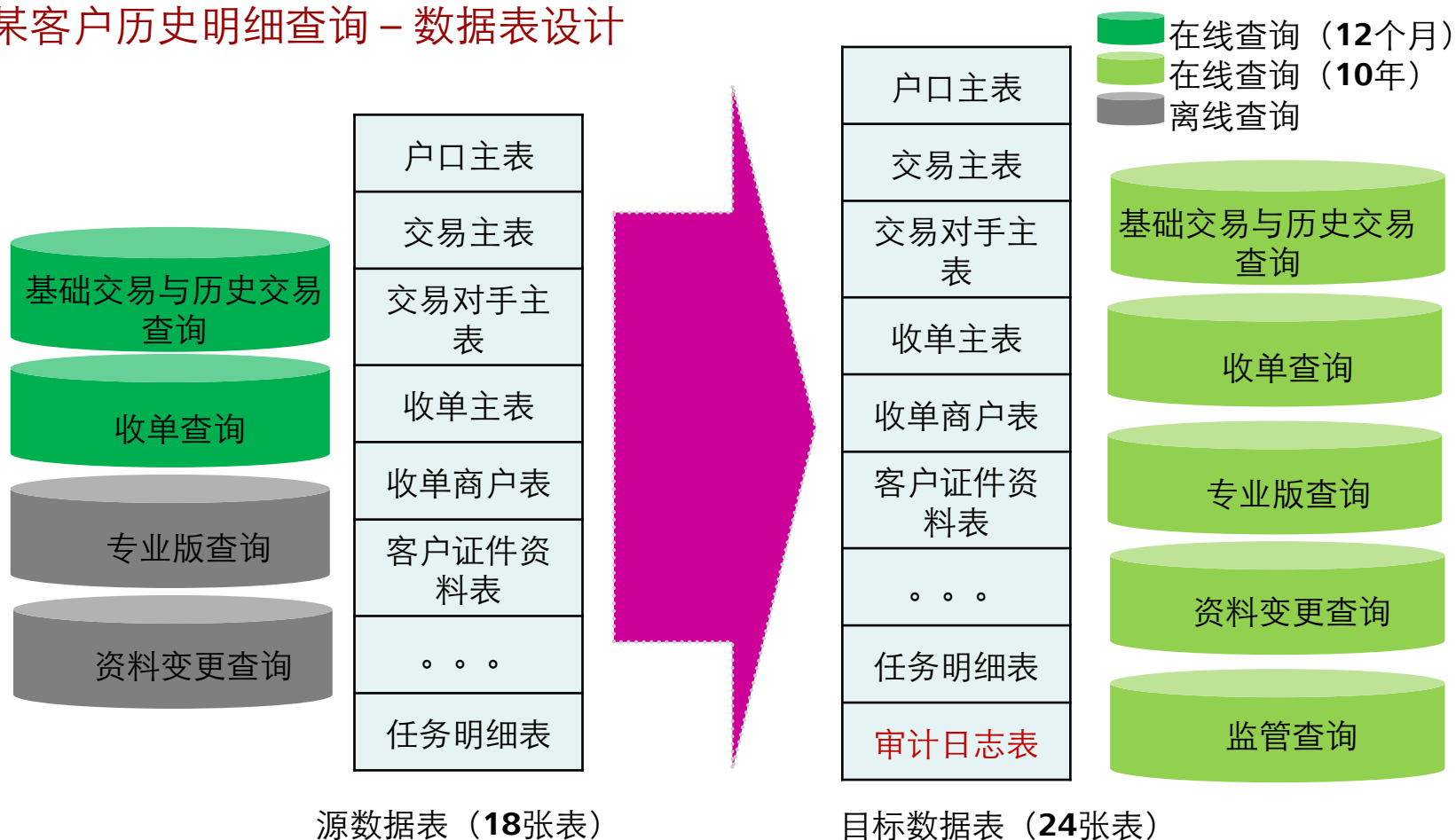
应用开发案例分析1

某客户历史明细查询 – 业务目标



应用开发案例分析2

某客户历史明细查询 – 数据表设计



应用开发案例分析3

某客户历史明细查询 – HBase表设计原则



应用开发案例分析4

某客户历史明细查询 – 用户历史资金交易表传统表索引设计

查询场景：

1. 【客户号】 【起始日期】 【结束日期】
2. 【客户号】 【账户号】 【起始日期】 【结束日期】
3. 【账户号】 【起始日期】 【结束日期】

通过这三种查询条件
可以一次性实时查询
某客户/账户起始日期
到结束日期期间的户口
历史资金交易数据

黄色字段为主键
绿色字段为索引字段

表名	XXXdetailTable		
描述	XXX交易基础信息表		
主键	DATE+SYSTEM_CODE+TRADE_NUMBER		
索引	CUSTOM_ID+ACCOUNT_NUMBER		
数据量			
字段	类型	长度	描述
DTAE	DATE	4	交易日期
SYSTEM_CODE	CHARACTER	3	系统代码
TRADE_NUMBER	CHARACTER	20	交易顺序号
TRADE_CONTENT	VARCHAR	200	交易内容
CUSTOM_ID	CHARACTER	15	客户号
ACCOUNT_NUMBER	CHARACTER	8	账户号
CUSTOM_INFORMATION	CHARACTER	60	客户详细信息

