mpush目前支持的SPI列表 所有受支持的SPI都在mpuhs-api模块的com.mpush.api.spi包下面。

- ServerEventListenerFactory 监听Server 产生的相关事件
- BindValidatorFactory 绑定用户身份是校验用户身份是否合法
- ClientClassifierFactory 同一个用户多设备登录互踢策略，默认Android和ios会互踢
- PushHandlerFactory 接收客户端发送到服务端的上行推送消息
- ExecutorFactory 整个系统的线程池工厂，可以按自己的策略创建线程池
- CacheManagerFactory 系统使用的缓存实现，默认是redis，支持自定义
- MQClientFactory 系统使用的MQ实现，默认是redis的pub/sub，支持自定义
- ServiceRegistryFactory 服务注册组件，默认实现是zookeeper，支持自定义
- ServiceDiscoveryFactory 服务发现组件，默认实现是zookeeper，支持自定义
- PushListenerFactory 消息来源自定义，默认使用Gateway模块，支持自定义，比如使用MQ

SpiLoader.java

```java
package com.mpush.api.spi;

import java.util.*;
import java.util.concurrent.ConcurrentHashMap;

public final class SpiLoader {
    private static final Map<String, Object> CACHE = new ConcurrentHashMap<>();

    public static void clear() {
        CACHE.clear();
    }

    public static <T> T load(Class<T> clazz) {
        return load(clazz, null);
    }

    @SuppressWarnings("unchecked")
    public static <T> T load(Class<T> clazz, String name) {
        String key = clazz.getName();
        Object o = CACHE.get(key);
        if (o == null) {
            T t = load0(clazz, name);
            if (t != null) {
                CACHE.put(key, t);
                return t;
            }
        } else if (clazz.isInstance(o)) {
            return (T) o;
        }
        return load0(clazz, name);
    }

    public static <T> T load0(Class<T>  clazz, String name) {
        ServiceLoader<T> factories = ServiceLoader.load(clazz);
        T t = filterByName(factories, name);

        if (t == null) {
            factories = ServiceLoader.load(clazz, SpiLoader.class.getClassLoader());
            t = filterByName(factories, name);
        }

        if (t != null) {
            return t;
        } else {
            throw new IllegalStateException("Cannot find META-INF/services/" + clazz.getName() +
" on classpath");
        }
    }

    private static <T> T filterByName(ServiceLoader<T> factories, String name) {
        Iterator<T> it = factories.iterator();
        //不指定名称，则升序排序取第一个

        if (name == null) {
            List<T> list = new ArrayList<T>(2);
            while (it.hasNext()) {
                list.add(it.next());
            }
            if (list.size() > 1) {
                list.sort((o1, o2) -> {
                    Spi spi1 = o1.getClass().getAnnotation(Spi.class);
                    Spi spi2 = o2.getClass().getAnnotation(Spi.class);
                    int order1 = spi1 == null ? 0 : spi1.order();
                    int order2 = spi2 == null ? 0 : spi2.order();
                    return order1 - order2;
                });
            }
            if (list.size() > 0) return list.get(0);
        } else { //返回指定名称的实例
            while (it.hasNext()) {
                T t = it.next();
                if (name.equals(t.getClass().getName()) ||
                        name.equals(t.getClass().getSimpleName())) {
                    return t;
                }
            }
        }
        return null;
    }
}
```

ServerEventListenerFactory

mpush-core/resources/META-INF/services/com.mpush.api.spi.core.ServerEventListenerFactory

==这个类好像没有什么用 ....==
监听Server 产生的相关事件

```
public interface ServerEventListenerFactory extends Factory<ServerEventListener> {
    static ServerEventListener create() {
        return SpiLoader.load(ServerEventListenerFactory.class).get();
    }
}
```

```
@Spi(order = 1)
public final class DefaultServerEventListener implements ServerEventListener,
ServerEventListenerFactory {
    @Override
    public ServerEventListener get() {
        return this;
    }
}
```

BindValidatorFactory

==这个类好像没有什么用 ....==

绑定用户身份是校验用户身份是否合法

mpush-core/resources/META-INF/services/com.mpush.api.spi.handler.BindValidatorFactory

```
BindValidator bindValidator = BindValidatorFactory.create();
bindValidator.init(mPushServer);
//验证用户身份
boolean success = bindValidator.validate(message.userId, message.data);
```

```
public interface BindValidatorFactory extends Factory<BindValidator> {
    static BindValidator create() {
        return SpiLoader.load(BindValidatorFactory.class).get();
    }
}

@Spi(order = 1)
public static class DefaultBindValidatorFactory implements BindValidatorFactory {
    private final BindValidator validator = (userId, data) -> true;
    @Override
    public BindValidator get() {
        return validator;
    }
}
```

