目录

数据迁移实践

KS3为用户提供多种数据迁移方式，同时也提供KS3之间的数据迁移、从第三方数据源迁移至KS3的完整方案。

迁移方式

本地数据迁移

本地小规模数据迁移可选择：

- 命令行工具
- 可视化工具

若数据规模较大，建议使用KS3-import数据迁移工具。

KS3之间数据迁移

可使用命令行工具、KS3-import数据迁移工具均可实现KS3之间的数据迁移，使用KS3的在线迁移服务可以实现：

- 同一region下，不同Bucket之间的数据迁移
- 不同region下，Bucket之间的数据迁移
- 不同KS3账号之间的数据迁移

KS3跨区域数据复制

可使用跨区域复制来实现不同Region之前的数据复制。

第三方数据源迁移至KS3

- KS3-import数据迁移工具目前支持将阿里云OSS、百度云、七牛云和AWS S3的数据迁移到KS3，修改配置文件的参数项即可轻松实现。 KS3-import数据迁移工具还支持增量上传、断点续传、流量控制、并发上传、进度查询和存储类型选择。
- KS3还为开发者提供支持各种语言的API和SDK，为用户应用提供更多的场景支持。

迁移方案

无缝数据迁移方案

当用户的服务已经在自己建立的源站或者在其他云产品上运行，需要迁移到KS3上，但是又不能停止服务，此时可利用重定向回源回源功能实现。

1. 将第三方存储T0之前的数据全量迁移至KS3
2. 数据迁移完成后，在KS3上配置重定向回源回源到第三方存储
3. 用户业务切换到KS3，记此时时间为T1
4. 用户将T0至T1时间段内新增或改动的文件，使用KS3-import或其他工具上传到KS3
5. 删除第三方存储

全量迁移方式

全量迁移有三种方式可供选择：

- 邮寄服务器：将服务器寄送到第三方存储，通过内网告诉网络，下载数据到服务器后，回寄到KS3机房，将数据上传至KS3
- 互联网带宽：购买第三方存储的云服务器，通过内网告诉网络下载数据的同时，利用互联网带宽上传到KS3
- 专线：购买第三方存储到KS3的专线，购买第三方存储或金山云的云服务器，通过内网高速网络下载数据的同时，利用专线上传到KS3

不使用主账号

如果用户担心主账户的AKSK泄露，最安全的策略是不使用主账号的AK/SK来访问KS3，而是创建一个子用户，并赋予这个子用户足够的权限，使用这个子用户的AK/SK这样可以规避AccessKey或者密码泄露导致的问题。具体操作步骤如下：

1. 进入控制台，点击左侧导航栏下方的**访问控制**，点击左侧导航栏的**人员管理** > **子用户**界面。
2. 点击子用户页面上方**新建用户**按钮，按照需求填写**用户登录信息**以及其他必填字段，勾选 **编程访问** （启用AccessKeyID和AccessKeySecret，支持通过API或其他开发工具访问），**控制台密码登录**勾选后会生成子账号的账户密码信息。
3. 点击确定之后，生成该账号的AccessKey，一定要在这一步通过**复制**或者下载CSV文件 **保存** 下来AK/SK,用于后续的访问。
4. 返回**子用户**界面，找到新创建的账号。创建完成之后，该子账号还是没有任何权限，点击右边的 **添加权限** ，给该账号赋予KS3FullAccess系统权限，添加完毕之后点击**确定**按钮。
5. 使用刚刚创建的IAM子用户的AK/SK,生成签名信息，构造对应的访问URL，就可以访问KS3的相关资源了。
6. 进入子用户控制台登录界面，输入主账号ID或用户名，输入IAM用户名，输入密码，点击登录。

读写权限分离

当用户要使用应用服务器对外服务的时候，KS3可以作为后端静态资源的存储。这个时候应用服务器只获取KS3的读权限，不对KS3进行写入和设置。可以设置读写权限分离，给应用服务器一个只读权限的用户即可。

