

```
;()); inetAddresses.hasMore
```

```
60         InetAddress inetAddr = (InetAddress) inetAddresses.nextElement();
61     }
62     //排除LoopbackAddress、SiteLocalAddress、LinkLocalAddress、MulticastAddress类;
63     if(!inetAddr.isLoopbackAddress() /*&& !inetAddr.isSiteLocalAddress()*/
64         && !inetAddr.isLinkLocalAddress() && !inetAddr.isMulticastAddress());
65         result.add(inetAddr);
66     }
67 }
68 }
69
70     return result;
71 }
72
73 /**
74  * 获取某个网络接口的Mac地址
75  * @return void
76  */
77 protected String getMacByInetAddress(InetAddress inetAddr){
78     try {
79         byte[] mac = NetworkInterface.getByInetAddress(inetAddr).getHardwareAddress();
80         StringBuffer stringBuffer = new StringBuffer();
81
82         for(int i=0;i<mac.length;i++){
83             if(i != 0) {
84                 stringBuffer.append("-");
85             }
86
87             //将十六进制byte转化为字符串
88             String temp = Integer.toHexString(mac[i] & 0xff);
89             if(temp.length() == 1){
90                 stringBuffer.append("0" + temp);
91             }else{
92                 stringBuffer.append(temp);
93             }
94         }
95
96         return stringBuffer.toString().toUpperCase();
97     } catch (SocketException e) {
98         e.printStackTrace();
99     }
100
101     return null;
102 }
103 }
```

获取客户Linux服务器的基本信息

```
1 public class LinuxServerInfos extends AbstractServerInfos{
2
3     @Override
4     protected List<String> getIpAddress() throws Exception {
5         List<String> result = null;
6
7         //获取所有网络接口
8         List<InetAddress> inetAddresses = getLocalAllInetAddress();
9
10        if(inetAddresses != null && inetAddresses.size() > 0){
11            result = inetAddresses.stream().map(InetAddress::getHostAddress).distinct().map(S
12        }
13
14        return result;
15    }
16
17    @Override
18    protected List<String> getMacAddress() throws Exception {
19        List<String> result = null;
20
21        //1. 获取所有网络接口
22        List<InetAddress> inetAddresses = getLocalAllInetAddress();
23
24        if(inetAddresses != null && inetAddresses.size() > 0){
25            //2. 获取所有网络接口的Mac地址
26            result = inetAddresses.stream().map(this::getMacByInetAddress).distinct().collect
27        }
28
29        return result;
30    }
31
32    @Override
33    protected String getCPUSerial() throws Exception {
34        //序列号
35        String serialNumber = "";
36
37        //使用dmidecode命令获取CPU序列号
38        String[] shell = {"/bin/bash","-c","dmidecode -t processor | grep 'ID' | awk -F ':' '
39        Process process = Runtime.getRuntime().exec(shell);
40        process.getOutputStream().close();
41
42        BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStre
43
44        String line = reader.readLine().trim();
45        if(StringUtils.isNotBlank(line)){
46            serialNumber = line;
47        }
48
49        reader.close();
50        return serialNumber;
51    }
52
53    @Override
54    protected String getMainBoardSerial() throws Exception {
55        //序列号
56        String serialNumber = "";
57
58        //使用dmidecode命令获取主板序列号
59        String[] shell = {"/bin/bash","-c","dmidecode | grep 'Serial Number' | awk -F ':' '
60        Process process = Runtime.getRuntime().exec(shell);
61        process.getOutputStream().close();
62
63        BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStre
64
65        String line = reader.readLine().trim();
66        if(StringUtils.isNotBlank(line)){
67            serialNumber = line;
68        }
69
70        reader.close();
71        return serialNumber;
72    }
73 }
```

获取客户Windows服务器的基本信息

```
1 public class WindowsServerInfos extends AbstractServerInfos{
2
3     @Override
4     protected List<String> getIpAddress() throws Exception {
5         List<String> result = null;
6
7         //获取所有网络接口
8         List<InetAddress> inetAddresses = getLocalAllInetAddress();
9
10        if(inetAddresses != null && inetAddresses.size() > 0){
11            result = inetAddresses.stream().map(InetAddress::getHostAddress).distinct().map(S
12        }
13
14        return result;
```