应用开发案例分析5

某客户历史明细查询 – 用户历史资金交易新表设计1

Rowkey		value	
日期	流水号	客户号	其他字段
20150101	001	客户A	...
20150101	002	客户B	...
20150101	003	客户B	...
20150101	004	客户C	...
20150102	001	客户A	...
20150102	002	客户A	...
20150102	003	客户B	...
20150102	004	客户C	...
20150103	001	客户A	...
20150103	002	客户B	...

在此种**Rowkey**设计下，如果要查询“客户B在2015/1/01到2015/1/02的刷卡记录”，那么在进行**HBase**数据扫描时就会扫描到中间三条不必要的记录。

Rowkey			value
客户号	日期	流水号	其他字段
客户A	20150101	001	...
客户A	20150102	001	...
客户A	20150102	002	...
客户A	20150103	001	...
客户B	20150101	002	...
客户B	20150101	003	...
客户B	20150102	003	...
客户B	20150103	002	...
客户C	20150101	004	...
客户C	20150102	004	...

在此种**Rowkey**设计下，如果要查询“客户B在2015/1/01到2015/1/02的刷卡记录”，那么将只查询中间客户B的三条记录，减少了不必要的查询。

应用开发案例分析6

某客户历史明细查询 – 用户历史资金交易新表设计2

Rowkey			value	
客户号	日期	流水号	其他字段	分布 region
客户A	20150101	001	...	全部分布在以“客户”字符开头的 region 中
客户A	20150102	001	...	
客户A	20150102	002	...	
客户A	20150103	001	...	
客户B	20150101	002	...	
客户B	20150101	003	...	
客户B	20150102	003	...	
客户B	20150103	002	...	
客户C	20150101	004	...	
客户C	20150102	004	...	

所有记录分布在同一个 **region** 中，数据分布不均匀。比如客户号均以省份代码开头，导致用户数据以省份进行聚集，可能会造成超大热点 **region**。

Rowkey			value	
客户号	日期	流水号	其他字段	分布 region
A客户	20150101	001	...	以“A”字符开头的 region
A客户	20150102	001	...	
A客户	20150102	002	...	
A客户	20150103	001	...	
B客户	20150101	002	...	以“B”字符开头的 region
B客户	20150101	003	...	
B客户	20150102	003	...	
B客户	20150103	002	...	
C客户	20150101	004	...	以“C”字符开头的 region
C客户	20150102	004	...	

所有记录按照用户特征分类分布，数据分布均匀。比如将卡号后4位提前，这样就可以让 **region** 均匀分布。

应用开发案例分析7

某行历史明细查询 – 用户历史资金交易表建表结果

表名	XXXdetailTable		
描述	XXX交易基础信息表		
主键	DATE+SYSTEM_CODE+TRADE_NUMBER		
索引	CUSTOM_ID+ACCOUNT_NUMBER		
数据量			
字段	类型	长度	描述
<i>DTAE</i>	DATE	4	交易日期
<i>SYSTEM_CODE</i>	CHARACTER	3	系统代码
<i>TRADE_NUMBER</i>	CHARACTER	20	交易顺序号
TRADE_CONTENT	VARCHAR	200	交易内容...
CUSTOM_ID	CHARACTER	15	客户号
ACCOUNT_NUMBER	CHARACTER	8	账户号
CUSTOM_INFORMATION	CHARACTER	60	客户详细信息...
...



应用开发案例分析7

某行历史明细查询 – 用户历史资金交易表建表结果

表名		XXXdetailTable		
描述		XXX交易基础信息表		
RowKey		CUSTOM_ID (客户编号)+ DATE (交易时间) + TRADE_NUMBER (交易顺序号) + SYSTEM_CODE (系统代码)		
Family	字段	类型	长度	描述
H (热Family)	CUSTOM_ID	CHAR	15	客户号
	ACCOUNT_NUMBER	CHAR	8	账户号
	DATE	DATE	4	交易日期
	TRADE_NUMBER	CHAR	20	交易顺序号
	SYSTEM_CODE	CHAR	3	系统代码
	TRADE_CONTENT	VARCHAR	200	交易内容...

