Javax / Usage

Development

Introduce

Dependent plugin： Plugin DevKit (This plugin is included by default in the idea.)

Source Code:　　　No source code is required, just add content using the Plugin Devkit plugin.
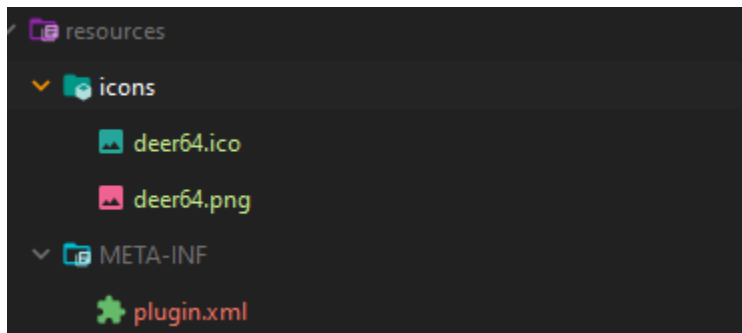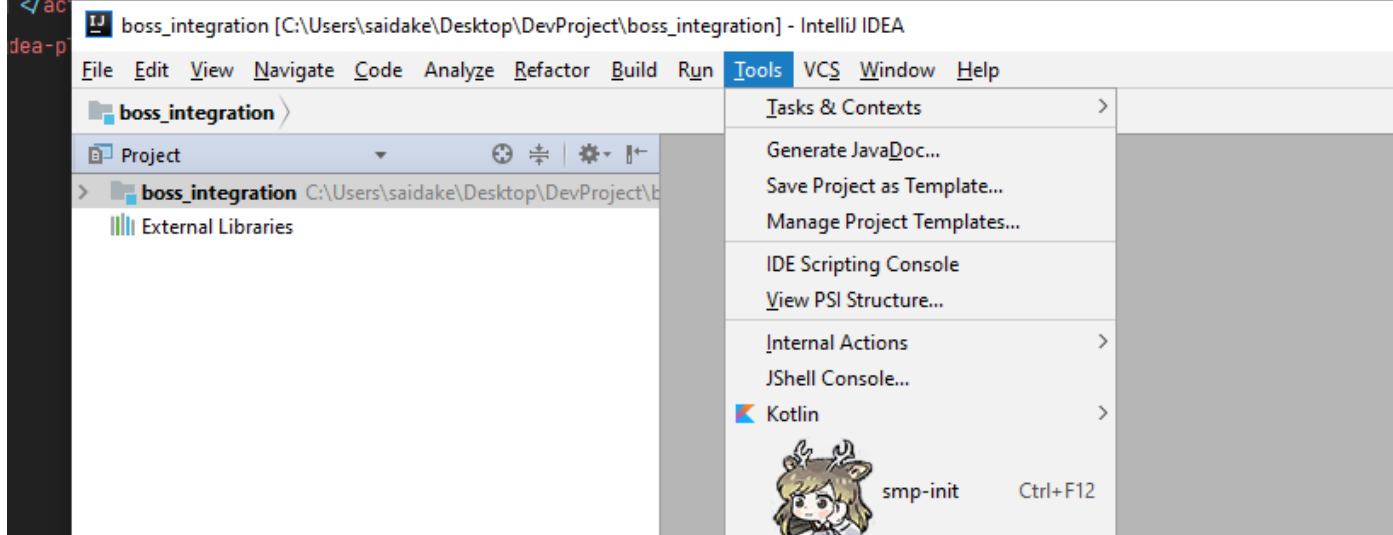
Icon

```
package icons;

public interface MyIcons {    // This class will be referced by plugin.xml

  Icon Action = IconLoader.getIcon("/icons/myAction.png", MyIcons.class);
  Icon Structure = IconLoader.getIcon("/icons/myStructure.png", MyIcons.class);
  Icon FileType = IconLoader.getIcon("/icons/myFileType.png", MyIcons.class);

}
```
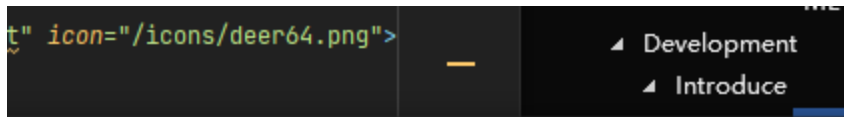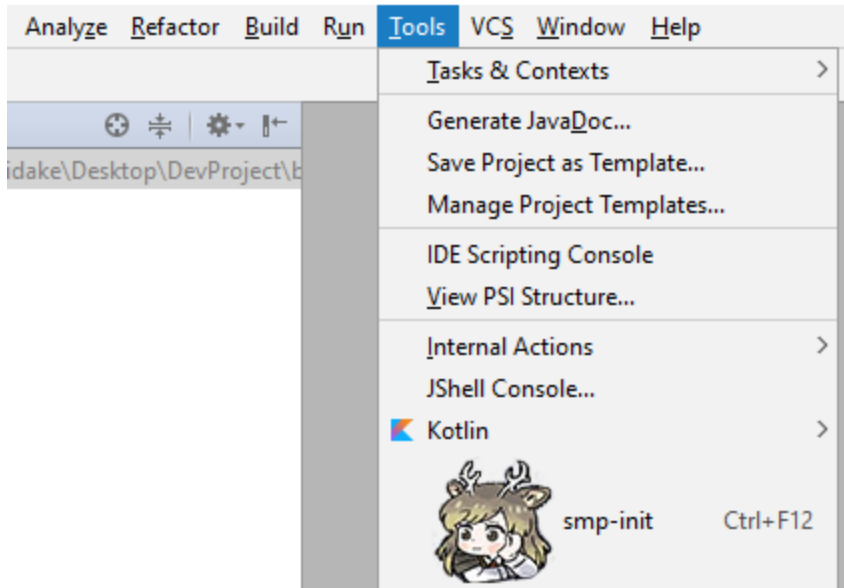
```
t" icon="/icons/deer64.png">
```

� Development
  � Introduce

ke\Desktop\DevProject\boss_integration] - IntelliJ IDEA

Analyze Refactor Build Run Tools VCS Window Help

dake\Desktop\DevProject\b

Tasks & Contexts	>
Generate JavaDoc...	
Save Project as Template...	
Manage Project Templates...	
IDE Scripting Console	
View PSI Structure...	
Internal Actions	>
JShell Console...	
Kotlin	>
smp-init	Ctrl+F12

Action

Create An action

New	>	Java Class
Cut	Ctrl+X	Groovy Class
Copy	Ctrl+C	Kotlin Class/File
Copy Path/Reference...		File
Paste	Ctrl+V	Scratch File Ctrl+Alt+Shift+Insert
Find Usages	Alt+F7	Package
Find in Files...	Ctrl+Shift+F	FXML File
Replace in Files...	Ctrl+Shift+R	JavaFX Application
Analyze	>	package-info.java
Refactor	>	HTML File
Bookmarks	>	Kotlin Script
Reformat Code	Ctrl+Alt+L	Kotlin Worksheet
Optimize Imports	Ctrl+Alt+O	Stylesheet
Delete...	Delete	PlantUML File
Build Module 'smp-init.main'		JavaScript File
Rebuild 'com.saidake.plugin.init'	Ctrl+Shift+F9	TypeScript File
Run 'Tests in 'com.saidake.plugin.init''	Ctrl+Shift+F10	package.json File
Debug 'Tests in 'com.saidake.plugin.init''		OpenAPI Specification
More Run/Debug	>	Swing UI Designer >
Open In	>	Resource Bundle
Local History	>	EditorConfig File
Git	>	XML Configuration File >
Reload from Disk		HTTP Request
Compare With...	Ctrl+D	Diagram >
Mark Directory as	>	Groovy Script
Diagrams	>	Data Source in Path
Convert Java File to Kotlin File	Ctrl+Alt+Shift+K	Plugin DevKit > Action
		Theme

action configuration in plugin.xml and Action class will be created automatically.

ToolsMenu

EditorPopupMenu 邮件编辑弹出菜单

<div style="background-color: yellow; text-align: center;">Toolwindow</div>

添加 GUI Form，添加 ToolWindowFacotory 初始化类，plugin.xml 添加窗口

控件变量名

添加事件

Create a NotificationGroup

NotificationGroup is registered in plugin.xml

Create a notification

com.intellij.notification.NotificationGroupManager

```
Editor editor = FileEditorManager.getInstance(project).getSelectedTextEditor();
if(editor!=null){
    VirtualFile file = FileDocumentManager.getInstance().getFile(editor.getDocument());
    int offset = editor.getCaretModel().getOffset();
    PsiElement element = PsiManager.getInstance(project).findFile(file).findElementAt(offset);
}
```

Build Project

- ∨ smp-init
  - ∨ Tasks
    - ∨ build
      - assemble
      - build
      - buildDependents
      - buildNeeded
      - classes
      - clean
      - jar
      - testClasses
    - › build setup
    - › documentation
    - › help
    - ∨ intellij
      - buildPlugin
      - buildSearchableOptions
      - downloadRobotServerPlugin
      - jarSearchableOptions
      - patchPluginXml
      - prepareSandbox
      - prepareTestingSandbox
      - prepareUiTestingSandbox
      - publishPlugin
      - runIde
      - runIdeForUiTests
      - runPluginVerifier
      - verifyPlugin
    - › other
    - › verification
  - › Dependencies
  - › Run Configurations

build
> classes
distributions
smp-init-1.0-SNAPSHOT.zip
smp-init-1.3.zip

Marketplace   Installed   Updates   ⋮

Manage Plugin Repositories...
HTTP Proxy Settings...
Install Plugin from Disk...
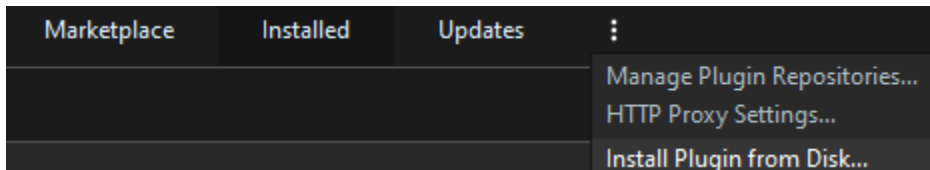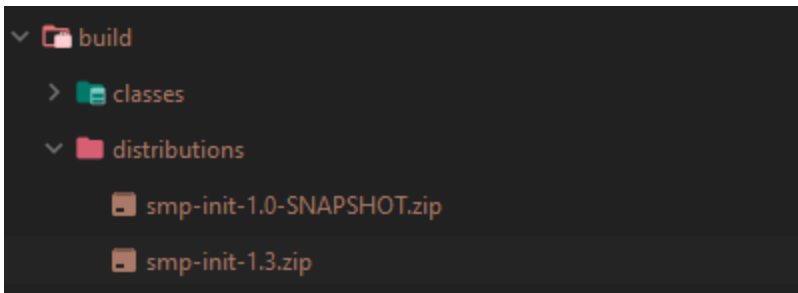
Javax / javax
crypto

Cipher

package javax.crypto;

public class **Cipher**

    The javax.crypto.Cipher class in Java provides the functionality for cryptographic operations including encryption, decryption, and key management.

    It's part of the Java Cryptography Extension (JCE) framework, which provides a comprehensive API for cryptographic operations.

    java.security.MessageDigest

public static final int ENCRYPT_MODE = 1;
public static final int DECRYPT_MODE = 2;
public static final int WRAP_MODE = 3;   Mode for wrapping a key.
public static final int UNWRAP_MODE = 4;   Mode for unwrapping a key.

public static final int PUBLIC_KEY = 1;
public static final int PRIVATE_KEY = 2;
public static final int SECRET_KEY = 3;

public final void init(int opmode, Key key) throws InvalidKeyException
public final void init(int mode, Key source, AlgorithmParameterSpec algorithm) throws InvalidKeyException, InvalidAlgorithmParameterException

    Initializes the cipher with the specified key and mode (ENCRYPT_MODE, DECRYPT_MODE).

    int opmode

        The operation mode of this cipher. It can be one of the following:

            Cipher.ENCRYPT_MODE: To initialize the cipher for encryption.
            Cipher.DECRYPT_MODE: To initialize the cipher for decryption.
            Cipher.WRAP_MODE: To initialize the cipher for key wrapping.
            Cipher.UNWRAP_MODE: To initialize the cipher for key unwrapping.

