Bundesverwaltungsamt

IsyFact-Standard

Detailkonzept Komponente Service

Version 1.12
01.03.2017

IsyFact Standards

Ansprechpartner:

Referat Z II 2
Bundesverwaltungsamt
E-Mail: isyfact@bva.bund.de
Internet: www.isyfact.de

Dokumentinformationen

Dokumenten-ID:	Detailkonzept_Komponente_Service.docx

Java Bibliothek / IT-System

Name	Art	Version
isy-serviceapi-core	Bibliothek	siehe isyfact-bom v1.3.6
isy-serviceapi-sst	Bibliothek	siehe isyfact-bom v1.3.6

Inhaltsverzeichnis

1. Einleitung

Entsprechend der Referenzarchitektur (siehe [IsyFactReferenzarchitektur]) stellen Anwendungen Services über die technische Komponente „Service“ zur Verfügung. Das vorliegende Dokument beschreibt die Aufgaben und die Architektur dieser technischen Komponente. Zu beachten ist, dass die Komponente „Service“ die Services einer Anwendung lediglich innerhalb der Plattform verfügbar macht. Um diese Services auch externen Anwendungen zur Verfügung zu stellen, müssen die Services durch ein so genanntes Service-Gateway exportiert werden. Dies ist jedoch nicht Inhalt des vorliegenden Dokuments, sondern wird im Dokument [ServiceGatewaySystementwurf] ausführlich beschrieben.

Die technischen Grundlagen der Service-Kommunikation innerhalb der Plattform sind im Dokument [ServicekommunikationKonzept] beschrieben.

Das Dokument gliedert sich wie folgt. In Abschnitt 2 wird kurz beschrieben, wie sich die Komponente “Service“ in die Gesamtarchitektur eines IT-Systems der Plattform einfügt. In Abschnitt 4 werden dann die Aufgaben der Komponente erläutert und Abschnitt 5 stellt den Standard-Aufbau der Komponente dar.

2. Überblick

Der Zugriff anderer Anwendungen auf die Funktionalität des Anwendungskerns eines IT-Systems innerhalb der Plattform erfolgt über die technische Komponente „Service“ (siehe Abbildung 1).

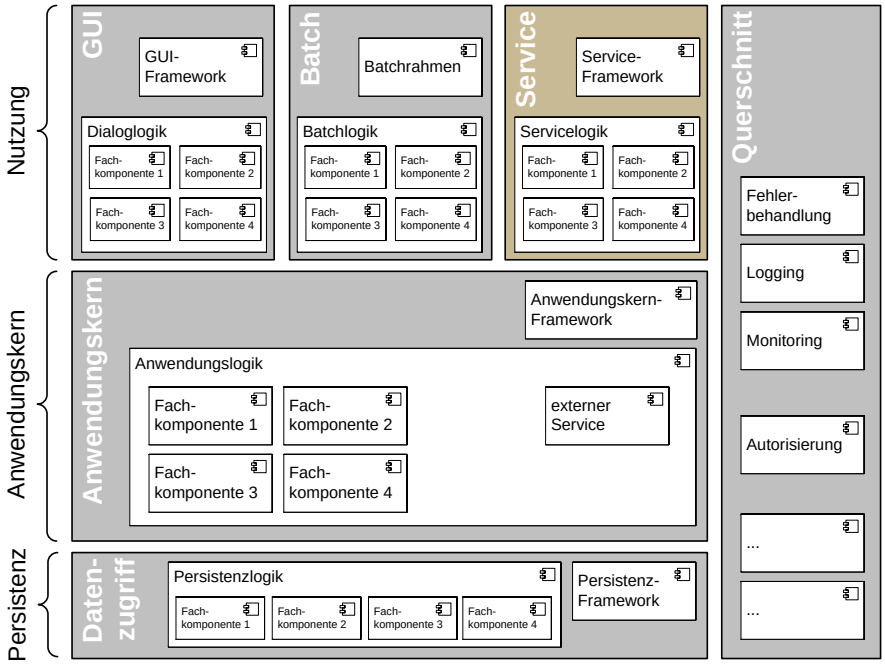

Abbildung 1: Referenzarchitektur eines IT-Systems

Intern besteht die Komponente „Service“ aus zwei Komponenten, dem Service-Framework und der Service-Logik:

Service-Framework: Das Service-Framework ist die Bezeichnung für die Technologie, mit der die Services des Anwendungskerns zur Verfügung gestellt werden. Hierfür wird das Framework Spring HTTP-Invoker verwendet. In der Regel wird ein extern angebotener Service noch durch zusätzliche Daten oder Logik ergänzt. Diese werden in der Komponente Service-Logik implementiert.