Z (冷Family)	CUSTOM_INFORMATION	CHARACTER	60	客户详细信息...



目录

1. HBase应用场景
2. HBase应用开发流程
3. 应用开发案例分析
4. 表设计指导
5. 常用开发接口示例
6. 应用开发实践

HBase表设计-总体原则

设计目标	设计原则	实现方法
提高吞吐量	预分 region ，使 region 分布均匀，提高并发	Rowkey 范围和分布已知，建议预分 region
提高写入性能	避免过多的热点 region	根据应用场景，可考虑将时间因素引入 Rowkey
提高查询性能	同时访问的数据尽量连续存储	同时读取的数据相邻存储 同时读取的数据存放在同一行 同时读取的数据存放在同一 cell
	查询频繁属性放在 Rowkey 前面部分	Rowkey 的设计在排序上必须与主要的查询条件契合
	离散度较好的属性作为 RowKey 组成部分	分析数据离散度特点以及查询场景，综合各种场景进行设计
	存储冗余信息，提高检索性能	使用二级索引，适应更多查询场景

HBase表设计-设计内容

设计内容通过不同维度，可以分为：

- **Table**设计（表粒度的设计）
- **RowKey**设计
- **Family**设计
- **Qualifier**设计

HBase表设计-Table设计

- 建表方法
 - 周期建表
 - 分表
- 预分**region**
 - 识别可能的热点**Key**区域
- **Family**属性
 - **TTL/Versions/Compression/Bloomfilter/Cache**
- 系统并发能力、数据清理能力
 - 利用集群的分布式能力（并发能力），提高集群业务吞吐量
 - 利用过期时间、版本个数设置等操作，让表能自动清除过期数据

HBase表设计-RowKey设计1

- 原则
 - 需要同时访问的数据，尽量连续存储。
- 访问效率
 - 分散写：提高并发度，但又不能过于分散。
 - 连续读：使用**scan**接口。

HBase表设计-RowKey设计2

- 属性值内容
 - 常用的查询场景属性
- 属性值顺序
 - 可枚举值较少的属性值放在前面
 - 访问权重高的属性值放在前面
- 时间属性
 - 循环**Key + TTL**
 - 周期建表

HBase表设计-RowKey设计3

- 属性值顺序
 - 可枚举值较少的属性值放在**RowKey**前面

数据来源—互联网数据：

查询所有http的数据：
Scan 2010-10,http,...

2010-10,http,cp001,s,guangdong,10
2010-10,http,cp002,s,guangdong,10
2010-10,http,cp003,s,guangdong,10
2010-10,http,cp004,s,guangdong,10
2010-10,http,cp005,s,guangdong,10
2010-10,rtsp,cp001,s,guangdong,10
2010-10,rtsp,cp002,s,guangdong,10
2010-10,rtsp,cp003,s,guangdong,10
2010-10,rtsp,cp004,s,guangdong,10
2010-10,rtsp,cp005,s,guangdong,10

查询cp001的所有数据：
scan 2010-10,http,cp001
scan 2010-10,rtsp,cp001

HBase表设计-RowKey设计3

- 属性值顺序
 - 可枚举值较少的属性值放在前面

数据来源—互联网数据：

查询cp001的所有数据：
scan 2010-10,cp001,...

2010-10,cp001,http,s,guangdong,10
2010-10,cp001,rtsp,s,guangdong,10
2010-10,cp002,http,s,guangdong,10
2010-10,cp002,rtsp,s,guangdong,10
2010-10,cp003,http,s,guangdong,10
2010-10,cp003,rtsp,s,guangdong,10
2010-10,cp004,http,s,guangdong,10
2010-10,cp004,rtsp,s,guangdong,10
2010-10,cp005,http,s,guangdong,10
2010-10,cp005,rtsp,s,guangdong,10

查询http的所有数据：
scan 2010-10,cp001,http,...
scan 2010-10,cp002,http,...
scan 2010-10,cp003,http,...
scan 2010-10,cp004,http,...
scan 2010-10,cp005,http,...