Beta

Beta

举报

```
17 | @Override | protected List<String> getMacAddress() throws Exception {
18 | List<String> result = null;
19 |
20 |
21 | //1. 获取所有网络接口
22 | List<InetAddress> inetAddresses = getLocalAllInetAddress();
23 |
24 | if(inetAddresses != null && inetAddresses.size() > 0){
25 |     //2. 获取所有网络接口的Mac地址
26 |     result = inetAddresses.stream().map(this::getMacByInetAddress).distinct().collect
27 | }
28 |
29 | return result;
30 | }
31 |
32 | @Override
33 | protected String getCPUSerial() throws Exception {
34 | //序列号
35 | String serialNumber = "";
36 |
37 | //使用WMIC获取CPU序列号
38 | Process process = Runtime.getRuntime().exec("wmic cpu get processorid");
39 | process.getOutputStream().close();
40 | Scanner scanner = new Scanner(process.getInputStream());
41 |
42 | if(scanner.hasNext()){
43 |     scanner.next();
44 | }
45 |
46 | if(scanner.hasNext()){
47 |     serialNumber = scanner.next().trim();
48 | }
49 |
50 | scanner.close();
51 | return serialNumber;
52 | }
53 |
54 | @Override
55 | protected String getMainBoardSerial() throws Exception {
56 | //序列号
57 | String serialNumber = "";
58 |
59 | //使用WMIC获取主板序列号
60 | Process process = Runtime.getRuntime().exec("wmic baseboard get serialnumber");
61 | process.getOutputStream().close();
62 | Scanner scanner = new Scanner(process.getInputStream());
63 |
64 | if(scanner.hasNext()){
65 |     scanner.next();
66 | }
67 |
68 | if(scanner.hasNext()){
69 |     serialNumber = scanner.next().trim();
70 | }
71 |
72 | scanner.close();
73 | return serialNumber;
74 | }
75 | }
```

添加一个自定义的可被允许的服务器硬件信息的实体类（如果校验其他参数，可自行补充）

```
1 | @Data
2 | public class LicenseCheckModel implements Serializable {
3 |
4 |
5 |     private static final long serialVersionUID = -2314678441082223148L;
6 |     /**
7 |      * 可被允许的IP地址
8 |      */
9 |     private List<String> ipAddress;
10 |
11 |     /**
12 |      * 可被允许的MAC地址
13 |      */
14 |     private List<String> macAddress;
15 |
16 |     /**
17 |      * 可被允许的CPU序列号
18 |      */
19 |     private String cpuSerial;
20 |
21 |     /**
22 |      * 可被允许的主板序列号
23 |      */
24 |     private String mainBoardSerial;
25 | }
```

添加一个License生成需要的参数的实体类

```
1 | @Data
2 | public class LicenseCreatorParam implements Serializable {
3 |
4 |     private static final long serialVersionUID = 2832129012982731724L;
5 |     /**
6 |      * 证书subject
7 |      */
8 |     private String subject;
9 |
10 |     /**
11 |      * 密钥别称
12 |      */
13 |     private String privateAlias;
14 |
15 |     /**
16 |      * 密钥密码（需要妥善保管，不能让使用者知道）
17 |      */
18 |     private String keyPass;
19 |
20 |     /**
21 |      * 访问秘钥库的密码
22 |      */
23 |     private String storePass;
24 |
25 |     /**
26 |      * 证书生成路径
27 |      */
28 |     private String licensePath;
29 |
30 |     /**
31 |      * 密钥库存储路径
32 |      */
33 |     private String privateKeysStorePath;
34 |
35 |     /**
36 |      * 证书生效时间
37 |      */
38 |     @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8")
39 |     private Date issuedTime = new Date();
40 |
41 |     /**
42 |      * 证书失效时间
43 |      */
44 |     @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8")
45 |     private Date expiryTime;
46 |
47 |     /**
48 |      * 证书生成时间
49 |      */
50 |     private Date createTime = new Date();
51 | }
```