1. 进入控制台，点击进入左侧导航栏下方的 **访问控制**，再点击左侧导航栏中**人员管理** > **子用户** ，进入子用户界面。

2. 点击 **新建用户**按钮 ，在访问方式中勾选 **编程访问** 。
3. 生成该账号的AccessKey，一定要在这一步保存下来用于后续的访问。
4. 返回**子用户**界面，找到刚刚创建的子用户。创建完成之后，该子账号还是没有任何权限，点击右边的"添加权限"，给该账号赋予KS3ReadOnlyAccess系统权限。
5. 使用刚刚创建的IAM子用户的AK/SK,生成签名信息，构造对应的访问URL，就可以访问KS3的相关资源了。
6. 也可以使用SDK，在SDK配置文件中写入刚刚创建的IAM子用户的AK/SK，就可以访问KS3的相关资源了。

存储空间权限分离

如果客户有两个部门，分别为研发部门和运维部门，客户在KS3已经创建了两个存储空间，rd_bucket和op_bucket,分别存放研发数据和运维数据，要求研发部门只能访问rd_bucket，运维部门只能访问op_bucket。 这时就需要创建两个子用户rd_user和op_user，分别授予rd_bucket和op_bucket的权限。具体的操作步骤如下：

1. 进入控制台，点击左侧导航栏下方的**访问控制**，再点击左侧导航栏的**人员管理**>**子用户** ，进入子用户界面，创建rd_user和op_user，并生成对应的AKSK。
2. 进入控制台，点击左侧导航栏下方的**访问控制**，再点击左侧导航栏的**权限管理**>**策略** ，进入策略管理界面。
3. 由于系统权限没有按照bucket粒度的权限，点击 **自定义策略** 标签页，点击**新建策略** 按钮，在**设置策略类型**中选择**策略语法** ，在**选择策略模板**中选择**复制系统模板**，选择**KS3FullAccess** 模板，输入策略名称“rdfullaccess”,修改策略语言如下：

```
{
"Version": "2015-11-01",
"Statement": [
{
    "Effect": "Allow",
    "Action": "ks3:ListBuckets",
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": "ks3:*",
    "Resource":[
        "krn:ksc:ks3:::rd_bucket",
        "krn:ksc:ks3:::rd_bucket/*"
    ]
 }
]
}
```

4. 同样，创建新策略“opfullaccess”。

```
{
"Version": "2015-11-01",
"Statement": [
{
    "Effect": "Allow",
    "Action": "ks3:ListBuckets",
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": "ks3:*",
    "Resource":[
        "krn:ksc:ks3:::op_bucket",
        "krn:ksc:ks3:::op_bucket/*"
    ]
 }
]
}
```

5. 返回用户管理界面，给rd_user和op_user分别赋予rdfullaccess和opfullaccess策略。

业务权限分离

如果用户的应用服务器，不仅仅只是读取KS3的文件，还需要写入文件时，那么只把只读权限(KS3ReadOnlyAccess)授予应用服务是不够的，授予KS3FullAccess 风险又比较大（KS3FullAccess包含删除和设置权限），那么就需要按照业务来进行权限的分配，具体的操作步骤如下：

1. 进入控制台，点击进入左侧导航栏下方的**访问控制**，再点击左侧导航栏中的**权限管理** > **策略** ，进入策略管理界面。
2. 点击 **自定义策略** 标签页，点击**新建策略** 按钮，在**设置策略类型**中选择 **产品功能/项目权限** ，选择配置服务类型为 **对象存储**，按照实际需求在操作列选择需要开启的权限，选择完成之后点击**创建策略**。
3. 进入控制台，点击进入左侧导航栏下方的**访问控制**，再点击左侧导航栏中的**人员管理** > **子用户** ，进入用户管理界面，点击**添加权限**将应用服务器所对应的子用户授予新创建好的策略。