ClientClassifierFactory

同一个用户多设备登录互踢策略，默认Android和ios会互踢

mpush-common/resources/META-INF/services/com.mpush.api.spi.router.ClientClassifierFactory

```
public interface ClientClassifier {
    ClientClassifier I = ClientClassifierFactory.create();
    int getClientType(String osName);
}
```

```
public interface ClientClassifierFactory extends Factory<ClientClassifier> {
    static ClientClassifier create() {
        return SpiLoader.load(ClientClassifierFactory.class).get();
    }
}
```

```java
@Spi(order = 1)
public final class DefaultClientClassifier implements ClientClassifier, ClientClassifierFactory
{
    @Override
    public int getClientType(String osName) {
        return ClientType.find(osName).type;
    }
    @Override
    public ClientClassifier get() {
        return this;
    }
}
```

```java
public enum ClientType {
    MOBILE(1, "android", "ios"),
    PC(2, "windows", "mac", "linux"),
    WEB(3, "web", "h5"),
    UNKNOWN(-1);

    public final int type;
    public final String[] os;

    ClientType(int type, String... os) {
        this.type = type;
        this.os = os;
    }

    public boolean contains(String osName) {
        return Arrays.stream(os).anyMatch(osName::contains);
    }

    public static ClientType find(String osName) {
        for (ClientType type : values()) {
            if (type.contains(osName.toLowerCase())) return type;
        }
        return UNKNOWN;
    }

    public static boolean isSameClient(String osNameA, String osNameB) {
        if (osNameA.equals(osNameB)) return true;
        return find(osNameA).contains(osNameB);
    }
}
```

SessionContext.java
ClientLocation.java

```
public int getClientType() {
    if (clientType == 0) {
        clientType = (byte) ClientClassifier.I.getClientType(osName);
    }
    return clientType;
}
```

PushHandlerFactory

接收客户端发送到服务端的上行推送消息

mpush-core/resources/META-INF/services/com.mpush.api.spi.handler.PushHandlerFactory

ConnectionServer.java
WebsocketServer.java

```
messageDispatcher.register(Command.PUSH, PushHandlerFactory::create);
```

```
public interface PushHandlerFactory extends Factory<MessageHandler> {
    static MessageHandler create() {
        return SpiLoader.load(PushHandlerFactory.class).get();
    }
}
```

```
@Spi(order = 1)
public final class ClientPushHandler extends BaseMessageHandler<PushMessage> implements
PushHandlerFactory {
    @Override
    public PushMessage decode(Packet packet, Connection connection) {
        return new PushMessage(packet, connection);
    }
    @Override
    public void handle(PushMessage message) {
        Logs.PUSH.info("receive client push message={}", message);

        if (message.autoAck()) {
            AckMessage.from(message).sendRaw();
            Logs.PUSH.info("send ack for push message={}", message);
        }
        //biz code write here
    }
    @Override
    public MessageHandler get() {
        return this;
    }
}
```

