2-

.

```
public class Config {
    private String b = "11";
        private Date date = new Date();
        private List<Date> birth = new ArrayList<Date>;
    private List<Test> tests = new ArrayList<Test>();
    private Map<String,Integer> maps = new HashMap<>();
    private Test test = new Test();
    private Provider provider  = new ProviderImpl();
    private List<Provider> providers = new ArrayList<>();
    private Map<String,Provider> absMap = new HashMap<>();
    private Map<String,Test> expMap = new HashMap<>();
    private List<String> priCol = new ArrayList<>();
        public String getNameSpace(){
                return "config";
        }
}
public class Test {
   private int a;

    public Test() {
    }

    public Test(int a) {
       this.a = a;
    }
    public void setA(int a) {
      this.a =a;
    }

    public int getA() {
        return this.a);
    }
}
public interface Provider {
}
public class ProviderImpl extends UniqueKey implements Provider  {
    private String a = "22";
    public ProviderImpl(){

    }
    public void setA(String a){
        this.a = a;
    }
    public String getA(){
        return this.a;
    }
}
```

getNameSpace

```
package com.fr.config.holder.factory;

import com.fr.config.holder.Conf;
import com.fr.config.holder.impl.ColConf;
import com.fr.config.holder.impl.MapConf;
import com.fr.config.holder.impl.ObjConf;
import com.fr.config.holder.impl.ObjectColConf;
import com.fr.config.holder.impl.ObjectMapConf;
import com.fr.config.holder.impl.SimConf;

import java.util.Collection;
import java.util.Map;
```

```
/**
 *
 * http://www.finedevelop.com/pages/viewpage.action?pageId=18648778
 */
public class Holders {
    /**
     * HolderTConfigHolder
     *
     * @param property
     * @param t
     * @param nameSpace
     * @param <T>
     * @return
     * @see Holders#simple(Object)
     */
    @Deprecated
    public static <T> Conf<T> simple(String property, T t, String nameSpace) {
        return new SimConf<T>(property, t).setNameSpace(nameSpace);
    }

    public static <T> Conf<T> simple(T t) {
        return new SimConf<T>(t);
    }

    /**
     * THolderTConfigHolder
     *
     * @param property
     * @param t
     * @param type
     * @param nameSpace
     * @param <T>
     * @return
     * @see Holders#obj(Object, Class)
     */
    @Deprecated
    public static <T> Conf<T> obj(String property, T t, Class<T> type, String nameSpace) {
        return new ObjConf<T>(property, t, type).setNameSpace(nameSpace);
    }

    public static <T> Conf<T> obj(T t, Class<T> type) {
        return new ObjConf<T>(t, type);
    }


    /**
     * KCollectionHolderConfigHolder
     *
     * @param property
     * @param collection
     * @param valueType
     * @param nameSpace
     * @param <K>
     * @return
     * @see Holders#objCollection(Collection, Class)
     */
    @Deprecated
    @SuppressWarnings("unchecked")
    public static <K> Conf<Collection<K>> collection(String property, Collection<K> collection, Class<K>
valueType, String nameSpace) {
        return new ColConf(property, collection, valueType).setNameSpace(nameSpace);
    }

    @SuppressWarnings("unchecked")
    public static <K> ColConf<Collection<K>> collection(Collection<K> collection, Class<K> valueType) {
        return new ColConf(collection, valueType);
    }

    /**
     * TCollectionHolderTConfigHolder
     *
```

```
     * @param property
     * @param collection
     * @param type
     * @param nameSpace
     * @param <T>
     * @return
     * @see Holders#collection(Collection, Class)
     */
    @Deprecated
    public static <T> ObjectColConf<Collection<T>> objCollection(String property, Collection<T> collection,
Class<T> type, String nameSpace) {
        return new ObjectColConf<Collection<T>>(property, collection, type).setNameSpace(nameSpace);
    }

    public static <T> ObjectColConf<Collection<T>> objCollection(Collection<T> collection, Class<T> type) {
        return new ObjectColConf<Collection<T>>(collection, type);
    }

    public static <T> ObjectColConf<Collection<T>> objCollection(Collection<T> collection, Class<T> type,
boolean order) {
        return new ObjectColConf<Collection<T>>(collection, type, order);
    }

    /**
     * keyvalueMapHolderK,VConfigHolder,keyTypevalueType
     *
     * @param property
     * @param map
     * @param keyType
     * @param valueType
     * @param nameSpace
     * @param <K>
     * @param <V>
     * @return
     * @see Holders#map(Map, Class, Class)
     */
    @Deprecated
    @SuppressWarnings("unchecked")
    public static <K, V> MapConf<Map<K, V>> map(String property, Map<K, V> map, Class<K> keyType, Class<V>
valueType, String nameSpace) {
        return new MapConf(property, map, keyType, valueType).setNameSpace(nameSpace);
    }

    @SuppressWarnings("unchecked")
    public static <K, V> MapConf<Map<K, V>> map(Map<K, V> map, Class<K> keyType, Class<V> valueType) {
        return new MapConf(map, keyType, valueType);
    }

    /**
     * valueVMapHolderVConfigHolder
     * K KKString,KValueWriterKString
     *  KValueReaderStringKValueReader.registerReader,  ValueWriter.registerWriter
     * keyTypemapkeyStringValueType
     *
     * @param property
     * @param map
     * @param keyType
     * @param valueType
     * @param nameSpace
     * @param <K>
     * @param <V>
     * @return
     * @see Holders#objMap(Map, Class, Class)
     */
    @Deprecated
    public static <K, V> ObjectMapConf<Map<K, V>> objMap(String property, Map<K, V> map, Class<K> keyType,
Class<V> valueType, String nameSpace) {
        return new ObjectMapConf<Map<K, V>>(property, map, keyType, valueType).setNameSpace(nameSpace);
    }

    public static <K, V> ObjectMapConf<Map<K, V>> objMap(Map<K, V> map, Class<K> keyType, Class<V> valueType) {
```

```
        return new ObjectMapConf<Map<K, V>>(map, keyType, valueType);
    }

    public static <K, V> ObjectMapConf<Map<K, V>> objMap(Map<K, V> map, Class<K> keyType, Class<V> valueType,
boolean ordered) {
        return new ObjectMapConf<Map<K, V>>(map, keyType, valueType, ordered);
    }
}
```