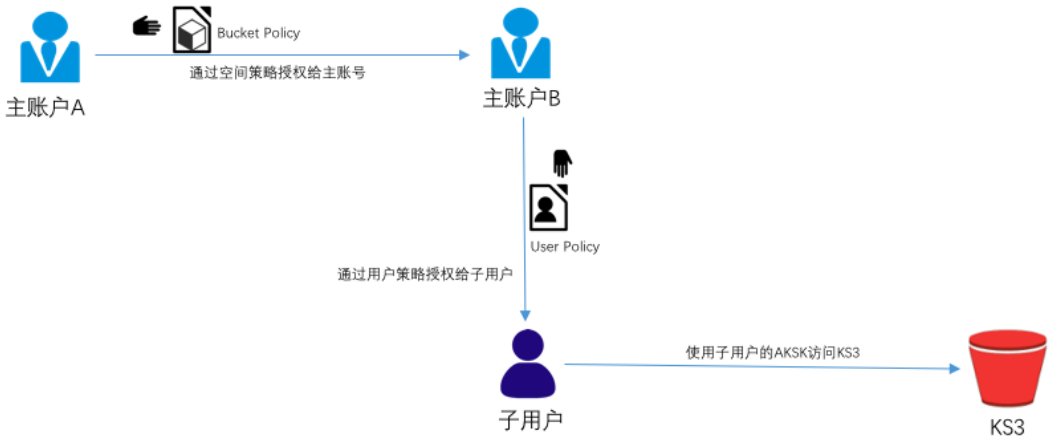
跨账户授权

假设主账户A想把自己的存储空间example_bucket的部分权限授权给主账号B下的研发人员。

那么要完成上述的功能，需要配置两条策略：

1. 需要将权限通过空间策略授予主账户B;
2. 主账户B再将这些权限通过用户策略委托给他的子用户rd_user。

原理图如下：



具体步骤如下：

1. 登录 KS3控制台，进入到A账户下需要授权的存储空间（Bucket），例如“exampleBucket”。点击**空间设置**页签 ，再点击右边**空间策略**页签，，点击**添加策略**，用户填想要授权的账号B，操作选择ListBucket、GetObject。设置完成后点击确定。具体参考如下：

```
{
    "Version":"2015-11-01",
    "Statement": [
    {
        "Sid":"1",
        "Effect":"Allow",
        "Principal":{krn:ksc:iam::AccountB_ID:root},
        "Action":[
            "ks3:ListBucket",
            "ks3:GetObject"
         ],
        "Resource":["krn:ksc:ks3:::example_bucket",
"krn:ksc:ks3:::example_bucket/*"]
    }
  ]
}
```

2. 进入到B账户下的 **策略管理** 页面，点击**添加策略**按钮新建一条自定义策略，具体参考如下：

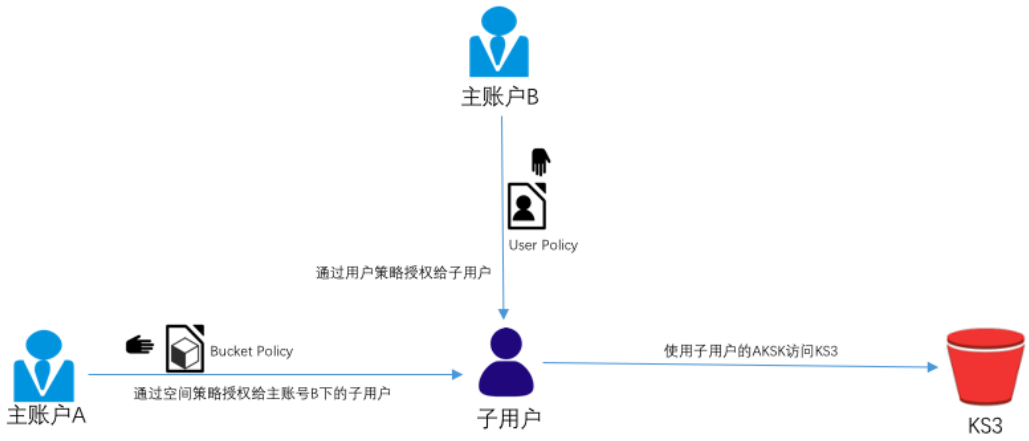
```
{
    "Version": "2015-11-01",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ks3:ListObject",
            "ks3:GetObject"
         ],
        "Resource":[
            "krn:ksc:ks3:::example_bucket",
            "krn:ksc:ks3:::example_bucket/*"
         ]
     }
   ]
}
```

3. 登录控制台，进入到B账户下的左侧导航栏下方的**访问控制** > **子用户** 页面，向rd_user授权第2步创建的策略。
4. 使用rd_user这个IAM子用户的AK/SK,生成签名信息，构造对应的访问URL，就可以访问example_bucket的相关资源了。