    Key key

The cryptographic key used for the operation. It can be a `SecretKey`, `PublicKey`, or `PrivateKey`, depending on the algorithm and operation mode.

AlgorithmParameterSpec params:

The algorithm parameters used to initialize the cipher. This can include parameters like initialization vectors (IVs), which are essential for certain modes of operation such as CBC (Cipher Block Chaining).

Symmetric Encryption Algorithms

- AES (Advanced Encryption Standard)

    Modes: CBC, CFB, CTR, CTS, ECB, OFB, GCM

    Padding: NoPadding, PKCS5Padding, PKCS7Padding

- DES (Data Encryption Standard)

    Modes: CBC, CFB, CTR, CTS, ECB, OFB

    Padding: NoPadding, PKCS5Padding, PKCS7Padding

- DESede (Triple DES)

    Modes: CBC, CFB, CTR, CTS, ECB, OFB

    Padding: NoPadding, PKCS5Padding, PKCS7Padding

- Blowfish

    Modes: CBC, CFB, CTR, CTS, ECB, OFB

    Padding: NoPadding, PKCS5Padding, PKCS7Padding

- RC2

    Modes: CBC, CFB, ECB, OFB

    Padding: NoPadding, PKCS5Padding

- RC4 (Stream Cipher)

    No specific modes or padding are used.

Asymmetric Encryption Algorithms

- RSA

    Modes: ECB (Used in conjunction with padding schemes, as RSA does not support block modes)

    Padding: NoPadding, PKCS1Padding, OAEPWithSHA-1AndMGF1Padding, OAEPWithSHA-256AndMGF1Padding

- ECIES (Elliptic Curve Integrated Encryption Scheme)

    Typically used with a default or specified padding and mode.

Examples of Cipher Algorithms

```
AES/CBC/PKCS5Padding
AES/ECB/NoPadding
DES/CBC/PKCS5Padding
DESede/CFB/NoPadding
Blowfish/OFB/NoPadding
RC2/ECB/PKCS5Padding
RSA/ECB/PKCS1Padding
RSA/ECB/OAEPWithSHA-256AndMGF1Padding
```

public static final Cipher getInstance(String var0) throws NoSuchAlgorithmException, NoSuchPaddingException

This static method returns a `Cipher` object that implements the specified transformation.

public final byte[] doFinal(byte[] var1) throws IllegalBlockSizeException, BadPaddingException

Performs the encryption or decryption operation on the provided input data and returns the result.

Usage

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
```

```java
public class EncryptionExample {

    // Method to generate a new AES Secret Key
    public static SecretKey generateSecretKey() throws Exception {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(128); // AES-128 bit key
        return keyGenerator.generateKey();
    }

    // Method to encrypt a string
    public static String encrypt(String plainText, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    // Method to decrypt a string
    public static String decrypt(String encryptedText, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
        return new String(decryptedBytes);
    }

    public static void main(String[] args) {
        try {
            // Generate a new AES Secret key
            KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
            keyGenerator.init(128); // AES-128 bit key
            SecretKey secretKey = keyGenerator.generateKey();

            // Plain text to be encrypted
            String plainText = "Hello, World!";

            // Encrypt the plain text
            Cipher cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
            String encryptedText = Base64.getEncoder().encodeToString(encryptedBytes);
            System.out.println("Encrypted Text: " + encryptedText);

            // Decrypt the encrypted text
            Cipher cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
            String decryptedText = new String(decryptedBytes);
            System.out.println("Decrypted Text: " + decryptedText);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

KeyGenerator

package javax.crypto;

public class **KeyGenerator**

public static final KeyGenerator getInstance(String algorithm) throws NoSuchAlgorithmException

        AES、ARCFOUR、Blowfish、DES 、DESede、RC2

        HmacMD5、HmacSHA1、HmacSHA256、HmacSHA384、HmacSHA512

        SunTlsPrf, SunTls12Prf, SunTlsMasterSecret, SunTlsKeyMaterial, SunTlsRsaPremasterSecret

public final void init(SecureRandom var1)

public final SecretKey generateKey()

KeyManager

package javax.net.ssl;

public interface **KeyManager**

Key manager, used for HTTPS mutual authentication, where the client sends the certificate and public key to the server. When interacting with different servers, clients may use different identities (certificates), so KeyManager is needed for management.

If it is only one-way authentication, KeyManager [] can be empty.

TrustManager

package javax.net.ssl;

public interface **TrustManager**

Trust manager, used for client detection of certificates sent by the server

KeyManagerFactory

package javax.net.ssl;

public class **KeyManagerFactory**

KeyManagerFactory is responsible for managing the key material used by an SSL/TLS server (or client in some cases) to establish a secure connection.

This includes the private keys and corresponding certificates that identify the server to the client.

public static final String getDefaultAlgorithm()

public static final KeyManagerFactory getInstance(String algorithm) throws NoSuchAlgorithmException

public final void init(KeyStore keystore, char[] password) throws KeyStoreException, NoSuchAlgorithmException, UnrecoverableKeyException

Usage
```
KeyStore keyStore = KeyStore.getInstance("JKS");
keyStore.load(new FileInputStream("keystore.jks"), "keystorePassword".toCharArray());

KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
kmf.init(keyStore, "keyPassword".toCharArray());

KeyManager[] keyManagers = kmf.getKeyManagers();
```

TrustManagerFactory

package javax.net.ssl;

public class **TrustManagerFactory**

TrustManagerFactory is responsible for managing the trust material, which is used to validate the certificates presented by peers during SSL/TLS handshake.

This includes verifying the authenticity of the certificates, typically against a set of trusted Certificate Authorities (CAs).

the tmf.getTrustManagers() method will not return a separate TrustManager for each certificate in the JKS file.

Instead, it will return a single array of TrustManager instances, typically containing one X509TrustManager, which is responsible for managing all the certificates in the KeyStore.

public static final String getDefaultAlgorithm()

public static final TrustManagerFactory getInstance(String algorithm) throws NoSuchAlgorithmException

public final void init(KeyStore keyStore) throws KeyStoreException

初始化信任管理人工厂，包含 keystore 中所有的信任管理人（设置 null 时，获取系统默认的所有信任管理人）

public final TrustManager[] getTrustManagers()

获取信任管理人列表，可以从空的 keyStore 中获取 X509TrustManger

X509TrustManager

package javax.net.ssl;

public interface **X509TrustManager** extends TrustManager

void checkClientTrusted(X509Certificate[] var1, String var2) throws CertificateException;　检查客户端证书信任

void checkServerTrusted(X509Certificate[] var1, String var2) throws CertificateException;　检查服务端证书验证

X509Certificate[] getAcceptedIssuers();　　　检查颁发者

SSLContext

package javax.net.ssl;

public class **SSLContext**　　SSL 上下文

应用层将数据丢给 TCP 时，需要经过 SSL 层的加密处理；TCP 层的数据丢给应用层时，需要经过 SSL 层的解密处理。因此网络中传输的都是加密后的数据，提高的通信的安全性。

HTTP：http 服务器不会验证客户端身份，客户端也不会验证响应的数据是请求的服务器发出。而且也不会对通信内容进行加密。

对称加密：加密和解密的密钥为同一个，加密速度快，钥匙越大解密时间越长

前提是双方都必须知道加密规则，所以只能事前约定好或者发送加密规则（密钥），发送显然是不可取的。

非对称加密：加密密钥是成对的（公钥和私钥），私钥由自己安全保管不外泄，公钥可以发给任何人

加密使用公钥或者私钥，解密需要私钥（公钥与私钥没有联系）

数字证书：HTTP 不会验证通信双方的身份，为了解决这个问题产生了数字证书

服务器首先向一个大家都信任的第三方机构申请一个身份证书

客户端向服务器建立通信之前首先向服务器 发送请求

服务器收到请求后 返回包含公钥的服务端证书

客户端收到证书后，与可信任的第三方机构 检验证书有效性，通过后则进行正常内容通信

服务器证书必须有 CA 机构签名，而这个权威证书又有可能经过更权威的证书签名，最顶层的权威证书就是根证书

根证书直接内置在浏览器中，浏览器可以利用自己自带的根证书验证某个服务器的证书是否有效

public static SSLContext getInstance(String var0) throws NoSuchAlgorithmException 　　 获 取 实 例 　　 // SSLContext.getInstance("TLS")

public final void init(KeyManager[] keyManager, TrustManager[] trustManager, SecureRandom scrureRandom) throws KeyManagementException　初始化

keyManager　　原始证书管理人（负责提供证书和私钥，证书发给对方 peer）

trustManager　　信任证书管理人（负责验证 peer 发来的证书）

public static synchronized void setDefault(SSLContext sslContext)　　设置默认的 SSL 连接上下文

使用———

只需要信任自己生成的证书或系统的证书：

```
TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init((KeyStore) null);                                     // 用空 keyStore 初始化 信任管理人工厂

X509TrustManager defaultTm = null;                             // 获取内部默认信任管理人
for (TrustManager tm : tmf.getTrustManagers()) {
    if (tm instanceof X509TrustManager) {
        defaultTm = (X509TrustManager) tm;
        break;
    }
}
```

```java
}

FileInputStream myKeys = new FileInputStream("truststore.jks");
KeyStore myTrustStore = KeyStore.getInstance(KeyStore.getDefaultType());
myTrustStore.load(myKeys, "password".toCharArray());           //加载自己的 keyStore
myKeys.close();

tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init(myTrustStore);                                         // 用自己的证书初始化信任管理人工厂


X509TrustManager myTm = null;                    // 获取内部默认信任管理人
for (TrustManager tm : tmf.getTrustManagers()) {
    if (tm instanceof X509TrustManager) {
        myTm = (X509TrustManager) tm;
        break;
    }
}

final X509TrustManager finalDefaultTm = defaultTm;
final X509TrustManager finalMyTm = myTm;
X509TrustManager customTm = new X509TrustManager() {
    @Override
    public X509Certificate[] getAcceptedIssuers() {
        return finalDefaultTm.getAcceptedIssuers();          // 如果您计划使用客户端证书身份验证，请合并"defaultTm"
和"myTm"的结果。
    }

    @Override
    public void checkServerTrusted(X509Certificate[] chain,
            String authType) throws CertificateException {
        try {
            finalMyTm.checkServerTrusted(chain, authType);
        } catch (CertificateException e) {
            finalDefaultTm.checkServerTrusted(chain,  authType);          //如果这也失败，这将引发另一个
CertificateException。
        }
    }

    @Override
    public void checkClientTrusted(X509Certificate[] chain,
            String authType) throws CertificateException {
        finalDefaultTm.checkClientTrusted(chain, authType);           //如果您计划使用客户端证书身份验证，请执行与检查
服务器相同的操作。
    }
};


SSLContext sslContext = SSLContext.getInstance("TLS");
sslContext.init(null, new TrustManager[] { customTm }, null);
```

SSLSocketFactory

package javax.net.ssl;

public abstract class **SSLSocketFactory** extends SocketFactory   SSL 套接字工厂

HttpsURLConnection

package javax.net.ssl;

public abstract class **HttpsURLConnection** extends HttpURLConnection   HTTPS 连接

public static void setDefaultSSLSocketFactory(SSLSocketFactory sslSocketFactory)   设置默认的 ssl 连接工厂

swing

JComponent

package javax.swing;

public abstract class **JComponent** extends Container implements Serializable, TransferHandler.HasGetTransferHandler 组件

public void add(Component comp, Object constraints) 添加组件 // panel.add(tfTitle, BorderLayout.NORTH) 指定位置添加

public void setPreferredSize(Dimension preferredSize) 设置大小

public void setEnabled(boolean enabled) 启用组件

JPanel

package javax.swing;

public class **JPanel** extends JComponent implements Accessible 面板

public JPanel(LayoutManager layout) 添加一个布局

JScrollPane

package javax.swing;

public class **JScrollPane** extends JComponent implements ScrollPaneConstants, Accessible 滚轮面板，放在 JPanel 内，会占据整个 Panel