可枚举性值较少的属性放在**RowKey**前面在处理不同业务需求的时候，平衡性更强。

HBase表设计-RowKey设计4

- 属性值顺序
 - 访问权重高的属性值放在前面

业务需求是访问某一天的所有数据，**date**放在前面

```
2010-10-1,http,cp001,video001
2010-10-1,http,cp001,video002
2010-10-2,http,cp001,video001
2010-10-2,http,cp001,video002
```

业务需求是访问一段时间内某一个**URL**的数据，**URL**放在前面

```
2010-10,video001,s,guangdong,1
2010-10,video001,s,guangdong,2
2010-10,video002,s,guangdong,1
2010-10,video002,s,guangdong,2
```

HBase表设计-RowKey设计5

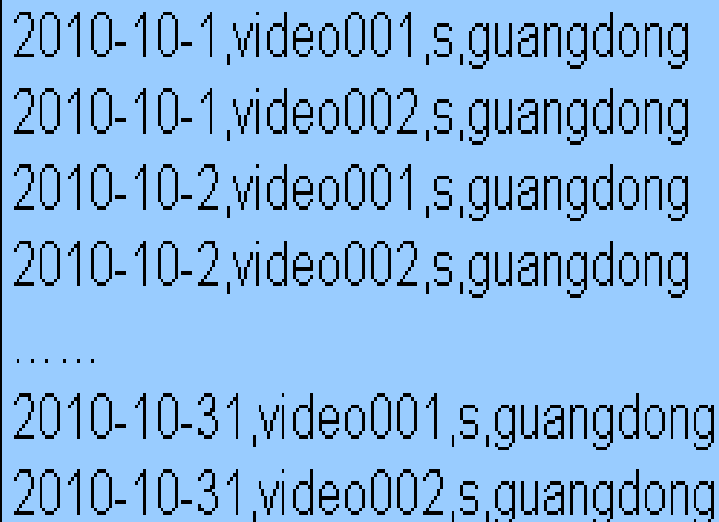
- 多业务场景共用**RowKey**会导致数据访问矛盾的问题，需考虑具体业务场景去解决，如下是常用的几个方法：
 - 折中法
 - 冗余法
 - HuaWei HBase—二级索引

HBase表设计-RowKey设计6

- 折中法

数据来源—互联网数据:

scan startrow = 2010-10-1,
endrow = 2010-10-1,z



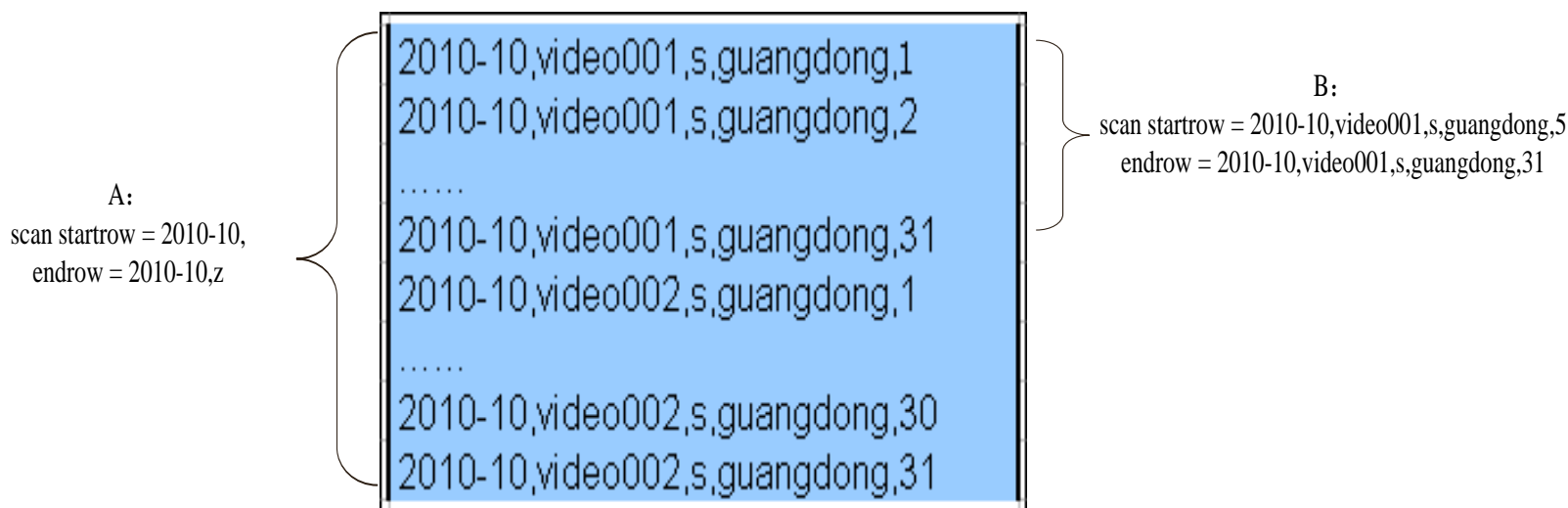
```
2010-10-1,video001,s,guangdong
2010-10-1,video002,s,guangdong
2010-10-2,video001,s,guangdong
2010-10-2,video002,s,guangdong
.....
2010-10-31,video001,s,guangdong
2010-10-31,video002,s,guangdong
```