   注意 ：

   - 账户A还可以使用空间策略直接向账户B中的子用户授予权限。但是该子用户仍需要来自用户所属的。
   - 父账户(账户B)的权限，该子用户必须同时拥有来自资源拥有者和父账户的权限，便能够访问资源。

原理图如下：



临时授权访问

基本原理

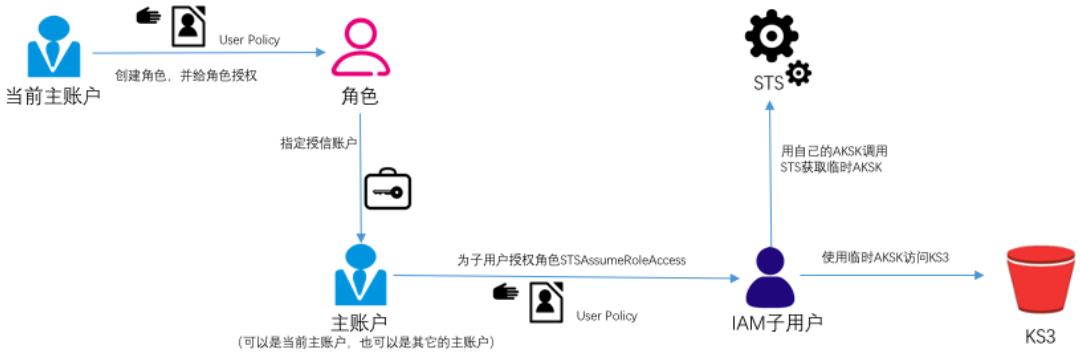
无论是主账号还是IAM子用户都是可以长期正常使用的，如果对应的AK/SK发生泄露之后如果无法及时解除权限的话会很危险。如果众多的客户端需要直接与KS3交互，可以让客户端获取一个临时的最小权限的凭证，并且该权限有一定的有效期即临时身份凭证，从而最大程度上保护用户数据安全。

考虑到如下的案例：

客户开发的App会分发给终端用户，终端用户的数据需要直接上传到KS3，如果将主账号或者IAM子用户的AK/SK嵌入到App都是非常危险的行为，那如何才能安全的授权给众多的App用户上传数据呢，以及如何保证多个用户之间存储的隔离。

类似这种需要临时访问的场景可以使用临时身份凭证来应对，其主要操作步骤为：

1. 在当前的主账号下创建一个角色，指定角色授信的主账户，并且给角色通过用户策略（user policy）授予一个最小权限。
2. 然后在授信主账号下创建一个子用户，并授权给子用户角色（让这个子用户扮演这个角色），子用户就可以通过STS服务获取临时AK/SK及token，去访问KS3了，为了提升安全性该token还有一定的过期时间。
   原理图如下：



具体的步骤如下：

创建策略

1. 安全的STS服务，需要设置两个自定义策略： a. 子用户策略（保证子用户有扮演角色的权限） b. 角色策略（保证角色对目标资源有对应权限）
   在控制台上操作，进入控制台，点击左侧导航栏下方中的**访问控制**，再点击左侧导航栏中**权限管理** > **策略**，进入策略管理界面，选择**自定义策略**页签> **新建策略**。

2. 在自定义策略中，输入策略名，此处假设为"test-policy"，然后选择 **可视化配置** > **添加策略语句**。

3. 侧边弹出**添加策略语句弹窗**，策略语句下拉框中选择产品服务为**临时身份验证**，在操作中选择**AssumeRole**，该权限表示可以扮演角色，点击**确定**按钮子用户策略创建完成，点击下方**创建策略**按钮。

4. 再次创建自定义策略，假设名称为“role-test-policy”，设置策略类型为 **策略语法** ，可以通过模板来简化创建策略过程，选择**复制系统模**

板，模板名称选择KS3ReadOnlyAccess，点击下一步。

5. 编辑相应的权限，如下载文件、查看bucket列表等，编辑完成之后点击**创建策略**按钮，角色策略创建完成。
   如下例子中对桶app_bucket及其内文件有get和ListBucket权限，分别对Action和Resource进行编辑，

```
{
    "Version": "2019-11-01",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ks3:GetObject",
            "ks3:ListBucket"
         ],
        "Resource":[
            "krn:ksc:ks3:::app_bucket",
            "krn:ksc:ks3:::app_bucket/*"
         ]
     }
   ]
}
```