JLabel

package javax.swing;

public class **JLabel** extends JComponent implements SwingConstants, Accessible 标签

public JLabel(String text)

public void setText(String text)

JTree

package javax.swing;

public class **JTree** extends JComponent implements Scrollable, Accessible

public JTree(Vector<?> value)

public JTree(Hashtable<?,?> value)

public TreeSelectionModel getSelectionModel() 获取选择模型

public void setModel(TreeModel newModel) 设置数的数据

public void setShowsRootHandles(boolean newValue) 是否显示节点句柄

public TreeCellRenderer getCellRenderer() 获取元素渲染器 // DefaultTreeCellRenderer renderer = (DefaultTreeCellRenderer) tree.getCellRenderer();

public void setRootVisible(boolean rootVisible) 根元素是否可见，隐藏根节点，同时也会隐藏所有子节点

public void expandRow(int row) 先展开，再隐藏根元素会出现一个列表

public void setRowHeight(int rowHeight) 设置行高 // 33

public void setAutoscrolls(boolean autoscrolls) 自动滚动

public void setScrollsOnExpand(boolean newValue) 张开自动滚动

public void addTreeSelectionListener(TreeSelectionListener tsl) 为 TreeSelection 事件添加侦听 TreeSelection 。

JTable

package javax.swing;

public class **JTable** extends JComponent implements TableModelListener, Scrollable, 表格
    TableColumnModelListener, ListSelectionListener, CellEditorListener,
    Accessible, RowSorterListener

public void setModel(TableModel dataModel) 设置 table 的数据

JTextField

package javax.swing;

public class **JTextField** extends JTextComponent implements SwingConstants 文本输入框

JButton

package javax.swing;

public class **JButton** extends AbstractButton implements Accessible

public JButton(String text)

ImageIcon

package javax.swing;

public class **ImageIcon** implements Icon, Serializable, Accessible　　图片图标

public ImageIcon(String filename, String description)

public ImageIcon (URL location)

AbstractButton

package javax.swing;

public abstract class **AbstractButton** extends JComponent implements ItemSelectable, SwingConstants

public void addActionListener(ActionListener actionListener)　　添加动作监听

BoxLayout

package javax.swing;

public class **BoxLayout** implements LayoutManager2, Serializable　箱式布局，允许在容器中纵向或者横向放置多个控件

SpringLayout

package javax.swing;

public class **SpringLayout** implements LayoutManager2　　弹簧布局，根据一组约束条件放置条件

DefaultTableModel

package javax.swing.table;

public class **DefaultTableModel** extends AbstractTableModel implements Serializable　　默认表格数据

public DefaultTableModel(Object[][] data, Object[] columnNames)　　行数据二维数组，表头数组

public void setDataVector(Object[][] dataVector, Object[] columnIdentifiers)　设置初始数据

public void addRow(Object[] rowData) 添加一行数据

TreeModel

package javax.swing.tree;

public interface **TreeModel**　　树的数据模型

TreeSelectionModel

package javax.swing.tree;

public interface **TreeSelectionModel**　　JTree 树的选择模型

public static final int　　　　SINGLE_TREE_SELECTION = 1;　　　　一次只能选择一个

public static final int　　　　CONTIGUOUS_TREE_SELECTION = 2;

public static final int　　　　DISCONTIGUOUS_TREE_SELECTION = 4;

void　　setSelectionMode(int　　mode)　　　设　置　选　择　模　式　　　　　　　　　　　　//
tree.getSelectionModel().setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION)

DefaultTreeModel

package javax.swing.tree;

public class **DefaultTreeModel** implements Serializable, TreeModel　　　默认数据模型

public DefaultTreeModel(TreeNode root)　　添加一个根元素，默认允许有子元素

public DefaultTreeModel(TreeNode root, boolean asksAllowsChildren)

DefaultMutableTreeNode

package javax.swing.tree;

public class **DefaultMutableTreeNode** implements Cloneable, MutableTreeNode, Serializable　　默认 JTree 树节点

public DefaultMutableTreeNode(Object userObject)　默认允许有子元素

public DefaultMutableTreeNode(Object userObject, boolean allowsChildren)

public void add(MutableTreeNode newChild)　　添加一个 child

TreeCellRenderer

package javax.swing.tree;
public interface **TreeCellRenderer**

  覆写，对特定元素设置 icon：

    @Override

    public Component getTreeCellRendererComponent(JTree tree,

     Object value, boolean selected, boolean expanded,

     boolean leaf, int row, boolean hasFocus) {

      super.getTreeCellRendererComponent(tree, value, selected,expanded, leaf, row, hasFocus);

      DefaultMutableTreeNode nodo = (DefaultMutableTreeNode) value;

      if (tree.getModel().getRoot().equals(nodo)) {

       setIcon(root);

      } else if (nodo.getChildCount() > 0) {

       setIcon(parent);

      } else {

       setIcon(leaf);

      }

      return this;

    }

Component getTreeCellRendererComponent(

  JTree tree,

  Object value,

  boolean selected,

  boolean expanded,

  boolean leaf,

  int row,

  boolean hasFocus)

DefaultTreeCellRenderer

package javax.swing.tree;
public class **DefaultTreeCellRenderer** extends JLabel implements TreeCellRenderer
public void setLeafIcon(Icon newIcon)　　设置用于表示叶节点的图标。
public void setOpenIcon(Icon newIcon)　　设置用于表示展开的非叶节点的图标。
public void setClosedIcon(Icon newIcon)　　设置用于表示未展开的非叶节点的图标。
public void setBackgroundNonSelectionColor(Color newColor)　　未选中的节点背景色（仅仅是节点的文字背景，不是整行背景）

package javax.swing;
public class **BorderFactory** 边框工厂，提供各种固定样式的边框对象创建。

xml (java.xml)
parsers

SAXParserFactory

package javax.xml.parsers;
public abstract class **SAXParserFactory**　　　　SAX XML Parser
public static SAXParserFactory newInstance()　　Use default implementation class
"com.sun.org.apache.xerces.internal.jaxp.SAXParserFactoryImpl".

DocumentBuilderFactory [Core]

package javax.xml.parsers;

public abstract class **DocumentBuilderFactory**   DOM XML Parser

Node data type:

Interface	nodeName	nodeValue	attributes
Attr	same as Attr.name	same as Attr.value	null
CDATASection	"#cdata-section"	same as CharacterData.data, the content of the CDATA Section	null
Comment	"#comment"	same as CharacterData.data, the content of the comment	null
Document	"#document"	null	null
DocumentFragme	"#document-fragment"	null	null
DocumentType	same as DocumentType.name	null	null
Element	same as Element.tagName	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInst	same as ProcessingInstruction.target	same as ProcessingInstruction.data	null
Text	"#text""	same as CharacterData.data, the content of the text node	null

@see org.w3c.dom.Node

public static DocumentBuilderFactory newInstance()   Use default implementation class "com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderFactoryImpl".
public abstract DocumentBuilder newDocumentBuilder() throws ParserConfigurationException;   create a new instance of DocumentBuilder.

DocumentBuilder

package javax.xml.parsers;

public abstract class **DocumentBuilder**

public Document parse(File f) throws SAXException, IOException   Obtain Document by parsing the file.

xpath

XPathFactory

package javax.xml.xpath;

public abstract class **XPathFactory**

public static XPathFactory newInstance()

public abstract XPath newXPath();   Get a new instance of XPath.

XPath

package javax.xml.xpath;

public interface **XPath**

public Object evaluate(
        String expression,
        Object item,
        QName returnType   Specifys the return type and references XPathConstants.
) throws XPathExpressionException;

```
// NodeList nodes = (NodeList)xp.evaluate("//@*", nlist.item(0), XPathConstants.NODESET);
```

XPathConstants

package javax.xml.xpath;

public class **XPathConstants**

public static final QName NUMBER = new QName("http://www.w3.org/1999/XSL/Transform", "NUMBER");   Number
data type

public static final QName STRING = new QName("http://www.w3.org/1999/XSL/Transform", "STRING");   String data
type.

public static final QName BOOLEAN = new QName("http://www.w3.org/1999/XSL/Transform", "BOOLEAN");   Boolean
data type.

public static final QName NODESET = new QName("http://www.w3.org/1999/XSL/Transform", "NODESET");   NodeSet data
type, Maps to NodeList.

public static final QName NODE = new QName("http://www.w3.org/1999/XSL/Transform", "NODE");   NodeSet data
type, Maps to Node.

public static final String DOM_OBJECT_MODEL = "http://java.sun.com/jaxp/xpath/dom";

transform

TransformerFactory

package javax.xml.transform;

public abstract class **TransformerFactory**

public static TransformerFactory newInstance() throws TransformerFactoryConfigurationError

public abstract Transformer newTransformer() throws TransformerConfigurationException;

Transformer

package javax.xml.transform;

public abstract class **Transformer**

public abstract void transform(Source xmlSource, Result outputTarget) throws TransformerException;   Transform the XML
Source to a Result.

DOMSource

package javax.xml.transform.dom;

public class **DOMSource** implements Source

public DOMSource(Node n)   Construct a DOMSource.

StreamResult

package javax.xml.transform.stream;

public class **StreamResult** implements Result

 public StreamResult(File f)   Construct a SteamResult from a file.

sql

DataSource

package javax.sql;

public interface **DataSource** extends CommonDataSource, Wrapper

imageio

ImageInputStream

package javax.imageio.stream;

public interface **ImageInputStream** extends DataInput, Closeable

int read(byte[] b) throws IOException;

Javax / jakarta
activation (jakarta.activation-api)

DataSource

package jakarta.activation;

public interface **DataSource**

注解

package jakarta.annotation;

@Resource　　　从 IOC 容器中注入 Bean 组件，按照 bean 名称匹配，找不到时再按 bean 类型匹配。JSR250 规范中的注解【TYPE, FIELD, METHOD】

　　　　　　name　　注入 bean 的 id

　　　　　　type　　注入 bean 的类型，接受参数为 class 类型

@PostConstruct　　　修饰一个非静态的 void（）方法。被@PostConstruct 修饰的方法会在服务器加载 Servlet 的时候运行，并且只会被服务器执行一次。【METHOD】

　　　　　　　　　　PostConstruct 在构造函数之后执行，init（）方法之前执行

@PreDestroy　　　　设置该类作为 bean 对应的生命周期方法【METHOD】

@Nonnull　　　加了@Nonnull 后编码传入 Null 时有非空警告，必须有 FindBugs 插件配合，或者使用 IDEA；运行时不检查。

　　　　　　　它是 JSR 305（缺陷检查框架）的注解，是告诉编译器这个域不可能为空

@Generated　　用于标识生成的代码或类是由哪个工具或程序自动生成的，一般是由代码生成器或自动化构建工具生成的代码所使用的，以便于开发者识别代码是否是手动编写的还是由工具生成的。

　　　　　　value="io.swagger.codegen.languates.JavaJerseyServerCodegen"

　　　　　　date="2019-03-13T09:00:29.816Z"

package javax.annotation.meta;

@TypeQualifierNickname　　　当前注解上其他注解别称，当前注解使用时，会连带使用当前注解上使用的所有注解（除了 TypeQualiirNickname 自己）

ELManager

package jakarta.el;

public class **ELManager**　　　EL 管理器

MessagingException

package jakarta.mail;

public class **MessagingException** extends Exception　　　消息异常

MimeMessage

package jakarta.mail.internet;

public class **MimeMessage** extends Message implements MimePart　　　mimie 邮件消息对象

ByteArrayDataSource

package jakarta.mail.util;

public class **ByteArrayDataSource** implements DataSource   字节数组数据资源

public ByteArrayDataSource(

      InputStream is,

      String type   mime 类型 // image/png

) throws IOException

EntityManager

package javax.persistence;

public interface **EntityManager**   实体管理

void persist(Object var1) 保存一个数据

GenerationType

public enum **GenerationType**   通用策略生成器

      IDENTITY   采用数据库 ID 自增长的方式来自增主键字段，Oracle 不支持这种方式

      AUTO   自动选择合适的策略，是默认选项;

      SEQUENCE   通过序列产生主键，通过@SequenceGenerator 注解指定序列名，MySql 不支持这种方式

      TABLE   通过表产生主键，框架借由表模拟序列产生主键，使用该策略可以使应用更易于数据库移植。

CascadeType

public enum **CascadeType**   级联类型

  ALL

  PERSIST

  MERGE   指向表 级联更新

  REMOVE   指向表 级联删除

  REFRESH   指向表 级联刷新（并发情况下，为了防止其他线程抢占更新数据，在保存数据的时，先刷新指向表数据和 关联外键表数据，再保存）

  DETACH   指向表 级联脱管（删除指向表 整条数据前，撤销这条数据 所有的关联外键表数据）

FetchType

public enum **FetchType**   sql 加载模式

  LAZY   懒加载

  EAGER   饥饿加载

EnumType

public enum **EnumType** 字段枚举类型

  ORDINAL   持久化的为 自定义枚举类型 的值 （根据数据库内的名称获取 值）

  STRING   持久化的为 自定义枚举类型 的名称（根据数据库内的值获取 名称）

AttributeConverter

public interface **AttributeConverter**<X, Y>   模板方法，自定义数据库类型转换器

Y convertToDatabaseColumn(X var1)   存储值时调用，把 Y 存入数据库

X convertToEntityAttribute(Y var1)   查询值时调用，转成 X 赋值到实体类中

Version

package jakarta.persistence;

@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface **Version** {}

The version checking process with the @Version annotation occurs within the context of a transaction.