Beta

📖

Beta

📄

📝

🔄

🚩


```
123 |         if(!checkIpAddress(expectedCheckModel.getIpAddress(),serverCheckModel.getIpAddress(){
124 |             throw new LicenseContentException("当前服务器的IP没在授权范围内");124 |
125 |         }125 |         //校验Mac地址
127 |
128 |         if(!checkIpAddress(expectedCheckModel.getMacAddress(),serverCheckModel.getMacAdress
129 |             throw new LicenseContentException("当前服务器的Mac地址没在授权范围内");129 |
130 |         }130 |
131 |         //校验主板序列号
132 |
133 |         if(!checkSerial(expectedCheckModel.getMainBoardSerial(),serverCheckModel.getMainBoa
134 |             throw new LicenseContentException("当前服务器的主板序列号没在授权范围内");134 |
135 |         }135 |
136 |         //校验CPU序列号
137 |
138 |         if(!checkSerial(expectedCheckModel.getCpuSerial(),serverCheckModel.getCpuSerial())){
139 |             throw new LicenseContentException("当前服务器的CPU序列号没在授权范围内");139 |
140 |         }else{
141 |             throw new LicenseContentException("不能获取服务器硬件信息");
142 |         }
143 |     }
144 |
145 |
146 | /**
147 |  * 重写XMLDecoder解析XML
148 |  * @param encoded XML类型字符串
149 |  * @return java.Lang.Object
150 |  */
151 | private Object load(String encoded){
152 |     BufferedInputStream inputStream = null;
153 |     XMLDecoder decoder = null;
154 |     try {
155 |
156 |         inputStream = new BufferedInputStream(new ByteArrayInputStream(encoded.getBytes(XML
157 |
158 |         decoder = new XMLDecoder(new BufferedInputStream(inputStream, DEFAULT_BUFSIZE),null
159 |         return decoder.readObject();
160 |     } catch (UnsupportedEncodingException e) {
161 |         e.printStackTrace();
162 |     } finally {
163 |         try {
164 |             if(decoder != null){
165 |                 decoder.close();
166 |             }
167 |             if(inputStream != null){
168 |                 inputStream.close();
169 |             }
170 |         } catch (Exception e) {
171 |             logger.error("XMLDecoder解析XML失败",e);
172 |         }
173 |     }
174 |
175 |     return null;
176 | }
177 |
178 | /**
179 |  * 获取当前服务器需要额外校验的License参数
180 |  * @return demo.LicenseCheckModel
181 |  */
182 | private LicenseCheckModel getServerInfos(){
183 |     //操作系统类型
184 |     String osName = System.getProperty("os.name").toLowerCase();
185 |     AbstractServerInfos abstractServerInfos = null;
186 |
187 |     //根据不同操作系统类型选择不同的数据获取方法
188 |     if (osName.startsWith("windows")) {
189 |         abstractServerInfos = new WindowsServerInfos();
190 |     } else if (osName.startsWith("linux")) {
191 |         abstractServerInfos = new LinuxServerInfos();
192 |     }else{//其他服务器类型
193 |         abstractServerInfos = new LinuxServerInfos();
194 |     }
195 |
196 |     return abstractServerInfos.getServerInfos();
197 | }
198 |
199 | /**
200 |  * 校验当前服务器的IP/Mac地址是否在可被允许的IP范围内<br/>
201 |  * 如果存在IP在可被允许的IP/Mac地址范围内，则返回true
202 |  * @return boolean
203 |  */
204 | private boolean checkIpAddress(List<String> expectedList, List<String> serverList){
205 |     if(expectedList != null && expectedList.size() > 0){
206 |         if(serverList != null && serverList.size() > 0){
207 |             for(String expected : expectedList){
208 |                 if(serverList.contains(expected.trim())){
209 |                     return true;
210 |                 }
211 |             }
212 |         }
213 |
214 |         return false;
215 |     }else {
216 |         return true;
217 |     }
218 | }
219 |
220 | /**
221 |  * 校验当前服务器硬件（主板、CPU等）序列号是否在可允许范围内
222 |  * @return boolean
223 |  */
224 | private boolean checkSerial(String expectedSerial,String serverSerial){
225 |     if(StringUtils.isNotBlank(expectedSerial)){
226 |         if(StringUtils.isNotBlank(serverSerial)){
227 |             if(expectedSerial.equals(serverSerial)){
228 |                 return true;
229 |             }
230 |         }
231 |
232 |         return false;
233 |     }else{
234 |         return true;
235 |     }
236 | }
237 | }
```