- 支持**A**聚合某一段时间段所有**URL**的需求
- 无法满足**B**查询某个**URL**某一段时间段数据的需求

HBase表设计-RowKey设计6

- 折中法

数据来源—互联网数据:

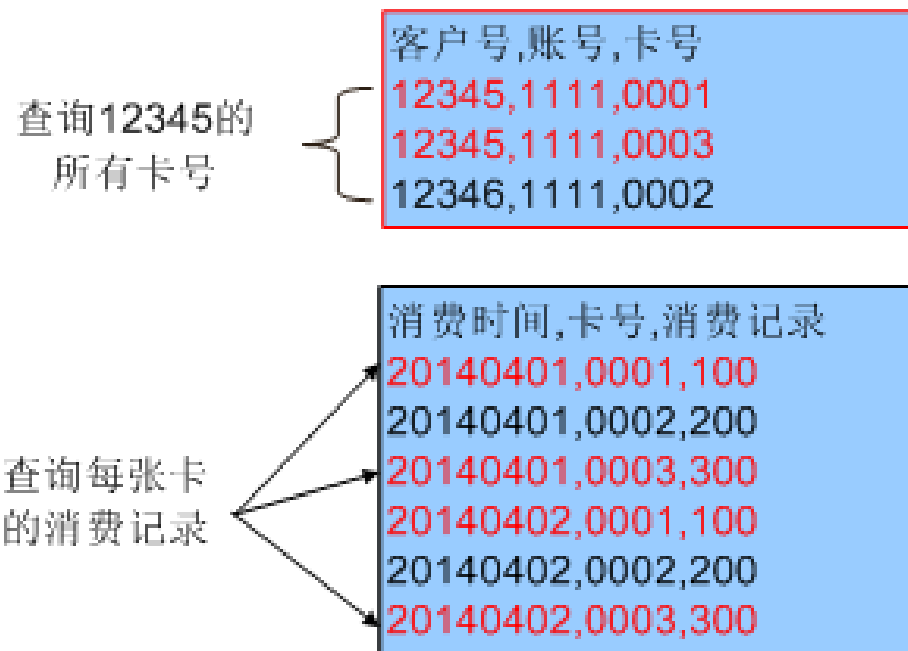


- 将时间属性分成两段，兼顾**A**和**B**的业务需求

HBase表设计-RowKey设计7

- 冗余法

数据来源—消费数据：查询一段时间某客户所有消费记录



- 在消费记录表中冗余存储客户号和账号，减少关联查询和RPC次数

HBase表设计-RowKey设计7

- 冗余法

数据来源—消费数据：查询一段时间某客户所有消费记录

查询
12345的
所有消费
记录

客户号,账号,卡号,消费时间,消费记录
20140401,12345,1111,0001,100
20140401,12345,1111,0001,100
20140401,12345,1111,0003,300
20140402,12345,1111,0003,300
20140402,12346,1111,0002,200
20140402,12346,1111,0002,200

- 在消费记录表中冗余存储客户号和账号，减少关联查询和RPC次数

HBase表设计-RowKey设计8

- 二级索引

数据来源—银行交易信息：对身份证件号码创建二级索引

基于二级索引的查询，先通过索引查找到**RowKey**，然后再根据**RowKey**提取实际的数据，达到快速查询的目标。

HBase表设计—Family设计

- 可枚举数量少扩展性弱的属性作为**Family**。
- 考虑因素
 - 分表 **vs.** 分**Family**。
 - 同时读取的数据存放在同一个**Family**。
 - 均衡各**Family**的数据量。
- 不同的**Family**设置不同的属性。

HBase表设计—Qualifier设计

- 不可枚举、数量多且扩展性强的属性作为**Qualifier**。
- 原则
 - 同时访问的数据存放到同一个**Cell**。
 - 列名尽量简短。



目录

1. HBase应用场景
2. HBase应用开发流程
3. 应用开发案例分析
4. 表设计指导
5. 常用开发接口示例
6. 应用开发实践

HBase常用接口示例1

- 常用**JAVA**接口清单

方法	说明
create()	通过 HBaseConfiguration 的 create 静态方法创建 Configuration 实例，用来处理配置的属性。可以调用其 set 方法来设置具体配置项的值。
createConnection()	HBase 通过 ConnectionFactory.createConnection(configuration) 方法创建 Connection 对象。
createTable(HTableDescriptor <i>desc</i>)	开始工作前，首先要做的是建表。常规用法是从 Connection 对象获得 Admin 对象，并调用 Admin 的 createTable 接口。
put(Put <i>put</i>)	Table 实例有 put 方法来向表中写入数据。
get(Get <i>get</i>)	Table 实例有 get 方法读表数据，返回值存储在 Result 对象中。
getScanner(Scan <i>scan</i>)	Table 实例有 getScanner 方法读取连续数据，返回值存储在 ResultScanner 对象中。