How @Version Works

Version Field:

The @Version annotation is applied to a field in an entity class, typically a numeric or timestamp field. This field tracks the version of the entity.

Initial Version:

When a new entity is created, the version field is initialized to a default value (e.g., 0 for numeric types).

Increment on Update:

Each time the entity is updated and the transaction is committed, the version field is incremented by the persistence provider (e.g., Hibernate).

This increment is handled automatically and is typically done by the database itself or by the JPA implementation.

Concurrency Check:

When a transaction attempts to update an entity, the persistence provider checks the version field.

If the version in the database does not match the version in the entity being updated,

it indicates that another transaction has modified the entity since it was read. This leads to an OptimisticLockException, which the application can handle to resolve the conflict.

OneToMany

package jakarta.persistence;

@Target({ElementType.METHOD, ElementType.FIELD})

@Retention(RetentionPolicy.RUNTIME)

public @interface **OneToMany**   多对一关联查询，当前为 少的一方（关联他表 Entity 对象 字段）

If two new objects are created simultaneously, when performing an operation, it should first perform a save operation for multiple tables, and then perform a save operation for fewer tables

If an object already exists, you can do the following:

TestPersonEntity firstByPerId = testPersonRepository.findFirstByPerId(1L);

List<TestStudentEntity> testStudentEntities = firstByPerId.getTestStudentEntities();

TestStudentEntity testStudentEntity = new TestStudentEntity();

testStudentEntity.setPerId(firstByPerId.getPerId());

testStudentEntity.setRole("normal");

testStudentEntity.setClassName("lala");

testStudentRepository.save(testStudentEntity);

testStudentEntities.add(testStudentEntity);

testPersonRepository.save(firstByPerId);

Class targetEntity() default void.class;   指定 关联他表 Entity 对象 类型   // BookingContainerDetailLogEntity.class

CascadeType[] cascade() default {};   级联策略   // CascadeType.ALL

FetchType fetch() default FetchType.LAZY;   Load data mode, in debug mode, cascading queries are automatically performed, LAZY will fail.   // FetchType.EAGER

String mappedBy() default "";   // "user"

boolean orphanRemoval() default false;

JoinColumn

package jakarta.persistence;
@Repeatable(JoinColumns.class)
@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface **JoinColumn**　　　关联外键（关联他表 Entity 对象 字段）

String name() default "";　　　　　　　　　当前外键表　外键列名　// "card_id "
String referencedColumnName() default "";　指向表 外键列（默认为指向表的主键）　//　"id"

boolean unique() default false;
boolean nullable() default true;
boolean insertable() default true;
boolean updatable() default true;
String columnDefinition() default "";
String table() default "";
ForeignKey foreignKey() default @ForeignKey(ConstraintMode.PROVIDER_DEFAULT);

GeneratedValue

package jakarta.persistence;
@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface **GeneratedValue**　　　指定自定义生成策略
GenerationType strategy() default GenerationType.AUTO;　　不使用自定义生成策略，而使用通用策略生成器　　//
GenerationType.IDENTITY
String generator() default "";　指定自定义生成策略　　//"idGenerator"

Enumerated

@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface **Enumerated**
EnumType value() default EnumType.ORDINAL;
　　value= EnumType.STRING　映射方式，默认存储枚举的数字序列

@Entity　　　　　　　　表明这是一个表实体 【TYPE】
@Table　　　　　　　　数据库表【TYPE】
　　　　　name="t_user"　指定表名
　　　　　schema="OWNER"　指定表拥有者，oracle 中为
@MappedSuperclass　　　　　标注父实体类（Entity 实体需要被继承使用）【TYPE】
@EmbeddedId　　　　　组合主键，搭配@Embeddable 的组合类使用
@Embeddable　　　　　　表示这个对象不是单独的数据表，它里面的字段会在其它实体中公用【TYPE】
@EntityListeners　　　　　监听实体属性变化，可以监听的事件：保存前，保存后，更新前，更新后，删除前，删除后【TYPE】
　　　　　value= ShippingChangeListener.class　自定义的监听器类

@Transient　　　不是表的字段映射，使用这个注解，ORM 框架将会忽略该属性

@Id　　　　　　　　　　指定为 主键　【METHOD，FIELD】
@Enumerated　　　　　　　枚举字段映射 【METHOD，FIELD】
　　　　　value= EnumType.STRING　 映射方式，默认存储枚举的数字序列

@Column　　　　　　　　　　指定为 表字段【METHOD，FIELD】
　　　　　length=50　　　　属性表示字段的长度，当字段的类型为 varchar 时，该属性才有效，默认为 255 个字符。
　　　　　name="lala"　　　　字段在数据库表中所对应字段的名称，默认为变量名；
　　　　　precision=2　　　　表示精度，当字段类型为 double 时，precision 表示数值的总长度。
　　　　　scale=2　　　　　表示精度，当字段类型为 double 时， scale 表示小数点所占的位数。

　　　　　unique=true　　　　字段是否为唯一标识，默认为 false，也可以使用@Table 标记中的@UniqueConstraint。
　　　　　nullable=true　　　　属性表示该字段是否可以为 null 值，默认为 true
　　　　　insertable　　　　属性表示在使用"INSERT"脚本插入数据时，是否需要插入该字段的值。
　　　　　updatable　　　　执行"UPDATE"插入数据时，是否更新该字段的值，用于只读自动生成的主键或外键
　　　　　columnDefinition="JSON"　　创建表时，指定该字段的 类型定义（如果 DB 中表已经建好，该属性没有必要使用。）
　　　　　table　　　　　　定义了包含当前字段的表名。
@OneToOne　　　　　　　　一对一关联查询（关联他表 Entity 对象 字段）　【METHOD，FIELD】
　　　　　cascade = CascadeType.ALL　　　级联策略
@ManyToOne　　　　　　　　多对一关联查询，当前为 多的一方（关联他表 Entity 对象 字段）　【METHOD，FIELD】
　　　　　cascade = CascadeType.ALL　　　级联策略
@Convert　　　　数据库存储类型 与 程序中类型的转换，converter 必须实现 AttributeConverter
@NamedNativeQuery　　在类名上预定义一个 native sql
　　　　　name = "UsersSeq.NextVal"
　　　　　query = "select CCBCM_OWNER.USERS_SEQ.nextval from dual"
　　　　　converter = MyConverter.class

servlet (jakarta.servlet-api)

Filter

public interface **Filter**
　　Related Classes:
　　　　org.springframework.web.filter.OncePerRequestFilter
　　　　org.springframework.web.filter.CorsFilter
　　　　org.springframework.web.filter.CharacterEncodingFilter
　　　　org.springframework.web.filter.HiddenHttpMethodFilter
　　　　org.springframework.web.filter.ForwardedHeaderFilter
　　　　org.springframework.web.filter.ShallowEtagHeaderFilter
　　　　org.springframework.web.filter.RequestContextFilter

void init(FilterConfig filterConfig) throws ServletException;
void doFilter(ServletRequest var1, ServletResponse var2, FilterChain var3)

ServletRequest

package javax.servlet;

public interface **ServletRequest**　　　在当前的请求或请求转发之间实现数据共享（是 Servlet 规范中四大域对象的请求域）

String getParameter(String var1);　　　　　用键名获取值（请求体 ?m=2）
RequestDispatcher getRequestDispatcher(String var1);　　获 取 派 遣 器 ， 可 以 用 于 转 发　　　　　　　　//
request.getRequestDispatcher("static/index.html").forward(request,response); 获取派遣器
ServletInputStream getInputStream()　　　　获取请求体输入流（仅能获取一次，springboot 的@RequestMapping 也会调用）
BufferedReader getReader()
ServletContext getServletContext();
Map<String, String[]> getParameterMap()
void setCharacterEncoding(String var1) throws UnsupportedEncodingException　设置编码
String getCharacterEncoding();　　　　　获取编码

ServletResponse

public interface **ServletResponse**

void setContentType(String var1);　　　　设置响应体格式
void setCharacterEncoding(String var1);　设置响应体编码
PrintWriter getWriter()　　　　　　　响应字符串　// getWriter().print(str);
void reset();　　　　　　　清空输出流
void addCookie(Cookie var1);　　　添加 cookie

ServletContextListener

public interface **ServletContextListener** extends EventListener　　上下文监听器

default void contextInitialized(ServletContextEvent sce)
default void contextDestroyed(ServletContextEvent sce)

ServletRequestWrapper

package jakarta.servlet;
public class **ServletRequestWrapper** implements ServletRequest
public ServletRequestWrapper(ServletRequest request)　　　Save the source request object to an internal field.

Cookie

package javax.servlet.http;
public class **Cookie**

public String getName()
public String getValue()
public void setDomain(String pattern)　// setDomain("localhost")

ServletContext

package javax.servlet;
public interface **ServletContext**
　　　　　应用上下文对象，web 应用中最大的作用域，每一个应用中只有一个 ServletContext 对象（是 Servlet 规范中四大域对象的
　　　　　应用域）
　　　　　域对象指的是对象有作用域，也就是有作用范围。域对象可以实现数据的共享，不同作用范围的域对象，共享数据的能力也不

一样。

获取：通过 javax.servlet.ServletRequest.getServletContext 方法

作用：可以配置和获得应用的全局初始化参数，可以实现 Servlet 之间的数据共享.