自定义KeyStoreParam，用于将公私钥存储文件存放到其他磁盘位置而不是项目中

```
1 | public class CustomKeyStoreParam extends AbstractKeyStoreParam {
2 |
3 |     /**
4 |      * 公钥/私钥在磁盘上的存储路径
5 |      */
6 |     private String storePath;
7 |     private String alias;
8 |     private String storePwd;
9 |     private String keyPwd;
10 |
11 |
12 |     public CustomKeyStoreParam(Class clazz, String resource,String alias,String storePwd,String
13 |         super(clazz, resource);13 |         this.storePath = resource;
14 |         this.alias = alias;
15 |         this.storePwd = storePwd;
16 |         this.keyPwd = keyPwd;
17 |     }
18 |
19 |
20 |     @Override
21 |     public String getAlias() {
```

```
25 | @Override
    |         26 |         public String getStorePwd() {
27 |             return storePwd;
28 |         }
29 |
30 | @Override
31 | public String getKeyPwd() {
32 |     return keyPwd;
33 | }
34 |
35 | /**
36 |  * 复写de.schlichtherle.License.AbstractKeyStoreParam的getStream()方法<br/>
37 |  * 用于将公私钥存储文件存放到其他磁盘位置而不是项目中
38 |  * @param
39 |  * @return java.io.InputStream
40 |  */
41 | @Override
42 | public InputStream getStream() throws IOException {
43 |     final InputStream in = new FileInputStream(new File(storePath));
44 |     if (null == in){
45 |         throw new FileNotFoundException(storePath);
46 |     }
47 |
48 |     return in;
49 | }
50 | }
```

最后是license的生成类，用于生成license证书

```
1 | public class LicenseCreator {
2 |
3 |     private static Logger logger = LogManager.getLogger(LicenseCreator.class);
4 |     private final static X500Principal DEFAULT HOLDER_AND_ISSUER = new X500Principal("CN=loca
5 |     private LicenseCreatorParam param;
6 |
7 |     public LicenseCreator(LicenseCreatorParam param) {
8 |         this.param = param;
9 |     }
10 |
11 | /**
12 |  * 生成License证书
13 |  * @return boolean
14 |  */
15 | public boolean generatelicense(){
16 |     try {
17 |         LicenseManager licenseManager = new CustomLicenseManager(initLicenseParam());
18 |         LicenseContent licenseContent = initLicenseContent();
19 |
20 |         licenseManager.store(licenseContent,new File(param.getLicensePath()));
21 |
22 |         return true;
23 |     }catch (Exception e){
24 |         logger.error(MessageFormat.format("证书生成失败: {0}",param),e);
25 |         return false;
26 |     }
27 | }
28 |
29 | /**
30 |  * 初始化证书生成参数
31 |  * @return de.schlichtherle.License.LicenseParam
32 |  */
33 | private LicenseParam initLicenseParam(){
34 |     Preferences preferences = Preferences.userNodeForPackage(LicenseCreator.class);
35 |
36 |     //设置对证书内容加密的秘钥
37 |     CipherParam cipherParam = new DefaultCipherParam(param.getStorePass());
38 |
39 |     KeyStoreParam privateStoreParam = new CustomKeyStoreParam(LicenseCreator.class
40 |         ,param.getPrivateKeysStorePath()
41 |         ,param.getPrivateAlias()
42 |         ,param.getStorePass()
43 |         ,param.getKeyPass());
44 |
45 |     LicenseParam licenseParam = new DefaultLicenseParam(param.getSubject()
46 |         ,preferences
47 |         ,privateStoreParam
48 |         ,cipherParam);
49 |
50 |     return licenseParam;
51 | }
52 |
53 | /**
54 |  * 设置证书生成正文信息
55 |  * @return de.schlichtherle.License.LicenseContent
56 |  */
57 | private LicenseContent initLicenseContent(){
58 |     LicenseContent licenseContent = new LicenseContent();
59 |     licenseContent.setHolder(DEFAULT HOLDER_AND_ISSUER);
60 |     licenseContent.setIssuer(DEFAULT HOLDER_AND_ISSUER);
61 |
62 |     licenseContent.setSubject(param.getSubject());
63 |     licenseContent.setIssued(param.getIssuedTime());
64 |     licenseContent.setNotBefore(param.getIssuedTime());
65 |     licenseContent.setNotAfter(param.getExpiryTime());
66 |     licenseContent.setConsumerType(param.getConsumerType());
67 |     licenseContent.setConsumerAmount(param.getConsumerAmount());
68 |     licenseContent.setInfo(param.getDescription());
69 |
70 |     //扩展校验服务器硬件信息
71 |     licenseContent.setExtra(param.getLicenseCheckModel());
72 |
73 |     return licenseContent;
74 | }
75 | }
```