HBase常用接口示例2

- 创建**Configuration**实例及**kerberos**安全认证

1.通过调用

HBaseConfiguration类

方法要求配置文件放在
classpath路径下。

2.安全认证通过统一调用

LoginUtil类完成。

```
// 安全版本
```

```
Configuration conf = HBaseConfiguration.create();
String userdir = System.getProperty("user.dir") +
File.separator + "conf" + File.separator;
conf.addResource(new Path(userdir + "core-site.xml"));
conf.addResource(new Path(userdir + "hdfs-site.xml"));
conf.addResource(new Path(userdir + "HBase-site.xml"));
if (User.isHBaseSecurityEnabled(conf)) {
    String userdir = System.getProperty("user.dir") +
File.separator + "conf" + File.separator;
    userName = "HBaseDeveloper";
    userKeytabFile = userdir + "user.keytab";
    krb5File = userdir + "krb5.conf";
    LoginUtil.setJaasConf(ZOOKEEPER_DEFAULT_LOGIN_CONTEXT_NAME,
userName, userKeytabFile);

LoginUtil.setZookeeperServerPrincipal(ZOOKEEPER_SERVER_PRINCIPAL_KEY,
ZOOKEEPER_DEFAULT_SERVER_PRINCIPAL);
LoginUtil.Login(userName, userKeytabFile, krb5File, conf);
}
```

HBase常用接口示例3

- 创建表

通过Admin的
createTable方法来创建
一张表，指定表名、
Family名称。

1. 快速建表

2. 预分region建表（
RowKey范围和分布已知
）

示例：

// 表名和Family名称

```
HTableDescriptor htd = new HTableDescriptor(tableName);
```

```
HColumnDescriptor hcd = new HColumnDescriptor("info");
```

```
htd.addFamily(hcd);
```

```
admin = conn.getAdmin();
```

// 表的描述信息，指定表名、Family名称

```
HTableDescriptor tableDesc = new HTableDescriptor(tableName);
```

```
tableDesc.addFamily(new HColumnDescriptor(families[i]));
```

// 不预分region建表

```
admin.createTable(hcd);
```

// 预分region建表的两种方式：

// ①指定起止RowKey和region个数；此时的起始RowKey为第一个region的
endKey，结束key为最后一个region的startKey。

```
admin.createTable(hcd, Bytes.toBytes(10), Bytes.toBytes(800000), 30);
```

// ②指定RowKey数组，不包括第一个region的startKey和最后一个region的
endKey，因此region个数等于数组长度+1

// 例如以下语句创建的表包括4个region，各region的起止key分别为[a), [a,k),
[k,z), [z,)，可以看成左闭右开区间，a属于[a,k)这个region，k属于[k,z)这个region
，z属于[z,)这个region。

```
byte[][] keys = {Bytes.toBytes("a"), Bytes.toBytes("k"), Bytes.toBytes("z")};
```

```
admin.createTable(hcd, keys);
```

HBase常用接口示例4

- 开启压缩方式

示例：

```
columnDesc = new HColumnDescriptor(families[i]);  
// 设置前缀压缩，HBase提供了PREFIX、DIF、FAST_DIFF三种前缀压缩方法  
columnDesc.setDataBlockEncoding(DataBlockEncoding.FAST_DIFF);  
// 设置文件压缩，文件压缩通常需要安装压缩算法共享库，这里建议先通过  
// org.apache.hadoop.HBase.util.CompressionTest检查一下压缩算法可用性  
columnDesc.setCompressionType(Compression.Algorithm.SNAPPY);  
tableDesc.addFamily(columnDesc);
```

HBase常用接口示例5

- 写入数据put

1. 写入数据需要指定要写入的列（含Family名称和列名称）。
2. 多条记录写入建议用putlist接口。

示例：

// 表的名称为person

```
private TableName tableName = TableName.valueOf("person");
```

// Family的名称为privateInfo

```
byte[] familyName = Bytes.toBytes("privateInfo");
```

// FamilyprivateInfo中有两个列name和address

```
byte[][] qualifiers = { Bytes.toBytes("name"), Bytes.toBytes("address") };
```

// 实例化一个put对象, 012005000201为RowKey

```
Put put = new Put(Bytes.toBytes("012005000201"));
```

```
put.addColumn(familyName, qualifiers[0], Bytes.toBytes("Zhang San"));
```

```
put.addColumn(familyName, qualifiers[1], Bytes.toBytes("Shenzhen, Guangdong"));
```