生命周期：应用一加载则创建，应用被停止则销毁。
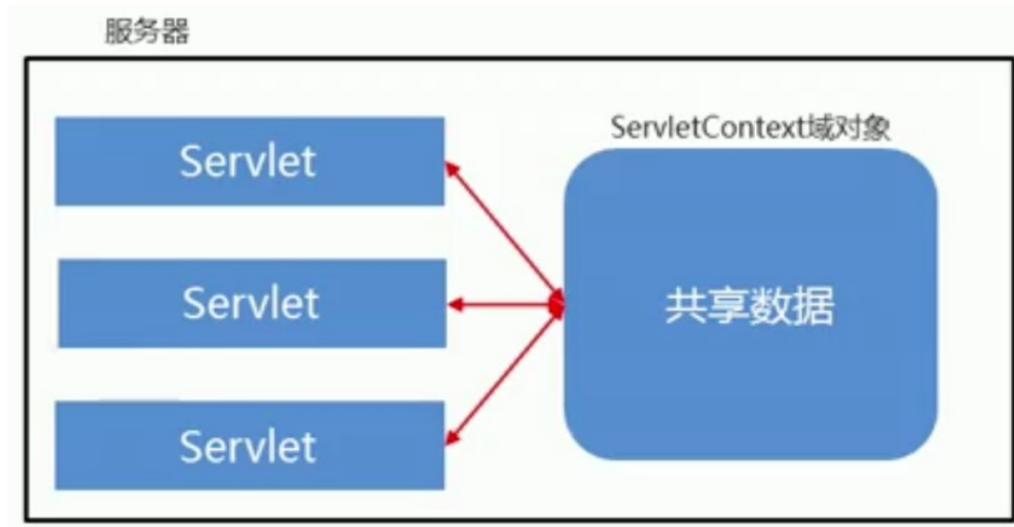
String getInitParameter(String var1);　　　根据名称获取全局配置的参数

String getContextPath();　　　　　　获取当前应用的访问虚拟目录　　　　//　/demo2

String getRealPath(String var1);　　　　　根据虚拟目录获取应用部署的磁盘绝对路径　//　D:\develop

Object getAttribute(String var1);　　　　过名称获取应用域对象中的数据

Enumeration<String> getAttributeNames();

void setAttribute(String var1, Object var2);　向应用域对象中存储数据

void removeAttribute(String var1);　　　　通过名称移除应用域对象中的数据

javax.servlet.FilterRegistration.Dynamic addFilter(String var1, Filter var2);　添加一个过滤器

web.xml >>

<context-param>　　配置全局参数

　　<param-name>xxname<param- name >

　　<param-value>xxval<param-value>

</context-param>

ServletConfig

package javax.servlet;

public interface **ServletConfig**　　　ServletConfig 是 Servlet 的配置参数对象，在 Servlet 的规范中，允许为每一个 Servlet 都提供一些初始化的配置。每个 Servlet 都有一个自己的 ServletConfig.

　　　　　　　　　通过 HttpServlet 的 init 方法获取 ServletConfig

　　　　　　　　　web.xml

　作用：在 Servlet 的初始化时，把一些置信息传递给 Servlet,

　生命周期：和 Servlet 相同。
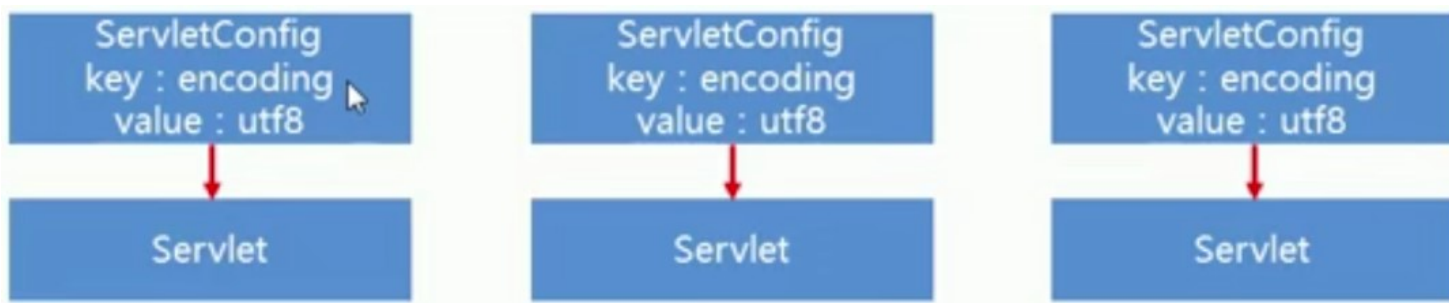
String getServletName();　　　　　　　获取 Servlet 的名称

ServletContext getServletContext();　　　　获取 ServletContext 对象

String getInitParameter(String var1);　　　根据参数名称获取参数的值

Enumeration<String> getInitParameterNames();　获取所有参数名称的枚举

web.xml >>

<servlet>　　服务执行类

　　<init-param>　　配置全局参数

　　　<param-name>xxname<param- name >

　　　<param-value>xxval<param-value>

　　</init-param>

</servlet>

FilterRegistration

package javax.servlet;

public interface FilterRegistration　　　动态注册过滤器，不需要重新加载应用程序就可以安装新的 Web 对象

public interface Dynamic

void addMappingForUrlPatterns(EnumSet<DispatcherType> var1, boolean var2, String... var3);

HttpServlet

package javax.servlet.http;

public abstract class **HttpServlet**　　　路由执行类

protected void doGet(HttpServletRequest req, HttpServletResponse resp)　　处理 get 请求

protected void doPost(HttpServletRequest req, HttpServletResponse resp)　　处理 post 请求

ServletInputStream getInputStream() throws IOException;

HttpServletRequest

package javax.servlet.http;

public interface **HttpServletRequest** extends ServletRequest　　　HttpServletRequest 比 ServletRequest 多了一些针对于 Http 协议的方法（ServletRequest 是为了多种协议设计的，目前主流还是 Http 协议）

　　　　　org.springframework.web.context.request.RequestContextHolder　　　　　(ServletRequestAttributes)

　　　　　RequestContextHolder.getRequestAttributes().getRequest　　获取当前 HttpServletRequest

　　　　　@Autowired　private HttpServletRequest request;　　　　　　　注入的 request 对象的确只注入一个，

　　　　　request 是个 spring 的代理对象，每次使用 request 时，通过代理，使用的是当前线程对应的 request 对象。

　　　　　　　　　　httpservlectRequest 是 web 容器的，不是交给 Spring 容器管理的，需要添加依赖 spring-boot-starter-web

public String getMethod();   获取请求方式
public String getHeader(String name);   获取请求头，获取单个请求头 name 对应的 value 值
Enumeration<String> getHeaders(String var1);   获取请求头，该方法将返回指定名字的 request header 的所有值，其结果是一个枚举对象。

public StringBuffer getRequestURL();   请求全路径   http://localhost:8080/BMI/index.html
public String getRequestURI();   除去 host 的路径   /BMI/index.html
public String getContextPath();   工程名（如果工程映射为/，此处返回则为空）   /BMI
public String getServletPath();   除去 host 和工程名的路径   /index.html
public HttpSession getSession();   获取会话   // getSession().getServletContext().getRealPath("") 获取项目路径
E:\Tomcat\webapps\TEST
                    携带 JSESSIONID 时，根据该值在服务器端查找是否有 HttpSession 对象，没有就创建新的 HttpSession 对象分配唯一标识 JSESSIONID
                    没带 JSESSIONID 时，创建建新的 HttpSession 对象分配唯一标识 JSESSIONID 将唯一标识发送给客户端

public HttpSession getSession(boolean var1);   获取 HttpSession 对象,未获取到是否自动创建
public Enumeration<String> getHeaderNames()   获取所有 Header 键名

使用———
———————————————————————————————

```java
import java.io.IOException;
import java.lang.reflect.Field;
import java.util.UUID;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.catalina.connector.Request;
import org.apache.catalina.connector.RequestFacade;
import org.apache.tomcat.util.http.MimeHeaders;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
import org.springframework.util.ReflectionUtils;



@WebFilter(urlPatterns = "/*")
@Component
@Order(-999)
public class RequestIdGenFilter extends HttpFilter {

    /**
     *
     */
    private static final long serialVersionUID = 1787347739651657706L;

    @Override
    protected void doFilter(HttpServletRequest req, HttpServletResponse res, FilterChain chain) throws
IOException, ServletException {
        try {
            // 从 RequestFacade 中获取 org.apache.catalina.connector.Request
            Field connectorField = ReflectionUtils.findField(RequestFacade.class, "request", Request.class);
            connectorField.setAccessible(true);
```

```
            Request connectorRequest = (Request) connectorField.get(req);

            // 从 org.apache.catalina.connector.Request 中获取 org.apache.coyote.Request
            Field coyoteField = ReflectionUtils.findField(Request.class, "coyoteRequest",
org.apache.coyote.Request.class);
            coyoteField.setAccessible(true);
            org.apache.coyote.Request coyoteRequest = (org.apache.coyote.Request)
coyoteField.get(connectorRequest);

            // 从 org.apache.coyote.Request 中获取 MimeHeaders
            Field mimeHeadersField = ReflectionUtils.findField(org.apache.coyote.Request.class, "headers",
MimeHeaders.class);
            mimeHeadersField.setAccessible(true);
            MimeHeaders mimeHeaders = (MimeHeaders) mimeHeadersField.get(coyoteRequest);

            this.mineHeadersHandle(mimeHeaders);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        super.doFilter(req, res, chain);
    }

    protected void mineHeadersHandle (MimeHeaders mimeHeaders) {
        // 添加一个 Header，随机生成请求 ID
        mimeHeaders.addValue("x-request-id").setString(UUID.randomUUID().toString());;
        // 移除一个 header
        mimeHeaders.removeHeader("User-Agent");
    }
}
```

HttpServletRequestWrapper

package jakarta.servlet.http;

public   class   **HttpServletRequestWrapper**   extends   ServletRequestWrapper   implements   HttpServletRequest

    HttpServletRequest 对象的参数是不可改变的，可以通过装饰模式来改变其状态

    jakarta.servlet.ServletRequestWrapper

        org.apache.catalina.connector.RequestFacade

        org.apache.catalina.connector.Request

public HttpServletRequestWrapper(HttpServletRequest request)   Directly call the parent constructer

HttpServletResponse

package javax.servlet.http;

public interface HttpServletResponse

void sendRedirect(String var1)   路由重定向

void setHeader(String var1, String var2) 设置请求头   // response.setHeader("Content-Disposition", "attachment;filename="
+ new String(fileName.getBytes("UTF-8"), "ISO-8859-1"));//设定输出文件头

void reset();   清空输出流，首要步骤

ServletOutputStream getOutputStream() throws IOException;   获取输出流

      while ((len = fin.read(tempBytes)) > 0) {

        out.write(tempBytes, 0, len);

    }

    response.getOutputStream().close();

```
InputStream fileInputStream = new FileInputStream(file);
BufferedInputStream bufferedInputStream = new BufferedInputStream(fileInputStream);
OutputStream output = response.getOutputStream();

BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(output);
response.reset();
response.setContentType("application/x-download");
response.setContentLength((int) (file.length()));
response.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");

int bytesRead;
byte[] buffer = new byte[1024];

while ((bytesRead = bufferedInputStream.read(buffer)) != -1) {
    bufferedOutputStream.write(buffer, 0, bytesRead);          //将缓冲区的数据输出到客户端浏览器
}
bufferedOutputStream.flush();
```

HttpSession [Core]

[2: store data, in memory]

package javax.servlet.http;

public interface HttpSession

The HttpSession class is used to track and manage user sessions.

It allows you to store data about a user across multiple requests and provides a mechanism to identify users during their interaction with the web application.

By default, many servlet containers (such as Tomcat, Jetty, or WildFly) store session data in memory on the server where the application is running.

You can use spring-session-data-redis to store session data in redis.

void setMaxInactiveInterval(int var1);

Specifies the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.