ExecutorFactory

整个系统的线程池工厂，可以按自己的策略创建线程池

mpush-core/resources/META-INF/services/com.mpush.api.spi.common.ExecutorFactory

ExecutorFactory.create();

```
public interface ExecutorFactory {
    String PUSH_CLIENT = "push-client";
    String PUSH_TASK = "push-task";
    String ACK_TIMER = "ack-timer";
    String EVENT_BUS = "event-bus";
    String MQ = "mq";

    Executor get(String name);

    static ExecutorFactory create() {
        return SpiLoader.load(ExecutorFactory.class);
    }
}
```

```java
@Spi(order = 1)
public final class ServerExecutorFactory extends CommonExecutorFactory {

    @Override
    public Executor get(String name) {
        final ThreadPoolConfig config;
        switch (name) {
            case MQ:
                config = ThreadPoolConfig
                        .build(T_MQ)
                        .setCorePoolSize(CC.mp.thread.pool.mq.min)
                        .setMaxPoolSize(CC.mp.thread.pool.mq.max)
                        .setKeepAliveSeconds(TimeUnit.SECONDS.toSeconds(10))
                        .setQueueCapacity(CC.mp.thread.pool.mq.queue_size)
                        .setRejectedPolicy(ThreadPoolConfig.REJECTED_POLICY_CALLER_RUNS);
                break;
            case PUSH_TASK:
                return new ScheduledThreadPoolExecutor(push_task, new
NamedPoolThreadFactory(T_PUSH_CENTER_TIMER),
                        (r, e) -> {
                            throw new PushException("one push task was rejected. task=" + r);
                        }
                );
            case ACK_TIMER: {
                ScheduledThreadPoolExecutor executor = new
ScheduledThreadPoolExecutor(ack_timer,
                        new NamedPoolThreadFactory(T_ARK_REQ_TIMER),
                        (r, e) -> Logs.PUSH.error("one ack context was rejected, context=" + r)
                );
                executor.setRemoveOnCancelPolicy(true);
                return executor;
            }
            default:
                return super.get(name);
        }

        return get(config);
    }
}
```

ClientExecutorFactory.java

mpush-client/resources/META-INF/services/com.mpush.api.spi.common.ExecutorFactory

```
@Spi(order = 1)
public final class ClientExecutorFactory extends CommonExecutorFactory {

    @Override
    public Executor get(String name) {
        switch (name) {
            case PUSH_CLIENT: {
                ScheduledThreadPoolExecutor executor = new
ScheduledThreadPoolExecutor(push_client
                        , new NamedPoolThreadFactory(T_PUSH_CLIENT_TIMER), (r, e) -> r.run() //
run caller thread
                );
                executor.setRemoveOnCancelPolicy(true);
                return executor;
            }
            case ACK_TIMER: {
                ScheduledThreadPoolExecutor executor = new
ScheduledThreadPoolExecutor(ack_timer,
                        new NamedPoolThreadFactory(T_ARK_REQ_TIMER),
                        (r, e) -> Logs.PUSH.error("one ack context was rejected, context=" + r)
                );
                executor.setRemoveOnCancelPolicy(true);
                return executor;
            }
            default:
                return super.get(name);
        }
    }
}
```

CacheManagerFactory

系统使用的缓存实现，默认是redis，支持自定义

mpush-cache/resources/META-INF/services/com.mpush.api.spi.common.CacheManagerFactory

```
CacheManagerBoot.java
MPushClient.java
PushClient.java

public interface CacheManagerFactory extends Factory<CacheManager> {
    static CacheManager create() {
        return SpiLoader.load(CacheManagerFactory.class).get();
    }
}
```

```
@Spi(order = 1)
public final class RedisCacheManagerFactory implements CacheManagerFactory {
    @Override
    public CacheManager get() {
        return RedisManager.I;
    }
}
```

MQClientFactory

系统使用的MQ实现，默认是redis的pub/sub，支持自定义

mpush-cache/resources/META-INF/services/com.mpush.api.spi.common.MQClientFactory

```
MPushServer.java
MPushClient.java

public interface MQClientFactory extends Factory<MQClient> {
    static MQClient create() {
        return SpiLoader.load(MQClientFactory.class).get();
    }
}
```

测试

```
//@Spi(order = 2)
@Spi(order = -1)
public final class SimpleMQClientFactory implements com.mpush.api.spi.common.MQClientFactory {
    private MQClient mqClient = new MQClient() {
        @Override
        public void subscribe(String topic, MQMessageReceiver receiver) {
            System.err.println("subscribe " + topic);
        }

        @Override
        public void publish(String topic, Object message) {
            System.err.println("publish " + topic + " " + message);
        }
    };

    @Override
    public MQClient get() {
        return mqClient;
    }
}
```

redis

```
@Spi(order = 1)
public final class RedisMQClientFactory implements MQClientFactory {
    private ListenerDispatcher listenerDispatcher = new ListenerDispatcher();

    @Override
    public MQClient get() {
        return listenerDispatcher;
    }
}
```

ServiceRegistryFactory

服务注册组件，默认实现是zookeeper，支持自定义

mpush-zk/resources/META-INF/services/com.mpush.api.spi.common.ServiceRegistryFactory

```
MPushServer.java
MPushClient.java

public interface ServiceRegistryFactory extends Factory<ServiceRegistry> {
    static ServiceRegistry create() {
        return SpiLoader.load(ServiceRegistryFactory.class).get();
    }
}
```