1.bConfDateConf

Conf<String> b = Holders.simple("11");

Conf<Date> date = Holders.simple(new Date());

Holders.simpleStringjdk,simple

2.testConfproviderconf

Conf<Provider> provider =Holders.obj(new ProviderImple(),Provider.class); ProviderImpl

Conf<Test> test = Holders.obj(new Test(),Test.class);TestConfTestConfigtestTestHolders.simpleUniquekey

```
public class Test extends UniqueKey {
    private Conf<Integer> a = Holders.simple(1);

    public Test() {
    }

    public Test(int a) {
        this.a.set(a);
    }

    public void setA(int a) {
        this.a.set(a);
    }

    public int getA() {
        return this.a.get();
    }
}
public class ProviderImpl extends UniqueKey implements Provider  {
    private Conf<String> a = Holders.simple("22");
    public ProviderImpl(){

    }
    public void setA(String a){
        this.a.set(a);
    }
    public String getA(){
        return this.a.get();
    }
}
```

3.mapsconfMapsimple

MapConf<Map<String, Integer>> maps = Holders.map(new HashMap(),String.class,Integer.class);Mapkeyvaluesimple

4.private List<Test> tests = new ArrayList<Test>(); private List<Provider> providers = new ArrayList<>();holder

```
ObjectColConf<Collection<Test>> tests = Holders.objCollection(new ArrayList<Test>(),Test.class));//Test

ObjectColConf<Collection<Provider>> providers=Holders.objCollection(new ArrayList<Provider>(),Provider.class);
//Provider
```

5. private Map<String,Provider> absMap = new HashMap<>(); private Map<String,Test> expMap = new HashMap<>(); holdervalue

ObjectMapConf<Map<String,Provider>> absMap = Holders.objMap(new HashMap<String,Provider>(),String.class,Provider.class);

ObjectMapConf<Map<String,Test>> expMap = Holders.objMap(new HashMap<String,Test>(),String.class,Test.class);

absMapputConfig

void putAbsMapObject key,Object value{

this.absMap.put(key,value);

}

ObjectMapConfObjectColConf map

.XmlHolders

XmlXmlablexmlable

XmlableXmlableXmlable

```
package com.fr.config.holder.factory;


import com.fr.config.holder.Conf;
import com.fr.config.holder.impl.xml.XmlColConf;
import com.fr.config.holder.impl.xml.XmlConf;
import com.fr.config.holder.impl.xml.XmlMapConf;
import com.fr.stable.xml.XMLable;

import java.util.Collection;
import java.util.Map;

/**
 * http://www.finedevelop.com/pages/viewpage.action?pageId=18648778
 * xml
 * xml
 */
@Deprecated
public class XmlHolders {
    //XmlablereadxmlwriteXml
    // config,property  arr config.arr XXXXXX   xxxxmlable
    @Deprecated
    @SuppressWarnings("unchecked")
    public static <T extends XMLable> Conf<T> obj(String property, T t, Class<T> clazz, String nameSpace) {
        return new XmlConf(property, t, clazz).setNameSpace(nameSpace);
    }

    @SuppressWarnings("unchecked")
    public static <T extends XMLable> Conf<T> obj(T t, Class<T> clazz) {
        return new XmlConf(t, clazz);
    }

    //XmlablereadxmlwriteXml
    // config,property  arr id
    // config.arr.id XXXXXX   xxxxmlable
    @Deprecated
    public static <T extends XMLable> XmlColConf<Collection<T>> collection(String property, Collection<T> t,
Class<T> clazz, String nameSpace) {
        return new XmlColConf<Collection<T>>(property, t, clazz).setNameSpace(nameSpace);
    }

    public static <T extends XMLable> XmlColConf<Collection<T>> collection(Collection<T> t, Class<T> clazz) {
        return new XmlColConf<Collection<T>>(t, clazz);
    }

    //XmlableMapreadxmlwriteXml
    //// config,property  arr
    // config.arr.key XXXXXX   xxxxmlable
    //K KKString,KValueWriterKString
    // KValueReaderStringK
    @Deprecated
    public static <K, V extends XMLable> XmlMapConf<Map<K, V>> map(String property, Map<K, V> t, Class keyType,
Class<V> clazz, String nameSpace) {
        return new XmlMapConf<Map<K, V>>(property, t, keyType, clazz).setNameSpace(nameSpace);
    }

    public static <K, V extends XMLable> XmlMapConf<Map<K, V>> map(Map<K, V> t, Class keyType, Class<V> clazz) {
        return new XmlMapConf<Map<K, V>>(t, keyType, clazz);
    }
}
```