Service-Logik: Die Komponente Service-Logik enthält Daten und Funktionalität, die für die Bereitstellung des Services relevant sind. Im Normalfall ist der Funktionsumfang der Komponente Service-Logik viel geringer als der Funktionsumfang der genutzten Services im Anwendungskern. Dies liegt darin begründet, dass die Komponente Service-Logik die Funktionalität des Anwendungskerns nutzt, um den Dienst bereitzustellen. Die Kernfunktionalität des Dienstes ist also im Anwendungskern implementiert. Die Schnittstelle zwischen den Schichten „Service“ und „Anwendungslogik“ ist daher eine interne Service-Schnittstelle. Eine Kernaufgabe der Service-Logik ist die Umsetzung der internen Datenstrukturen und Exceptions des IT-

Systems auf Transportobjekte[1] und Exceptions der Service-Schnittstelle sowie die Autorisierung der Nutzung von angebotenen Services.

Da durch Spring HTTP-Invoker das Service-Framework vorgeben ist, muss für eine konkrete Anwendung lediglich die Service-Logik realisiert werden. Die folgenden Abschnitte beschränken sich daher auf die Beschreibung der Aufgaben und des Aufbaus der Service-Logik.

[1] Transportobjekte sind Java-Klassen, deren einziger Zweck es ist, Daten zu transportieren. Sie definieren lediglich eine Datenstruktur.

3. Namenskonventionen

Die Service-URLs der HTTP-Invoker-Schnittstellen müssen einheitlich aufgebaut sein. Hierzu werden im Dokument [Namenskonventionen] einheitliche Vorgaben definiert.

4. Aufgaben der Service-Logik

Die Service-Logik hat die folgenden Aufgaben:

Transformation: Bei einem Serviceaufruf müssen die Transportobjekte der Service-Schnittstelle in passende Objekte des Anwendungskerns umgewandelt werden. Außerdem muss das Ergebnis des Anwendungskerns wieder in ein Transportobjekt der Service-Schnittstelle umgewandelt werden.

Exception-Behandlung: Tritt bei der Verarbeitung eines Service-Aufrufs im Anwendungskern oder in der Komponente "Service" eine Exception auf, so muss diese Exception in eine Exception der Service-Schnittstelle umgewandelt werden.

Transaktion: Die Komponente „Service“ übernimmt in der Regel auch die vollständige Transaktionssteuerung. Das bedeutet, die Komponente öffnet und schließt Transaktionsklammern und führt im Fehlerfall ein Rollback durch.

Autorisierung: Für einen Service-Aufruf muss eine Anwendung überprüfen, ob der Aufrufer autorisiert ist die Service-Operation auszuführen.

Logging: Die Komponente muss die Korrelation-ID in den Logging-Kontext einfügen.

5. Aufbau der Service-Logik

Der Aufbau der Service-Logik ist in Abbildung 2 dargestellt.