添加一个生成证书的Controller，对外提供了两个RESTful接口，分别是「获取服务器硬件信息」和「生成证书」

```
1 | @RestController
2 | @RequestMapping("/license")
3 | public class LicenseCreatorController {
4 |
5 |     /**
6 |     * 证书生成路径
7 |     */
8 |     @Value("${license.licensePath}")
9 |     private String licensePath;
10 |
11 |     /**
12 |     * 获取服务器硬件信息
13 |     * @param osName 操作系统类型，如果为空则自动判断
14 |     * @return com.ccx.models.License.LicenseCheckModel
15 |     */
16 |
17 |     @RequestMapping(value = "/getServerInfos",produces = {MediaType.APPLICATION_JSON_UTF8_VALUE
18 |
19 | public LicenseCheckModel getServerInfos(@RequestParam(value = "osName",required = false) St
20 |     //操作系统类型19 |         if(StringUtils.isBlank(osName)){20 |         }
21 |     osName = System.getProperty("os.name");21 |
22 |     osName = osName.toLowerCase();
23 |
24 |     AbstractServerInfos abstractServerInfos = null;
25 |
26 |     //根据不同操作系统类型选择不同的数据获取方法
27 |     if (osName.startsWith("windows")) {
28 |         abstractServerInfos = new WindowsServerInfos();
29 |     } else if (osName.startsWith("linux")) {
```



也曾被风温柔以待

关注



0



19



0



专栏目录

Beta

Beta

举报

配置文件

客户端代码的配置文件中需要加上如下配置：

```
1 #License相关配置
2 license.subject=license_demo
3 license.publicAlias=publicCert
4 license.storePass=public_password1234
5 license.licensePath=E:/LicenseDemo/license.lic
6 license.publicKeysStorePath=E:/LicenseDemo/publicCerts.keystore
7 license.uploadPath=E:/LicenseDemo/
```

License校验类需要的参数类

```
1 @Data
2 public class LicenseVerifyParam {
3
4     /**
5      * 证书subject
6      */
7     private String subject;
8
9     /**
10      * 公钥别称
11      */
12     private String publicAlias;
13
14     /**
15      * 访问公钥库的密码
16      */
17     private String storePass;
18
19     /**
20      * 证书生成路径
21      */
22     private String licensePath;
23
24     /**
25      * 密钥库存储路径
26      */
27     private String publicKeysStorePath;
28 }
```

创建de.schlichtherle.license.LicenseManager类的单例

```
1 public class LicenseManagerHolder {
2
3     private static volatile LicenseManager LICENSE_MANAGER;
4
5     public static LicenseManager getInstance(LicenseParam param){
6         if/LICENSE_MANAGER == null){
7             synchronized (LicenseManagerHolder.class){
8                 if/LICENSE_MANAGER == null){
9                     LICENSE_MANAGER = new CustomLicenseManager(param);
10                }
11            }
12        }
13
14        return LICENSE_MANAGER;
15    }
16 }
```