A negative time or zero indicates the session should never timeout.

void setAttribute(String var1, Object var2);

Object getAttribute(String var1);

void removeAttribute(String var1);

void invalidate();   Invalidate the session

ServletContext getServletContext();

annotation

package javax.servlet.annotation;

@WebServlet   对 servlet 使用（例如继承 HttpServlet）

      name="";   指定的 servlet 名称，默认为 Servlet 类名，XXServlet   【TYPE】

      urlPatterns="";   拦截路径

      loadOnStartup= -1;   设置大于0的值(默认值为-1)，表示启动应用程序后就要初始化 Servlet（而不是实例化几个 Servlet）

@WebFilter   对 filter 使用（例如继承 Filter）

      urlPatterns="";   拦截路径

WebFilter

@Target({ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface **WebFilter**

      // @WebFilter(filterName = "testFilter", urlPatterns = "/*", initParams = @WebInitParam(name = "noFilterUrl", value = "/test"))

String description() default "";

String displayName() default "";

WebInitParam[] initParams() default {};   自定义过滤器初始化参数的数组，此参数可以通过自定义过滤器 init() 的入参 FilterConfig 对象的 getInitParameter() 方法获取；

        （由于过滤器没有直接排除自定义 URL 不拦截的设定，如果我们需要在自定义拦截的 URL 中排除部分不需要拦截的 URL，可以通过将需要排除的 URL 放到 initParams 参数中再在 doFilter 方法中排除）

String filterName() default "";   自定义过滤器的名称

String smallIcon() default "";

String largeIcon() default "";

String[] servletNames() default {};

String[] value() default {};

String[] urlPatterns() default {};   自定义需要拦截的 URL，可以使用正则匹配，若没指定该参数值，则默认拦截所有请求

DispatcherType[] dispatcherTypes() default {DispatcherType.REQUEST};

boolean asyncSupported() default false;

validation （jakarta.validation-api）
annotation

Valid

package jakarta.validation;

@Target({ METHOD, FIELD, CONSTRUCTOR, PARAMETER, TYPE_USE })

@Retention(RUNTIME)

@Documented

public @interface **Valid**

  jakarta.validation.Valid

- Belongs to **Jakarta Bean Validation**
- Triggers automatic validation of method parameters or fields in objects when used with @RequestBody, @ModelAttribute,
    or other binding annotations.
- Works primarily on **single objects**.
    ```
    public ResponseEntity<?> createUser(@Valid @RequestBody UserDto user) {
        // If validation fails, an exception is thrown automatically
    }
    ```
- Commonly used in Spring MVC controllers, service layers, or entity validation.

  org.springframework.validation.annotation.Validated

- Belongs to **Spring Framework**
- Supports **validation groups**, allowing selective validation of constraints.
- Works at both **class and method levels**, including within service layers.
    ```
    @Validated(OnCreate.class) // Specify validation groups
    public ResponseEntity<?> createUser(@Valid @RequestBody UserDto user) {
    ```

```
        }
```
- Used when **multiple validation scenarios** are needed.

@Valid　依赖 hibernater-validator 的校验工具，设定对当前实体类类型参数进行校验，可以使用在内部嵌套对象中，需要导入 JSR303
规范　【METHOD，FIELD，CONSTRUCTOR，PARAMETER，TYPE_USE】

package javax.validation.constraints;

@Null　　必须为 null【METHOD，FIELD，ANNOTATION_TYPE，CONSTRUCTOR，PARAMETER，TYPE_USE】

　　　　message="error"　　错误提示消息

NotNull

package javax.validation.constraints;

@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER, TYPE_USE })

@Retention(RUNTIME)

@Repeatable(List.class)

@Documented

@Constraint(validatedBy = { })

public @interface NotNull

String message() default "{jakarta.validation.constraints.NotNull.message}";

　　Error Message

Class<?>[] groups() default { };

　　Different fields are grouped together for validation, you can specify the group category to which the fields involved in
　　the validation belong.

　　// groups={GroupA.class}

表单数据	@NotNull	@NotEmpty	@NotBlank
String s = null;	false	false	false
String s = "" ;	true	false	false
String s = " ";	true	true	false
String s = "Jock" ;	true	true	true

@NotBlank　　　字符串必须非 null，并且长度大于 0

　　　　message="error"　　　错误提示消息

　　　　groups={GroupA.class}　　对不同种类的属性进行分组，在校验时可以指定参与校验的字段所属的组类别

@NotEmpty　　　字符串必须非 null，并且长度大于 0，不能全是空格

　　　　message="error"　　　错误提示消息

　　　　groups={GroupA.class}　　对不同种类的属性进行分组，在校验时可以指定参与校验的字段所属的组类别

@AssertTrue　　必须为 true

　　　　message="error"　　　错误提示消息

@AssertFalse　　必须为 false

　　　　message="error"　　　错误提示消息

@Max　　　　必须为数字，值必须小于于指定的最大值

　　　　message="error"　　　错误提示消息

　　　　value=60　　　　最大值

@Min　　　　必须为数字，值必须大于于指定的最小值

　　　　message="error"　　　错误提示消息

        value=60   最小值

@DecimalMax   必须为数字，值必须小于于指定的最大值

        message="error"   错误提示消息

        value=60   最大值

@DecimalMin   必须为数字，值必须大于于指定的最小值

        message="error"   错误提示消息

        value=60   最小值

@Size   值的大小必须在范围内

        message="error"   错误提示消息

        max=60   最大值

        min=60   最小值

@Email   值必须是邮箱地址

        message="error"   错误提示消息

        regexp=".*"   正则

@Digits   小数的值位数 限制

        message="error"   错误提示消息

        integer=3   此数字可接受的最大整数位数

        fraction=4   此数字可接受的最大小数位数

@Past   值必须是一个过去的日期

        message="error"   错误提示消息

@Future   值必须是一个将来的日期

        message="error"   错误提示消息

@Pattern   值必须符合正则表达式

        message="error"   错误提示消息

        regex=".*"

        flags={Pattern.Flag.UNIX_LINES}

使用——

```java
@RequestMapping(value = "/addemployee")
public String addEmployee(@Valid Employee employee,Errors errors){
    System.out.println(errors.hasErrors());
    return "success.jsp";
}
```

```java
@RequestMapping(value = "/addemployee")
public String addEmployee(@Valid Employee employee, Errors errors, Model model){
    System.out.println(employee);
    if(errors.hasErrors()){
        for(FieldError error : errors.getFieldErrors()){
            model.addAttribute(error.getField(),error.getDefaultMessage());
        }
        return "addemployee.jsp";
    }
    return "success.jsp";
}
```

xml (jakarta.xml.bind-api)

parsers

DocumentBuilderFactory

package javax.xml.parsers;

public abstract class **DocumentBuilderFactory**   文件构建器工厂

public static DocumentBuilderFactory newInstance() 返回新文件构建器

public void setValidating(boolean validating)   是否指定此代码生成的解析器将在文档解析时验证文档

public void setIgnoringComments(boolean ignoreComments)   是否指定此代码生成的解析器将忽略注释

DocumentBuilder

package javax.xml.parsers;

public abstract class **DocumentBuilder**

public Document parse(File f) throws SAXException, IOException   Parses XML file.

bind

JAXBContext

package javax.xml.bind;

public abstract class **JAXBContext**   主要用来构建 JAXB 实例，并提供与 XML/Java 绑定信息相关的抽象方法

public static JAXBContext newInstance(Class... classesToBeBound) throws JAXBException   构 建 JAXB 实 例   //
JAXBContext.newInstance(com.liantuo.contrast.util.jaxb.User.class);

public static JAXBContext newInstance(String contextPath) throws JAXBException   构建 JAXB 实例，对某一个 package
包下所有的对象编组   // JAXBContext.newInstance("com.liantuo.contrast.util.jaxb");

public static JAXBContext newInstance(String contextPath, ClassLoader classLoader)   构建 JAXB 实例

public abstract Marshaller createMarshaller() throws JAXBException;   编组

public abstract Unmarshaller createUnmarshaller() throws JAXBException;   解组

public abstract Validator createValidator() throws JAXBException;   验证

Unmarshaller

package javax.xml.bind;

public interface **Unmarshaller**   解组对象

public Object unmarshal(Reader reader) throws JAXBException;   解组，可以使用 StringReader

transform

Source

package javax.xml.transform;

public interface **Source**   资源

public void setSystemId(String systemId);

public String getSystemId();

xpath

XpathFactory [CORE]

package javax.xml.xpath;

public abstract class **XPathFactory**

public static XPathFactory newInstance()

public abstract XPath newXPath();

```
XPath xPath =  XPathFactory.newInstance().newXPath();
String expression = "/class/student";
NodeList nodeList = (NodeList) xPath.compile(expression).evaluate( document, XPathConstants.NODESET);
```

XPath

package javax.xml.xpath;

public interface **XPath**

public XPathExpression compile(String expression) throws XPathExpressionException;   Compile a xpath string.

XPathExpression

package javax.xml.xpath;

public interface **XPathExpression**

public Object evaluate(Object item, QName returnType) throws XPathExpressionException;

XPathConstants

package javax.xml.xpath;

public class **XpathConstants**

public static final QName NUMBER = new QName("http://www.w3.org/1999/XSL/Transform", "NUMBER");

public static final QName STRING = new QName("http://www.w3.org/1999/XSL/Transform", "STRING");

public static final QName BOOLEAN = new QName("http://www.w3.org/1999/XSL/Transform", "BOOLEAN");

public static final QName NODESET = new QName("http://www.w3.org/1999/XSL/Transform", "NODESET");

public static final QName NODE = new QName("http://www.w3.org/1999/XSL/Transform", "NODE");

public static final String DOM_OBJECT_MODEL = "http://java.sun.com/jaxp/xpath/dom";

websocket (jakarta.ws.rs-api)

@PathParam   路劲参数

Response

public abstract class **Response** implements AutoCloseable

```
  public static enum Status implements Response.StatusType {
    private final int code;
    private final String reason;
    private final Response.Status.Family family;

    private Status(int statusCode, String reasonPhrase) {
      this.code = statusCode;
      this.reason = reasonPhrase;
      this.family = Response.Status.Family.familyOf(statusCode);
    }

    public static enum Family {
      INFORMATIONAL,
      SUCCESSFUL,
      REDIRECTION,
      CLIENT_ERROR,
      SERVER_ERROR,
```

```java
        OTHER;

        private Family() {
        }

        public static Response.Status.Family familyOf(int statusCode) {
            switch(statusCode / 100) {
            case 1:
                return INFORMATIONAL;
            case 2:
                return SUCCESSFUL;
            case 3:
                return REDIRECTION;
            case 4:
                return CLIENT_ERROR;
            case 5:
                return SERVER_ERROR;
            default:
                return OTHER;
            }
        }
    }
OK(200, "OK"),
CREATED(201, "Created"),
ACCEPTED(202, "Accepted"),
NO_CONTENT(204, "No Content"),
RESET_CONTENT(205, "Reset Content"),
PARTIAL_CONTENT(206, "Partial Content"),
MULTIPLE_CHOICES(300, "Multiple Choices"),
MOVED_PERMANENTLY(301, "Moved Permanently"),
FOUND(302, "Found"),
SEE_OTHER(303, "See Other"),
NOT_MODIFIED(304, "Not Modified"),
USE_PROXY(305, "Use Proxy"),
TEMPORARY_REDIRECT(307, "Temporary Redirect"),
PERMANENT_REDIRECT(308, "Permanent Redirect"),
BAD_REQUEST(400, "Bad Request"),
UNAUTHORIZED(401, "Unauthorized"),
PAYMENT_REQUIRED(402, "Payment Required"),
FORBIDDEN(403, "Forbidden"),
NOT_FOUND(404, "Not Found"),
METHOD_NOT_ALLOWED(405, "Method Not Allowed"),
NOT_ACCEPTABLE(406, "Not Acceptable"),
PROXY_AUTHENTICATION_REQUIRED(407, "Proxy Authentication Required"),
REQUEST_TIMEOUT(408, "Request Timeout"),
CONFLICT(409, "Conflict"),
GONE(410, "Gone"),
LENGTH_REQUIRED(411, "Length Required"),
```

PRECONDITION_FAILED(412, "Precondition Failed"),
REQUEST_ENTITY_TOO_LARGE(413, "Request Entity Too Large"),
REQUEST_URI_TOO_LONG(414, "Request-URI Too Long"),
UNSUPPORTED_MEDIA_TYPE(415, "Unsupported Media Type"),
REQUESTED_RANGE_NOT_SATISFIABLE(416, "Requested Range Not Satisfiable"),
EXPECTATION_FAILED(417, "Expectation Failed"),
PRECONDITION_REQUIRED(428, "Precondition Required"),
TOO_MANY_REQUESTS(429, "Too Many Requests"),
REQUEST_HEADER_FIELDS_TOO_LARGE(431, "Request Header Fields Too Large"),
UNAVAILABLE_FOR_LEGAL_REASONS(451, "Unavailable For Legal Reasons"),
INTERNAL_SERVER_ERROR(500, "Internal Server Error"),
NOT_IMPLEMENTED(501, "Not Implemented"),
BAD_GATEWAY(502, "Bad Gateway"),
SERVICE_UNAVAILABLE(503, "Service Unavailable"),
GATEWAY_TIMEOUT(504, "Gateway Timeout"),
HTTP_VERSION_NOT_SUPPORTED(505, "HTTP Version Not Supported"),
NETWORK_AUTHENTICATION_REQUIRED(511, "Network Authentication Required");

transaction (jakarta.transaction-api)

Transactional

package jakarta.transaction;
@Inherited
@InterceptorBinding
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface **Transactional**

Update Process:

AAA--Spring.docx#aopJdkDynamicAopProxy
AAA--Spring.docx#aopReflectiveMethodInvocation

AAA--Spring.docx#jpaCrudMethodMetadataPostProcessor
AAA--Spring.docx#transactionTransactionSynchronizationMan

AAA--SpringBoot.docx#hibernateDefaultMergeEventListener
AAA--SpringBoot.docx#hibernatetypeTypeHelper
AAA--SpringBoot.docx#hibernateCollectionType

Cascade Update：

New or deleted data will not be obtained in the cascading data obtained from the @ OneToMany annotation.