// 多个put存放到一个List中

```
puts.add(put);
```

// 提交一次put数据请求

```
Table table = conn.getTable(tableName);  
table.put(puts);
```

HBase常用接口示例6

- 读取一行数据get

1. 读取一条数据需要实例化该表对应的**Table**对象。

2. 创建一个**Get**对象。也可以为**get**对象设定参数值，如**Family**的名称和列的名称。

3. 查询结果被存储在**Result**对象中。

示例：

```
Table table = null;
try {
    table = conn.getTable(tableName);
    Get get = new Get(rowKey.getBytes());
    // 设定Family名和列名
    get.addColumn(family, qualifier);
    // 设定Family名
    get.addFamily(family);
    Result result = table.get(get);
    for (KeyValue kv : result.raw()) {
        System.out.println(kv.getRow() + " " + (kv.getFamily()) + " "
            + (kv.getQualifier()) + " " + kv.getTimestamp() + " "
            + kv.getValue() + " ");
    }
} catch (IOException e) {
    // TODO
} finally {
    if (null != table) {
        try {
            table.close();
        } catch (IOException e) {
            // TODO
        }
    }
}
```

HBase常用接口示例7

- 读取多行数据scan

1.实例化该表对应的**Table**对象。

2.创建一个**Scan**对象，并针对查询条件设置**scan**的参数值。

3.查询结果的多行数据保存在**ResultScanner**对象，每行数据以**Result**对象形式存储，**Result**中存储了多个**KeyValue**对。

示例：

```
Table table = null;
ResultScanner rScanner = null;
try {
    table = conn.getTable(tableName);
    Scan scan = new Scan();
    scan.setStartRow(startRow);
    scan.setStopRow(stopRow);
    // 重要参数：每次RPC从服务端取回的记录数
    scan.setCaching(1000);
    rScanner = table.getScanner(scan);
    for (Result r = rScanner.next(); r != null; r = rScanner.next()) {
        for (KeyValue kv : r.raw()) {
            // TODO
        }
    }
} catch (IOException e) {
    // TODO
} finally {
    if (null != rScanner) {
        rScanner.close();
    }
    if (null != table) {
        try {
            table.close();
        } catch (IOException e) {
            // TODO
        }
    }
} }
```

HBase常用接口示例8

- 过滤器filter

示例：

```
Scan s = new Scan();  
// 前缀过滤器，查找RowKey以 "132"开头的行  
s.setFilter(new PrefixFilter(Bytes.toBytes("132")));  
// 查找符合条件的6行数据  
s.setFilter(new PageFilter(6));  
// 多个过滤器结合使用  
FilterList list = new FilterList(Operator.MUST_PASS_ALL);  
list.addFilter(new SingleColumnValueFilter(family,  
    qualifier,  
    CompareOp.EQUAL, value));  
list.addFilter(new PageFilter(6));  
s.setFilter(list);
```

- **HBase Filter**主要通过设置一些过滤条件，在查询（**Scan**）过程中进行**Row**级别的数据过滤。可以设置**RowKey**过滤条件，也可以设置列名或者列值的过滤条件。
- **HBase**中提供了多种过滤器，这些过滤器都在**org.apache.hadoop.HBase.filter**包中。单个过滤器的使用，首先需要创建一个**Scan**对象，然后设置该对象的**Filter**参数为过滤器对象。也可以多个过滤器条件配合使用，采用**FilterList**。具体**HBase**中提供了哪些**filter**，可以查询**API**清单。

HBase常用接口示例9

- 创建索引
- HBase通过 **IndexAdmin** 可以进行二级索引相关操作：创建索引、增加索引、删除索引等。
- 可以通过将索引信息和 **Family**信息添加到 **HTableDescriptor**里面，实现创建表的同时创建索引。
- 用**IndexSpecification**来描述索引的定义信息，包括索引名、索引列、索引列类型、索引列长度。

示例：

// new **IndexAdmin**对象进行索引管理操作

```
IndexAdmin IAdmin = new IndexAdmin(conf);
```

// 创建表的**Desc**对象和**Family**对象

```
String userTableName = "testAddIndex";
```

```
HTableDescriptor htd = new
```

```
HTableDescriptor(TableName.valueOf(userTableName));
```

```
HColumnDescriptor hcd = new HColumnDescriptor("cf");
```

```
htd.addFamily(hcd);
```

// 创建一个二级索引对象

```
TableIndices tableIndices = new TableIndices();
```

```
tableIndices.addIndex(iSpec);
```

```
IndexSpecification spec = new IndexSpecification("index1");
```

```
spec.addIndexColumn(hcd, "q1", ValueType.String, 10);
```