License校验类，安装/校验证书：

```
1 public class LicenseVerify {
2
3     private static final Logger logger = LoggerFactory.getLogger(LicenseVerify.class);
4
5     /**
6      * 安装License证书
7      */
8     public synchronized LicenseContent install(LicenseVerifyParam param){
9         LicenseContent result = null;
10         DateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
11
12         //1. 安装证书
13         try{
14             LicenseManager licenseManager = LicenseManagerHolder.getInstance(initLicenseParam
15             licenseManager.uninstall());
16
17             result = licenseManager.install(new File(param.getLicensePath()));
18             logger.info(MessageFormat.format("证书安装成功, 证书有效期: {0} - {1}",format.format
19             }catch (Exception e){
20                 logger.error("证书安装失败! ",e);
21             }
22
23             return result;
24         }
25
26         /**
27          * 校验License证书
28          * @return boolean
29          */
30         public boolean verify(){
31             LicenseManager licenseManager = LicenseManagerHolder.getInstance(null);
32             DateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
33
34             //2. 校验证书
35             try {
36                 LicenseContent licenseContent = licenseManager.verify();
37
38                 logger.info(MessageFormat.format("证书校验通过, 证书有效期: {0} - {1}",format.format
39                 return true;
40             }catch (Exception e){
41                 logger.error("证书校验失败! ",e);
42                 return false;
43             }
44         }
45
46         /**
47          * 初始化证书生成参数
48          * @param param License校验类需要的参数
49          * @return de.schlichtherle.license.LicenseParam
50          */
51         private LicenseParam initLicenseParam(LicenseVerifyParam param){
52             Preferences preferences = Preferences.userNodeForPackage(LicenseVerify.class);
53
54             CipherParam cipherParam = new DefaultCipherParam(param.getStorePass());
55
56             KeyStoreParam publicStoreParam = new CustomKeyStoreParam(LicenseVerify.class
57             ,param.getPublicKeysStorePath()
58             ,param.getPublicAlias()
59             ,param.getStorePass()
60             ,null);
61
62             return new DefaultLicenseParam(param.getSubject()
63             ,preferences
64             ,publicStoreParam
65             ,cipherParam);
66         }
67     }
```

在项目启动的时候安装证书的Listener类：


```
3 | 4
private static Logger logger = LoggerFactory.getLogger(LicenseCheckListener.class); 5 |
6 /**
7  * 证书subject
8  */
9 @Value("${license.subject}")
10 private String subject;
11
12 /**
13  * 公钥别称
14  */
15 @Value("${license.publicAlias}")
16 private String publicAlias;
17
18 /**
19  * 访问公钥库的密码
20  */
21 @Value("${license.storePass}")
22 private String storePass;
23
24 /**
25  * 证书生成路径
26  */
27 @Value("${license.licensePath}")
28 private String licensePath;
29
30 /**
31  * 密钥库存储路径
32  */
33 @Value("${license.publicKeysStorePath}")
34 private String publicKeysStorePath;
35
36 @Override
37 public void onApplicationEvent(ContextRefreshedEvent event) {
38     //root application context 没有parent
39     ApplicationContext context = event.getApplicationContext().getParent();
40     if(context == null){
41         if(StringUtils.isNotBlank(licensePath)){
42             logger.info("+++++++ 开始安装证书 ++++++");
43
44             LicenseVerifyParam param = new LicenseVerifyParam();
45             param.setSubject(subject);
46             param.setPublicAlias(publicAlias);
47             param.setStorePass(storePass);
48             param.setLicensePath(licensePath);
49             param.setPublicKeysStorePath(publicKeysStorePath);
50
51             LicenseVerify licenseVerify = new LicenseVerify();
52             //安装证书
53             licenseVerify.install(param);
54
55             logger.info("+++++++ 证书安装结束 ++++++");
56         }
57     }
58 }
59 }
```