创建子用户

主账户是不能扮演角色的，需要受信主账户通过用户权限（user policy）把扮演角色的权限授权给IAM子用户。

1. 在控制台上操作请点击进入 KS3控制台，点击左侧导航栏下方的**访问控制 > 子用户** ，进入子用户管理界面，点击**新建用户**按钮，输入子用户登录账户等信息，此处假设为test-subuser选中编程访问复选框为子用户创建AK和SK。

2. 在**子用户**页面选中刚创建的子用户，点击右侧**添加权限**来为子用户授权。

3. 在**选择权限**的下拉框中选择**自定义策略**，需要选中在**创建策略**步骤中创建的test-policy策略，这样子用户就具有了扮演角色的权限，但是现在还没有角色，因此接下来要创建角色。

创建角色

1. 请点击进入控制台,点击左侧导航栏下方的**访问控制**，点击左侧导航栏下方的**角色管理**，进入角色管理界面。

2. 点击**新建角色**按钮，选择金山云账号，可以选择当前的账号，也可以选择其它云账号。我们创建一个名为test-role的角色，设置载体信息为当前账号（也可以选择其他账号，载体信息即授信账号，比如你在账号1下创建角色授信给账号2则账号2的子用户就可以扮演账号1下的角色）。新创建的角色是没有任何权限的，点击**下一步**来设置角色权限，点击完成按钮。

3. 创建完角色之后，回到**角色管理**页面，点击刚刚创建的角色名，进入角色详情页。**需要记录下角色的KRN字段**，角色是没有任何权限的，点击**关联策略**页签，点击**添加授权**按钮，在策略下拉框中选择自定义策略，选择策略role-test-policy，点击确定按钮。这样角色test-role就有的相应的权限。

子用户扮演角色

主账户是不能扮演角色的，需要受信主账户通过用户权限（user policy）把扮演角色的权限授权给IAM子用户。

1. 再次进入 **访问控制 > 子用户** 页面，点击刚创建的子用户test-subuser右侧的**添加权限**。

2. 在选择权限下拉框中，选择为子用户创建的策略test-policy，这样子用户就具有扮演角色的权限，而且所扮演的角色对目标资源有相应的权限。

3. 同样此时，该子用户仅具有了扮演角色的权限，但还没有关联具体的角色，因此还需要添加权限对应的资源，即角色ID，在导航栏左侧**策略**页面中点击选择所创建的"test-policy"，在**策略内容**中的Resource字段值中填入角色的KRN。

```
{
    "Version": "2015-11-01",
    "Statement": [
    {
    "Sid": "Stmt15253327178180",
    "Effect": "Allow",
    "Action": ["sts:AssumeRole"],
    "Resource": [" krn:ksc:iam::Account_ID:role/test-role"]
    }
    ]
}
```

使用STS授权访问

1. 使用子用户test-subuser的AK/SK 去调用STS服务获取临时权限。详见文档获取角色的临时身份。

```
http://sts.cn-beijing-6.api.ksyun.com/?Action=AssumeRole
&Version=2019-11-01
&RoleSessionName=Bob
&RoleKrn=krn:ksc:iam::Account_ID:role/test-role
&AUTHPARAMS
```

返回的内容如下：

```
<AssumeRoleResponse>
  <AssumeRoleResult>
      <Credentials>
        <SecretAccessKey>wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY</SecretAccessKey>
        <Expiration>2017-07-15T23:28:33.359Z</Expiration>
        <AccessKeyId>AKIAIOSFODNN7EXAMPLE</AccessKeyId>
        <SecurityToken>V1xxxxxxxxxxxx</SecurityToken>
      </Credentials>
      <AssumedRoleUser>
         <Krn>krn:ksc:sts::123456789012:assumed-role/demo/Bob</Krn>
         <AssumedRoleId>ARO123EXAMPLE123:Bob</AssumedRoleId>
```

```
      </AssumedRoleUser>
   </AssumeRoleResult>
   <ResponseMetadata>
     <RequestId>c6104cbe-af31-11e0-8154-cbc7ccf896c7</RequestId>
   </ResponseMetadata>
</AssumeRoleResponse>
```