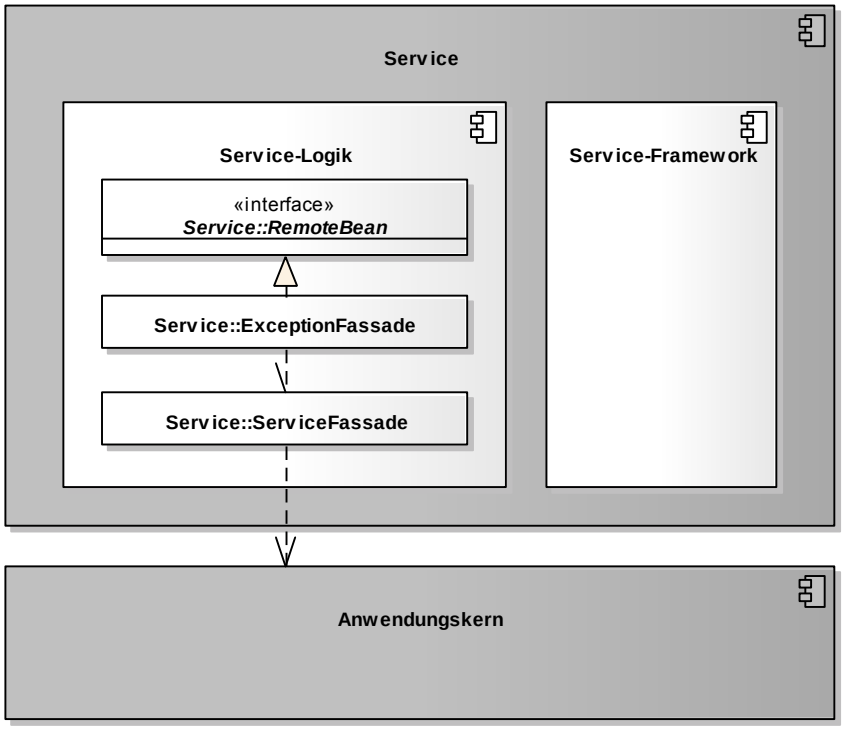

Abbildung 2: Aufbau Service-Logik

Eine Service-Schnittstelle wird durch eine Fachanwendung entsprechend der Referenzarchitektur in Form einer HTTP-Invoker-Schnittstelle angeboten. Zum Aufruf dieser HTTP-Invoker-Schnittstelle definiert die Fachanwendung eine JAR-Datei, die die `RemoteBean` definiert und alle direkt oder indirekt verwendeten Transportobjekte der `RemoteBean`.[2] Die JAR-Datei hat typischerweise den Namen `<Anwendungsname>-httpinvoker-sst-<servicename>-vx.y-z.jar`.

Jeder Methode der RemoteBean wird eine Instanz der Klasse `AufrufKontextTo` übergeben. Diese Klasse ist in der JAR-Datei `serviceapi.jar` definiert. Durch die Klasse werden jeder Methode der internen Service-Schnittstelle die Login-Daten (Benutzer, Behörde, Passwort), die Rollen und die Correlation-ID übergeben.

Im Wesentlichen besteht die Service-Logik aus zwei Klassen:

Exception-Fassade: Die Exception-Fassade ist verantwortlich für die Umwandlung der durch den Anwendungskern oder die Service-Logik geworfenen Exceptions in Exceptions der Service-Schnittstelle. Hierzu

[2] Als `RemoteBean` wird das Java-Interface bezeichnet, welches die Service-Schnittstelle definiert. Mit diesem Interface wird durch die passende Spring-Konfiguration in der Fachanwendung die HTTP-Invoker-Schnittstelle definiert.

implementiert die Exception-Fassade das Remote-Bean-Interface der Service-Schnittstelle und definiert in jeder Methode einen try-catch-Block, der alle Throwables abfängt und in Exceptions der Service-Schnittstelle umwandelt.

In Abbildung 3 ist ein Beispiel für eine Exception-Fassade einer Fachanwendung angegeben. Die Service-Operationen sind in diesem Fall die Methoden des Interfaces `BeispielRemoteBean`. Konkret handelt es sich lediglich um eine Service-Operation nämlich `holeBeispielAnfrage`. Die Service-Operation ist mit der Annotation `@StelltLoggingKontextBereit` versehen, die eine mit dem `AufrufKontext` übergebene Korrelation-ID[3] im Logging-Kontext registriert und diesen beim Verlassen der Methode wieder aufräumt. Es ist wichtig den Logging-Kontext zu setzen, bevor die Exception-Fassade aktiv wird. Die Implementierung der Service-Operation reicht den Methodenaufruf an die implementierende Klasse (`BeispielService`) weiter, fängt auftretende Fehler jedoch über einen try-catch-Block ab. Der try-catch-Block unterscheidet zwischen Exceptions der Datenbankzugriffsschicht (`HibernateException`) und allen anderen Exceptions (`Throwable`), um einen passenden Fehlertext in die Log-Dateien zu schreiben.