Child child1 = chidRepository.findById(1l);
chidRepository.save(child1);
Child child2 = new Child();
chidRepository.save(child2);
Parent parent= parentRepository.findById(1l);   // Does not include the new child data.

Refresh jpa cache data:
Parent parent1= parentRepository.findById(1l);   // Does not include the new child data.

Child child1 = chidRepository.findById(1l);
chidRepository.save(child1);

```
Child child2 = new Child();
chidRepository.save(child2);
Parent parent2= parentRepository.findById(1l);     // Does not include the new child data.
parent2.getChildList.add(child2);         // parent1 and parent2 will be updated.
```

1. When a transaction method calls other methods, regardless of whether the other methods have transaction annotations or not, the transaction will be passed.
   但在同一个类中，A 方法是非事务性方法，但是 B 方法是事务性方法，此时 A 调用 B 就会导致 B 的事务失效。
2. @Transactional 注解修饰的方法必须是 public 修饰的，同样的@Transactional 修饰类时，也只有类中使用 pulbic 修饰的方法才能成为事务。
3. unchecked Exception：Spring 默认抛出了未检查 unchecked 异常（继承自 RuntimeException 的异常）或者 Error 才回滚事务；
   其他异常不会触发回滚事务。如果在事务中抛出其他类型的异常，但却期望 Spring 能够回滚事务，就需要指定 rollbackFor 属性；
   若在目标方法中抛出的异常是 rollbackFor 指定的异常的子类，事务同样会回滚。
4. When a transaction method calls other methods surrounded by try catch, regardless of whether the other methods have transaction annotations or not, the transaction will be passed, and exceptions within the methods surrounded by try catch will also be rolled back

   当事务方法 try catch 调用其他方法，其他方法中抛出了异常，此时就会导致事务方法的事务注解@Transactional 失效【老版本】
   UnexpectedRollbackException 【老版本】

   ```
   @Service
   Class AServiceImpl implements IAService{
       @Transactional
       public Result A(Student s) {
        try {
               bService.save(s);
           } catch (Exception e) {
               e.printStackTrace();
           }
           return Result.ok();
       }
   }
   ```
5. 数据库引擎不支持事务
6. 未启用事务
7. Spring 的事务管理核心是动态代理，不是动态代理的 Bean 是无法进行被 Spring 进行事务管理的

TxType value() default Transactional.TxType.REQUIRED;
@Nonbinding
Class[] rollbackOn() default {};

        rollbackFor = Exception.class   表示 Exception 及其子类的异常都会触发回滚，同时不影响 Error 的回滚。
                        说明 rollbackFor = Exception.class 不会覆盖 Error 的回滚
        @Transactional(propagation = Propagation.REQUIRES_NEW)同   一个 Service 类中，spring 并不重新创建新事务，
        REQUIRES_NEW 失效，如果是两不同的 Service，就会创建新事务了
@Nonbinding
Class[] dontRollbackOn() default {};

Javax / jdk

misc

Unsafe

package jdk.internal.misc;

public final class **Unsafe**

The Unsafe class offers methods for performing low-level, unsafe operations that include:

- Memory management (allocating and freeing memory)
- Low-level synchronization (park and unpark threads)
- Direct access to fields (bypassing usual access checks)
- CAS (Compare-And-Swap) operations
- Creating instances of classes without invoking constructors

util

ArraysSupport

package jdk.internal.util;

public class **ArraysSupport**

```
public static final int SOFT_MAX_ARRAY_LENGTH = Integer.MAX_VALUE - 8;

public static int newLength(int oldLength, int minGrowth, int prefGrowth) {
    // preconditions not checked because of inlining
    // assert oldLength >= 0
    // assert minGrowth > 0

    int prefLength = oldLength + Math.max(minGrowth, prefGrowth); // might overflow
    //A. check whether the new length is a positive number.
    if (0 < prefLength && prefLength <= SOFT_MAX_ARRAY_LENGTH) {
        return prefLength;
    } else {
        // put code cold in a separate method
        return hugeLength(oldLength, minGrowth);
    }
}

private static int hugeLength(int oldLength, int minGrowth) {
    int minLength = oldLength + minGrowth;
    if (minLength < 0) { // overflow
        throw new OutOfMemoryError(
                "Required array length " + oldLength + " + " + minGrowth + " is too large");
    } else if (minLength <= SOFT_MAX_ARRAY_LENGTH) {
        return SOFT_MAX_ARRAY_LENGTH;
    } else {
        return minLength;
    }
}
```

Javax / sun

security

SecurityConstants

package sun.security.util;

public final class **SecurityConstants**　　安全常量

public static final RuntimePermission GET_CLASSLOADER_PERMISSION = new RuntimePermission("getClassLoader");　　获取 classLoader 的权限

util

BuddhistCalendar

package sun.util;

public class **BuddhistCalendar** extends GregorianCalendar

public String getCalendarType()

public String getDisplayName(int field, int style, Locale locale)

public Map<String,Integer> getDisplayNames(int field, int style, Locale locale)

Javax / org.w3c.dom

Document [CORE]

package org.w3c.dom;

public interface **Document** extends Node　　文件实体对象

　　　　javax.xml.xpath.XPathFactory

public NodeList getElementsByTagName(String tagname);　根据标签名 获取元素（Document 为根标签下，Node 为当前节点下）

public Text createTextNode(String data);

public Element createElement(String tagName) throws DOMException;

NodeList

package org.w3c.dom;

public interface **NodeList**　　节点列表

public Node item(int index);　　获取第几个节点

Node

package org.w3c.dom;

public interface **Node**

public static final short ELEMENT_NODE　　　　= 1;　　　　The most commonly used type.

public static final short ATTRIBUTE_NODE　　　= 2;

public static final short TEXT_NODE　　　　= 3;

public static final short CDATA_SECTION_NODE　　= 4;

public static final short ENTITY_REFERENCE_NODE　= 5;

public static final short ENTITY_NODE　　　　= 6;

public static final short PROCESSING_INSTRUCTION_NODE = 7;

public static final short COMMENT_NODE　　　　= 8;

public static final short DOCUMENT_NODE　　　　= 9;

public static final short DOCUMENT_TYPE_NODE　　= 10;

public static final short DOCUMENT_FRAGMENT_NODE　= 11;

public static final short NOTATION_NODE　　　　= 12;

public short getNodeType();

public String getNodeName();　　Get tag name.

public Node getFirstChild();

public NodeList getChildNodes();

public String getNodeValue() throws DOMException;

public NamedNodeMap getAttributes();

public void setTextContent(String textContent) throws DOMException;

public Node appendChild(Node newChild) throws DOMException;　Adds a new child to the current node.

NamedNodeMap

package org.w3c.dom;

public interface **NamedNodeMap**

public Node getNamedItem(String name);　　Retrieves a node specified by name.

Element

package org.w3c.dom;

public interface **Element** extends Node　元素节点

public NodeList getElementsByTagName(String name);　根据标签名 获取元素获取所有标签

DefaultHandler

package org.xml.sax.helpers;

public class **DefaultHandler**

   implements EntityResolver, DTDHandler, ContentHandler, ErrorHandler

public void startElement (

      String uri,   命名空间链接

      String localName,   没有前缀的本地名称

      String qName,   带有前缀的本地名称   // of:OfficeDocumentSettings

      Attributes attributes   元素的属性列表

)   每解析到一个元素（element）的时候都会触发这个函数，并且将这个 element 的属性 attributes 和值 value 当作参数传进
      来

public void endElement (String uri, String localName, String qName) throws SAXException

public void characters (

      char ch[],   字符数组

      int start,   字符数组开始位置

      int length   使用的长度

) throws SAXException   // new String(ch, start, length)

public void startDocument () throws SAXException

public void endDocument () throws SAXException

public void fatalError (SAXParseException e) throws SAXException

public void error (SAXParseException e) throws SAXException

public void warning (SAXParseException e) throws SAXException

XMLReaderFactory

package org.xml.sax.helpers;

public final class **XMLReaderFactory**

public static XMLReader createXMLReader() throws SAXException

public static XMLReader createXMLReader(String var0) throws SAXException

fxml

FXMLLoader

package javafx.fxml;

public class **FXMLLoader**   加载 fxml 文件

codeInspection

LocalQuickFix

package com.intellij.codeInspection;

public interface **LocalQuickFix** extends QuickFix<ProblemDescriptor>, FileModifier   本地快速检查

credentialStore

CredentialAttributes

package com.intellij.credentialStore;

public final class **CredentialAttributes**　凭证属性

public CredentialAttributes(@NotNull String serviceName, @Nullable String userName)

AllIcons

package com.intellij.icons;

public class **AllIcons**　所有图标

public static class Nodes {

      public static final Icon ModuleGroup = IconLoader.getIcon("/nodes/moduleGroup.png"); // 16x16

}

PasswordSafe

package com.intellij.ide.passwordSafe;

public abstract class **PasswordSafe** implements PasswordStorage, CredentialStore　密码安全

public static final PasswordSafe getInstance()　获取一个实例

PasswordStorage

package com.intellij.ide.passwordSafe;

public interface **PasswordStorage**　密码存储

Credentials get(@NotNull CredentialAttributes var1)　获取凭证

Notifications

package com.intellij.notification;

public interface **Notifications**　　　Notification tool

    public static class **Bus**

        public static void notify(@NotNull Notification notification)　　　Make an announcement

NotificationGroup

package com.intellij.notification;

public final class **NotificationGroup**　　　通知框，idea 右下角更新通知等

public NotificationGroup(@NotNull String displayId, @NotNull NotificationDisplayType defaultDisplayType, boolean logByDefault)

    displayId="firstplugin_id"

    defaultDisplayType=NotificationDisplayType.BALLON　气泡通知

    logByDefault=true　　　　　　　　　日志记录，开启 Event Log

public Notification createNotification(@NotNull String content, @NotNull MessageType type)

    content="xxx"

    type=MessageType.INFO　　　Notification type (MessageType, NotificationType)

NotificationDisplayType

package com.intellij.notification;

public enum **NotificationDisplayType** {

 NONE("notification.type.no.popup"),

BALLOON("notification.type.balloon"),   Expires automatically after 10 seconds
STICKY_BALLOON("notification.type.sticky.balloon"),   Needs to be closed by user.
TOOL_WINDOW("notification.type.tool.window.balloon");

NotificationGroupManager

package com.intellij.notification;

public interface **NotificationGroupManager**

       Create a Notification:

            // NotificationGroupManager.getInstance().getNotificationGroup("smp.group.notification")
               .createNotification("init project success", NotificationType.INFORMATION)
               .notify(e.getProject());

static NotificationGroupManager getInstance()

NotificationGroup getNotificationGroup(@NotNull String groupId);

openapi
actionSystem

AnAction

package com.intellij.openapi.actionSystem;

public abstract class **AnAction** implements PossiblyDumbAware   一个动作，通过 Plugin DevKit 创建

public abstract void actionPerformed(@NotNull AnActionEvent var1)   触发执行

AnActionEvent

package com.intellij.openapi.actionSystem;

public class **AnActionEvent** implements PlaceProvider<String>   动作事件

public <T> T getRequiredData(@NotNull DataKey<T> key)   获取需要的数据（Editor）

       Editor editor = e.getRequiredData(CommonDataKeys.EDITOR);

       String name = e.getRequiredData(CommonDataKeys.PSI_FILE).getViewProvider().getVirtualFile().getName();