2. 从返回的XML结果中找到临时权限：AccessKeyId、SecretAccessKey及SecurityToken，然后通过KS3 API或JAVA SDK去访问相应的资源。

GO示例：

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "github.com/KscSDK/ksc-sdk-go/ksc"
    "github.com/KscSDK/ksc-sdk-go/ksc/utils"
    "github.com/KscSDK/ksc-sdk-go/service/sts"
    "github.com/ks3sdklib/aws-sdk-go/aws"
    "github.com/ks3sdklib/aws-sdk-go/aws/credentials"
    "github.com/ks3sdklib/aws-sdk-go/service/s3"
    "os"
)

func main() {
    ak := "<AccessKeyID>"
    sk := "<AccessKeySecret>"
    stsRegion := "cn-beijing-6"
    assumeRole := assumeRoleRequest(ak, sk, stsRegion) //调用ksc-go-sdk的sts服务
    var data Response
    _ = json.Unmarshal(assumeRole, &data)
    stsAk := data.AssumeRoleResult.Credentials.AccessKeyId      // sts ak
    stsSk := data.AssumeRoleResult.Credentials.SecretAccessKey  // sts sk
    stsToken := data.AssumeRoleResult.Credentials.SecurityToken // sts token

    //create ks3 client with sts ak,sk,token
    credentials := credentials.NewStaticCredentials(stsAk, stsSk, stsToken)
    client := s3.New(&aws.Config{
        Region:           "BEIJING",
        Credentials:      credentials,
        Endpoint:         "ks3-cn-beijing.ksyuncs.com", //ks3地址
        DisableSSL:       true,                         //是否禁用https
        LogLevel:         1,                            //是否开启日志,0为关闭日志，1为开启日志 n
        S3ForcePathStyle: false,                        //是否强制使用path style方式访问
        LogHTTPBody:      true,                         //是否把HTTP请求body打入日志
        Logger:           os.Stdout,                    //打日志的位置
    })

    // example for uploading file
    params := &s3.PutObjectInput{
        Bucket:      aws.String("BucketName"),              // bucket名称
        Key:         aws.String("ObjectKey"),               // object key
        ACL:         aws.String("public-read"),             //权限，支持private(私有)，public-read(公开读)
        Body:        bytes.NewReader([]byte("PAYLOAD")),    //要上传的内容
        ContentType: aws.String("application/ocet-stream"), //设置content-type
        Metadata: map[string]*string{
            //"Key": aws.String("MetadataValue"), // 设置用户元数据
            // More values...
        },
    }
    resp, err := client.PutObject(params)
    if err != nil {
        panic(err)
    }
    fmt.Println(resp)
    //获取新的文件名
    fmt.Println(*resp.NewFileName)

}

func assumeRoleRequest(ak string, sk string, region string) []byte { //调用ksc-go-sdk的sts服务
    svc := sts.SdkNew(ksc.NewClient(ak, sk /*,true*/), &ksc.Config{Region: &region}, &utils.UrlInfo{ //debug模式的话,打开true开关
        UseSSL:      true,
        UseInternal: false,
    })
    var resp *map[string]interface{}
    var err error

    //set your assumeRole here
    assumeRoleInput := make(map[string]interface{})
    assumeRoleInput["RoleKrn"] = "<YourRoleKrn>"
    assumeRoleInput["RoleSessionName"] = "Bob"
    assumeRoleInput["DurationSeconds"] = "3600"
    //assumeRole["Policy"] = ""

    resp, err = svc.AssumeRole(&assumeRoleInput)
    if err != nil {
        fmt.Println("error:", err.Error())
        return nil
    }
    var str []byte
    if resp != nil {
        str, _ = json.Marshal(&resp)
        //fmt.Printf("%+v\n", string(str))
    }
    return str
}