在项目启动的时候安装证书的Listener类:

```
1 @Component
2 public class LicenseCheckListener implements ApplicationListener<ContextRefreshedEvent> {
3
4     private static Logger logger = LoggerFactory.getLogger(LicenseCheckListener.class);
5
6     /**
7      * 证书subject
8      */
9     @Value("${license.subject}")
10    private String subject;
11
12    /**
13     * 公钥别称
14     */
15    @Value("${license.publicAlias}")
16    private String publicAlias;
17
18    /**
19     * 访问公钥库的密码
20     */
21    @Value("${license.storePass}")
22    private String storePass;
23
24    /**
25     * 证书生成路径
26     */
27    @Value("${license.licensePath}")
28    private String licensePath;
29
30    /**
31     * 密钥库存储路径
32     */
33    @Value("${license.publicKeysStorePath}")
34    private String publicKeysStorePath;
35
36    @Override
37    public void onApplicationEvent(ContextRefreshedEvent event) {
38        //root application context 没有parent
39        ApplicationContext context = event.getApplicationContext().getParent();
40        if(context == null){
41            if(StringUtils.isNotBlank(licensePath)){
42                logger.info("+++++++ 开始安装证书 ++++++");
43
44                LicenseVerifyParam param = new LicenseVerifyParam();
45                param.setSubject(subject);
46                param.setPublicAlias(publicAlias);
47                param.setStorePass(storePass);
48                param.setLicensePath(licensePath);
49                param.setPublicKeysStorePath(publicKeysStorePath);
50
51                LicenseVerify licenseVerify = new LicenseVerify();
52                //安装证书
53                licenseVerify.install(param);
54
55                logger.info("+++++++ 证书安装结束 ++++++");
56            }
57        }
58    }
59 }
```

上述安装证书的类型与服务端生成证书的类型要在同一个包路径下

尤其是LicenseCheckModel类需要在同一个包路径下，防止XML反序列化失败，我的都是在【com.bw.license】路径下。

拦截器校验license证书

```
1 @Component
2 public class LoginInterceptor implements HandlerInterceptor {
3
4     @Override
5     public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
6         LicenseVerify licenseVerify = new LicenseVerify();
7
8     //校验证书是否有效
9     boolean verifyResult = licenseVerify.verify();
10
11     if(verifyResult){
12         return true;
13     }
14 }
```

Beta

Beta

举报

```
15 |         JSONObject obj = new JSONObject();
16 |         obj.put("errcode", "0319");
17 |         obj.put("errmsg", "您的证书无效, 请核查服务器是否取得授权或重新申请证书!");
18 |         response.getWriter().print(obj);
19 |         response.getWriter().flush();
20 |         return false;
21 |     }
22 | }
23 | }
```