Service-Fassade: Die Service-Fassade übernimmt die restlichen Aufgaben der Service-Logik:

- Sie transformiert die Transportobjekte der Service-Schnittstelle in Objekte des Anwendungskerns und umgekehrt. Hierzu wird in der Regel die Java-Bibliothek Dozer verwendet. Falls die Transformation kompliziert ist, kann die Transformation auch vollständig ausprogrammiert werden. Hierbei sind Kosten und Nutzen genau abzuwägen.
- Sie öffnet und schließt die Transaktionsklammer und führt gegebenenfalls Rollbacks durch. Hierzu werden Annotationen verwendet (siehe [DatenzugriffDetailkonzept]).
- Sie führt gegebenenfalls die Autorisierung des Aufrufs aus. Hierzu verwendet sie den Berechtigungsmanager (siehe [SicherheitNutzerdok]).

In Abbildung 4 ist ein Beispiel für eine Service-Fassade angegeben. Die Implementierung der Service-Fassade erfolgt hier analog zur Implementierung der Exception-Fassade. Die nach außen angebotene Service-Operation (`holeBeispielAnfrage`) wird jedoch nicht 1:1 an die implementierende Klasse weitergeleitet, da sich die Parameter und der Rückgabewert des Aufrufs unterscheiden. Nach außen hin werden Transportobjekte angeboten. Intern arbeitet die Anwendung mit ihren eigenen Entitäten. Diese können sich von den nach außen hin angebotenen Transportobjekten unterscheiden, z. B. weil sie zusätzliche Attribute enthalten, einzelne Attribute anders benennen

[3] Falls im AufrufKontext keine Korrelation-ID vorhanden ist, so erzeugt die Annotation eine neue Korrelation-ID.

oder die Daten in irgendeiner Form anders repräsentieren als die Transportobjekte.

In der Service-Fassade erfolgt auch die Autorisierung eines Zugriffs auf eine Servicemethode. Voraussetzung für die Autorisierung ist die Auswertung des mitgelieferten AufrufKontextes über die Annotation `@StelltAufrufKontextBereit` an der Servicemethode. Anschließend kann über die Annotation `@Gesichert` die Berechtigung zum Zugriff auf die Methode geprüft werden. Hier werden alle benötigten Rechte des Aufrufers überprüft. Alternativ kann die Annotation `@Gesichert` auch an der Service-Klasse verwendet werden, wenn alle Methoden die gleiche Autorisierung erfordern. Die Annotationen sind Bestandteil der T-Komponente Sicherheit (siehe [SicherheitNutzerdok]).

Das Mapping im Beispiel wird durch dozer umgesetzt. Vor dem Aufruf werden die Parameter gemappt (Klasse `BeispielHolenAnfrageTo` auf Klasse `BeispielHolenAnfrage`), nach dem Aufruf der Rückgabewert (Klasse `BeispielHolenAntwort` auf Klasse `BeispielHolenAntwortTo`).

Die Komponente Service-Logik wird durch eine entsprechende Spring-Konfigurationsdatei „service.xml“ verschaltet (siehe Abbildung 5).

```
public class BeispielExceptionFassade
implements BeispielRemoteBean
{
  private static final Logger logger = ...

  private BeispielService service;

  ...

  @StelltLoggingKontextBereit
  public BeispielHolenAntwortTo
    holeBeispielAnfrage(
      AufrufKontextTo kontext,
      BeispielHolenAnfrageTo anfrage)
    throws BeispielTechnicalToException {

    try {
      return
        service.holeBeispielAnfrage(kontext,
          anfrage);
    } catch (HibernateException e) {
      logger.error("Fehler bei Transaktion", e);
      throw new BeispielTechnicalToException(
        ...);
    } catch (Throwable t) {
      logger.error("...", t);
      throw new BeispielTechnicalToException(
        ...);
    }
  }

  ...
}
```

Abbildung 3: Beispiel für eine Exception-Fassade

```
@Transactional(rollbackFor = Throwable.class,
  propagation = Propagation.REQUIRED)
public class BeispielServiceFassade{
  private static final Logger logger = ...

  private MapperIF dozer;
  private Beispiel beispiel;

  @StelltAufrufKontextBereit
  @Gesichert(Rechte.RECHT_ZUGRIFFBEISPIEL)
  public BeispielHolenAntwortTo
    holeBeispielAnfrage(
      AufrufKontextTo kontext,
      BeispielHolenAnfrageTo anfrage)
    throws VisaPortalException {

    try {
      BeispielHolenAnfrage anfrageAwk =
        (BeispielHolenAnfrage) dozer.map(anfrage,
          BeispielHolenAnfrage.class);

      BeispielHolenAntwort antwortAwk =
        beispiel.holeBeispielAnfrage(
          anfrageAwk);

      return (BeispielHolenAntwortTo)
        dozer.map(antwortAwk,
          BeispielHolenAntwortTo.class);
    } catch (MappingException e) {
      logger.error("...", e);
      throw new TechnicalException(...);
  }
  ...
}
```