测试

```
//@Spi(order = 2)
@Spi(order = -1)
public final class SimpleRegistryFactory implements ServiceRegistryFactory {
    @Override
    public ServiceRegistry get() {
        return FileSrd.I;
    }
}
```

zookeeper

```
@Spi(order = 1)
public final class ZKRegistryFactory implements ServiceRegistryFactory {
    @Override
    public ServiceRegistry get() {
        return ZKServiceRegistryAndDiscovery.I;
    }
}
```

ServiceDiscoveryFactory

服务发现组件，默认实现是zookeeper，支持自定义

mpush-zk/resources/META-INF/services/com.mpush.api.spi.common.ServiceDiscoveryFactory

```
MPushServer.java
MPushClient.java
ConnClientBoot.java
PushClient.java

public interface ServiceDiscoveryFactory extends Factory<ServiceDiscovery> {
    static ServiceDiscovery create() {
        return SpiLoader.load(ServiceDiscoveryFactory.class).get();
    }
}
```

测试

```
//@Spi(order = 2)
@Spi(order = -1)
public final class SimpleDiscoveryFactory implements ServiceDiscoveryFactory {
    @Override
    public ServiceDiscovery get() {
        return FileSrd.I;
    }
}
```

zookeeper

```
@Spi(order = 1)
public final class ZKDiscoveryFactory implements ServiceDiscoveryFactory {
    @Override
    public ServiceDiscovery get() {
        return ZKServiceRegistryAndDiscovery.I;
    }
}
```

PushListenerFactory

消息来源自定义，默认使用Gateway模块，支持自定义，比如使用MQ

mpush-core/resouces/META-INF/services/com.mpush.api.spi.push.PushListenerFactory

```
PushCenter.java

public interface PushListenerFactory<M extends IPushMessage> extends Factory<PushListener<M>> {
    @SuppressWarnings("unchecked")
    static <M extends IPushMessage> PushListener<M> create() {
        return (PushListener<M>) SpiLoader.load(PushListenerFactory.class).get();
    }
}
```

GatewayPushListener.java

```
@Spi(order = 1)
public final class GatewayPushListener implements PushListener<GatewayPushMessage>,
PushListenerFactory<GatewayPushMessage> {
    @Override
    public PushListener<GatewayPushMessage> get() {
        return this;
    }
}
```

MQPushListener.java

此类暂时没用到，可能是用在推送端基于MQ来消费推送的mpush server反馈结果

```java
@Spi(order = 2)
public final class MQPushListener implements PushListener<MQPushMessage>,
PushListenerFactory<MQPushMessage> {
    private final MQClient mqClient = new MQClient();

    @Override
    public void init(MPushContext context) {
        mqClient.init();
        MQMessageReceiver.subscribe(mqClient, ((MPushServer) context).getPushCenter());
    }


    @Override
    public void onSuccess(MQPushMessage message, Object[] timePoints) {
        //publish messageId to mq:[success/queue]
        mqClient.publish("/mpush/push/success", message);
    }

    @Override
    public void onAckSuccess(MQPushMessage message, Object[] timePoints) {
        //publish messageId to mq:[success/queue]
        mqClient.publish("/mpush/push/success", message);
    }

    @Override
    public void onBroadcastComplete(MQPushMessage message, Object[] timePoints) {
        //publish messageId to mq:[broadcast/finish/queue]
        mqClient.publish("/mpush/push/broadcast_finish", message);
    }

    @Override
    public void onFailure(MQPushMessage message, Object[] timePoints) {
        //publish messageId to mq:[failure/queue], client can retry
        mqClient.publish("/mpush/push/failure", message);
    }

    @Override
    public void onOffline(MQPushMessage message, Object[] timePoints) {
        //publish messageId to mq:[offline/queue], client persist offline message to db
        mqClient.publish("/mpush/push/offline", message);
    }

    @Override
    public void onRedirect(MQPushMessage message, Object[] timePoints) {
        //publish messageId to mq:[route/change/queue], client should be try again
        mqClient.publish("/mpush/push/route_change", message);
    }

    @Override
    public void onTimeout(MQPushMessage message, Object[] timePoints) {
        //publish messageId to mq:[ack/timeout/queue], client can retry
        mqClient.publish("/mpush/push/ack_timeout", message);

    }
    @Override
    public PushListener<MQPushMessage> get() {
        return this;
    }
}
```