文章知识点与官方知识档案匹配，可进一步学习相关知识

Java技能树 首页 概览 125069 人正在系统学习中

springboot实现web系统Licence验证	08-07
教程请见https://blog.csdn.net/Lammonpeter/article/details/78602862	
SpringBoot -- 软件许可（License）证书生成+验证+应用完整流程 热门推荐 Appleyk的专栏 12万+	
一、项目目录树结构 由于时间有限，不可能在博客上花太多时间、也不可能每一个细节都说的很细，所以，下面的内容呈...	
License授权方案_司晓杰的博客	8-2
需要使用License认证,生成一个License证书,该证书中包含客户服务器信息(IP地址、MAC地址、CPU序列号、主板序列号)...	
...License证书的授权和许可到期验证_license授权_想养一只!的博客-CSD...	8-3
在这里讲解的是使用license证书的形式实现授权和许可验证(已通过测试)。主要是通过IP地址、MAC地址、CPU序列号、...	
Java反编译代码混淆代码保护知识产权保护License授权 最新发布 tianmaxingkonger的专栏 181	
目前，市场上有许多Java的反编译工具，黑客能够对这些程序进行更改，或者复用其中的程序。因此，如何保护Java程序...	
SpringBoot整合TrueLicense实现License的授权与服务器许可	m0_37947644的博客 683
license制作	
H3C License授权之实战_h3c ac license查询_艺博东的博客	8-3
1、输入授权信息—>导入&追加 2、下载授权码Excel清单模版 3、扫码或者手动写(以扫描为例) 4、使用微信扫码—>复制 3...	
软件授权与加密技术原理_license原理_带o的玫瑰123的博客	8-3
license授权机制的原理: (1)生成密钥对,包含私钥和公钥。(2)授权者保留私钥,使用私钥对授权信息诸如使用截止日期,mac ...	
java license生成验证的实现	11-21
java license生成验证的实现	
springboot增加license授权认证	醉鱼的博客 1050
环境 MacOS 10.14.6JDK1.8 源码链接: https://github.com/zuiyu-main/springboot-demo/tree/master/springboot-license 使...	
Java truelicense 实现License授权许可和验证_Genmer的博客	8-4
其中还有ftp的校验没有尝试,本demo详细介绍的是本地校验license授权机制的原理: 生成密钥对,方法有很多。我们使用true...	
license授权什么意思_到底什么是开源协议和ARM授权模式?_weixin_39963...	8-2
License是软件的授权许可,里面详尽表述了你获得代码后拥有的权利,可以对别人的作品进行何种操作,何种操作又是被禁止...	
Spring Boot项目中使用 TrueLicense 生成和验证License（服务器许可）	weixin_34111790的博客 1311
一简介 License，即版权许可证，一般用于收费软件给付费用户提供的访问许可证明。根据应用部署位置的不同，一般可以...	
无法识别服务器硬件信息,请教：无法获取服务器硬件信息	weixin_31001313的博客 526
请教各位一个问题，我的程序在本地可以正常获取服务器的硬件信息(CPU，硬盘序号等)，但是，一放到服务器上，就提示...	
java-license服务授权_java license授权_超级小龙虾的博客	8-3
SpringBoot增加license授权认证_有站网 SpringBoot整合TrueLicense生成和验证License证书(一)_哔哩哔哩_bilibili 打开cm...	
使用truelicense实现用于JAVA工程license机制（包括license生成和验证）	04-03
使用truelicense实现用于JAVA工程license机制（包括license生成和验证）	
基于springboot项目License的生成和验证	Lonels的博客 1652
1.使用JDK自带的keytool工具生成公私钥证书库.私钥用于生成License文件,公钥用于验证License文件,我这里只是给一个...	
软件许可（License）授权方案	u010249118的博客 1491
软件许可（License）授权方案	
License授权方案	weixin_46007214的博客 1万+
源码地址: https://github.com/sixj0/license解决的问题：将项目卖给其他公司，需要将jar包在客户的服务器上部署，为了避...	
java web springboot License生成器 (JAVA源码+图形界面)	09-23
保证java web ,springboot,tomcate web安全，可以现在IP，mac,自定义参数，License生成器 (JAVA源码+界面) 其中包括il...	
SpringBoot 整合 TrueLicense 实现 License 的授权与服务器许可1	08-03
前言License，即版权许可证，一般用于收费软件给付费用户提供的访问许可证明。根据应用部署位置的不同，一般可以分...	
keystore文件_HTTPS之密钥知识与密钥工具Keytool和Keystore-Explorer	weixin_39698217的博客 304
1 简介之前文章《Springboot整合https原来这么简单》讲解过一些基础的密码学知识和Springboot整合HTTPS。本文将更深...	
spring boot api文档_spring-boot-plus是易于使用，快速高效，功能丰富，开源...	weixin_39521009的博客 162
spring-boot-plus是一套集成spring boot常用开发组件的后台快速开发框架 Spring-Boot-Plus是易于使用，快速，高效，功能...	
安装vcs时license无法验证	03-26
如果您在安装vcs时遇到了无法验证license的问题，请按照以下步骤进行排除： 1. 检查网络连接 确保您的计算机已连接到...	

“相关推荐”对你有帮助么？

😞 非常没帮助 😐 没帮助 😐 一般 😊 有帮助 😄 非常有帮助

关于我们 招贤纳士 商务合作 寻求报道 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00
公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心 家长监护 网络110报警服务 中国互联网举报中心 Chrome商店下载 账号管理规范 版权与免责声明 版权申诉 出版物许可证 营业执照
©1999-2023北京创新乐知网络技术有限公司

Beta 图标库 Beta 收藏夹 草稿箱 举报

也曾被风温柔以待

关注

0

0

19

0

0

专栏目录