// 创建表和二级索引

```
IAdmin.createTable(htd);
```

HBase常用接口示例10

- 增加索引

示例：

```
// new IndexAdmin对象
```

```
HBaseAdmin admin = new IndexAdmin(conf);
```

```
String userTableName = "testAddIndex";
```

```
// new 索引对象
```

```
IndexSpecification spec = new IndexSpecification("index1");
```

```
spec.addIndexColumn(hcd, "q1", ValueType.String, 10);
```

```
// new 增加索引
```

```
admin.addIndex(TableName.valueOf(userTableName), spec);
```

- HBase通过 **IndexAdmin**可以进行二级索引相关操作：创建索引、增加索引、删除索引等。
- 如果表已经创建了，需要增加索引，则需要采用**IndexAdmin**的**addIndex**接口来实现。
- 用**IndexSpecification**来描述索引的定义信息，包括索引名、索引列、索引列类型、索引列长度。



思考题

1. **HBase**相比于传统关系型数据库有什么优势?
2. **HBase**应用开发的核心步骤?
3. **HBase**各项操作的权限如何控制?

总结

- 分析了几个应用场景的背景、存在的问题，引出**HBase**如何解决这类问题。
- 通过简洁完整的样例工程，详细地指导完成**HBase**应用开发所需要的基本步骤。
- 结合**HBase**的特点，给出了**HBase**的表设计原则，指导开发者充分利用**HBase**的优势。



习题

- 判断题

1. **HBase**预分**region**能解决数据分布不均匀场景。 (T or F)
2. 不同**Family**可以设置不同的**TTL**属性。 (T or F)
3. **Scan**时指定**StartKey**和**EndKey**能提升性能。 (T or F)

- 单选题

1. 通过**createTable**方法来创建一张表，必须传入的参数为（）？
 - A.表名
 - B.表名和列
 - C.表名和**Family**
 - D.可以为空

习题

- 多选题

1. HBase表设计中RowKey设计有哪些策略? ()

- A.可枚举属性值少的放在前面
- B.访问权重高的属性值放在前面
- C.采用部分字段的冗余存储减少关联查询的时间消耗
- D.RowKey长度越长越好

附录：HBase各操作所需权限汇总1/3

Action	Scope	Permission
CreateTable	namespace	CREATE
DeleteTable	table	ADMIN CREATE
TruncateTable	table	ADMIN CREATE
ModifyTable	table	ADMIN CREATE
AddColumn	table	ADMIN CREATE
ModifyColumn	table.family	ADMIN CREATE
DeleteColumn	table.family	ADMIN CREATE
EnableTable	table	ADMIN CREATE
DisableTable	table	ADMIN CREATE
Moveregion	table	ADMIN
Assignregion	table	ADMIN

附录：HBase各操作所需权限汇总2/3

RestoreSnapshot	如果用户SnapShot的owner，需要table的ADMIN，如果不是owner则需要GLOBAL ADMIN	
DeleteSnapshot	如果用户SnapShot的owner，可以直接执行，如果不是owner则需要GLOBAL ADMIN	
GetNamespaceDescriptor	namespace	ADMIN
ListNamespaceDescriptors	所有需要列出的namespace,ADMIN	ADMIN
TableFlush	table	ADMIN CREATE
Openregion	如果属于系统表则需要是超级管理员用户 如果属于普通表 需要 GLOBAL ADMIN	
Flush	table	ADMIN CREATE
Split	table	ADMIN
multisplit	table	ADMIN
Compact	table	ADMIN CREATE
GetClosestRowBefore	column family	READ
internalRead	column family	READ
Put	column family	WRITE
Delete	column family	WRITE
BatchMutate	column family	WRITE
CheckAndPut	column family	READ WRITE

附录：HBase各操作所需权限汇总3/3

IncrementColumnValue	column family	WRITE	
Append	column family	WRITE	
IncrementAfterRowLock	column family	WRITE	
ScannerOpen	需要是打开scanner的owner		
BulkLoadHFile	table	CREATE	
PrepareBulkLoad	table	CREATE	
CleanupBulkLoad	table	CREATE	
EndpointInvocation	CoprocessorEnvironment.getTableNames	EXEC	所有表中有实现EndpointObserver的coprocessor都需要EXEC权限 包括一些从客户端执行的命令，如MR任务，Phoenix sql line,hive on HBase,可以通过describe '表名' 查看实现的Endpoint
Close	GLOBAL	ADMIN	
StopregionServer	GLOBAL	ADMIN	
GetTableDescriptors	tableNames	ADMIN CREATE	
Merge	table	ADMIN	
RollWALWriterRequest	GLOBAL	ADMIN	

Thank you

www.huawei.com