Abbildung 4: Beispiel für eine Service-Fassade

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>
  <!-- ============================================
       Zusammenschaltung der T-Komponente Service
       ============================================ -->

  <bean id="beispielExceptionFassade"
    class="....BeispielExceptionFassade">

    <property name="service">
      <bean class="....BeispielServiceFassade">
        <property name="beispiel" ref="..."/>
        <property name="dozer" ref="..."/>
      </bean>
    </property>
  </bean>
</beans>
```

Abbildung 5: Beispiel für service.xml

6. Realisierung

Service-Gateway (d. h. Service-Consumer oder Service-Provider) und Fachanwendungen teilen sich die Java-Klassen der RemoteBean-Schnittstelle:

- Java-Interface der RemoteBean
- Transport-Objekte
- Exceptions

Diese Java-Klassen und -Interfaces existieren in allen Versionen der Schnittstelle und unterscheiden sich inhaltlich durch die in der neuen Version durchgeführten Änderungen.

Bei Transport-Objekten ist zu beachten, dass die UID stets 0 ist:

```
private static final long serialVersionUID = 0L;
```

Um die Klassen und Interfaces zu versionieren, wird die Versionsnummer als Teil des Paket-Pfads geführt. Dazu nachfolgend ein Beispiel:

```
<Organisation>.<Domäne>.<Anwendungsname>.service.<Servicename
>.httpinvoker.v29_1
```

Die Versionsnummer ist außerdem im Namen der JAR-Datei enthalten, die alle Klassen und Interfaces der HTTP-Invoker-Schnittstelle enthält:

```
<Anwendungsname>-httpinvoker-sst-<Servicename>-v29.1-
0.jar
```

Um die Implementierung einer Schnittstelle in zwei unterschiedlichen Versionen zu ermöglichen, wird die zweistellige Versionsnummer Teil der Maven-Artifact-ID. Um analog zu Komponenten eine dreistellige Versionsnummer zu erhalten, wird in der Maven-Datei eine einstellige Versionsnummer gesetzt. Dies würde im Beispiel wie folgt aussehen:

```
<project …>
    <modelVersion>4.0.0</modelVersion>

<groupId><Organisation>.<Domäne>.<Anwendungsname></grou
pId>
    <artifactId><Anwendungsname>-httpinvoker-sst-
<Servicename>-v29.1</artifactId>
    <version>0</version>
```

Diese einstellige Version kann bei kompatiblen Änderungen erhöht werden, ohne dass eine neue ArtifactID vergeben werden muss. Kompatible Änderungen sind beispielsweise neue Operationen in der Schnittstelle oder neue, optionale Attribute im Datenmodell.

Bei inkompatiblen Änderungen der Schnittstelle[4], wird die zweistellige Versionsnummer angepasst und damit ein neues Projekt erzeugt. Bei umfangreichen Anpassungen kann eine Erhöhung der ersten Stelle gerechtfertigt sein, dies ist mit den entsprechenden Gremien abzustimmen.

Die Schnittstelle wird in der Regel in einer älteren Java-Version kompiliert als die Anwendung kompiliert ist, um die Schnittstelle auch in älteren Anwendungen einsetzen zu können. Wenn die Schnittstelle jedoch ausschließlich von einem Service Gateway bzw. einer Fachanwendung genutzt wird, welche die aktuelle Java-Version einsetzen, kann auch die Schnittstelle in der aktuellen Java-Version kompiliert werden.

[4] Entfernen von Attributen oder Operationen, Hinzufügen von Pflichtfeldern, sonstige inkompatible Änderungen.

7. Nutzung

Zur Nutzung einer entfernten Schnittstelle bindet ein Anwendungssystem das erstellte Schnittstellen-JAR ein und initialisiert die RemoteBeans damit.

Das geschieht über die vom Spring Framework bereitgestellte Factory-Klasse `HttpInvokerProxyFactoryBean`, wie im folgenden Beispiel dargestellt. Auf dieser Bean können dann die entfernten Methoden aufgerufen werden.