       获取当前文件

CommonDataKeys

package com.intellij.openapi.actionSystem;

public class **CommonDataKeys**   公共数据键

public static final DataKey<Editor> EDITOR = DataKey.create("editor")   编辑器内容

public static final DataKey<PsiFile> PSI_FILE = DataKey.create("psi.File");   文件内容

components

ServiceManager

package com.intellij.openapi.components;

public class **ServiceManager**   服务管理器

public static <T> T getService(@NotNull Class<T> serviceClass)   获取服务

ApplicationComponent

package com.intellij.openapi.components;

public interface **ApplicationComponent** extends BaseComponent 在 IDEA 启动的时候初始化,整个 IDEA 中只有一个实例。

       plugins.xml 加载元素：<application-componens>

ProjectComponent

package com.intellij.openapi.components;

public interface **ProjectComponent** extends BaseComponent

       plugins.xml 加载元素：<project-componens>

ModuleComponent

package com.intellij.openapi.components;

public interface **ModuleComponent** extends BaseComponent

          plugins.xml 加载元素：<module-componens>

PersistentStateComponent

package com.intellij.openapi.components;

public interface **PersistentStateComponent**<T>　　持久状态组件，持久化数据存放

      plugin.xml >>

```xml
<extensions defaultExtensionNs="com.intellij">
 <applicationService serviceImplementation=""com.saidake.MyIdeaDemoPluginStates" />
</extensions>
```

      Datastate >>

      @State(name = "DataSetting" ,storages = @Storage("plugin.xml"))

      public class DataSetting implements PersistentStateComponent<DataState> {

         private Datastate state = new Datastate( );

         public static DataSetting getInstance() {

            return ServiceManager.getService(DataSetting.class);

         }

         @Nullable@override

         public DataState getstate( ) {

            return state;

         }

         @Override

         public void loadstate(@NotNull Datastate state) i

            this.state = state;

         }

         public ProjectConfigvo getProjectConfig(){

            return state.getProjectConfigvo();

         }

      }

T getState()

void loadState(@NotNull T var1)

注解

package com.intellij.openapi.components;

public @interface **State**　组件状态

      String name()

      Storage[] storages() default {};

public @interface **Storage**　组件存储

      String value() default "";　　　　// "my-idea-demo.xml"

editor

Editor

package com.intellij.openapi.editor;

public interface **Editor** extends UserDataHolder　　　编辑器对象

SelectionModel getSelectionModel()　　　获取选中的文本

SelectionModel

package com.intellij.openapi.editor;

public interface **SelectionModel**　　　选中文本

String getSelectedText()　　获取选中的文本

FileChooser

package com.intellij.openapi.fileChooser;

public class **FileChooser**　　文件选择器

public static VirtualFile[] chooseFiles(

　　　　　　@NotNull FileChooserDescriptor descriptor,

　　　　　　@Nullable Project project,

　　　　　　@Nullable VirtualFile toSelect　　其实打开目录　// project.getBaseDir()

　　　　　　)　　选择文件

public static void chooseFile(

　　　　　　@NotNull FileChooserDescriptor descriptor,

　　　　　　@Nullable Project project,

　　　　　　@Nullable VirtualFile toSelect,

　　　　　　@NotNull Consumer<? super VirtualFile> callback

　　　　　　)　选择文件

FileChooserDescriptor

package com.intellij.openapi.fileChooser;

public class **FileChooserDescriptor** implements Cloneable　　文件选择描述器

public FileChooserDescriptor(

　　　　　　boolean chooseFiles,

　　　　　　boolean chooseFolders,

　　　　　　boolean chooseJars,

　　　　　　boolean chooseJarsAsFiles,

　　　　　　boolean chooseJarContents,

　　　　　　boolean chooseMultiple

　　　　　　)

public void setTitle(@Nls(capitalization = Capitalization.Title) String title)　　设置标题

public void setDescription(@Nls(capitalization = Capitalization.Sentence) String description)　　设置描述

FileChooserDescriptorFactory

package com.intellij.openapi.fileChooser;

public class **FileChooserDescriptorFactory**　　文件装饰工厂

public static FileChooserDescriptor createSingleFolderDescriptor()　　指定一个单目录

Project

package com.intellij.openapi.project;

public interface **Project** extends ComponentManager, AreaInstance　项目

VirtualFile getBaseDir()　　获取基础路径

MessageDialogBuilder

package com.intellij.openapi.ui;

public abstract class **MessageDialogBuilder**<T extends MessageDialogBuilder>

public static MessageDialogBuilder.YesNo yesNo(@NotNull String title, @NotNull String message)　警告弹框

DialogWrapper

package com.intellij.openapi.ui;

public abstract class **DialogWrapper**　　对话框

protected DialogWrapper(boolean canBeParent)　　　承接父构造，true

protected abstract JComponent createCenterPanel()　　创建 中间面板

protected JComponent createNorthPanel()

protected JComponent createSouthPanel()

public void setTitle(@Nls(capitalization = Capitalization.Title) String title)　设置 title，之后 init 初始化

protected void init()　　　　　　　　初始化

public void show()　　　　　　　显示

protected void doOKAction()　　　点击 ok 动作

public boolean showAndGet()

Messages

package com.intellij.openapi.ui;

public class **Messages**　消息弹窗

public static void showMessageDialog(

　　　　　　@Nullable Project project,

　　　　　　String message,

　　　　　　@NotNull @Nls(capitalization = Capitalization.Title) String title,

　　　　　　@Nullable Icon icon

　　　　　　)　显示消息弹框

public static Icon getInformationIcon()　获取提示 icon

JBPopup

package com.intellij.openapi.ui.popup;

public interface **JBPopup** extends Disposable, LightweightWindow　　弹窗

void showCenteredInCurrentWindow(@NotNull Project var1);　当前窗口显示弹窗

ListPopup

package com.intellij.openapi.ui.popup;

public interface **ListPopup** extends JBPopup　弹窗列表

JBPopupFactory

package com.intellij.openapi.ui.popup;

public abstract class **JBPopupFactory**　弹窗创建工厂

public static JBPopupFactory getInstance()

public abstract ListPopup createListPopup(@NotNull ListPopupStep var1, int var2)　创建弹窗列表

BaseListPopupStep

package com.intellij.openapi.ui.popup.util;

public class **BaseListPopupStep**<T> extends BaseStep<T> implements ListPopupStep<T>　基础弹框列表

　　　　　显示弹窗：JBPopupFactory.getInstance().createListPopup(testPopList,

　　　　　5).ShowCenteredInCurrentWindow(e.project)

public BaseListPopupStep(@Nullable String title, @NotNull T... values)

public PopupStep onChosen(T selectedValue, boolean finalChoice)　选择事件

util

IconLoader

package com.intellij.openapi.util;

public final class **IconLoader**　　图标加载器

public static Icon getIcon(@NonNls @NotNull final String path) 根据路径获取图标

UserDataHolderBase

package com.intellij.openapi.util;

public class **UserDataHolderBase** implements UserDataHolderEx, Cloneable　　用户数据储存器

vfs

VirtualFile

package com.intellij.openapi.vfs;

public abstract class **VirtualFile** extends UserDataHolderBase implements ModificationTracker 虚拟文件

public abstract String getPath()   获取路径

public abstract String getName() 获取文件或路径名

public abstract VirtualFile getParent();　　获取父文件（根目录返回 null）

public abstract VirtualFile[] getChildren();   获取所有子文件（不是目录返回 null）

VirtualFileManager

package com.intellij.openapi.vfs;

public abstract class **VirtualFileManager** implements ModificationTracker　　虚拟文件管理器

public static String constructUrl(@NotNull String protocol, @NotNull String path) 构建文件 URL

public abstract VirtualFile refreshAndFindFileByUrl(@NotNull String url);　　刷新并通过 URL 查找文件

wm

ToolWindow

package com.intellij.openapi.wm;

public interface **ToolWindow** extends BusyObject 工具窗口

　　　　窗口初始化步骤：

　　　　　　　为主 Panel 添加 field 名称，并且设置 getter

　　　　　　　新窗口添加构造：　　public ControllerWindow(Project project, ToolWindow toolWindow)

　　　　　　　plugin.xml 中添加 Factory 工厂类

ContentManager getContentManager(); 获取内容管理器

void hide(@Nullable Runnable var1)　　隐藏　// hide(null)

ToolWindowFactory

package com.intellij.openapi.wm;

public interface **ToolWindowFactory**　　工具窗口工厂，添加工具窗口时，必须添加工具窗口工厂

void createToolWindowContent(@NotNull Project var1, @NotNull ToolWindow var2) 创建工具窗口

　　　　　需要在 window 类中构造添加参数

```
public NoteListWindow(Project project, ToolWindow toolWindow)
```

　　　　　构造一个 toolWindow：

　　　　　　public void init(ToolWindow window) {

　　　　　　}

　　　　　　@Override

　　　　　　public void createToolWindowContent(@NotNull Project project, @NotNull ToolWindow toolWindow) {

　　　　　　　ControllerAnchorWindow controllerAnchorWindow=new ControllerAnchorWindow(project,toolWindow);

　　　　　//创建 window 都西昂

　　　　　　　ContentFactory instance = ContentFactory.SERVICE.getInstance(); //获取内容工厂实例

```
            Content content = instance.createContent(controllerAnchorWindow.getMainPanel(),"",false);  //获取用于
    toolwindow 显示的内容
            toolWindow.getContentManager().addContent(content); //给 toolwindow 设置内容
        }
```

ui

EditorTextField

package com.intellij.ui;

public class **EditorTextField** extends NonOpaquePanel implements DocumentListener, TextComponent, DataProvider, TextAccessor, DocumentBasedComponent, FocusListener, MouseListener

public EditorTextField(@NotNull String text) 编辑文本域，属于 JComponent

public String getText()   获取文本

ContentManager

package com.intellij.ui.content;

public interface **ContentManager** extends Disposable, BusyObject 内容管理器

void addContent(@NotNull Content var1); 添加内容

ContentFactory

package com.intellij.ui.content;

public interface **ContentFactory** 内容工厂

public static class SERVICE

        public static ContentFactory getInstance()

Content createContent(JComponent var1, String var2, boolean var3) 创建内容

JBLabel

package com.intellij.ui.components;

public class **JBLabel** extends JLabel implements AnchorableComponent, JBComponent<JBLabel>

public JBLabel(@NotNull String text)

public void setComponentStyle(@NotNull ComponentStyle componentStyle) 设置文字样式 // setComponentStyle(UIUtil.ComponentStyle.SMALL)

public void setFontColor(@NotNull FontColor fontColor) 设置文字颜色 // setFontColor(UIUtil.FontColor.BRIGHTER)

public JBLabel withBorder(Border border) 添加 border // withBorder(JBUI.Borders.empty(0, 5, 2, 0))

uiDesigner

AbstractLayout

package com.intellij.uiDesigner.core;

public abstract class **AbstractLayout** implements LayoutManager2

public static final int DEFAULT_HGAP = 10;

public static final int DEFAULT_VGAP = 5;

util

GridBag

package com.intellij.util.ui;

public class **GridBag** extends GridBagConstraints   用于在布局时为我们的组件提供约束

public GridBag setDefaultInsets(int top, int left, int bottom, int right) 设置默认边框

public GridBag setDefaultInsets(@Nullable Insets insets) 设置默认边框

public GridBag setDefaultWeightX(double weight)

public GridBag setDefaultFill(int fill)   设置组件排列方向// setDefaultFill(GridBagConstraints.HORIZONTAL)

public GridBag nextLine()   获取下一行

public GridBag next()   获取下一个

public GridBag weightx(double weight)

UIUtil

package com.intellij.util.ui;

public class **UIUtil**　　样式工具

DomElementsInspection

package com.intellij.util.xml.highlighting;

public abstract class **DomElementsInspection**<T extends DomElement> extends XmlSuppressableInspectionTool　　dom 元素检查工具

　　　可以通过 rootElement 检查 pom 文件的依赖

public void checkFileElement(DomFileElement<T> domFileElement, final DomElementAnnotationHolder holder)　　检查文件元素