type (
```

```
    Response struct {
        AssumeRoleResult struct {
            Credentials struct {
                SecretAccessKey string `json:"SecretAccessKey"`
                Expiration      string `json:"Expiration"`
                AccessKeyId     string `json:"AccessKeyId"`
                SecurityToken   string `json:"SecurityToken"`
            } `json:"Credentials"`
            AssumedRoleUser struct {
                Krn           string `json:"Krn"`
                AssumedRoleId string `json:"AssumedRoleId"`
            } `json:"AssumedRoleUser"`
        } `json:AssumeRoleResult`
        RequestId string `json:"RequestId"`
    }
)
```

跨域资源访问实践

使用场景

浏览器的同源策略限制从一个源加载的文档或脚本与来自另一个源的资源进行交互的方式，是用于隔离潜在恶意文件的关键安全机制。同协议、同域名（或IP）、以及同端口视为同一个域，一个域内的脚本仅仅具有本域内的权限，即本域脚本只能读写本域内的资源，而无法访问其它域的资源。

跨域资源共享（Cross-Origin Resource Sharing，简称 CORS），是 HTML5 提供的标准跨域解决方案。CORS允许WEB端的应用程序访问不属于本域的资源。开发者可以利用KS3提供的接口控制跨域访问的各种权限，开发灵活的WEB应用程序。

使用规则

如果需要使用浏览器跨域访问资源，可以登录 KS3控制台，设置存储桶的相关CORS解决跨域访问问题。

控制台

登录KS3控制台,选择需要进行CORS配置的bucket,选择**空间设置** > **CORS配置** > **添加规则**，添加一条自定义CORS策略。

配置

您可以配置CORS的以下各项参数

参数设置说明：

- **Allow Origin**

  设定允许跨境请求的来源，多个域名以英文逗号分隔。

- **Method**

  设定允许的跨域请求方法，包含：GET、POST、DELETE、PUT、HEAD

- **Allow Header**

  设定允许的跨域请求header，多个Header以英文逗号分隔。

- **Exposed Header**

  设定允许从应用程序进行访问的响应头部。

- **Cache Time**

  设定浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间。

  注： KS3会根据跨域请求匹配相对应的bucket下的CORS规则，根据规则设定的权限进行检查，并依次匹配每一条规则，根据规则的设定来允许请求并返回相对应的header，如想了解CROS相关操作请参考CORS配置。

防盗链实践

KS3提供的防盗链通过黑白名单的方式控制，其中黑名单用于添加禁止访问的来源域名，白名单用于添加允许访问的来源域名。防盗链功能可手动设置关闭。

录入域名的时候需要注意以下规则：

- 域名之间用逗号(,)分开，不需要写 http://
- 支持域名前使用通配符 *，可用于指代当前域名下的多级子域名

由于有些合法的请求是不会带referer来源头部的,所以有时候不能拒绝来源头部（referer）为空的请求，在KS3中，我们还提供了手动设置referer的选项。

操作步骤：

1. 登录 KS3控制台,进入想要设置的空间里，防盗链功能位于**空间设置**菜单下

2. 设置黑白名单：
   - 黑名单设置：

   - 白名单设置：

3. 手动设置referer

删除风险控制

数据删除方式

通过以下五种操作方式可以删除您的数据。

1. 通过KS3控制台

操作步骤参见文档删除文件。

2. 命令行工具util

操作步骤参见文档KS3Util命令行工具。

3. 可视化工具explorer

操作步骤参见文档GUI图形化界面工具。

4. SDK

操作步骤参见文档SDK概览。

5. 生命周期管理

若客户在生命周期规则中配置了定期删除文件，KS3会根据生命周期的配置定期删除符合条件的文件。

操作步骤参见文档生命周期管理。

删除风险

以下情况均会导致您的数据被误删，请谨慎操作。

- 在使用上述五种删除方式时操作失误，导致误删。
- 没有正确地配置Bucket访问权限，导致文件被他人恶意删除。

如何避免误删

为了避免数据被误删，需要配置正确的访问权限，配置时注意遵循以下原则：

- 不使用主账号访问KS3。
- 读写分离，对于只需要读数据的业务，只使用具有读权限的子账号。
- 针对不同的业务，授予“够用且最小的范围”的KS3权限。如对子账号设置用户策略(User Policy)，限制用户调用DeleteObject删除文件权限。
- 妥善保管数据访问的凭据，如账号密码、IAM子账号访问凭据。 配置权限详情参见文档权限概述。