```
    <!-- HttpInvoker-Proxy für Virenscanner -->
    <bean id="virenscanRemoteBean"
class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean"
>
        <property name="serviceUrl"
value="${virenscan.service.url}/VirenscanBean_v1_0"/>
        <property name="serviceInterface"

value="de.bund.bva.pliscommon.avscanner.service.virenscan.httpinvoker.v1_0.V
irenscanRemoteBean"/>
        <property name="httpInvokerRequestExecutor">
            <bean

class="de.bund.bva.pliscommon.serviceapi.core.httpinvoker.TimeoutWiederholun
gHttpInvokerRequestExecutor">
                <property name="anzahlWiederholungen"
value="${virenscan.service.wiederholungen}"/>
                <property name="timeout"
value="${virenscan.service.timeout}"/>
            </bean>
        </property>
    </bean>
```

Die FactoryBean erwartet eine Service-URL und ein Interface zur Initialisierung. Der Host-Teil der URL muss in jedem Fall in der betrieblichen Konfiguration der Anwendung zu finden sein. Das Interface ist im Schnittstellen-JAR verfügbar.

Die Nutzung des hier im Beispiel verwendeten `TimeoutWiederholungHttpInvokerRequestExecutor` ist optional. Dieser Executor bricht nach dem angegebenen Timeout die Anfrage ab und wiederholt sie bis zur maximalen angegebenen Wiederholungsanzahl.

Wenn die Anwendung IsyLogging [IsyLogging] nutzt, muss statt der Spring-eigenen Factory die erweiterte IsyHttpInvokerProxyFactoryBean genutzt werden. Sie versieht die RemoteBeans automatisch mit einem LogMethodInterceptor, der die Aufrufzeiten der ausgehenden Aufrufe misst und loggt. Die Konfiguration erfolgt in diesem Fall wie folgt:

```
    <bean id="konfigurationRemoteBean"
class="de.bund.bva.pliscommon.serviceapi.core.httpinvoker.IsyHttpInvokerProx
yFactoryBean">
        <property name="serviceUrl">

<value>${anwendung.xyz.service.url}/KonfigurationBean_v1_0</value>
        </property>
        <property name="serviceInterface"

value="de.bund.bva.yz.anwendung.service.konfiguration.httpinvoker.v1_0.Konfi
gurationRemoteBean"/>
        <property name="httpInvokerRequestExecutor"
ref="httpInvokerRequestExecutor"/>
        <property name="remoteSystemName"
value="${httpinvoker.anwendungxyz.name}"/>
```

```
    </bean>
```

Die erweiterte ProxyFactoryBean erwartet nur einen zusätzlichen Parameter `remoteSystemName`. Dieser wird genutzt, um einen sprechenden Systemnamen bei den Logausgaben auszugeben.

8. Quellenverzeichnis

[DatenzugriffDetailkonzept]

Detailkonzept Komponente Datenzugriff
10_Blaupausen\technische_Architektur\Detailkonzept_Komponente_Datenzugriff.pdf.

[IsyFactReferenzarchitektur]

IsyFact – Referenzarchitektur
00_Allgemein\IsyFact-Referenzarchitektur.pdf.

[IsyLogging]

IsyFact - Logging
20_Bausteine\Logging\Nutzungsvorgaben_Logging.pdf.

[Namenskonventionen]

IsyFact – Namenskonventionen
00_Allgemein\IsyFact-Namenskonventionen.pdf.

[ServiceGatewaySystementwurf]

Systemdokumentation Service-Gateway
20_Bausteine/Service-Gateway/Systemdokumentation_Service-Gateway.pdf .

[ServicekommunikationKonzept]

Grundlagen der Servicekommunikation innerhalb der Plattform
10_Blaupausen\Integrationsplattform\Grundlagen_interne_Servicekommunikation.pdf .

[SicherheitNutzerdok]

Nutzerdokumentation Sicherheit
20_Bausteine\Sicherheitskomponente\Nutzerdokumentation_Sicherheit.pdf .

9. Abbildungsverzeichnis